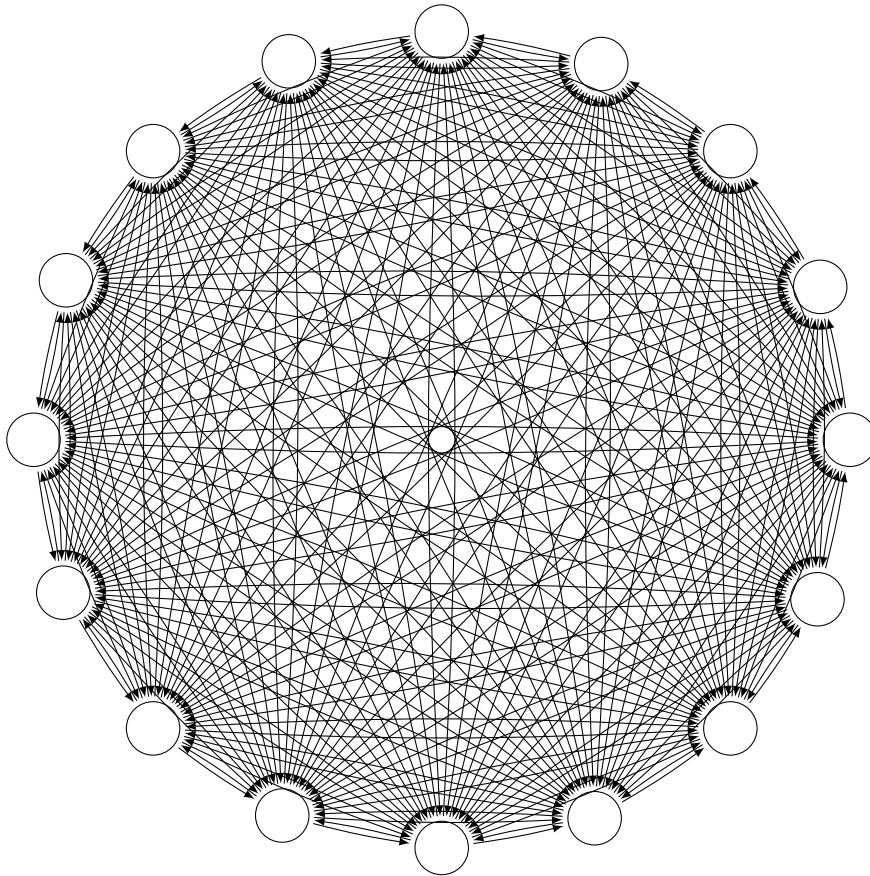


# The Learning Problem, Classification case



PONTIFICIA UNIVERSIDAD JAVERIANA

FACULTAD DE CIENCIAS

DEPARTAMENTO DE MATEMÁTICAS

BOGOTÁ - COLOMBIA

2021



---

PONTIFICIA UNIVERSIDAD JAVERIANA

FACULTAD DE CIENCIAS  
DEPARTAMENTO DE MATEMÁTICAS

## **The Learning Problem, Classification case**

Jerome Pierre V Marquet

Tesis presentada al Departamento de Matemáticas para optar al grado de  
Magister en Matemática

---

Dirigida por:  
Lina Maria Acosta Avena, Ph.D.

Bogotá - Colombia  
January 21, 2022

## **NOTA DE ADVERTENCIA**

“La universidad no se hace responsable por los conceptos emitidos por sus alumnos en sus trabajos de tesis. Solo velará por que no se publique nada contrario al dogma y a la moral católica y porque las tesis no contengan ataques personales contra alguna persona, antes bien se vea en ellas el anhelo de buscar la verdad y la justicia”.

Artículo 23 de la Resolución N°13 de Julio de 1996

THE LEARNING PROBLEM, CLASSIFICATION CASE

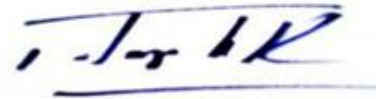
Jerome Marquet

APROBADO:



---

Jhon Jairo Sutachan Rubio, Ph.D.  
Director de Posgrado  
Facultad de Ciencias



---

Alba Alicia Trespalacios Rangel, Ph.D.  
Decana  
Facultad de Ciencias

Bogotá, febrero de 2022



THE LEARNING PROBLEM, CLASSIFICATION CASE

Jerome Marquet

Advisors:

Lina Maria Acosta Avena Lina Maria Acosta Avena

Jury:

Marisol García Peña, ph.D. Marisol García Peña

Camilo Ortiz Astorquiza, ph.D. Camilo Ortiz Astorquiza

A thesis presented for the degree of  
Master in Mathematics

Department of Mathematics  
Pontificia Universidad Javeriana  
Colombia

# Contents

|   |           |
|---|-----------|
| <b>Agradecimientos</b>  | <b>7</b>  |
| <b>Introducción</b>   | <b>11</b> |
| <b>1 Foundation of learning</b>                                     | <b>13</b> |
| 1.1 Components of the learning problem . . . . .                    | 13        |
| 1.1.1 Data generation process: . . . . .                            | 13        |
| 1.1.2 Learner's output and measure of success: . . . . .            | 14        |
| 1.1.3 Empirical Risk Minimization (ERM): . . . . .                  | 14        |
| 1.2 Is learning Feasible . . . . .                                  | 15        |
| 1.2.1 Hypothesis class ( $\mathcal{H}$ ): . . . . .                 | 15        |
| 1.2.2 Probably Approximately Correct (PAC): . . . . .               | 16        |
| 1.2.3 Hoeffding's Inequalities: . . . . .                           | 17        |
| 1.3 Finite classes are PAC learnable . . . . .                      | 19        |
| 1.4 Infinite class are PAC learnable . . . . .                      | 20        |
| 1.4.1 Growth Function . . . . .                                     | 20        |
| 1.4.2 Bound for the growth function: . . . . .                      | 21        |
| 1.4.3 VC-dimension: . . . . .                                       | 26        |
| 1.4.4 VC generalization bound: . . . . .                            | 27        |
| 1.4.5 Sample complexity: . . . . .                                  | 33        |
| <b>2 Neural network</b>   | <b>35</b> |
| 2.1 General description of the artificial neural network: . . . . . | 35        |
| 2.2 Notations: . . . . .  | 37        |
| 2.3 Gradient descent and stochastic gradient descent: . . . . .     | 38        |
| 2.4 Equation for backpropagation: . . . . .                         | 39        |
| 2.5 Backpropagation algorithm: . . . . .                            | 41        |
| 2.6 Activation functions: . . . . .                                 | 42        |
| 2.7 Cost functions: . . . . .                                       | 42        |
| 2.8 L2 Regularization: . . . . .                                    | 44        |
| 2.9 Overfitting and validation set . . . . .                        | 46        |
| <b>3 Analysis of a dataset</b>                                      | <b>47</b> |
| 3.1 The MNIST dataset . . . . .                                     | 47        |
| 3.2 Results . . . . .   | 49        |

|          |  |           |
|----------|--|-----------|
| <b>4</b> | <b>Conclusions and Future Directions</b> | <b>56</b> |
| 4.1      | Conclusion . . . . .                     | 56        |
| 4.2      | Steps further . . . . .                  | 57        |

## **The Learning Problem, Classification case**

Jerome Pierre V Marquet

Departamento de Matemáticas,  
Facultad de Ciencias  
Pontificia Universidad Javeriana

### **Resumen**

“Machine learning” o aprendizaje automático se refiere a un conjunto de algoritmos destinados a hacer las predicciones más precisas posibles de una variable de salida basada en los valores de algunas variables de entrada. Cuando la variable de salida es categórica, el proceso de generación de una predicción se llama clasificación. Problemas de este tipo ocurren muy a menudo en la práctica (por ejemplo: predecir el género de una persona, si un cliente de un banco va a incumplir su hipoteca, o si el precio de una acción en particular va a subir o bajar). Un problema importante en la clasificación es el reconocimiento de imágenes. Por ejemplo, hay reconocimiento facial en las redes sociales, apoyo diagnóstico en imágenes médicas o descubrimiento de productos (encontrar un producto similar usando una imagen de referencia). Presentamos y resolvimos el problema de aprendizaje en clasificación desde una perspectiva teórica y práctica. Primero, explicamos lo que entendemos por “aprender” para un algoritmo. Introducimos la notación matemática de las diferentes partes del problema de aprendizaje en clasificación, y demostramos matemáticamente que el aprendizaje es factible bajo nuestra definición de “capacidad de aprendizaje”. A continuación, nos concentramos en un método de aprendizaje llamado red neuronal artificial. Este método es una forma muy flexible de modelar fenómenos altamente no lineales. Introdujimos la notación matemática, y demostramos las diferentes ecuaciones que rigen su funcionamiento (el algoritmo de backpropagation en particular). Luego, mostramos cómo podemos implementar el método de red neuronal en el paquete de software R. Por último, presentamos las actuaciones del programa en un famoso conjunto de datos de prueba, a saber, la base de datos MNIST, y comparamos nuestros resultados con los mencionados en el sitio web de Lecun, que estudió ampliamente esta base de datos.

**Palabras Claves:** Algoritmo de retropropagación, clasificación, reconocimiento de imágenes, capacidad de aprendizaje, aprendizaje automático, red neuronal.



# The Learning Problem, Classification Case

Jerome Pierre V Marquet

Department of Mathematics  
Faculty of Sciences  
Pontificia Universidad Javeriana

## Abstract

Machine learning refers to a set of algorithms aimed at making the most accurate possible predictions of an output variable based on the values of some input variables. When the output variable is categorical, the task of generating a prediction is called classification. Classification problems occur very frequently in practice (e.g.: predicting the gender of a person, if a client of a bank is going to default on his mortgage, or if a particular share price is going to go up or down). A major problem in classification is image recognition. There is for example face recognition on social networks, diagnostic support in medical imaging, or product discoverability (finding a similar product using a reference image). We presented and solved the classification learning problem from a theoretical and practical perspective. First, we explained what we mean by “learning” for an algorithm. We introduced the mathematical notation of the different parts of the classification learning problem, and we mathematically demonstrated that learning is feasible under our definition of “learnability”. Next, concentrated on one method of learning named artificial neural network. This method is a very flexible way of modeling highly nonlinear phenomena. We introduced the mathematical notation, and we demonstrated the different equations that govern its functioning (the backpropagation algorithm in particular). Then, we showed how we can implement the neural network method in the R software package. Finally, we presented the performances of the program on a famous testing data set, namely the MNIST database, and compared our results with those mentioned on the web site of Yann Lecun, who studied this database extensively.

**Key Words:** Backpropagation algorithm, Classification, Image recognition, Learnability, Machine Learning, Neural Network.

# Introduction

The aim of the present work is to state a problem, namely the learning problem limited to the classification case, to define it properly, to model it conceptually, and finally to propose one way of solving it. The final results will be a set of definitions setting out the problem, a theoretical development showing that it is solvable and a computer program that brings a concrete solution to a practical instance of the problem. As written above, the problem at hand is the learning problem in classification, the method for solving it will be artificial neural networks and the practical instance will be handwritten digit recognition. Very broadly stated, the learning problem consists conceptually in using empirical instances of a data generating process in order to predict new instances of an output variable from the same process in particular when there exists no analytical way to model the phenomenon at hand.

When the output variable of a problem is categorical, the task of generating a prediction is called classification. Classification problems occurs very frequently in practice. We may want to predict the gender of a person, if a client of a bank is going to default on his mortgage, or if a particular share price is going to go up or down tomorrow.

A major problem in classification is image recognition. There is for example face recognition on social network, diagnostic support in medical imaging, or product discoverability (finding a similar product using a reference image).

The thesis is organized the following way. First, we are going to explain what we mean by “learning” for an algorithm. We will introduce the mathematical notation of the different parts of the classification learning problem, and we will mathematically demonstrate that learning is feasible under our definition of “learnability”. Next, we will concentrate on one method of learning, namely the neural network method. We will introduce the mathematical notation and we will demonstrate the different equations that govern its functioning. Then, we will show how we can implement the neural network method in the R software package. Finally, we will present the performances of the program on a famous testing data set, namely the MNIST data base, and compare our results with those mentioned by Lecun, who studied this data base extensively on his web site and in his paper Lecun *et al.* (1998b).

# Chapter 1

## Foundation of learning

Machine learning refers to a set of algorithms aimed at making the most accurate possible predictions of an output variable based on the values of some input variables. It differs from general statistical modeling which focuses on inference, trying to understand the relationship between inputs and outputs by building a mathematical structure of causes and effects.

In order to define properly the learning problem, we can give a name and symbol to its main components.

There is a set  $\mathcal{S}$  of data consisting of input-output examples  $\{(x_1, y_1), \dots, (x_m, y_m)\}$ .

Those data points were generated by an unknown function describing the relationship between the inputs and the outputs. This function  $f$  is our unknown target function.

Every input  $x$ , is coming from the input space  $\mathcal{X}$  (set of all possible inputs) and is related to a particular output  $y$  of the output space  $\mathcal{Y}$  (set of all possible outputs). Hence, the unknown target function  $f$  goes from  $\mathcal{X}$  to  $\mathcal{Y}$ , and the outputs  $y$  is equal to  $f(x)$ .

A learning algorithm uses the data set  $\mathcal{S}$ , to find a function  $h : \mathcal{X} \rightarrow \mathcal{Y}$  that approximates  $f$ . The function  $h$  is chosen from a set  $\mathcal{H} = \{h_1, h_2, \dots\}$  called hypothesis set, or hypothesis class. The goal of the algorithm is to find the function  $h$  that best matches  $f$  on the training examples hoping that it will continue to match  $f$  on new data.

### 1.1 Components of the learning problem

In this section we are going to describe a formal model aimed to capture the learning problem. We adopted the notation of Shalev-Shwartz (2014) and some of the interpretations of Abu-Mostafa *et al.* (2012).

#### 1.1.1 Data generation process:

In the classification learning problem, the data are divided in two parts, the inputs  $\mathcal{X}$  and the outputs  $\mathcal{Y}$ .

The set  $\mathcal{X}$  contains the objects we may wish to label and the set  $\mathcal{Y}$  contains the possible outputs associated with the inputs in  $\mathcal{X}$ . Every element of  $\mathcal{X}$  is linked with a certain element of  $\mathcal{Y}$ .

In the classification case,  $\mathcal{Y}$  is the label set  $\{0, 1\}$  or  $\{-1, 1\}$ .

If we sample a finite sequence of pairs in  $\mathcal{X} \times \mathcal{Y}$ , we obtain the set  $\mathcal{S} = \{(x_1, y_1), \dots, (x_m, y_m)\}$  of labeled examples called training data set. This is the input that the learner has access to along with the corresponding “true” labels.

We assume that the instances of the training set  $\mathcal{S}$  are generated by some unknown probability distribution  $\mathcal{D}$ . We also assume that there exists some “correct” but unknown target function  $f : \mathcal{X} \rightarrow \mathcal{Y}$  such that  $y_i = f(x_i)$  for all  $i \in \{1, \dots, m\}$ . Each pair in the training data set  $\mathcal{S}$  is generated by first sampling a point  $x_i$  according to  $\mathcal{D}$  and then defining its output with  $f$ .

### 1.1.2 Learner’s output and measure of success:

As we mentioned in the introduction of the section, the objective of a learning algorithm is to find a function that best approximates the target function. We name such a function  $h : \mathcal{X} \rightarrow \mathcal{Y}$ , predictor, hypothesis or classifier.

Once we have defined a predictor, we need to be able to assess its performance at the task of predicting the output of a new domain point.

We define the error of a predictor to be the probability that it does not predict the correct output on a random data point generated by the data-generating distribution  $\mathcal{D}$ . That is, the error of  $h$  is the probability to draw a random instance  $x$ , according to the distribution  $\mathcal{D}$ , such that  $h(x) \neq f(x)$ .

Formally, we define the error of a predictor  $h : \mathcal{X} \rightarrow \mathcal{Y}$ , to be

$$C_{\mathcal{D},f}(h) \stackrel{\text{def}}{=} \mathbb{P}_{x \sim \mathcal{D}} [h(x) \neq f(x)] \stackrel{\text{def}}{=} \mathcal{D}(\{x : h(x) \neq f(x)\})$$

The subscript  $(\mathcal{D}, f)$  indicate that the error is measured with respect to the probability distribution  $\mathcal{D}$  and the correct labeling function  $f$ .

$C_{\mathcal{D},f}(h)$  has several synonymous names such as generalization error, risk, or true error.

We use the letter  $C$  for the error since we view the error as a loss function or as a cost function.

The  $f$  subscript is omitted in the rest of the document.

Given that the error is defined as a probability on the unknown probability distribution  $\mathcal{D}$  over the input space  $\mathcal{X}$ , we cannot directly measure  $C_{\mathcal{D}}(h)$ . The best we can do is to measure a related error on the training examples which we have access to. Next, we are going to define such a measure called the empirical risk.

### 1.1.3 Empirical Risk Minimization (ERM):

A learning algorithm receives as input a training set  $\mathcal{S}$ , sampled from the unknown distribution  $\mathcal{D}$  and labeled by the target function  $f$ , and it outputs a predictor rule  $h_{\mathcal{S}} : \mathcal{X} \rightarrow \mathcal{Y}$  that depends

on the particular training set used. The goal of the algorithm is to find  $h_S$  that minimizes the error with respect to the unknown  $\mathcal{D}$  and  $f$ . Since the learner does not know  $\mathcal{D}$  and  $f$ , the true error is not directly available. What is available is the so-called training error (also referred to as empirical risk), the error that the predictor incurs over the training sample.

Formally, and in the case of a classifier, the empirical risk is defined to be,

$$C_S(h) \stackrel{\text{def}}{=} \frac{|\{i \in \{1, \dots, m\} : h(x_i) \neq y_i\}|}{m}$$

Since the training sample is the “vision” of the world that the learner has available, it makes sense to search for solutions that works well on those data. This learning paradigm is called Empirical Risk Minimization (ERM).

In the next section we will see how  $C_S$  and  $C_{\mathcal{D}}$  can be used to define “learnability”.

## 1.2 Is learning Feasible

Now that we have defined the learning problem (finding a predictor that approximates well the target function) we may ask ourselves if this problem has a solution. We first have to define what we mean by a “solution” in the context of learning and then to show that under this paradigm the learning problem indeed has a solution.

### 1.2.1 Hypothesis class ( $\mathcal{H}$ ):

Something may go wrong with ERM. There is always some probability that the sampled training data happens to be very nonrepresentative of the underlying  $\mathcal{D}$ . In this case, it may happen that a given predictor has an excellent performance on the training set but behaves very poorly on new previously unseen data. This problem is called “overfitting”. Intuitively, overfitting occurs when our hypothesis fits the training data “too well” or when the model is learning the training data “by heart” instead of understanding them.

If we do not want to give up on ERM we have to find a way to rectify it by searching conditions under which it does not overfit. A common solution is to apply the ERM learning rule (find  $h$  that minimizes  $C_S(h)$ ) over a restricted search space. The learner should choose in advance (before seeing the data) a set of predictors. This set is called hypothesis class or hypothesis set and is denoted by  $\mathcal{H}$ . Each element  $h$  of  $\mathcal{H}$  is a function from  $\mathcal{X}$  to  $\mathcal{Y}$ .

For a given class  $\mathcal{H}$ , and a training sample  $\mathcal{S}$ , the  $ERM_{\mathcal{H}}$  rule uses the  $ERM$  rule to choose a predictor  $h \in \mathcal{H}$  with the lowest possible error over  $\mathcal{S}$ .

Formally,  $ERM_{\mathcal{H}}(\mathcal{S}) \in \operatorname{argmin}_{h \in \mathcal{H}} C_S(h)$ , where  $\operatorname{argmin}_{h \in \mathcal{H}} C_S(h)$  is the set of hypotheses in  $\mathcal{H}$  that achieve minimum value of  $C_S$  over  $\mathcal{H}$ .

The simplest type of restriction on a class is imposing an upper bound on its size,  $|\mathcal{H}|$ .

Now that we justified the use of the hypothesis class, we can define what we mean by learning.

### 1.2.2 Probably Approximately Correct (PAC):

Intuitively we can consider that an algorithm learnt something if using the information contained “inside” a training data set, it is able to infer the same information “outside” this training data set.

We can make the comparison with a student studying for his calculus exam with exercises the teacher solved during the semester. The fact that the student can solve those exercises for which he knows the correct answer, do not necessarily mean that he learnt something. The real test is an exam with exercises for which he does not know the correct answer. Of course, we hope that if the student performed well on the training exercises, he will also perform well on the exam exercises. But it could be that the student did not understand the logic behind the exercises and simply memorizes the correct answers when he was performing well on the training exercises.

The same holds with algorithms. We would like to find an algorithm that performs well on the training examples ( $C_S$  small, or  $C_S \approx 0$ ) hoping that it will still preform well on any example of  $\mathcal{X}$  ( $C_D$  small, or  $C_D \approx 0$ ). However, finding an algorithm that performs well on the training examples does not imply that it will perform well on any example especially on new previously unseen data. The algorithm could just have memorized the “correct answers” of the training data set, and in this case it did not learn anything.

Consequently, learning is not possible in a deterministic paradigm.

However, if we place ourselves in a probabilistic framework, we could ask ourselves if there is a good enough probability that what we learnt from the training examples tells us something about the target function  $f$  on rest of the input space. In this framework, learning would be “finding something likely about  $f$  outside of  $\mathcal{S}$ ”.

Using the concept of training and empirical errors previously defined, the idea of “good performance on the training examples implies good performance on the whole input space” could be expressed as “ $C_S$  small implies  $C_D$  small”. Hence learning would be possible if we could find a hypothesis  $h$  such that  $C_S(h) \approx 0 \implies C_D(h) \approx 0$ .

This in turn can be divided into two parts:

1. Can we have  $C_D$  arbitrarily closed to  $C_S$  or equivalently, is  $C_S \approx C_D$ ?
2. Can we make  $C_S$  small enough or equivalently, is  $C_S \approx 0$ ?

Consequently, the task of learning would become “choosing a hypothesis  $h$  such that that  $\mathbb{P}[C_S(h) \approx C_D(h)]$  is high enough”. The only assumption we make in this probabilistic framework is that the examples in  $\mathcal{S}$  are generated independently from the same unknown distribution.

The second part of the problem  $C_S \approx 0$  is not affected by uncertainty since we can calculate  $C_S(h)$  for a certain hypothesis  $h$  and a certain data set  $\mathcal{S}$ .

Given that “ $\approx$ ” is not a mathematically well-defined concept we will rather express it as “ $\mathbb{P}[|C_S(h) - C_D(h)| \text{ small}]$  is high” or equivalently “ $\mathbb{P}[|C_S(h) - C_D(h)| \text{ high}]$  is small”.

A more formal notation is  $\mathbb{P}[|C_S(h) - C_D(h)| > \varepsilon]$  is small  $\forall \varepsilon > 0$ .

We can now formally define what we mean by learning with the concept of “Probably Approximately Correct”.

We took the definition of PAC from Shalev-Shwartz (2014) and adapted it with the explanation of Abu-Mostafa *et al.* (2012) on subsection 1.3.2 of “Learning from Data” which does not explicitly define PAC.

**Definition 1.1.** *Probably Approximately Correct (PAC)*

A hypothesis class  $\mathcal{H}$  is PAC learnable if there exists a function  $m_{\mathcal{H}}: (0, 1)^2 \rightarrow \mathbb{N}$ , and a learning algorithm  $\mathcal{A}$  such that:

for every tolerance level  $\varepsilon \in (0, 1)$ , and every confidence level  $\delta \in (0, 1)$ ,

for every distribution  $\mathcal{D}$  over  $\mathcal{X}$ ,

for every labeling function  $f: \mathcal{X} \rightarrow \mathcal{Y}$ ,

when running  $\mathcal{A}$  on a training set  $\mathcal{S}$  of  $m \geq m_{\mathcal{H}}$  i.i.d. examples, generated by  $\mathcal{D}$  and labeled by  $f$ , the algorithm returns a hypothesis  $h_S$  such that with probability of at least  $1 - \delta$  (over the choice of the  $m$  training examples),  $|C_S(h_S) - C_D(h_S)| \leq \varepsilon$ , or equivalently:  $\mathbb{P}[|C_S(h_S) - C_D(h_S)| > \varepsilon] \leq \delta$ .

Learning from a data set  $\mathcal{S}$  produces a hypothesis  $h_S$  to approximate the unknown target function  $f$ . If learning is successful,  $h_S$  approximates  $f$  well on the whole input space  $\mathcal{X}$ , which means that  $C_D(h_S) \approx 0$ . However, according to the probabilistic framework what we evaluate  $\mathbb{P}[C_S(h_S) \approx C_D(h_S)]$ . Consequently, we still have to make  $C_S(h_S) \approx 0$  in order to conclude that  $C_D(h_S) \approx 0$  with a certain probability.

We cannot guarantee to find  $h_S$  such that  $C_S(h_S) \approx 0$ , but we can check if it is the case.

Finally, we have traded the condition of learning  $C_D(h_S) \approx 0$  that we cannot ascertain for a condition  $C_S(h_S) \approx 0$  which we can ascertain given we can show that  $\mathbb{P}[C_S(h) \approx C_D(h)]$  is high.

The question  $C_S(h_S) \approx C_D(h_S)$  addresses the problem of generalization, but it does not guarantee by itself learning. If  $C_S(h_S)$  is high, the generalization condition, “ $\mathbb{P}[C_S(h) \approx C_D(h)]$  high” only tells us with high confidence that  $h_S$  is a bad result.

Next, we present two results that we call Hoeffding inequality 1 and 2 that will help us address the concept of generalization.

### 1.2.3 Hoeffding’s Inequalities:

<sup>2</sup>As we stated previously, we need to address mathematically the condition “ $\mathbb{P}[C_S(h) \approx C_D(h)]$  is high”. Given that “ $\approx$ ” is not a mathematically well-defined concept we will rather express it

as “ $\mathbb{P}[|C_S(h) - C_D(h)| \text{ small}]$  is high” or equivalently “ $\mathbb{P}[|C_S(h) - C_D(h)| \text{ high}]$  is small”. Consequently, we need an inequality that bound the probability  $\mathbb{P}[|C_S(h) - C_D(h)| > \varepsilon] \forall \varepsilon > 0$ .

Fortunately, such inequality exists, it is the Hoeffding’s inequality. We took the following formulation of Hoeffding’s Inequality from Shalev-Shwartz (2014).

**Teorema 1.1. Hoeffding’s Inequality 1**

Let  $Z_1, Z_2, \dots, Z_m$  be a sequence of i.i.d. random variables and let  $\bar{Z} = \frac{1}{m} \sum_{i=1}^m Z_i$ . Assume that  $\mathbb{P}[a \leq Z_i \leq b] = 1 \forall i$ . Then,  $\forall \varepsilon > 0$

$$\mathbb{P}\left[\left|\frac{1}{m} \sum_{i=1}^m Z_i - \mathbb{E}[\bar{Z}]\right| > \varepsilon\right] \leq 2e^{-2m\varepsilon^2/(b-a)^2}.$$

Let’s apply Hoeffding’s inequality 1 to our learning problem.

For a fixed hypothesis  $h$  suppose that  $Z_i = \begin{cases} 1 & \text{if } h(x_i) \neq y_i \text{ for } x_i \sim \mathcal{D} \\ 0 & \text{if } h(x_i) = y_i \text{ for } x_i \sim \mathcal{D} \end{cases}$

Then,  $\mathbb{P}[0 \leq Z_i \leq 1] = 1 \forall i$ , and  $\mathcal{L}_S(h) = \frac{\sum_{i=1}^m Z_i}{m} = \bar{Z}$ , and  $\mathcal{L}_D(h) = \mathbb{E}[\bar{Z}]$ .

This is because  $\forall i$ ,

$$\mathbb{E}[Z_i] = 1 \times \mathbb{P}[Z_i = 1] + 0 \times \mathbb{P}[Z_i = 0] = \mathbb{P}[Z_i = 1] = \mathbb{P}_{x_i \sim \mathcal{D}}[h(x_i) \neq f(x_i)] \stackrel{\text{def}}{=} \mathcal{L}_D(h),$$

and,  $\mathbb{E}[\bar{Z}] = \mathbb{E}\left[\frac{1}{m} \sum_{i=1}^m Z_i\right] = \frac{1}{m} \sum_{i=1}^m \mathbb{E}[Z_i] = \mathbb{E}[Z_i]$ .

Hence,  $\mathbb{E}[\bar{Z}] = \mathcal{L}_D(h)$

Thus, if we fixed a hypothesis  $h$  ahead of time, Hoeffding’s Inequality 1 tells us that

$$\mathbb{P}[|C_S(h) - C_D(h)| > \varepsilon] \leq 2e^{-2m\varepsilon^2}, \forall \varepsilon > 0.$$

Later, we will need another version of the previous Hoeffding’s inequality. It is convenient to introduce it here. We took this formulation of Hoeffding’s Inequality from appendix A.3 of Abu-Mostafa *et al.* (2012).

**Teorema 1.2. Hoeffding’s Inequality 2**

Let  $\mathcal{A} = (a_1, \dots, a_M)$  be a finite population of  $M$  points, such that  $a_i \in [0, 1] \forall i \in \{1, \dots, M\}$

Let  $X_1, \dots, X_m$  be a random sample drawn without replacement from  $\mathcal{A}$ .

Let  $\mu = \frac{1}{M} \sum_{i=1}^M a_i$  is the mean of  $\mathcal{A}$ .

Then,  $\forall \varepsilon > 0$

$$\mathbb{P}\left[\left|\frac{1}{m} \sum_{i=1}^m X_i - \mu\right| > \varepsilon\right] \leq 2e^{-2m\varepsilon^2}$$



Now that we have defined the concepts of hypothesis class and learnability, let's concentrate on a particular case, when the hypothesis class is finite.

### 1.3 Finite classes are PAC learnable

As we saw earlier, Hoeffding's inequality 1 gives us a bound for the probability of the discrepancy between the generalization error and the empirical error. This inequality applies to each member of a class of hypotheses  $\mathcal{H}$  individually. However, if we have " $M$ " hypotheses in  $\mathcal{H} = \{h_1, h_2, \dots, h_M\}$ , we need to see what happens for all hypothesis simultaneously. We followed the thread of (Abu-Mostafa *et al.*, 2012)

With multiple hypotheses in  $\mathcal{H}$ , the algorithm picks a final hypothesis  $h_S$  based on the data set  $\mathcal{S}$  after it was generated. Instead of proving that  $\mathbb{P}[|C_S(h) - C_D(h)| > \varepsilon]$  is small  $\forall h_i \in \mathcal{H}$ , we want to show that  $\mathbb{P}[|C_S(h) - C_D(h)| > \varepsilon]$  is small for the final hypothesis  $h_S$ . The problem is that  $h_S$  is not fixed ahead of time before generating  $\mathcal{S}$ , because it depends on  $\mathcal{S}$ . Consequently, we cannot just plug  $h_S$  in the Hoeffding's inequality 1. To get around this, we try to bound  $\mathbb{P}[|C_S(h_S) - C_D(h_S)| > \varepsilon]$  in a way that does not depend on the chosen  $h_S$  by the algorithm.

Regardless of the algorithm and the sample,  $h_S$  is one of the  $h_i$  in  $\mathcal{H}$ .

Hence,  $|C_S(h_S) - C_D(h_S)| > \varepsilon \implies$

$$|C_S(h_1) - C_D(h_1)| > \varepsilon \vee |C_S(h_2) - C_D(h_2)| > \varepsilon \vee \dots \vee |C_S(h_M) - C_D(h_M)| > \varepsilon$$

Given that if  $A \implies B$  then  $\mathbb{P}[A] \leq \mathbb{P}[B]$ ,

$$\mathbb{P}[|C_S(h_S) - C_D(h_S)| > \varepsilon] \leq \mathbb{P}[|C_S(h_1) - C_D(h_1)| > \varepsilon \vee \dots \vee |C_S(h_M) - C_D(h_M)| > \varepsilon]$$

Given that  $\mathbb{P}[A_1 \vee A_2 \vee \dots \vee A_M] \leq \mathbb{P}[A_1] + \mathbb{P}[A_2] + \dots + \mathbb{P}[A_M]$ ,

$$\mathbb{P}[|C_S(h_S) - C_D(h_S)| > \varepsilon] \leq \mathbb{P}[|C_S(h_1) - C_D(h_1)| > \varepsilon] + \dots + \mathbb{P}[|C_S(h_M) - C_D(h_M)| > \varepsilon]$$

$$\mathbb{P}[|C_S(h_S) - C_D(h_S)| > \varepsilon] \leq \sum_{i=1}^M \mathbb{P}[|C_S(h_i) - C_D(h_i)| > \varepsilon]$$

Now, we apply Hoeffding's inequality 1 to the " $M$ " terms of the sum, one at the time, which gives us  $\mathbb{P}[|C_S(h_S) - C_D(h_S)| > \varepsilon] \leq 2Me^{-2m\varepsilon^2}$ .

Where  $|\mathcal{S}| = m$ , the size of the data set.

This expression gives us a way to characterize the discrepancy between  $C_S$  and  $C_D$  with a probabilistic bound. We interpret it the following way. For a certain tolerance level  $\delta$  ( $= 0.05$  for example), we can find a bound  $\varepsilon$  such that with probability at least  $1 - \delta$ ,  $|C_S(h_S) - C_D(h_S)| \leq \varepsilon$ . Identifying  $\delta = 2Me^{-2m\varepsilon^2}$ , we deduce that  $\varepsilon = \sqrt{\frac{1}{2m} \ln \frac{2M}{\delta}}$ .

With a data set of size  $|\mathcal{S}| = m$ , there is a probability at most  $\delta$  that the generalization and empirical errors will differ by more than  $\varepsilon = \sqrt{\frac{1}{2m} \ln \frac{2M}{\delta}}$ .

Given that we found a way to bound  $\mathbb{P}[|C_S(h_S) - C_D(h_S)| > \varepsilon]$  in a way that does not depend on the chosen  $h_S$  by the algorithm, the bound holds for every hypothesis  $h_i$  in  $\mathcal{H}$ . Consequently,  $|C_S(h_S) - C_D(h_S)| \leq \varepsilon$ , holds for all  $h_i \in \mathcal{H}$ , this is  $C_D(h_S) \leq C_S(h_S) + \varepsilon$  and  $C_D(h_S) \geq C_S(h_S) - \varepsilon$ . This tells us that not only did we chose a hypothesis that generalizes well out of the sample ( $C_D(h_S) \leq C_S(h_S) + \varepsilon$ ), we also know that we did the best we could with our set  $\mathcal{H}$ .  $C_D(h_S) \geq C_S(h_S) - \varepsilon$  assures us that we could not do much better because every hypothesis with a higher  $C_S$  than  $h_S$  will have a comparably higher  $C_D$ .

Now, we would like to extend the preceding results for finite size hypothesis set in the more realistic case of infinite size hypothesis set.

## 1.4 Infinite class are PAC learnable

If we try to apply the preceding results for finite size hypothesis set in the case of infinite size hypothesis set  $\mathcal{H} = \{h_1, h_2, \dots\}$ , we immediately see that if “ $M$ ” the size of  $\mathcal{H}$  goes to infinity our bound also goes to infinity and is meaningless. Consequently, we need to replace the “ $M$ ” by another quantity that deals with the infinite size of  $\mathcal{H}$ . This quantity will be the growth function. We followed the thread of chapter 2 and Appendix of “Learning from data” (Abu-Mostafa *et al.*, 2012)

### 1.4.1 Growth Function

Remember that the  $M$  factor got in the formula when we took the disjunction of the events  $|C_S(h_1) - C_D(h_1)| > \varepsilon \vee |C_S(h_2) - C_D(h_2)| > \varepsilon \vee \dots \vee |C_S(h_M) - C_D(h_M)| > \varepsilon$  in order to be sure to include  $|C_S(h_S) - C_D(h_S)| > \varepsilon$ . This union bound strongly overestimated our probability. However, if two hypotheses  $h_1$  and  $h_2$  are similar to each other, the events  $|C_S(h_1) - C_D(h_1)| > \varepsilon$  and  $|C_S(h_2) - C_D(h_2)| > \varepsilon$ , are likely to coincide for most data sets. In many learning models, many hypotheses are indeed very similar.

We introduce the growth function, a combinatorial quantity that formalizes the effective number of hypotheses by accounting for their overlaps. We will replace the size of  $\mathcal{H}$  in our bound by the growth function which is finite even when  $|\mathcal{H}|$  is infinite.

The definition of the growth function is based on the number of different hypotheses that  $\mathcal{H}$  can implement, but only on a finite sample of points rather than on the entire input space  $\mathcal{X}$ .

To facilitate the definition of the growth function we first define the concept of dichotomies on a finite set of points.

#### Definition 1.2. Dichotomies

Let  $x_1, x_2, \dots, x_m \in \mathcal{X}$ . The dichotomies generated by  $\mathcal{H}$  on these points are defined by

$$\mathcal{H}(x_1, x_2, \dots, x_m) = \{(h(x_1), h(x_2), \dots, h(x_m)) \text{ st } h \in \mathcal{H}\}.$$

Given that we restrict our analysis to the binary classification case, a dichotomy on  $m$  points is an element of the set  $\{-1, +1\}^m$ .

Let's now define the growth function.

**Definition 1.3.** *Growth function  $m_{\mathcal{H}}(m)$*

The growth function is defined for a hypothesis set  $\mathcal{H}$  by

$$m_{\mathcal{H}}(m) = \max_{x_1, x_2, \dots, x_m \in \mathcal{X}} |\mathcal{H}(x_1, x_2, \dots, x_m)|$$

To compute the growth function, one needs to consider all possible choices of  $m$  points and consider the one that gives us the most dichotomies.

For any hypothesis set,  $\mathcal{H}(x_1, x_2, \dots, x_m) \subseteq \{-1, +1\}^m$  and thus,  $|\mathcal{H}(x_1, x_2, \dots, x_m)| \leq | \{-1, +1\}^m | = 2^m$ . Consequently,  $m_{\mathcal{H}}(m) \leq 2^m$ .

If  $\mathcal{H}$  is capable of generating all possible dichotomies, on  $(x_1, x_2, \dots, x_m)$ , then  $\mathcal{H}(x_1, x_2, \dots, x_m) = \{-1, +1\}^m$ , and we say that  $\mathcal{H}$  can *shatter*  $(x_1, x_2, \dots, x_m)$ .

In order to have a measure of the limit of a hypothesis set for generating dichotomies, we will use the concept of break point.

**Definition 1.4.** *Break point*

If no data set of size  $k$  can be shattered by  $\mathcal{H}$ , then  $k$  is said to be a break point for  $\mathcal{H}$ .

If  $k$  is a break point, then  $m_{\mathcal{H}}(k) < 2^k$ .

### 1.4.2 Bound for the growth function:

We have defined the growth function a candidate for replacing "M" in Hoeffding's inequality 1. Now we try to find a bound for this quantity.

**Definition 1.5.**  $B(m, k)$

$B(m, k)$  is the maximum number of dichotomies on  $m$  points such that no subset of size  $k$  of the  $m$  points can be shattered by these dichotomies.

Since  $B(m, k)$  is defined as a maximum it will serve as an upper bound for any  $m_{\mathcal{H}}(m)$  having a break point  $k$ , since if  $k$  is a break point for  $\mathcal{H}$ , then  $m_{\mathcal{H}}(m) \leq B(m, k)$ .

We will try to define  $B(m, k)$  recursively.

**Lemma 1.1.**

$$B(m, k) \leq B(m-1, k) + B(m-1, k-1) \quad \forall m, k.$$

*Proof:* We start calculating  $B(m, 1)$  and  $B(1, k)$ .

$B(m, 1) = 1 \quad \forall m$  since if  $k = 1$  no subset of size 1 (ie a singleton) can be shattered by the  $2^m$  available dichotomies and any two different dichotomies on  $m$  points will differ on at least one point (one

will have +1 and the other -1 on the same point). Consequently, any two dichotomies can shatter at least one point.

When we consider  $B(1, k)$ , we are dealing with a set  $\{x_1\}$  containing a single point. Hence, there are only 2 possible dichotomies possible on this single point, namely +1 and -1. If  $k > 1$ , there are no subset of size 1 of  $\{x_1\}$ . Consequently, the constraint “no subset of size  $k$  of the  $m$  points can be shattered by these dichotomies” is vacuously true. Hence, we deduce that  $B(1, k) = 2 \quad \forall k > 1$ .

We can now consider the case where  $k > 1$  and  $m > 1$  and try to define the recursion relationship.

$B(m, k)$  is a maximum number of dichotomies. Hence it can be seen as the size of a set  $S$  of different dichotomies. We can organize  $S$  in a way that is convenient to us.

Two different dichotomies on  $m$  points may differ only on the last point  $x_m$  (ie they have the same values on  $x_1, x_2, \dots, x_{m-1}$  and a different value for  $x_m$ ) or they may differ before the  $m - th$  points. For example, the two dichotomies +1, -1, -1, +1, +1 and +1, -1, -1, +1, -1 on 5 points differs only on  $x_5$  while +1, -1, +1, +1, +1 and +1, -1, -1, +1, -1 differ on some other points than possibly the last one. We will use this fact decomposing  $S$  into the union of two disjoint subsets  $S_1 \cup S_2$ .

$S_2$  is the subset of  $S$  that consists of pairs of dichotomies differing with each other only on the last point  $x_m$ , and  $S_1 = S \setminus S_2$ . Given that the elements of  $S_2$  can be organized in pairs, we can further divide  $S_2$  into two subsets,  $S_2^-$  and  $S_2^+$ . The subset  $S_2^-$  consists of dichotomies having -1 on  $x_m$  and the subset  $S_2^+$  consists of dichotomies having +1 on  $x_m$ . In the end,  $S = S_1 \cup S_2^- \cup S_2^+$ . Let's give a symbol to the size of those subsets,  $\beta = |S_1|$  and  $\alpha = |S_2^-| = |S_2^+|$ . In the end,  $B(m, k) = \beta + 2\alpha$ .

|       |         | # rows   | $x_1$    | $x_2$ | ... | $x_{m-1}$ | $x_m$ |     |
|-------|---------|----------|----------|-------|-----|-----------|-------|-----|
| $S_1$ |         | $\beta$  | +1       | +1    | ... | +1        | +1    |     |
|       |         |          | -1       | +1    | ... | +1        | -1    |     |
|       |         |          | ...      | ...   | ... | ...       | ...   |     |
|       |         |          | +1       | -1    | ... | -1        | -1    |     |
|       |         |          | -1       | +1    | ... | -1        | +1    |     |
| $S_2$ | $S_2^+$ | $\alpha$ | +1       | -1    | ... | +1        | +1    |     |
|       |         |          | -1       | -1    | ... | +1        | +1    |     |
|       |         |          | ...      | ...   | ... | ...       | ...   |     |
|       |         |          | +1       | -1    | ... | +1        | +1    |     |
|       |         |          | -1       | -1    | ... | -1        | +1    |     |
|       | $S_2^-$ | $\alpha$ | $\alpha$ | +1    | -1  | ...       | +1    | -1  |
|       |         |          |          | -1    | -1  | ...       | +1    | -1  |
|       |         |          |          | ...   | ... | ...       | ...   | ... |
|       |         |          |          | +1    | -1  | ...       | +1    | -1  |
|       |         |          |          | -1    | -1  | ...       | -1    | -1  |

Table 1.1: Dichotomies on “ $m$ ” points

Now let's focus on the dichotomies of  $S$ , but restricting ourselves to the first  $m - 1$  points. The total number different dichotomies on those points is  $\beta + \alpha$  since  $S_2^-$  and  $S_2^+$  are identical on the

first  $m - 1$  points.

|       |         | # rows   | $x_1$ | $x_2$ | ... | $x_{m-1}$ |
|-------|---------|----------|-------|-------|-----|-----------|
| $S_1$ |         | $\beta$  | +1    | +1    | ... | +1        |
|       |         |          | -1    | +1    | ... | +1        |
|       |         |          | ...   | ...   | ... | ...       |
|       |         |          | +1    | -1    | ... | -1        |
|       |         |          | -1    | +1    | ... | -1        |
| $S_2$ | $S_2^+$ | $\alpha$ | +1    | -1    | ... | +1        |
|       |         |          | -1    | -1    | ... | +1        |
|       |         |          | ...   | ...   | ... | ...       |
|       |         |          | +1    | -1    | ... | +1        |
|       |         |          | -1    | -1    | ... | -1        |
|       | $S_2^-$ | $\alpha$ | +1    | -1    | ... | +1        |
|       |         |          | -1    | -1    | ... | +1        |
|       |         |          | ...   | ...   | ... | ...       |
|       |         |          | +1    | -1    | ... | +1        |
|       |         |          | -1    | -1    | ... | -1        |

Table 1.2: Dichotomies on “ $m - 1$ ” points

By definition of  $B(m, k)$ ,  $k$  is a break point for the dichotomies in  $S$ . Consequently, no subset of size  $k$  of the points  $x_1, x_2, \dots, x_m$  can be shattered by the dichotomies in  $S$ . Hence no subset of size  $k$  of the points  $x_1, x_2, \dots, x_{m-1}$  can be shattered by the dichotomies in  $S$  either (otherwise,  $k$  would not be a break point for the dichotomies in  $S$ ), which imply that  $k$  is a break point for the dichotomies in  $S$  restricted to the first  $m - 1$  points (used for  $B(m, k)$ ). By definition,  $B(m - 1, k)$  is the maximum number of dichotomies on any  $m - 1$  points such that no subset of size  $k$  of the  $m - 1$  points can be shattered by these dichotomies. What we know is that no subset of size  $k$  of the first  $m - 1$  points of  $B(m, k)$  can be shattered by the  $\alpha + \beta$  dichotomies of  $S$  restricted to the first  $m - 1$  points. Maybe we can find  $m - 1$  other points (than the first  $m - 1$ ) that gives more dichotomies such that no subset of size  $k$  of them can be shattered. Since  $B(m - 1, k)$  is the maximum, we deduce that  $\alpha + \beta \leq B(m - 1, k)$ .

Now we want to show that  $k - 1$  is a break point for the sets of dichotomies  $S_2^\pm$ .

If there is a subset of size  $k - 1$  of the  $m - 1$  first points for which we have all possible dichotomies (i.e. this subset is shattered by the dichotomies in  $S_2^\pm$ ), then by adding both copies with +1 and -1 respectively, for  $x_m$ , we would have overall  $k$  columns that have all possible dichotomies, which we cannot have since  $k$  is a break point for the dichotomies in  $S$ . We deduce that  $\alpha \leq B(m - 1, k - 1)$ .

Since  $B(m, k) = \beta + 2\alpha$ , and  $\alpha + \beta \leq B(m - 1, k)$ , and  $\alpha \leq B(m - 1, k - 1)$ , we deduce that  $B(m, k) \leq B(m - 1, k) + B(m - 1, k - 1)$  ■

Next we will prove Sauer’s lemma. This result will give us an upper bound for  $B(m, k)$  which will serve as upper bound for  $m_{\mathcal{H}}$ , which was our original task.

**Lemma 1.2. Sauer's Lemma**

$$B(m, k) \leq \sum_{i=0}^{k-1} \binom{m}{i} \quad \forall m, k \text{ in } \mathbb{N}$$

*Proof:* By induction on  $m$ :

Basic step:  $B(m, k) \leq \sum_{i=0}^{k-1} \binom{m}{i}$  for  $m = 1$  and  $\forall k \in \mathbb{N}$

By definition,  $B(1, k)$  is the maximum number of dichotomies on one point such that no subset of size  $k$  of this point can be shattered by these dichotomies.

If  $k = 1$ , the only subset of size  $k$  is the point itself which would be shattered by two dichotomies. Hence,  $B(1, k) = 1$ .

$$\sum_{i=0}^{k-1} \binom{m}{i} = \sum_{i=0}^0 \binom{m}{i} = \binom{m}{0} = 1 \geq B(1, k) = 1$$

If  $k > 1$ , there is no subset of size  $k$  of the point under consideration. Consequently, the second condition ("no subset of size  $k$  of the  $m$  points can be shattered by these dichotomies") of the definition of  $B(m, k)$  is always true, and the definition reduces to "the maximum number of dichotomies on  $m$  points", which is two when  $m = 1$

$$\begin{aligned} \sum_{i=0}^{k-1} \binom{m}{i} &= \binom{1}{0} + \binom{1}{1} + \binom{1}{2} + \cdots + \binom{1}{k-1} \\ &= 1 + 1 + \binom{1}{2} + \cdots + \binom{1}{k-1} \\ &\geq 2 \\ &= B(1, k) \end{aligned}$$

This concludes the initial step.

Inductive step:  $B(m, k) \leq \sum_{i=0}^{k-1} \binom{m}{i} \quad \forall m \leq m_0 \text{ and } \forall k \implies B(m_0 + 1, k) \leq \sum_{i=0}^{k-1} \binom{m_0 + 1}{i} \quad \forall k$

Suppose  $B(m, k) \leq \sum_{i=0}^{k-1} \binom{m}{i} \quad \forall m \leq m_0 \text{ and } \forall k$ .

Notice that since any subset of size 1 of  $m$  points can be shattered by two dichotomies,  $B(m, 1) = 1$ , and  $\sum_{i=0}^{k-1} \binom{m}{i} = 1$  for  $k = 1$ , hence  $B(m, k) \leq \sum_{i=0}^{k-1} \binom{m}{i} \quad \forall m, k = 1$

Let's now concentrate on the case  $k \geq 2$ . By the recursion relation defined previously,  $B(m_0 + 1, k) \leq B(m_0, k) + B(m_0, k - 1)$ .

Applying the induction hypothesis,

$$\begin{aligned}
B(m_0 + 1, k) &\leq \sum_{i=0}^{k-1} \binom{m_0}{i} + \sum_{i=0}^{k-2} \binom{m_0}{i} \\
&= 1 + \sum_{i=1}^{k-1} \binom{m_0}{i} + \sum_{j=1}^{k-1} \binom{m_0}{j-1} \\
&= 1 + \sum_{i=1}^{k-1} \left[ \binom{m_0}{i} + \binom{m_0}{i-1} \right] \\
&= 1 + \sum_{i=1}^{k-1} \binom{m_0 + 1}{i} \\
&= \sum_{i=0}^{k-1} \binom{m_0 + 1}{i}
\end{aligned}$$

Which conclude the inductive step.

By the principle of strong induction,  $B(m, k) \leq \sum_{i=0}^{k-1} \binom{m}{i} \forall m, k$  ■

In the demonstration above we used the following lemma.

**Lemma 1.3.**

$$\binom{m_0 + 1}{i} = \binom{m_0}{i} + \binom{m_0}{i-1}$$

*Proof:*

$$\begin{aligned}
\binom{m_0}{i} + \binom{m_0}{i-1} &= \frac{m_0!}{(m_0 - i)!i!} + \frac{m_0!}{(m_0 - i + 1)!(i-1)!} \\
&= \frac{m_0!}{(m_0 - i)!i(i-1)!} + \frac{m_0!}{(m_0 - i + 1)(m_0 - i)!(i-1)!} \\
&= \frac{m_0!}{(m_0 - i)!(i-1)!} \times \left( \frac{1}{i} + \frac{1}{(m_0 - i + 1)} \right) \\
&= \frac{m_0!}{(m_0 - i)!(i-1)!} \times \left( \frac{m_0 - i + 1 + i}{i(m_0 - i + 1)} \right) \\
&= \frac{m_0!}{(m_0 - i)!(i-1)!} \times \left( \frac{m_0 + 1}{i(m_0 - i + 1)} \right) \\
&= \frac{(m_0 + 1)!}{(m_0 - i + 1)!i!} \\
&= \binom{m_0 + 1}{i}
\end{aligned}$$
■

We can now use Sauer's lemma to formally prove the upper bound of the growth function.

**Teorema 1.3.** *If  $m_{\mathcal{H}}(k) < 2^k$  for some value  $k$ , then*

$$m_{\mathcal{H}}(m) \leq \sum_{i=0}^{k-1} \binom{m}{i} \forall m.$$

*Proof:* If  $m_{\mathcal{H}}(k) < 2^k$  for some value  $k$ , then  $k$  is a break point for  $\mathcal{H}$ .

If  $k$  is a break point for  $\mathcal{H}$ , by definition of  $B(m, k)$ ,  $m_{\mathcal{H}}(m) \leq B(m, k)$ .

Then, by Sauer's Lemma,  $m_{\mathcal{H}}(m) \leq \sum_{i=0}^{k-1} \binom{m}{i}$  ■

The implication of this theorem is that if  $\mathcal{H}$  has a break point,  $m_{\mathcal{H}}(m)$  has a polynomial upper bound.

### 1.4.3 VC-dimension:

The last theorem of the previous section gave us a polynomial bound for the growth function in terms of any break point. The smaller the break point the better the bound. This allows us to define a single parameter that characterizes the growth function. This parameter is the VC dimension.

**Definition 1.6.** *VC dimension*

*The Vapnik-Chervonenkis dimension of a hypothesis set  $\mathcal{H}$  denoted by  $d_{VC}(\mathcal{H})$  or simply  $d_{VC}$  is the largest value of  $m$  for which  $m_{\mathcal{H}}(m) = 2^m$ . If  $m_{\mathcal{H}}(m) = 2^m$  for all  $m$ , then  $d_{VC}(\mathcal{H}) = \infty$ .*

If  $d_{VC}$  is the VC dimension of  $\mathcal{H}$ , then,  $k = d_{VC} + 1$  is a break point for  $m_{\mathcal{H}}$  since  $m_{\mathcal{H}}(m)$  cannot equal  $2^m$  for any  $m > d_{VC}$  by definition. Hence, the last theorem can be rewritten in terms of the VC dimension:

$$m_{\mathcal{H}}(m) \leq \sum_{i=0}^{d_{VC}} \binom{m}{i}.$$

Therefore, the order of the polynomial bound is  $d_{VC}$ .

We now prove a lemma that will help us deriving a simpler bound for the growth function.

**Lemma 1.4.**

$$\sum_{i=0}^D \binom{m}{i} \leq m^D + 1, \forall m, D \text{ in } \mathbb{N}$$

*Proof:* by induction:

Basic step:  $\sum_{i=0}^D \binom{m}{i} \leq m^D + 1$  for  $D = 1, \forall m \text{ in } \mathbb{N}$

For  $D = 1, \forall m \text{ in } \mathbb{N}, \sum_{i=0}^D \binom{m}{i} = \binom{m}{0} + \binom{m}{1} = 1 + m = m^D + 1 \leq m^D + 1$

Inductive step:  $\sum_{i=0}^{D_0} \binom{m}{i} \leq m^{D_0} + 1$  for some  $D_0, \forall m \text{ in } \mathbb{N} \implies \sum_{i=0}^{D_0+1} \binom{m}{i} \leq m^{D_0+1} + 1 \forall m \text{ in } \mathbb{N}$

If  $D_0 \geq m, \binom{m}{D_0+1} = 0$ .



Hence,  $\sum_{i=0}^{D_0+1} \binom{m}{i} = \sum_{i=0}^{D_0} \binom{m}{i} + \binom{m}{D_0+1} = \sum_{i=0}^{D_0} \binom{m}{i} \leq m^{D_0} + 1 \leq m^{D_0+1} + 1$

If  $D_0 < m$ ,  $\sum_{i=0}^{D_0+1} \binom{m}{i} = \sum_{i=0}^{D_0} \binom{m}{i} + \binom{m}{D_0+1} \leq m^{D_0} + 1 + \binom{m}{D_0+1}$

Notice that  $\binom{m}{D_0+1} \leq m^{D_0} (m-1)$ .

This is because  $\binom{m}{D_0+1}$  is the product of  $D_0+1$  terms, and  $m^{D_0} (m-1)$  can also be decomposed in the product of  $D_0+1$  term.

$$\binom{m}{D_0+1} = \frac{m}{D_0+1} \times \frac{(m-1)}{D_0} \times \frac{(m-2)}{D_0-1} \times \cdots \times \frac{(m-D_0)}{1}$$

$$m^{D_0} (m-1) = m \times (m-1) \times m \times \cdots \times m$$

Taken two by two the elements of the product in  $\binom{m}{D_0+1}$  are always inferior or equal to the elements of the product in  $m^{D_0} (m-1)$ .

Consequently,  $\sum_{i=0}^{D_0+1} \binom{m}{i} \leq m^{D_0} + 1 + m^{D_0} (m-1) = m^{D_0} + 1 + m^{D_0+1} - m^{D_0} = m^{D_0+1} + 1$

Which conclude the inductive step.

By the principle of induction,  $\sum_{i=0}^D \binom{m}{i} \leq m^D + 1, \forall m, D \text{ in } \mathbb{N}$ . ■

**Lemma 1.5.**

$$m_{\mathcal{H}}(m) \leq m^{d_{VC}} + 1.$$

*Proof:* Given the definition of  $d_{VC}$ , the smallest break point is 1 above the VC dimension. Hence  $d_{VC} + 1$  is a break point.

The last theorem of the preceding section tells us that  $m_{\mathcal{H}}(m) \leq \sum_{i=0}^{k-1} \binom{m}{i}$  if  $k$  is a break point.

Hence,  $m_{\mathcal{H}}(m) \leq \sum_{i=0}^{d_{VC}} \binom{m}{i}$ .

By the preceding lemma,  $\sum_{i=0}^{d_{VC}} \binom{m}{i} \leq m^{d_{VC}} + 1$ .

Consequently,  $m_{\mathcal{H}}(m) \leq m^{d_{VC}} + 1$ . ■

Now that the growth function is bounded in terms of the VC dimension, we can try to replace “ $M$ ”, the number of hypothesis in  $\mathcal{H}$  by  $m_{\mathcal{H}}(m)$  in the generalization bound.

#### 1.4.4 VC generalization bound:

In this section we try to demonstrate the VC generalization bound.

First we have to prove the Vapnik-Chervonenkis theorem which gave an upper bound for the probability  $\mathbb{P}[\sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{D}}(h)| > \varepsilon]$  for any target function and any input distribution.

The event  $\sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{D}}(h)| > \varepsilon$  is equivalent to the union over all  $h \in \mathcal{H}$  of the event  $|C_{\mathcal{S}}(h) - C_{\mathcal{D}}(h)| > \varepsilon$ . The use of the supremum allows  $\mathcal{H}$  to have a continuum of hypotheses.

Given that it is particularly challenging to manipulate  $C_{\mathcal{D}}(h)$  because it depends on the entire input space, we instead relate it to the in-sample loss  $C_{\mathcal{S}'}(h)$  for a new independent data set  $\mathcal{S}'$  sampled according to same distribution as  $\mathcal{S}$ .

The idea is to bound  $\mathbb{P}[|C_{\mathcal{S}}(h) - C_{\mathcal{D}}(h)| \text{ "is large"}]$  by another term  $\mathbb{P}[|C_{\mathcal{S}}(h) - C_{\mathcal{S}'}(h)| \text{ "is large"}]$ .

**Lemma 1.6.** *VCI:*

Let  $\mathcal{S}, \mathcal{S}'$  be two data sets sampled independently from the same distribution.

$$(1 - 2e^{-\frac{1}{2}\varepsilon^2 m}) \mathbb{P} \left[ \sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{D}}(h)| > \varepsilon \right] \leq \mathbb{P} \left[ \sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{S}'}(h)| > \frac{\varepsilon}{2} \right]$$

*Proof.*

$$\mathbb{P} \left[ \sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{S}'}(h)| > \frac{\varepsilon}{2} \right] \geq \mathbb{P} \left[ \sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{S}'}(h)| > \frac{\varepsilon}{2} \wedge \sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{D}}(h)| > \varepsilon \right] \quad (1.1)$$

$$\begin{aligned} &= \mathbb{P} \left[ \sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{D}}(h)| > \varepsilon \right] \\ &\times \mathbb{P} \left[ \sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{S}'}(h)| > \frac{\varepsilon}{2} \mid \sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{D}}(h)| > \varepsilon \right] \quad (1.2) \end{aligned}$$

(1.1) comes from the fact that for any two events  $A, B$ ,  $\mathbb{P}[A] \geq \mathbb{P}[A \wedge B]$

(1.2) comes from the fact that for any two events  $A, B$ ,  $\mathbb{P}[A \wedge B] = \mathbb{P}[A] \times \mathbb{P}[B|A]$

The condition  $\sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{D}}(h)| > \varepsilon$ , depends on the data set  $\mathcal{S}$ , through  $C_{\mathcal{S}}(h)$ . We condition on the set of all possible data sets  $\mathcal{S}$  sampled according to the distribution  $\mathcal{D}$  for which  $\sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{D}}(h)| > \varepsilon$ .

Fix one such data set  $\mathcal{S}$ . Let  $h^*$  be any hypothesis for which  $|C_{\mathcal{S}}(h^*) - C_{\mathcal{D}}(h^*)| > \varepsilon$ . One such hypothesis must exist since otherwise  $\varepsilon$  would be an upper bound for the set  $\{|C_{\mathcal{S}}(h) - C_{\mathcal{D}}(h)| \text{ st } h \in \mathcal{H}\}$  contradicting the condition  $\sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{D}}(h)| > \varepsilon$ . This hypothesis  $h^*$  depends on  $\mathcal{S}$  but not on  $\mathcal{S}'$ .

Observe that if  $|C_{\mathcal{S}}(h^*) - C_{\mathcal{S}'}(h^*)| > \frac{\varepsilon}{2}$ , then  $\sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{S}'}(h)| > \frac{\varepsilon}{2}$ , because  $\sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{S}'}(h)|$  is an upper bound of the set  $\{|C_{\mathcal{S}}(h) - C_{\mathcal{S}'}(h)| \text{ st } h \in \mathcal{H}\}$ , hence  $\frac{\varepsilon}{2} < |C_{\mathcal{S}}(h^*) - C_{\mathcal{S}'}(h^*)| \leq \sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{S}'}(h)|$ .

Consequently,

$$\mathbb{P} \left[ |C_{\mathcal{S}}(h^*) - C_{\mathcal{S}'}(h^*)| > \frac{\varepsilon}{2} \mid \sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{D}}(h)| > \varepsilon \right] \leq \mathbb{P} \left[ \sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{S}'}(h)| > \frac{\varepsilon}{2} \mid \sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{D}}(h)| > \varepsilon \right] \quad (1.3)$$

Also,

$$|C_{\mathcal{S}'}(h^*) - C_{\mathcal{D}}(h^*)| \leq \frac{\varepsilon}{2}, \quad \text{and} \quad |C_{\mathcal{S}}(h^*) - C_{\mathcal{D}}(h^*)| > \varepsilon$$

imply

$$|C_S(h^*) - C_{S'}(h^*)| > \frac{\varepsilon}{2}.$$

Indeed,

$$|C_{S'}(h^*) - C_{\mathcal{D}}(h^*)| \leq \frac{\varepsilon}{2} \implies -C_{S'}(h^*) - \frac{\varepsilon}{2} \leq -C_{\mathcal{D}}(h^*) \leq -C_{S'}(h^*) + \frac{\varepsilon}{2},$$

and

$$\begin{aligned} |C_S(h^*) - C_{\mathcal{D}}(h^*)| > \varepsilon &\implies C_S(h^*) - C_{\mathcal{D}}(h^*) > \varepsilon \vee C_S(h^*) - C_{\mathcal{D}}(h^*) < -\varepsilon \\ &\implies C_S(h^*) - C_{S'}(h^*) + \frac{\varepsilon}{2} > \varepsilon \vee C_S(h^*) - C_{S'}(h^*) - \frac{\varepsilon}{2} < -\varepsilon \\ &\implies C_S(h^*) - C_{S'}(h^*) > \frac{\varepsilon}{2} \vee C_S(h^*) - C_{S'}(h^*) < -\frac{\varepsilon}{2} \\ &\implies |C_S(h^*) - C_{S'}(h^*)| > \frac{\varepsilon}{2} \end{aligned}$$

Hence,

$$\mathbb{P} \left[ |C_{S'}(h^*) - C_{\mathcal{D}}(h^*)| \leq \frac{\varepsilon}{2} \mid \sup_{h \in \mathcal{H}} |C_S(h) - C_{\mathcal{D}}(h)| > \varepsilon \right] \leq \mathbb{P} \left[ |C_S(h^*) - C_{S'}(h^*)| > \frac{\varepsilon}{2} \mid \sup_{h \in \mathcal{H}} |C_S(h) - C_{\mathcal{D}}(h)| > \varepsilon \right]. \quad (1.4)$$

Finally, given that  $h^*$  is fixed with respect to  $S'$ , we may apply Hoeffding inequality 1 and obtain

$$\mathbb{P} \left[ |C_{S'}(h^*) - C_{\mathcal{D}}(h^*)| \leq \frac{\varepsilon}{2} \mid \sup_{h \in \mathcal{H}} |C_S(h) - C_{\mathcal{D}}(h)| > \varepsilon \right] \geq 1 - 2e^{-\frac{1}{2}\varepsilon^2 m} \quad (1.5)$$

Combining (1.3), (1.4), (1.5),

$$\begin{aligned} \mathbb{P} \left[ \sup_{h \in \mathcal{H}} |C_S(h) - C_{S'}(h)| > \frac{\varepsilon}{2} \mid \sup_{h \in \mathcal{H}} |C_S(h) - C_{\mathcal{D}}(h)| > \varepsilon \right] \\ &\geq \mathbb{P} \left[ |C_S(h^*) - C_{S'}(h^*)| > \frac{\varepsilon}{2} \mid \sup_{h \in \mathcal{H}} |C_S(h) - C_{\mathcal{D}}(h)| > \varepsilon \right] \\ &\geq \mathbb{P} \left[ |C_{S'}(h^*) - C_{\mathcal{D}}(h^*)| \leq \frac{\varepsilon}{2} \mid \sup_{h \in \mathcal{H}} |C_S(h) - C_{\mathcal{D}}(h)| > \varepsilon \right] \\ &\geq 1 - 2e^{-\frac{1}{2}\varepsilon^2 m} \end{aligned} \quad (1.6)$$

By (1.2) and (1.6),

$$\mathbb{P} \left[ \sup_{h \in \mathcal{H}} |C_S(h) - C_{S'}(h)| > \frac{\varepsilon}{2} \right] \geq \mathbb{P} \left[ \sup_{h \in \mathcal{H}} |C_S(h) - C_{\mathcal{D}}(h)| > \varepsilon \right] \times \left( 1 - 2e^{-\frac{1}{2}\varepsilon^2 m} \right)$$

■

We have related the generalization loss  $C_{\mathcal{D}}$ , to the in-sample loss for two independent data sets  $S$  and  $S'$ . Consequently, we can work with  $\mathcal{H}$  restricted to  $S$  and  $S'$  of size  $m$  each, rather than the infinite  $\mathcal{H}$ .

Now, we can try to bound the worst-case deviation between those two in-sample losses using the growth function. Specifically, we want to bound  $\mathbb{P}\left[\sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{S}'}(h)| > \frac{\varepsilon}{2}\right]$ , where the probability is over the joint distribution of the data sets  $\mathcal{S}$  and  $\mathcal{S}'$ .

An equivalent way of sampling two data sets is to first sample a data set  $\mathcal{S}_{2m}$  of size  $2m$ , then randomly partition it into  $\mathcal{S}$  and  $\mathcal{S}'$ . This is equivalent to randomly sampling without replacement  $m$  examples for  $\mathcal{S}$  and leaving the rest for  $\mathcal{S}'$ .

**Lemma 1.7. VC2:**

*Let  $\mathcal{S}, \mathcal{S}'$  be two data sets of size  $m$  sampled independently from the same distribution.*

$$\mathbb{P}\left[\sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{S}'}(h)| > \frac{\varepsilon}{2}\right] \leq m_{\mathcal{H}}(2m) \times \sup_{\mathcal{S}_{2m}} \left\{ \sup_{h \in \mathcal{H}} \left\{ \mathbb{P}\left[|C_{\mathcal{S}}(h) - C_{\mathcal{S}'}(h)| > \frac{\varepsilon}{2} \mid \mathcal{S}_{2m}\right]\right\} \right\}$$

*Proof.*

$$\begin{aligned} \mathbb{P}\left[\sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{S}'}(h)| > \frac{\varepsilon}{2}\right] &= \sum_{\mathcal{S}_{2m}} \mathbb{P}[\mathcal{S}_{2m}] \times \mathbb{P}\left[\sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{S}'}(h)| > \frac{\varepsilon}{2} \mid \mathcal{S}_{2m}\right] & (1.7) \\ &\leq \sum_{\mathcal{S}_{2m}} \mathbb{P}[\mathcal{S}_{2m}] \times \sup_{\mathcal{S}_{2m}} \left\{ \mathbb{P}\left[\sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{S}'}(h)| > \frac{\varepsilon}{2} \mid \mathcal{S}_{2m}\right]\right\} \\ &\leq \sup_{\mathcal{S}_{2m}} \left\{ \mathbb{P}\left[\sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{S}'}(h)| > \frac{\varepsilon}{2} \mid \mathcal{S}_{2m}\right]\right\} \times \sum_{\mathcal{S}_{2m}} \mathbb{P}[\mathcal{S}_{2m}] \\ &\leq \sup_{\mathcal{S}_{2m}} \left\{ \mathbb{P}\left[\sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{S}'}(h)| > \frac{\varepsilon}{2} \mid \mathcal{S}_{2m}\right]\right\} & (1.8) \end{aligned}$$

(1.7) comes from the law of total probability.

Let  $\mathcal{H}(\mathcal{S}_{2m})$  be the number of dichotomies that  $\mathcal{H}$  can implement on the  $2m$  points in  $\mathcal{S}_{2m}$ .

$\mathcal{H}(\mathcal{S}_{2m}) \leq m_{\mathcal{H}}(2m) \leq 2^{(2m)}$ . Suppose  $\mathcal{H}(\mathcal{S}_{2m}) = M$ , realized by  $h_1, \dots, h_M$ .

Thus,  $\sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{S}'}(h)| = \sup_{h \in \{h_1, \dots, h_M\}} |C_{\mathcal{S}}(h) - C_{\mathcal{S}'}(h)|$ .

Then,

$$\begin{aligned} \mathbb{P}\left[\sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{S}'}(h)| > \frac{\varepsilon}{2} \mid \mathcal{S}_{2m}\right] &= \mathbb{P}\left[\sup_{h \in \{h_1, \dots, h_M\}} |C_{\mathcal{S}}(h) - C_{\mathcal{S}'}(h)| > \frac{\varepsilon}{2} \mid \mathcal{S}_{2m}\right] \\ &\leq \sum_{m=1}^M \mathbb{P}\left[\sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h_m) - C_{\mathcal{S}'}(h_m)| > \frac{\varepsilon}{2} \mid \mathcal{S}_{2m}\right] \\ &\leq M \times \sup_{h \in \mathcal{H}} \left\{ \mathbb{P}\left[|C_{\mathcal{S}}(h) - C_{\mathcal{S}'}(h)| > \frac{\varepsilon}{2} \mid \mathcal{S}_{2m}\right]\right\} & (1.9) \end{aligned}$$

By (1.8), (1.9),

$$\begin{aligned}
\mathbb{P} \left[ \sup_{h \in \mathcal{H}} |C_S(h) - C_{S'}(h)| > \frac{\varepsilon}{2} \right] &\leq \sup_{\mathcal{S}_{2m}} \left\{ M \times \sup_{h \in \mathcal{H}} \left\{ \mathbb{P} \left[ |C_S(h) - C_{S'}(h)| > \frac{\varepsilon}{2} \mid \mathcal{S}_{2m} \right] \right\} \right\} \\
&= M \times \sup_{\mathcal{S}_{2m}} \left\{ \sup_{h \in \mathcal{H}} \left\{ \mathbb{P} \left[ |C_S(h) - C_{S'}(h)| > \frac{\varepsilon}{2} \mid \mathcal{S}_{2m} \right] \right\} \right\} \\
&\leq m_{\mathcal{H}}(2m) \times \sup_{\mathcal{S}_{2m}} \left\{ \sup_{h \in \mathcal{H}} \left\{ \mathbb{P} \left[ |C_S(h) - C_{S'}(h)| > \frac{\varepsilon}{2} \mid \mathcal{S}_{2m} \right] \right\} \right\}
\end{aligned}$$

■

We now proceed to bound  $\mathbb{P} \left[ |C_S(h) - C_{S'}(h)| > \frac{\varepsilon}{2} \mid \mathcal{S}_{2m} \right]$  for any  $h$  and any  $\mathcal{S}_{2m}$ .

**Lemma 1.8. VC3:**

For any  $h$  and any  $\mathcal{S}_{2m}$ ,

$$\mathbb{P} \left[ |C_S(h) - C_{S'}(h)| > \frac{\varepsilon}{2} \mid \mathcal{S}_{2m} \right] \leq 2e^{-\frac{1}{8}\varepsilon^2 m}$$

*Proof:* Let  $\mathcal{A} = (a_1, \dots, a_m, a_{m+1}, \dots, a_{2m})$  be a finite population of  $2m$  points, such that  $\forall i \in \{1, \dots, 2m\}$ ,  $a_i = 1$  if  $h(x_i) \neq y_i$  and  $a_i = 0$  otherwise.

Let  $X_1, \dots, X_m$  be a random sample drawn without replacement from  $\mathcal{A}$  and  $\mu = \frac{1}{2m} \sum_{i=1}^{2m} a_i$  be the mean of  $\mathcal{A}$ .

$$\mu = \frac{1}{2m} \sum_{i=1}^{2m} a_i = \frac{1}{2} \left( \frac{1}{m} \sum_{a_i \in \mathcal{S}} a_i + \frac{1}{m} \sum_{a_i \in \mathcal{S}'} a_i \right) = \frac{C_S(h) + C_{S'}(h)}{2}$$

It follows that  $|C_S(h) - \mu| > t \iff |C_S(h) - C_{S'}(h)| > 2t$ .

Then by Hoeffding's inequality 2,  $\forall \varepsilon > 0$ ,  $\mathbb{P} [|C_S(h) - \mu| > \varepsilon] \leq 2e^{-2m\varepsilon^2}$ .

This is because

Hence,  $\forall \varepsilon > 0$ ,  $\mathbb{P} [|C_S(h) - C_{S'}(h)| > 2t] \leq 2e^{-2mt^2}$ .

Take  $t = \varepsilon/4$ ,  $\mathbb{P} [|C_S(h) - C_{S'}(h)| > 2t] \leq 2e^{-\frac{1}{8}\varepsilon^2 m}$ .

■

The combination of lemmas VC1, VC2, VC3 proves the following theorem.

**Teorema 1.4. Vapnik-Chervonenkis**

For any  $h$  and any  $\mathcal{S}_{2m}$ ,

$$\mathbb{P} \left[ \sup_{h \in \mathcal{H}} |C_S(h) - C_{\mathcal{D}}(h)| > \varepsilon \right] \leq 4m_{\mathcal{H}}(2m) e^{-\frac{1}{8}\varepsilon^2 m}$$

*Proof:* By lemma VC1,

$$(1 - 2e^{-\frac{1}{2}\varepsilon^2 m}) \mathbb{P} \left[ \sup_{h \in \mathcal{H}} |C_S(h) - C_{\mathcal{D}}(h)| > \varepsilon \right] \leq \mathbb{P} \left[ \sup_{h \in \mathcal{H}} |C_S(h) - C_{S'}(h)| > \frac{\varepsilon}{2} \right]$$

In the right-hand side,  $e^{-\frac{1}{2}\varepsilon^2 m}$  is either bigger than or equal to  $1/4$  or smaller than  $1/4$ .

If  $e^{-\frac{1}{2}\varepsilon^2 m} \geq 1/4$ , then  $e^{-\frac{1}{8}\varepsilon^2 m} > e^{-\frac{1}{2}\varepsilon^2 m} \geq 1/4$ . In this case,  $4m_{\mathcal{H}}(2m) e^{-\frac{1}{8}\varepsilon^2 m} \geq 4m_{\mathcal{H}}(2m) 1/4 = m_{\mathcal{H}}(2m) \geq 1$ , given that the minimum number of dichotomies generated by  $\mathcal{H}$  is 1. Hence in this case we have  $\mathbb{P}[\sup_{h \in \mathcal{H}} |C_S(h) - C_{\mathcal{D}}(h)| > \varepsilon] \leq 1$ , which is always true.

If  $e^{-\frac{1}{2}\varepsilon^2 m} < 1/4$ , then  $1 - 2e^{-\frac{1}{2}\varepsilon^2 m} > 1/2$ , and lemma VC1 becomes:

$$\mathbb{P} \left[ \sup_{h \in \mathcal{H}} |C_S(h) - C_{\mathcal{D}}(h)| > \varepsilon \right] \leq 2 \mathbb{P} \left[ \sup_{h \in \mathcal{H}} |C_S(h) - C_{S'}(h)| > \frac{\varepsilon}{2} \right]$$

By lemma VC2,

$$\mathbb{P} \left[ \sup_{h \in \mathcal{H}} |C_S(h) - C_{\mathcal{D}}(h)| > \varepsilon \right] \leq 2m_{\mathcal{H}}(2m) \times \sup_{S_{2m}} \left\{ \sup_{h \in \mathcal{H}} \left\{ \mathbb{P} \left[ |C_S(h) - C_{S'}(h)| > \frac{\varepsilon}{2} \mid \mathcal{S}_{2m} \right] \right\} \right\}$$

By lemma VC3,

$$\mathbb{P} \left[ \sup_{h \in \mathcal{H}} |C_S(h) - C_{\mathcal{D}}(h)| > \varepsilon \right] \leq 2m_{\mathcal{H}}(2m) \times 2e^{-\frac{1}{8}\varepsilon^2 m}$$

Finally,

$$\mathbb{P} \left[ \sup_{h \in \mathcal{H}} |C_S(h) - C_{\mathcal{D}}(h)| > \varepsilon \right] \leq 4m_{\mathcal{H}}(2m) e^{-\frac{1}{8}\varepsilon^2 m}$$

■

Finally, we can use the Vapnik-Chervonenkis theorem to find a bound for the generalization error with respect to the experimental error, the size of the data set, and a confidence level  $\delta$ .

**Teorema 1.5.** *VC generalization bound.*

For any tolerance level  $\delta$ ,

$$C_{\mathcal{D}}(h) \leq C_S(h) + \sqrt{\frac{8}{m} \ln \left( \frac{4m_{\mathcal{H}}(2m)}{\delta} \right)}$$

$$C_{\mathcal{D}}(h) \leq C_S(h) + \sqrt{\frac{8}{m} \ln \left( \frac{4 \left( (2m)^{d_{VC}} + 1 \right)}{\delta} \right)}$$

With probability  $1 - \delta$ .

*Proof:* According to the Vapnik-Chervonenkis theorem, for any  $h$ ,

$$\mathbb{P} \left[ \sup_{h \in \mathcal{H}} |C_{\mathcal{S}}(h) - C_{\mathcal{D}}(h)| > \varepsilon \right] \leq 4m_{\mathcal{H}}(2m) e^{-\frac{1}{8}\varepsilon^2 m} \leq 4 \left( (2m)^{d_{VC}} + 1 \right) e^{-\frac{1}{8}\varepsilon^2 m}.$$

Consequently, if we have a data set of size  $m$ , and we want to find the generalization error  $\varepsilon$  that won't be violated with probability  $\delta$ , we set  $4 \left( (2m)^{d_{VC}} + 1 \right) e^{-\frac{1}{8}\varepsilon^2 m} = \delta$ , or  $4m_{\mathcal{H}}(2m) e^{-\frac{1}{8}\varepsilon^2 m} = \delta$  and find the corresponding  $\varepsilon = \sqrt{\frac{8}{m} \ln \left( \frac{4m_{\mathcal{H}}(2m)}{\delta} \right)}$  or  $\varepsilon = \sqrt{\frac{8}{m} \ln \left( \frac{4((2m)^{d_{VC}} + 1)}{\delta} \right)}$ . ■

Since  $m_{\mathcal{H}}(2m)$  is polynomial of order  $d_{VC}$ , in  $m$ , the error bar converges to zero as long as  $d_{VC}$  is finite. Consequently, with enough data each and every hypothesis in an infinite hypothesis set  $\mathcal{H}$  with finite VC dimension will generalize well from  $C_{\mathcal{S}}$  to  $C_{\mathcal{D}}$ .

The last result establishes the feasibility of learning with infinite hypothesis sets.

We will end our theoretical analysis of the feasibility of learning by analyzing “how big” a data set must be in order to achieve a certain performance at learning.

#### 1.4.5 Sample complexity:

The concept of sample complexity relates the number of examples in a training data set with generalization performance of a hypothesis set.

**Definition 1.7.** *Sample complexity*

*The sample complexity denotes how many training examples  $m$  are needed to achieve a certain generalization performance specified by a tolerance level  $\varepsilon$  and a confidence parameter  $\delta$ .*

In the definition above, the tolerance level  $\varepsilon$  determines the allowed generalization error, and the confidence parameter  $\delta$  determines how often the error tolerance  $\varepsilon$  is violated. The relationship between  $m, \varepsilon, \delta$ , indicates how much data is needed to get a certain generalization level with a certain probability.

The VC bound defined in the preceding section can help us estimate the sample complexity for a certain learning model defined by its VC dimension. If we fix a maximum tolerance  $\varepsilon$ , for a confidence level  $\delta > 0$ , the generalization error is bounded by  $\sqrt{\frac{8}{m} \ln \left( \frac{4m_{\mathcal{H}}(2m)}{\delta} \right)}$ , so it suffices to make  $\sqrt{\frac{8}{m} \ln \left( \frac{4m_{\mathcal{H}}(2m)}{\delta} \right)} \leq \varepsilon$ , which implies  $\frac{8}{\varepsilon^2} \ln \left( \frac{4m_{\mathcal{H}}(2m)}{\delta} \right) \leq m$ . In order to get rid of the growth function which is tedious to evaluate we replace it in our expression by the polynomial bound based on the VC dimension,  $m_{\mathcal{H}}(m) \leq m^{d_{VC}} + 1$ . This gives us the following implicit relation between,  $m, \varepsilon, \delta$ :  $\frac{8}{\varepsilon^2} \ln \left( \frac{4((2m)^{d_{VC}} + 1)}{\delta} \right) \leq m$ . This expression estimates how many examples  $m$  are needed in order to achieve a certain performance in terms of generalization error  $\varepsilon$ , with a certain confidence  $\delta$ .

In practical situation we are given a data set  $\mathcal{S}$ , so  $m$  is fixed and the relevant question would be which performance can we expect given this particular data set. The answer is (by the VC

generalization bound) that with probability  $1 - \delta$ ,

$$C_{\mathcal{D}} \leq C_{\mathcal{S}} + \sqrt{\frac{8}{m} \ln \left( \frac{4 \left( (2m)^{d_{VC}} + 1 \right)}{\delta} \right)}.$$

We can see that the bound on  $C_{\mathcal{D}}$  has two parts. The first one is  $C_{\mathcal{S}}$ , and the second one is a function  $\Omega(m, \mathcal{H}, \delta) = \sqrt{\frac{8}{m} \ln \left( \frac{4 \left( (2m)^{d_{VC}} + 1 \right)}{\delta} \right)}$ . When a set of hypotheses  $\mathcal{H}$  becomes more complex,  $d_{VC}$  increases and  $\Omega(m, \mathcal{H}, \delta)$  increases as well. Consequently, a simpler model that would achieve the same training error as a more complex model would have a better estimate for  $C_{\mathcal{D}}$ . We also see that  $\Omega(m, \mathcal{H}, \delta)$  increases if we want a higher confidence and decreases if we have more data. However, it is to be noted that when  $\mathcal{H}$  has a higher VC dimension it is more likely to have a lower training error  $C_{\mathcal{S}}$ . Consequently, we have a tradeoff, and the optimal model is a compromise that minimizes a combination of  $C_{\mathcal{S}}$  and  $\Omega(m, \mathcal{H}, \delta)$ .



## Chapter 2

# Neural network

In the preceding part we made a presentation of the classification learning problem and showed its feasibility in general. Now we go from theory to practice. We are going to concentrate on a particular method of learning (i.e. one particular hypothesis class  $\mathcal{H}_{NN}$ ) called neural network.

We will first give a conceptual explanation of the model, then we will define a notation system for the different parts of the model, and finally we will develop the different equations that govern its functioning.

### 2.1 General description of the artificial neural network:

An artificial neural network is a model of computation that can approximate any continuous function from  $\mathbb{R}^n$  to  $\mathbb{R}^m$ , with high flexibility capability, and high potential of non-linearity (Nielsen, 2019). It is inspired by the structure of neural network in the brain. It consists of a large number of basic computing devices (neurons) that are connected to each other in a complex communication network. The neurons are displayed along several layers made of a certain number of neurons. Actually as proved by Hornik (1989) and Cybenko (1989), multilayer feedforward networks are universal approximators in the sense that they can approximate any Borel measurable function from one finite dimensional space to another to any desired degree of accuracy, provided sufficiently many hidden units are available.

We limit our analysis to the fully connected case, where each neuron of a given layer is connected to every neurons of the preceding layer. The first layer is called the input layer, the last layer is called the output layer, and the layers in between are called hidden layers.

The value stored in each neuron depends on the values of the neurons in the preceding layers, on a set of coefficients called weights and biases, and on a function called activation function. Formally, it is the weighted sum of the outputs of the neurons connected to its incoming edges.

$$a_j^{(\ell)} = f \left( \sum_{k=1}^{n_{\ell-1}} w_{jk}^{(\ell)} \cdot a_k^{(\ell-1)} + b_j^{(\ell)} \right)$$

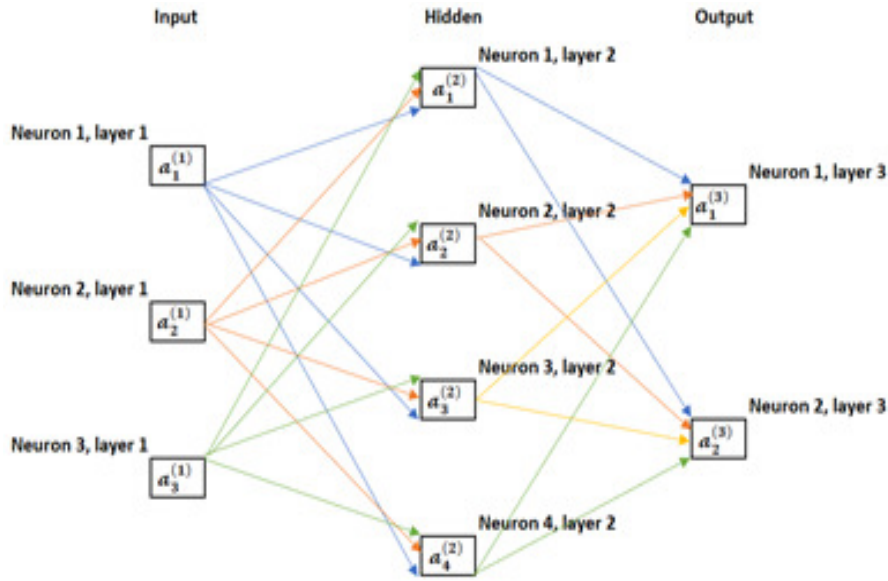


Figure 2.1: Neural network architecture

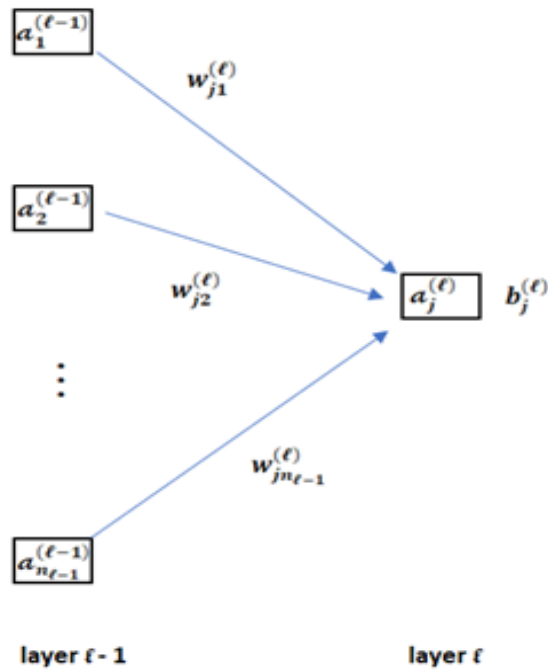


Figure 2.2: Neural network weights and biases

The values stored in the neurons of the input layer are the inputs values of the data set  $\mathcal{S}$ , and the values in the neurons of the output layer are the prediction of the model.

## 2.2 Notations:

We adopted the notation of Nielsen (2019).

$n$  = number of training examples

$x$  = index of individual training example

$l$  = index of the hidden layers (In particular, the input layer has index 1 and the output layer has index  $L$ ).

$w_{jk}^{(\ell)}$  = weight between the  $j$ -th neuron in the  $l$ -th layer and the  $k$ -th neuron in the  $l-1$ -th layer.

$$w^{(\ell)} = \begin{bmatrix} w_{11}^{(\ell)} & \cdots & w_{1n_{\ell-1}}^{(\ell)} \\ \vdots & \ddots & \vdots \\ w_{n_{\ell}1}^{(\ell)} & \cdots & w_{n_{\ell}n_{\ell-1}}^{(\ell)} \end{bmatrix} \text{ weight matrix.}$$

$b_j^{(\ell)}$  = value of the  $j$ -th bias on the  $l$ -th layer.

$b^{(\ell)}$  = vector value for the bias on the  $l$ -th layer

$z_j^{(\ell)}$  = value of the  $j$ -th neuron on the  $l$ -th layer before the activation function is applied.

$$z_j^{(\ell)} = \sum_{k=1}^{n_{\ell-1}} w_{jk}^{(\ell)} \cdot a_k^{(\ell-1)} + b_j^{(\ell)}$$

$z^{(\ell)}$  = vector value for the  $l$ -th layer before the activation function is applied.

$a_j^{(\ell)}$  = value of the  $j$ -th neuron on the  $l$ -th layer after the activation function is applied. Hence,  
 $a_j^{(\ell)} = f(z_j^{(\ell)})$ .

$a^{(\ell)}$  = vector value for the  $l$ -th layer after the activation function is applied. Hence,  $a^{(\ell)} = f(z^{(\ell)})$ .

$f$  = activation function.

$y(x)$  = output for the training examples "x".

$C(w, b, \mathcal{S})$  = Cost function depending on the weights, biases and data of the training set.

$$C(w, b, \mathcal{S}) = \frac{1}{n} \sum_{x=1}^n C_x$$

$C_x$  = value of the cost function for the individual training example labeled "x".

Next we will explain how the weights and biases are calculated through an optimization process.

## 2.3 Gradient descent and stochastic gradient descent:

Like every learning model, the goal of a neural network is to approximate an unknown target function. Selecting a neural network according to some data set  $\mathcal{S}$  consists in defining the different weights and biases parameters. The criterion for this process is the minimization of the cost function. Specifically, we need to find a way to select the weights and biases that minimize the value of the cost function.

Gradient descent is an iterative optimization procedure in which at each step we improve the solution by taking a step along the negative of the gradient of the function to be minimized at the current point. In our case, the procedure could be summarize by the following relations:

$$w_{jk}^{(\ell)} \longrightarrow w_{jk}^{(\ell)} - \eta \frac{\partial C}{\partial w_{jk}^{(\ell)}}$$
$$b_j^{(\ell)} \longrightarrow b_j^{(\ell)} - \eta \frac{\partial C}{\partial b_j^{(\ell)}}$$

In stochastic gradient descent (SGD) we allow the optimization procedure to take a step along a random direction, as long as the expected value of the direction is the negative of the gradient. That way, the minimization proceeds in the right direction but with random fluctuation that will cancel out in the long run.

Given that we define the cost function  $C$  as an average of individual cost functions  $C_x$  over every example in  $\mathcal{S}$ , its gradient is  $\nabla C = \frac{1}{n} \sum_{x=1}^n \nabla C_x$ .

If we pick examples successively according to a discrete uniform distribution between 1 and  $n$ ,  $\mathbb{P}[\nabla C_X = \nabla C_x] = \frac{1}{n}$ ,  $\forall x = 1 \dots n$ , where  $\nabla C_X$  is the random variable corresponding to the value of the gradient of the individual cost function for the uniformly drawn examples  $X$ . Hence,  $\mathbb{E}[\nabla C_X] = \frac{1}{n} \nabla C_1 + \frac{1}{n} \nabla C_2 + \dots + \frac{1}{n} \nabla C_n = \frac{1}{n} \sum_{x=1}^n \nabla C_x = \nabla C$ .

In practice, stochastic gradient descent is a sequential version of gradient descent. Instead of considering the gradient of the total cost function on all training data points at each step, we consider the gradient of the cost function at a data point chosen uniformly at random. The gradient of this single point's cost function is used for weight update in the same way that the full gradient was used in gradient descent.

$$w_{jk}^{(\ell)} \longrightarrow w_{jk}^{(\ell)} - \eta \frac{\partial C_x}{\partial w_{jk}^{(\ell)}}$$
$$b_j^{(\ell)} \longrightarrow b_j^{(\ell)} - \eta \frac{\partial C_x}{\partial b_j^{(\ell)}}$$

Consequently, if we find a way to calculate the partial derivatives of the  $C_x$  we would automatically have the partial derivatives of the cost function  $C$ .

Stochastic gradient descent is usually preferred to regular gradient descent for being much faster as explained in Lecun *et al.* (1998a).

The question now is: “how to apply stochastic gradient descent in the case of neural network?”. This can be achieved by the so-called backpropagation algorithm.

## 2.4 Equation for backpropagation:

According to the equations for the stochastic gradient descent we need to calculate  $\frac{\partial C_x}{\partial b_j^{(\ell)}}$  and  $\frac{\partial C_x}{\partial w_{jk}^{(\ell)}}$  to iteratively update our parameters  $w_{jk}^{(\ell)}$  and  $b_j^{(\ell)}$ , looking for the values that will minimize the cost function  $C$ . The equations of the backpropagation algorithm were developed according to Nielsen (2019).

Define  $\delta_j^{(L)} = \frac{\partial C_x}{\partial z_j^{(L)}}$

Recall that the cost is a function of  $a^{(L)}$  which in turn is a function of  $z^{(L)}$  which is function of  $w^{(L)}, b^{(L)}, a^{(L-1)}$ . Hence,  $C_x(a^{(L)}(z^{(L)}(w^{(L)}, b^{(L)}, a^{(L-1)})))$ .

We may apply the chain rule to compute  $\delta_j^{(L)} = \frac{\partial C_x}{\partial z_j^{(L)}}$ .

$$\begin{aligned}\frac{\partial C_x}{\partial z_j^{(L)}} &= \sum_{k=1}^{n_L} \frac{\partial C_x}{\partial a_k^{(L)}} \cdot \frac{\partial a_k^{(L)}}{\partial z_j^{(L)}} \\ \frac{\partial C_x}{\partial z_j^{(L)}} &= \frac{\partial C_x}{\partial a_j^{(L)}} \cdot \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}}\end{aligned}$$

Given that  $\frac{\partial a_k^{(L)}}{\partial z_j^{(L)}} = 0 \quad \forall k \neq j$ .

$$\begin{aligned}\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} &= \frac{\partial}{\partial z_j^{(L)}} f(z_j^{(L)}) \\ \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} &= f'(z_j^{(L)}) \\ \implies \delta_j^{(L)} &= \frac{\partial C_x}{\partial a_j^{(L)}} \cdot f'(z_j^{(L)})\end{aligned}$$

We can rewrite this equation in vector form  $\delta^{(L)} = \nabla_a C_x \odot f'(z^{(L)})$

Where:  $\odot$  is the Hadamard product

$$\begin{aligned}\delta^{(L)} &= \begin{pmatrix} \delta_1^{(L)} \\ \vdots \\ \delta_{n_L}^{(L)} \end{pmatrix} \\ \nabla_a C_x &= \begin{pmatrix} \frac{\partial C_x}{\partial a_1^{(L)}} \\ \vdots \\ \frac{\partial C_x}{\partial a_{n_L}^{(L)}} \end{pmatrix}\end{aligned}$$

$$f' \left( z^{(L)} \right) = \begin{pmatrix} f' \left( z_1^{(L)} \right) \\ \vdots \\ f' \left( z_{n_L}^{(L)} \right) \end{pmatrix}$$

Define  $\delta_j^{(\ell)} = \frac{\partial C_x}{\partial z_j^{(\ell)}}$

We are now trying to express  $\delta_j^{(\ell)}$  in terms of  $\delta_j^{(\ell+1)}$ .

$$C_x \left( a^{(L)} \left( z^{(L)} \left( a^{(L-1)} \left( z^{(L-1)} \left( \dots \left( a^{(\ell+1)} \left( z^{(\ell+1)} \left( a^{(\ell)} \left( z^{(\ell)} \left( \dots \left( a^{(1)} \left( z^{(1)} \right) \right) \right) \right) \right) \right) \right) \right) \right) \right) \right)$$

We may apply the chain rule to compute  $\delta_j^{(\ell)} = \frac{\partial C_x}{\partial z_j^{(\ell)}}$

$$\frac{\partial C_x}{\partial z_j^{(\ell)}} = \sum_{k=1}^{n_{\ell+1}} \frac{\partial C_x}{\partial z_k^{(\ell+1)}} \cdot \frac{\partial z_k^{(\ell+1)}}{\partial z_j^{(\ell)}}$$

$$\frac{\partial C_x}{\partial z_j^{(\ell)}} = \sum_{k=1}^{n_{\ell+1}} \delta_k^{(\ell+1)} \cdot \frac{\partial z_k^{(\ell+1)}}{\partial z_j^{(\ell)}}$$

Given that  $z_k^{(\ell+1)} = \left( \sum_{j=1}^{n_\ell} w_{kj}^{(\ell+1)} \cdot a_j^{(\ell)} \right) + b_k^{(\ell+1)} = \left( \sum_{j=1}^{n_\ell} w_{kj}^{(\ell+1)} \cdot f \left( z_j^{(\ell)} \right) \right) + b_k^{(\ell+1)}$ ,

$$\frac{\partial z_k^{(\ell+1)}}{\partial z_j^{(\ell)}} = w_{kj}^{(\ell+1)} \cdot f' \left( z_j^{(\ell)} \right)$$

$$\implies \delta_j^{(\ell)} = \sum_{k=1}^{n_{\ell+1}} w_{kj}^{(\ell+1)} \cdot f' \left( z_j^{(\ell)} \right) \cdot \delta_k^{(\ell+1)}$$

We can rewrite this equation in vector form  $\delta^{(\ell)} = \left( \left[ w^{(\ell+1)} \right]^T \delta^{(\ell+1)} \right) \odot f' \left( z^{(\ell)} \right)$

Where:  $\odot$  is the Hadamard product

$$\delta^{(\ell+1)} = \begin{pmatrix} \delta_1^{(\ell+1)} \\ \vdots \\ \delta_{n_{\ell+1}}^{(\ell+1)} \end{pmatrix}$$

$$\left[ w^{(\ell+1)} \right] = \begin{bmatrix} w_{11}^{(\ell)} & \cdots & w_{1n_\ell}^{(\ell)} \\ \vdots & \ddots & \vdots \\ w_{n_{\ell+1}1}^{(\ell)} & \cdots & w_{n_{\ell+1}n_\ell}^{(\ell)} \end{bmatrix}$$

Now we are ready to calculate  $\frac{\partial C_x}{\partial b_j^{(\ell)}}$  and  $\frac{\partial C_x}{\partial w_{jk}^{(\ell)}}$  applying again de chain rule.

$$\frac{\partial C_x}{\partial b_j^{(\ell)}} = \frac{\partial C_x}{\partial z_j^{(\ell)}} \cdot \frac{\partial z_j^{(\ell)}}{\partial b_j^{(\ell)}}$$

$$\implies \frac{\partial C_x}{\partial b_j^{(\ell)}} = \delta_j^{(\ell)} \cdot \frac{\partial z_j^{(\ell)}}{\partial b_j^{(\ell)}}$$

Given that  $z_j^{(\ell)} = \left(\sum_{k=1}^{n_{\ell-1}} w_{jk}^{(\ell)} \cdot a_k^{(\ell-1)}\right) + b_j^{(\ell)}$ , we deduce that  $\frac{\partial z_j^{(\ell)}}{\partial b_j^{(\ell)}} = 1$ , and hence  $\frac{\partial C_x}{\partial b_j^{(\ell)}} = \delta_j^{(\ell)}$ .

$$\begin{aligned}\frac{\partial C_x}{\partial w_{jk}^{(\ell)}} &= \frac{\partial C_x}{\partial z_j^{(\ell)}} \cdot \frac{\partial z_j^{(\ell)}}{\partial w_{jk}^{(\ell)}} \\ \implies \frac{\partial C_x}{\partial w_{jk}^{(\ell)}} &= \delta_j^{(\ell)} \cdot \frac{\partial z_j^{(\ell)}}{\partial w_{jk}^{(\ell)}}\end{aligned}$$

Given that  $z_j^{(\ell)} = \left(\sum_{k=1}^{n_{\ell-1}} w_{jk}^{(\ell)} \cdot a_k^{(\ell-1)}\right) + b_j^{(\ell)}$ , we deduce that  $\frac{\partial z_j^{(\ell)}}{\partial w_{jk}^{(\ell)}} = a_k^{(\ell-1)}$ , and hence  $\frac{\partial C_x}{\partial w_{jk}^{(\ell)}} = \delta_j^{(\ell)} a_k^{(\ell-1)}$ .

We can now rewrite the equations of the gradient descent:

$$\begin{aligned}w_{jk}^{(\ell)} &\longrightarrow w_{jk}^{(\ell)} - \eta \frac{\partial C_x}{\partial w_{jk}^{(\ell)}} = w_{jk}^{(\ell)} - \eta \left(a_k^{(\ell-1)} \delta_j^{(\ell)}\right) \\ b_j^{(\ell)} &\longrightarrow b_j^{(\ell)} - \eta \frac{\partial C_x}{\partial b_j^{(\ell)}} = b_j^{(\ell)} - \eta \delta_j^{(\ell)}\end{aligned}$$

In matrix form:

$$\begin{aligned}w^{(\ell)} &\longrightarrow w^{(\ell)} - \eta \left[ \left[ a^{(\ell-1)} \right] \left[ \delta^{(\ell)} \right]^T \right]^T \\ b^{(\ell)} &\longrightarrow b^{(\ell)} - \eta \delta^{(\ell)}\end{aligned}$$

Now that we defined the different equations of backpropagation we can implement them in a formalized algorithm.

## 2.5 Backpropagation algorithm:

The backpropagation equations gave us a way of computing the gradient of the cost function. We can implement an algorithm to apply the stochastic gradient descent and find the set of parameters  $w_{jk}^{(\ell)}$  and  $b_j^{(\ell)}$  that minimizes the cost function.

- Input x: Set the corresponding neurons on the input layer

$$a^{(1)} = f(x_x)$$

- Feedforward: For each  $\ell = 2, 3, \dots, L$  compute  $z^{(\ell)} = w^{(\ell)} a^{(\ell-1)} + b^{(\ell)}$  and

$$a^{(\ell)} = f(z^{(\ell)}).$$

- Output error  $\delta^{(L)}$  :  $\delta^{(L)} = \nabla_a C_x \odot f'(z^{(L)})$
- Backpropagate the error: For each  $\ell = L-1, L-2, \dots, 2$  compute

$$\delta^{(\ell)} = \left( \left[ w^{(\ell+1)} \right]^T \delta^{(\ell+1)} \right) \odot f'(z^{(\ell)})$$

- Output: Calculate the gradient of the cost function  $\frac{\partial C_x}{\partial w_{jk}^{(\ell)}} = \delta_j^{(\ell)} a_k^{(\ell-1)}$  and  $\frac{\partial C_x}{\partial b_j^{(\ell)}} = \delta_j^{(\ell)}$ .
- Update the weights and biases:

$$w^{(\ell)} \longrightarrow w^{(\ell)} - \eta \left[ a^{(\ell-1)} \left[ \delta^{(\ell)} \right]^T \right]^T$$

$$b^{(\ell)} \longrightarrow b^{(\ell)} - \eta \delta^{(\ell)}$$

The formal presentation of this algorithm was taken from Nielsen (2019).

## 2.6 Activation functions:

In this section, let us explain why we need to apply an activation function to the weighted sum of outputs of neurons of the preceding layer.

If we consider the biological analogy, a neuron in the brain receives electrical signal until it is enough “activated” so that it fires and gave information to another neuron. If we want to copy this mechanism we should use a step function as activation function.

However, with this sort of activation function, a small change in the weights or biases can sometimes cause the output to completely flip say from 0 to 1. This is not a desirable property of the network. We would prefer that small changes in some weights or biases only cause small corresponding changes in the output from the network. One way to achieve this is to use a smoother version of the step function, for example the sigmoid function. The exact shape of the sigmoid is not its interesting property, but rather its smoothness which causes small changes  $\Delta w_{jk}$  and  $\Delta b_j$  to produce small change  $\Delta output$ . Furthermore, if we did not apply an activation function our neural network would only be able to model linear relationship between the input of the network and the output. For the purpose of this work, we will use the sigmoid function  $\sigma(z) = \frac{1}{1+e^{-z}}$  as activation function.

Next, we will take a look at the cost function. It is a major part of a neural network given that the weights and biases are computed through a minimization process of this function.

## 2.7 Cost functions:

We have defined the cost function as the average of individual cost functions for every training example,  $C = \frac{1}{n} \sum_{x=1}^n C_x$ . We will now discuss a choice for those  $C_x$ .

A natural choice is the mean square error function,

$$C_x = \frac{1}{2} \|y(x) - a^{(L)}(x)\|^2 = \frac{1}{2} \sum_{j=1}^{n_L} (y_j - a_j^{(L)})^2.$$

However, this cost function can slow down the learning process which in neural network take the form of solving a minimization problem. Gradient descent works by changing the weights



and biases at rate determined by the partial derivative  $\frac{\partial C_x}{\partial w}$  and  $\frac{\partial C_x}{\partial b}$ . Thus, when those partial derivatives are small, the learning is “slow”.

Given that

$$C_x = \frac{1}{2} \|y - a^{(L)}\|^2 = \frac{1}{2} \|y - f(z^{(L)})\|^2 = \frac{1}{2} \sum_{j=1}^{n_L} (y_j - f(z_j^{(L)}))^2,$$

we have that

$$\frac{\partial C_x}{\partial w_{jk}} = (a_j^{(L)} - y_j) \cdot a_k^{(L-1)} \cdot f'(z_j^{(L)}) \qquad \frac{\partial C_x}{\partial b_j} = (a_j^{(L)} - y_j) \cdot f'(z_j^{(L)})$$

We see that both partial derivatives are proportional to the derivative of the activation function  $f'(z_j^{(L)})$ . However, if we look at the shape of the sigmoid function which is a typical activation function (see Figure 2.3), we see that when the neuron’s output is very close to 1, the curve gets very flat, and so  $f'(z_j^{(L)})$  is very small.

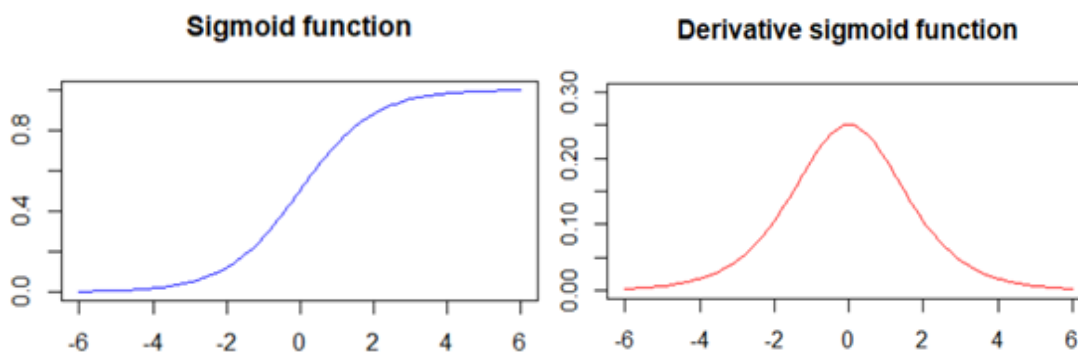


Figure 2.3: Graph of sigmoid function and its derivative.

This problem is known as “learning slow down” and it has been explained in Glorot and Bengio (2010) and Lecun *et al.* (1998a).

We can solve this problem of learning slow down by replacing the mean square error cost with a different cost function known as the cross-entropy.

The cross-entropy cost function for a single training example is given by

$$C_x = - \sum_{j=1}^{n_L} [y_j \ln(a_j^{(L)}) + (1 - y_j) \ln(1 - a_j^{(L)})]$$

It is not directly obvious that this function may be interpreted as a cost function. Let’s see why it is the case as explained in Nielsen (2019). First, we can see that the cost-entropy is nonnegative. This is because both logarithms are negative since they take as argument numbers between 0 and 1, and a minus sign is added in front of the expression. Second, if the neuron’s actual output is close to the desired output for all training inputs,  $x$ , then the cross-entropy will be close to zero. To see this, suppose for example that  $y = 0$  and  $a \approx 0$  for some input  $x$ . This means that the network is doing a good job on that input. Since  $y = 0$ , the first term of the expression vanishes while the

second term is  $-\ln(1-a) \approx 0$ . A similar analysis holds when  $y = 1$  and  $a \approx 1$ . Consequently, we see that the contribution to the cost function will be low provided the actual output is close to the desired output. Furthermore, the cross-entropy avoids the problem of the learning slow-down that appears with the mean square error cost function.

Remember that:

$$C_x = - \sum_{j=1}^{n_L} [y_j \ln(a_j^{(L)}) + (1 - y_j) \ln(1 - a_j^{(L)})]$$

Hence,

$$C_x = - \sum_{j=1}^{n_L} [y_j \ln(f(z_j^{(L)})) + (1 - y_j) \ln(1 - f(z_j^{(L)}))]$$

Then,

$$\frac{\partial C_x}{\partial w_{jk}} = - \left[ \frac{y_j}{a_j^{(L)}} + \frac{1 - y_j}{1 - a_j^{(L)}} \cdot (-1) \right] \cdot a_k^{(L-1)} \cdot f'(z_j^{(L)})$$

Considering that we work with sigmoid activation function,

$$\frac{\partial C_x}{\partial w_{jk}} = - \left[ \frac{y_j - y_j \cdot a_j^{(L)} - a_j^{(L)} + y_j \cdot a_j^{(L)}}{a_j^{(L)} \cdot (1 - a_j^{(L)})} \right] \cdot a_k^{(L-1)} \cdot \sigma'(z_j^{(L)})$$

Considering that  $\sigma'(z) = \sigma(z) \cdot (1 - \sigma(z))$  and that  $a_j^{(L)} = \sigma(z_j^{(L)})$

$$\begin{aligned} \frac{\partial C_x}{\partial w_{jk}} &= - \left[ \frac{y_j - a_j^{(L)}}{\sigma'(z_j^{(L)})} \right] \cdot a_k^{(L-1)} \cdot \sigma'(z_j^{(L)}) \\ &= (a_j^{(L)} - y_j) \cdot a_k^{(L-1)} \end{aligned}$$

Similary we deduce  $\frac{\partial C_x}{\partial b_j} = (a_j^{(L)} - y_j)$

As we can see the  $f'(z_j^{(L)})$  terms which causes the learning slow-down vanished.

Next, we are going to discuss a regularization technique, that helps to reduce overfitting by controlling the values of the weights and biases.

## 2.8 L2 Regularization:

Network with large weights may change its behavior significantly in response to small changes in the input. Network with small weight won't change too much their output if we change slightly the input. Consequently, it is desirable to develop networks with small coefficients. The technique that keeps the weights small is called regularization. The hope is that a regularized model will do real learning about the phenomenon at hand and will be resistant to learning peculiarities of the noise in the data.

There are several regularization techniques, but here we concentrate on L2 regularization. In L2 regularization, we add an extra term to the cost function called regularization term. This term is

the sum of the squares of all weight in the network scaled by a factor  $\lambda/2n$ , where  $\lambda > 0$  is the regularization parameter.

$$C = \frac{1}{n} \sum_{x=1}^n C_x + \frac{\lambda}{2n} \sum_{j,k} w_{jk}^2$$

$$C_x = - \sum_{j=1}^{n_L} \left[ y_j \ln(a_j^{(L)}) + (1 - y_j) \ln(1 - a_j^{(L)}) \right]$$

The effect of regularization is to make the network prefer using small weights. Large weight will only be allowed if they considerably improve the first part of the cost function. Regularization forces the network to compromise between finding small weight and minimizing the original cost function. The relative importance of the two elements of the compromise depends on the value of the parameter  $\lambda$ .

We can implement L2 regularization in the SGD procedure by replacing  $\nabla C = \begin{pmatrix} \nabla_w C \\ \nabla_b C \end{pmatrix}$  as the direction of steepest descent with  $\begin{pmatrix} \nabla_w C_x + \frac{\lambda}{n} w \\ \nabla_b C_x \end{pmatrix}$ . Therefore, we need to check that

$$\mathbb{E} \left[ \begin{pmatrix} \nabla_w C_x + \frac{\lambda}{n} w \\ \nabla_b C_x \end{pmatrix} \right] = \nabla C$$

$$\nabla C = \begin{pmatrix} \nabla_w C \\ \nabla_b C \end{pmatrix} = \begin{pmatrix} \frac{1}{n} \sum_{x=1}^n (\nabla_w C_x) + \frac{\lambda}{n} w \\ \frac{1}{n} \sum_{x=1}^n (\nabla_b C_x) \end{pmatrix}$$

$$\nabla_w C = \begin{pmatrix} \frac{\partial C}{\partial w_{11}^{(1)}} \\ \vdots \\ \frac{\partial C}{\partial w_{jk}^{(\ell)}} \\ \vdots \\ \frac{\partial C}{\partial w_{n_L n_{L-1}}^{(L)}} \end{pmatrix}; \nabla_b C = \begin{pmatrix} \frac{\partial C}{\partial b_1^{(1)}} \\ \vdots \\ \frac{\partial C}{\partial b_j^{(\ell)}} \\ \vdots \\ \frac{\partial C}{\partial b_{n_L}^{(L)}} \end{pmatrix}; w = \begin{pmatrix} w_{11}^{(1)} \\ \vdots \\ w_{jk}^{(\ell)} \\ \vdots \\ w_{n_L n_{L-1}}^{(L)} \end{pmatrix}; \nabla_w C_x = \begin{pmatrix} \frac{\partial C_x}{\partial w_{11}^{(1)}} \\ \vdots \\ \frac{\partial C_x}{\partial w_{jk}^{(\ell)}} \\ \vdots \\ \frac{\partial C_x}{\partial w_{n_L n_{L-1}}^{(L)}} \end{pmatrix}; \nabla_b C = \begin{pmatrix} \frac{\partial C_x}{\partial b_1^{(1)}} \\ \vdots \\ \frac{\partial C_x}{\partial b_j^{(\ell)}} \\ \vdots \\ \frac{\partial C_x}{\partial b_{n_L}^{(L)}} \end{pmatrix}$$

$$\mathbb{E} \left[ \begin{pmatrix} \nabla_w C_x + \frac{\lambda}{n} w \\ \nabla_b C_x \end{pmatrix} \right] = \frac{1}{n} \begin{pmatrix} \nabla_w C_1 + \frac{\lambda}{n} w \\ \nabla_b C_1 \end{pmatrix} + \dots + \frac{1}{n} \begin{pmatrix} \nabla_w C_n + \frac{\lambda}{n} w \\ \nabla_b C_n \end{pmatrix}$$

$$= \frac{1}{n} \begin{pmatrix} \sum_{x=1}^n (\nabla_w C_x) + \lambda w \\ \sum_{x=1}^n (\nabla_b C_x) \end{pmatrix}$$

$$= \begin{pmatrix} \frac{1}{n} \sum_{x=1}^n (\nabla_w C_x) + \frac{\lambda}{n} w \\ \frac{1}{n} \sum_{x=1}^n (\nabla_b C_x) \end{pmatrix}$$

$$= \nabla C$$

Consequently, the rules for updating the weights become:

$$w_{jk}^{(\ell)} \longrightarrow \left(1 - \frac{\eta}{n} \lambda\right) w_{jk}^{(\ell)} - \eta \frac{\partial C_x}{\partial w_{jk}^{(\ell)}}$$

$$b_j^{(\ell)} \longrightarrow b_j^{(\ell)} - \eta \frac{\partial C_x}{\partial b_j^{(\ell)}}$$

## 2.9 Overfitting and validation set

One major issue in Machine Learning is overfitting. As we said in paragraph 1.2.1 in Chapter 1, overfitting occurs, when our model fits the training data “too well” or when the model is learning the training data “by heart” instead of understanding them. One way to avoid overfitting is to make use of a so-called validation set. The way this set works is very much the same as the test set. It gives us an estimate of the true error, as explained in Abu-Mostafa *et al.* (2012).

When the model is affected by overfitting, its performance on the training set is very good even if the true performance is poor. Consequently, one way of dealing with this problem is to keep track of the training performance during the building process of our neural network model (i.e. at each epoch) and comparing it to an estimate of the true performance evaluated by means of the validation set.

Typically, overfitting occurs when the performance on the training set increases at each epoch while the performance on the validation set stays constant or even decreases, as illustrated in Figure 2.4.

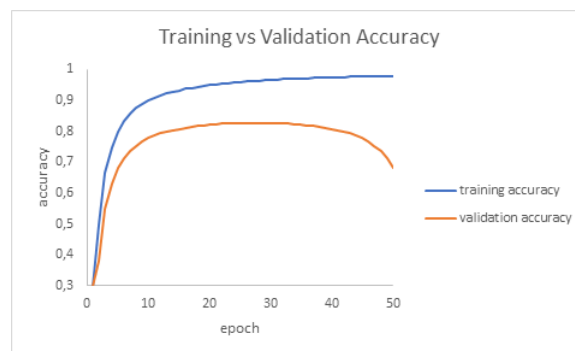


Figure 2.4: Typical validation vs training accuracy

Consequently, we should make sure that we stop the learning process (in our case the number of epoch), before the point where validation accuracy starts decreasing.

## Chapter 3

# Analysis of a dataset

In order to apply the developed theory about neural networks we wrote a code in the software R designed to solve a classification problem. We test this program on a famous data base of handwritten digits, the MNIST dataset. This dataset is freely downloadable on the website of Lecun and has been extensively studied by Lecun *et al.* (1998b).

### 3.1 The MNIST dataset

The MNIST data base consists of scanned images of handwritten digits in greyscale together with their correct classifications. Several examples of those images can be observed on Figure 3.1.

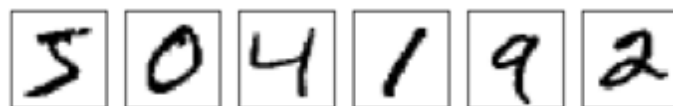


Figure 3.1: Examples from MNIST database

The pictures were centered in a 28x28 image by computing the center of mass of the pixels and translating the image so as to position this point at the center of the 28x28 pixels. The data set has two parts: a training set of 60.000 images sampled from 250 people and a test set of 10.000 images sampled from 250 different people. Following the thread from Lecun we divided the training set in two in order to obtain a validation data set of 10.000 images leaving 50.000 images in the training set.

Next, we are going to present the pseudo code that was implemented in R. Keep in mind that the notation “list[[k]]” is used to refer to the k-th element of a list. The symbol “\*” is used for matrix multiplication, and the notation “.” for scalar multiplication.

A. Initialization of the hyperparameters:

epoch

$\eta$

$\lambda$

n = vector containing the number of neurons on each layer

L= # layers  
 N= # training examples

B. Initialization of the weights and biases:

```

for k = 2 to L
  nk = # neurons on the k - th layer
  nk-1 = # neurons on the k - 1 - th layer
  b[[k]] = vector(nk × 1, N(0,1))
  w[[k]] = matrix(nk × nk-1, N(0,  $\frac{1}{\sqrt{n_{k-1}}}$ ))
end for

```

C. Applying backpropagation algorithm on the N training examples successively and repeating the process for the number of epochs chosen.

```

for k = 1 to epoch
  for j = 1 to N
    1. choose an example "x" at random
    2. create the label vector "y" with the right classification of the example at hand
    3. initialize neurons of the first layer of the network with the values of the training example
    4. Feedforward:
      for ℓ = 2 to L
        z[[ℓ]] = w[[ℓ]] * a[[ℓ - 1]] + b[[ℓ]]
        a[[ℓ]] = f(z[[ℓ]])
      end for
    5. Calculate the output error:
      delta[[L]] = a[[L]] - y
    6. Backpropagate the error:
      for ℓ = L - 1 to 2
        delta[[ℓ]] = (wT[[ℓ + 1]] * delta[[ℓ + 1]]) * f'(z[[ℓ]])
        w[[ℓ]] = (w[[ℓ]] · (1 -  $\eta \frac{\lambda}{N}$ )) - ( $\eta \cdot (a[[ℓ - 1]] * delta^T[[ℓ]])^T$ )
        b[[ℓ]] = b[[ℓ]] - ( $\eta \cdot delta[[ℓ]]$ )
      end for
    end for
    Evaluate the accuracy on the validation dataset.
    if (validation accuracy > highest validation accuracy recorded)
      Keep the weights and biases of the current epoch.
    end if
  end for
end for

```

## 3.2 Results

We tested the program with several network's architectures found on the web site of Lecun who studied the data set MNIST extensively with different machine learning methods. We selected five such architectures, ran our program on it and compared our results with those of Lecun.

Given the various sources of randomness in the building process of a neural network for a given architecture (random initialization of the weights, training example entered in a random order, ...) we expect our results to be fairly close to Lecun .

The first tested network has a single hidden layer of 300 neurons and uses the mean square cost function. The training process took 7.5 hours. Our model achieved a classification error rate of 4.53% while Lecun reported a classification error rate of 4.7% on the test data set. We ran the program on 30 epochs which seems enough to obtain a stable validation accuracy as we observe on Figure 3.2.

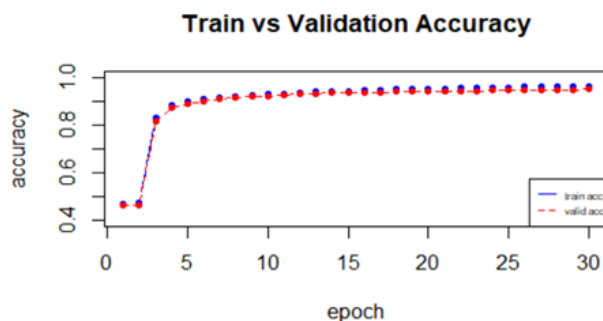


Figure 3.2: train vs validation accuracy for network 1

We can use a confusion matrix to analyze our results a little bit more in detail. This matrix is a table whose rows represent the predicted classes of our model and whose columns represent the actual classes. Hence, the entry at row “i”, and column “j”, represent the number of examples in the test set that the model classified in class “i”, and that are actually in class “j”. Consequently, on the diagonal of the matrix lie the examples correctly classified by the network.

Based on our confusion matrix, we can also define for every class the true positive rate or sensitivity (portion of the examples assigned by the network to a class they actually belong to) and the true negative rate or specificity (portion of the examples assigned by the network to a different class when they actually belong to a different class). Thus, we can evaluate the true positive rate (TPR) and the true negative rate (TPN) with the following expressions:

$$TPR = \frac{TP}{TP + FN} \quad ; \quad TNR = \frac{TN}{TN + FP}$$

where

- $TP$  is the number of true positive of a class (number of examples assigned by the network to the class they actually belongs to),
- $TN$  is the number of true negative of a class (number of examples assigned by the network to a different class when they actually belong to a different class),

- *FP* is the number of false positive of a class (number of examples assigned by the network to the class when they actually belong to a different class),
- *FN* is the number of false negative of a class (number of examples belonging to the class and assigned by the network to a different class).

In Tables 3.1 and 3.2 we present the matrix confusion and the distribution of the true positive and negative rates for network 1, respectively. Analyzing this tables, we can observe that the model performs well in general, but it works better on identifying some digits than others. In particular, we observe lower TPR for digits 2, 5, 8 and 9, even if they stay above 92%. It is worth mentioning that the TNR for those same digits were approximately 99%. On the other hand, digits 0 and 1 reached the highest TPR.

| Prediction | Actual |      |     |     |     |     |     |     |     |     |
|------------|--------|------|-----|-----|-----|-----|-----|-----|-----|-----|
|            | 0      | 1    | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   |
| 0          | 969    | 0    | 9   | 2   | 1   | 7   | 9   | 3   | 2   | 7   |
| 1          | 0      | 1121 | 2   | 1   | 2   | 2   | 4   | 11  | 2   | 6   |
| 2          | 1      | 1    | 975 | 12  | 2   | 2   | 1   | 18  | 4   | 1   |
| 3          | 1      | 4    | 5   | 959 | 2   | 14  | 0   | 4   | 11  | 12  |
| 4          | 1      | 0    | 7   | 0   | 937 | 2   | 6   | 3   | 5   | 21  |
| 5          | 1      | 1    | 1   | 13  | 0   | 822 | 6   | 0   | 16  | 3   |
| 6          | 4      | 4    | 5   | 1   | 12  | 14  | 928 | 0   | 9   | 2   |
| 7          | 1      | 2    | 13  | 8   | 2   | 1   | 0   | 977 | 5   | 12  |
| 8          | 2      | 2    | 14  | 13  | 4   | 22  | 4   | 1   | 919 | 5   |
| 9          | 0      | 0    | 1   | 1   | 20  | 6   | 0   | 11  | 1   | 940 |

Table 3.1: Confusion Matrix for network 1.

| Category | TP   | TN   | FP | FN | TPR   | TNR   |
|----------|------|------|----|----|-------|-------|
| 0        | 969  | 8980 | 40 | 11 | 98.9% | 99.6% |
| 1        | 1121 | 8835 | 30 | 14 | 98.8% | 99.7% |
| 2        | 975  | 8926 | 42 | 57 | 94.5% | 99.5% |
| 3        | 959  | 8937 | 53 | 51 | 95.0% | 99.4% |
| 4        | 937  | 8973 | 45 | 45 | 95.4% | 99.5% |
| 5        | 822  | 9067 | 41 | 70 | 92.2% | 99.5% |
| 6        | 928  | 8991 | 51 | 30 | 96.9% | 99.4% |
| 7        | 977  | 8928 | 44 | 51 | 95.0% | 99.5% |
| 8        | 919  | 8959 | 67 | 55 | 94.4% | 99.3% |
| 9        | 940  | 8951 | 40 | 69 | 93.2% | 99.6% |

Table 3.2: Performance indicators for network 1

The second tested network has a single hidden layer of 1000 neurons and uses the mean square cost function. The training process took 8.9 hours. Lecun reported a classification error rate of 4.5% on the test data set while our model achieved a classification error rate of 4.6%. We ran the program on 30 epochs which seems enough to obtain a stable validation accuracy as we observe on Figure 3.3.



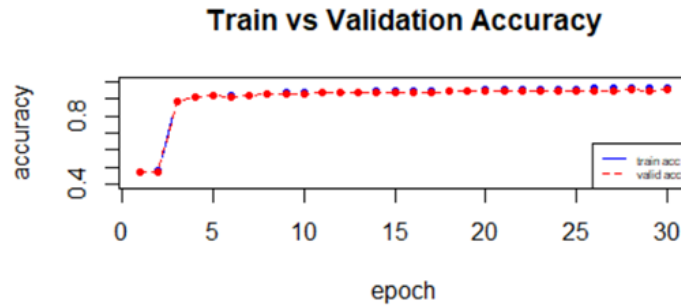


Figure 3.3: Train vs validation accuracy for network 2

| Prediction | Actual |      |     |     |     |     |     |     |     |     |
|------------|--------|------|-----|-----|-----|-----|-----|-----|-----|-----|
|            | 0      | 1    | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   |
| 0          | 965    | 0    | 8   | 0   | 1   | 8   | 12  | 1   | 6   | 7   |
| 1          | 0      | 1114 | 1   | 0   | 1   | 1   | 3   | 5   | 2   | 4   |
| 2          | 1      | 2    | 965 | 7   | 3   | 1   | 3   | 14  | 1   | 0   |
| 3          | 1      | 2    | 4   | 948 | 1   | 11  | 1   | 3   | 7   | 7   |
| 4          | 0      | 1    | 7   | 1   | 938 | 4   | 15  | 3   | 8   | 19  |
| 5          | 4      | 2    | 0   | 13  | 1   | 843 | 13  | 1   | 5   | 3   |
| 6          | 3      | 4    | 8   | 1   | 6   | 5   | 903 | 0   | 6   | 1   |
| 7          | 3      | 2    | 14  | 13  | 4   | 3   | 1   | 987 | 6   | 12  |
| 8          | 2      | 8    | 22  | 20  | 4   | 10  | 7   | 2   | 929 | 9   |
| 9          | 1      | 0    | 3   | 7   | 23  | 6   | 0   | 12  | 4   | 947 |

Table 3.3: Confusion Matrix for network 2

| Category | TP   | TN   | FP | FN | TPR   | TNR   |
|----------|------|------|----|----|-------|-------|
| 0        | 965  | 8977 | 43 | 15 | 98.5% | 99.5% |
| 1        | 1114 | 8848 | 17 | 21 | 98.1% | 99.8% |
| 2        | 965  | 8936 | 32 | 67 | 93.5% | 99.6% |
| 3        | 948  | 8953 | 37 | 62 | 93.9% | 99.6% |
| 4        | 938  | 8960 | 58 | 44 | 95.5% | 99.4% |
| 5        | 843  | 9066 | 42 | 49 | 94.5% | 99.5% |
| 6        | 903  | 9008 | 34 | 55 | 94.3% | 99.6% |
| 7        | 987  | 8914 | 58 | 41 | 96.0% | 99.4% |
| 8        | 929  | 8942 | 84 | 45 | 95.4% | 99.1% |
| 9        | 947  | 8935 | 56 | 62 | 93.9% | 99.4% |

Table 3.4: Performance measures for network 2

In Tables 3.3 we presented the confusion matrix and 3.4 the distribution of the true positive and negative rates for network 2. Analyzing this tables, we again observe a very good performance with this network albeit inferior to the one of the preceding model.

The third network has two hidden layers of 300 and 100 neurons and uses the cross-entropy cost function. The training process took 7.43 hours. Our model achieved a classification error rate of 2.18% while Lecun reported a classification error rate of 3.05% on the test data set. We ran the program on 30 epochs which seems enough to obtain a stable validation accuracy as we observe on Figure 3.4.

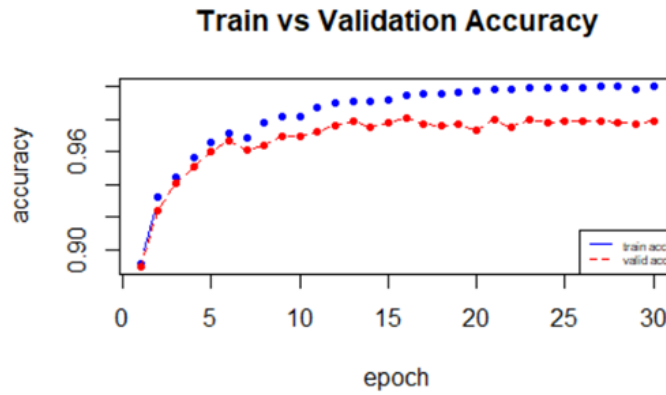


Figure 3.4: train vs validation accuracy for network 3

In Table 3.5 we presented the matrix confusion and in Table 3.6 the distribution of the true positive and negative rates for network 3. Analyzing this tables, we can observe that the model looks uniformly efficient on every category.

| Prediction | Actual |      |      |     |     |     |     |     |     |     |
|------------|--------|------|------|-----|-----|-----|-----|-----|-----|-----|
|            | 0      | 1    | 2    | 3   | 4   | 5   | 6   | 7   | 8   | 9   |
| 0          | 966    | 0    | 3    | 0   | 2   | 5   | 5   | 2   | 4   | 3   |
| 1          | 0      | 1128 | 1    | 0   | 0   | 0   | 3   | 9   | 0   | 4   |
| 2          | 2      | 1    | 1018 | 7   | 2   | 0   | 1   | 10  | 2   | 0   |
| 3          | 1      | 0    | 2    | 985 | 1   | 6   | 1   | 3   | 3   | 4   |
| 4          | 1      | 0    | 1    | 0   | 959 | 2   | 3   | 0   | 2   | 10  |
| 5          | 2      | 1    | 0    | 6   | 0   | 870 | 2   | 0   | 8   | 4   |
| 6          | 4      | 4    | 1    | 0   | 5   | 6   | 942 | 0   | 6   | 1   |
| 7          | 2      | 0    | 4    | 6   | 3   | 1   | 0   | 997 | 4   | 6   |
| 8          | 1      | 1    | 2    | 4   | 1   | 1   | 1   | 2   | 942 | 2   |
| 9          | 1      | 0    | 0    | 2   | 9   | 1   | 0   | 5   | 3   | 975 |

Table 3.5: Confusion Matrix for network 3

| Category | TP   | TN   | FP | FN | TPR   | TNR   |
|----------|------|------|----|----|-------|-------|
| 0        | 966  | 8996 | 24 | 14 | 98.6% | 99.7% |
| 1        | 1128 | 8848 | 17 | 7  | 99.4% | 99.8% |
| 2        | 1018 | 8943 | 25 | 14 | 98.6% | 99.7% |
| 3        | 985  | 8969 | 21 | 25 | 97.5% | 99.8% |
| 4        | 959  | 8999 | 19 | 23 | 97.7% | 99.8% |
| 5        | 870  | 9085 | 23 | 22 | 97.5% | 99.7% |
| 6        | 942  | 9015 | 27 | 16 | 98.3% | 99.7% |
| 7        | 997  | 8946 | 26 | 31 | 97.0% | 99.7% |
| 8        | 942  | 9011 | 15 | 32 | 96.7% | 99.8% |
| 9        | 975  | 8970 | 21 | 34 | 96.6% | 99.8% |

Table 3.6: Performance measures for network 3

The fourth network has two hidden layers of 500 and 150 neurons and uses the cross-entropy cost function. The training process took 8.64 hours. Our model achieved a classification error rate of 2.02% while Lecun reported a classification error rate of 2.95% on the test data set. We ran the program on 30 epochs which seems enough to obtain a stable validation accuracy as we observe on Figure 3.5.

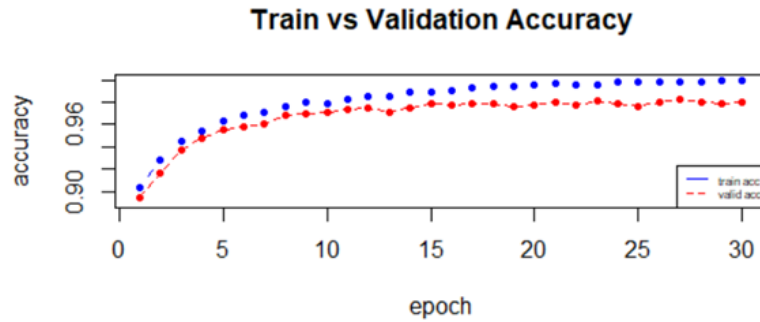


Figure 3.5: train vs validation accuracy for network 4

| Prediction | Actual |      |      |     |     |     |     |      |     |     |
|------------|--------|------|------|-----|-----|-----|-----|------|-----|-----|
|            | 0      | 1    | 2    | 3   | 4   | 5   | 6   | 7    | 8   | 9   |
| 0          | 970    | 0    | 1    | 0   | 1   | 3   | 5   | 1    | 3   | 2   |
| 1          | 0      | 1126 | 1    | 0   | 0   | 0   | 2   | 4    | 0   | 4   |
| 2          | 2      | 2    | 1016 | 6   | 4   | 0   | 1   | 6    | 3   | 0   |
| 3          | 0      | 0    | 0    | 981 | 0   | 9   | 1   | 2    | 3   | 4   |
| 4          | 1      | 0    | 2    | 0   | 958 | 3   | 4   | 0    | 3   | 10  |
| 5          | 0      | 1    | 0    | 4   | 0   | 869 | 2   | 0    | 3   | 2   |
| 6          | 2      | 4    | 3    | 0   | 4   | 3   | 942 | 0    | 2   | 1   |
| 7          | 1      | 1    | 4    | 9   | 4   | 1   | 0   | 1008 | 4   | 5   |
| 8          | 3      | 1    | 5    | 5   | 0   | 2   | 1   | 1    | 948 | 1   |
| 9          | 1      | 0    | 0    | 5   | 11  | 2   | 0   | 6    | 5   | 980 |

Table 3.7: Confusion Matrix for network 4

In Table 3.7 we presented the matrix confusion and in Table 3.8 the distribution of the true positive and negative rates for network 4. Analyzing this tables, we can observe that the model looks uniformly efficient on every category.

| Category | TP   | TN   | FP | FN | TPR   | TNR   |
|----------|------|------|----|----|-------|-------|
| 0        | 970  | 9004 | 16 | 10 | 99.0% | 99.8% |
| 1        | 1126 | 8854 | 11 | 9  | 99.2% | 99.9% |
| 2        | 1016 | 8944 | 24 | 16 | 98.4% | 99.7% |
| 3        | 981  | 8971 | 19 | 29 | 97.1% | 99.8% |
| 4        | 958  | 8995 | 23 | 24 | 97.6% | 99.7% |
| 5        | 869  | 9096 | 12 | 23 | 97.4% | 99.9% |
| 6        | 942  | 9023 | 19 | 16 | 98.3% | 99.8% |
| 7        | 1008 | 8943 | 29 | 20 | 98.1% | 99.7% |
| 8        | 948  | 9007 | 19 | 26 | 97.3% | 99.8% |
| 9        | 980  | 8961 | 30 | 29 | 97.1% | 99.7% |

Table 3.8: Performance indicators for network 4

The last tested network has a single hidden layer of 800 neurons and uses the cross-entropy cost function. The training process took 8.7 hours. Our model achieved a classification error rate of 2% while Lecun reported a classification error rate of 1.6% on the test data set. We ran the program on 30 epochs which seems enough to obtain a stable validation accuracy as we observe on Figure 3.6.

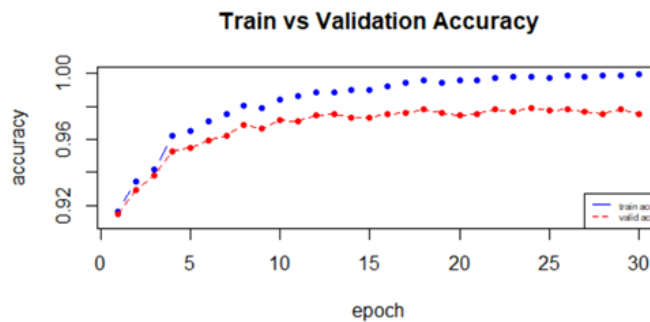


Figure 3.6: train vs validation accuracy for network 5

In Table 3.9 we presented the confusion matrix and in Table 3.10 the distribution of the true positive and negative rates for network 4. Analyzing this tables, we can observe that the model looks uniformly efficient on every category.

| Prediction | Actual |      |      |     |     |     |     |      |     |     |
|------------|--------|------|------|-----|-----|-----|-----|------|-----|-----|
|            | 0      | 1    | 2    | 3   | 4   | 5   | 6   | 7    | 8   | 9   |
| 0          | 970    | 0    | 2    | 0   | 2   | 4   | 6   | 2    | 5   | 3   |
| 1          | 0      | 1123 | 1    | 0   | 0   | 0   | 2   | 2    | 0   | 2   |
| 2          | 2      | 3    | 1018 | 2   | 3   | 0   | 1   | 9    | 4   | 0   |
| 3          | 1      | 1    | 1    | 996 | 1   | 8   | 1   | 3    | 2   | 7   |
| 4          | 1      | 0    | 1    | 0   | 957 | 1   | 4   | 0    | 0   | 6   |
| 5          | 1      | 1    | 0    | 4   | 0   | 867 | 2   | 0    | 3   | 3   |
| 6          | 2      | 2    | 3    | 0   | 3   | 5   | 938 | 0    | 3   | 0   |
| 7          | 1      | 1    | 4    | 2   | 2   | 1   | 0   | 1002 | 3   | 7   |
| 8          | 2      | 3    | 2    | 3   | 1   | 5   | 4   | 3    | 950 | 2   |
| 9          | 0      | 1    | 0    | 3   | 13  | 1   | 0   | 7    | 4   | 979 |

Table 3.9: Confusion Matrix for network 5

| Category | TP   | TN   | FP | FN | TPR   | TNR   |
|----------|------|------|----|----|-------|-------|
| 0        | 970  | 8996 | 24 | 10 | 99.0% | 99.7% |
| 1        | 1123 | 8858 | 7  | 12 | 98.9% | 99.9% |
| 2        | 1018 | 8944 | 24 | 14 | 98.6% | 99.7% |
| 3        | 996  | 8965 | 25 | 14 | 98.6% | 99.7% |
| 4        | 957  | 9005 | 13 | 25 | 97.5% | 99.9% |
| 5        | 867  | 9094 | 14 | 25 | 97.2% | 99.8% |
| 6        | 938  | 9024 | 18 | 20 | 97.0% | 99.8% |
| 7        | 1002 | 8951 | 21 | 26 | 97.5% | 99.8% |
| 8        | 950  | 9001 | 25 | 24 | 97.5% | 99.7% |
| 9        | 979  | 8962 | 29 | 30 | 97.0% | 99.7% |

Table 3.10: Performance indicators for network 5

## Chapter 4

# Conclusions and Future Directions

### 4.1 Conclusion

To conclude this work, we will recall the initial objectives and the final achievements of the thesis.

The problem of machine learning can be easily explained from an intuitive perspective: “Using data examples to feed an algorithm hoping that it learns how to replicate a data generating process”. However, “translating” this explanation in the area of mathematics is not so natural, and needs some efforts to define notations, concepts (especially what “learning” mathematically means) and prove that the objective of the problem is theoretically achievable. In the same manner, if we consider the method of neural network, the general idea is easily explainable; however, developing the necessary notations and modelling equations (especially for the backpropagation algorithm) is also not so obvious.

In this thesis we took the effort to develop and explain the mathematics behind the idea of a learning algorithm and in particular for neural networks.

Writing a code in R that evaluates the coefficients of a certain neural network and make corresponding predictions required an in-depth comprehension of the above-mentioned equations. This is another achievement of the thesis. This comprehension is necessary in order to propose some advanced improvement in the area of neural network in possible future academic works on the subject. We can for example try to propose new ways of randomly defining the initials weights and biases. We could also try new cost and activation functions.

The ultimate demonstration that a code is indeed working properly is to test it on a particular data set. We experienced the difficulties that arise when it comes to putting our hands on interesting data sets about a certain topic. Consequently, we refocused our attention on a data set which is maybe less interesting by itself as it has been studied many times but offers the advantages to be easily available.

Using the previously written code in order to train a neural network in practice is a third achievement of the thesis. In that last part of the work, we could show the use of the performance indicators of a prediction model (sensitivity, specificity and accuracy).

## 4.2 Steps further

A natural step further in the classification problem with image data set is to implement the so called Convolutional Neural Network (CNN).

The architecture of a fully connected layers neural network doesn't take into account the spatial structure of the images. For each pixel in the input image, we enter the pixel's intensity as the value of a neuron of the input layer. That way, pixels located very closed to each other and pixels far apart from each other are treated on the same footing. In an image however, it is very likely that a pixel is related to the adjacent pixels as part of some component of objects represented on the image. In a fully connected layers neural network, the spatial structure of the image must be "learned" by the network making use of the training examples.

In CNN we try to use the spatial structure of the image as a starting point instead of inferring it from the data. This is done by "scanning" the image with different "filters" hoping that each filter will extract a certain characteristic of the object represented on the image. Practically, the filters are matrices, and the "scanning" operation consists in applying the convolutional operation. The values of the coefficients of the filter's matrices are learned by the network the same way as the weight and biases in fully connected networks.

The derivation of the equations of the backpropagation algorithm and their implementation in a program are significantly more difficult in CNN. Furthermore, the number of filters to be used increases a lot the calculation time of the program making it difficult to apply in practice in the context of this work.

Another improvement can be made investigating methods for setting the hyperparameters (network architecture,  $\eta$ ,  $\lambda$ ).

The brute force approach is the grid search, where we define sets of different possible values for those parameters, and then systematically consider every possible combination of values of the parameters and select the one that gives the best results on a validation data set.

With grid search we just make a series of evaluation of the model with different sets of hyperparameters, but we do not keep track of the past evaluation of the model to guide our search of the best set of hyperparameter. Furthermore, the systematic character of grid search supposes that all the hyperparameters have the same relative importance which is empirically not true.

There exists another approach called Bayesian optimization where we keep track of the past evaluations when choosing the set of hyperparameters to evaluate next. This informed way of choosing hyperparameters enables to focus on pieces of the hyperparameter space that seems to be more significant.

Again, computation time increases dramatically with those methods making them difficult to apply in practice in the context of this work.

# Bibliography

- Abu-Mostafa et al.(2012)** Y. Abu-Mostafa, M. Magdon-Ismail and L. Hsuan-Tien. *Learning From Data*. AMLBook, United States. quoted on page.
- Cybenko(1989)** G. Cybenko. Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals, and Systems*, páginas 303–314. quoted on page.
- Glorot and Bengio(2010)** X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *Journal of Machine Learning Research.*, páginas 249–256. quoted on page.
- Hornik(1989)** K. Hornik. Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, páginas 359–366. quoted on page.
- Lecun()** Y. Lecun. THE MNIST DATA BASE of handwritten digits. URL <http://yann.lecun.com/exdb/mnist/>. quoted on page.
- Lecun et al.(1998a)** Y. Lecun, L. Bottou, G B Orr and K. R. Muller. Efficient BackProp. páginas 9–48. URL *Neural Networks: Tricks of the Trade*. quoted on page.
- Lecun et al.(1998b)** Y. Lecun, L. Bottou, Y. Bengio and P. Haffner. Gradient-Based Learning Applied to Document Recognition. páginas 2278–2324. URL *Institute of Electrical and Electronics Engineers*. quoted on page.
- Nielsen(2019)** M. Nielsen. *Neural Network and Deep Learning*. Springer. quoted on page.
- Shalev-Shwartz(2014)** S. Shalev-Shwartz, S.and Ben-David. *Understanding Machine Learning from Theory to Algorithms*. University Press, Cambridge. quoted on page.