INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN Y DESARROLLO DE TECNOLOGÍA DIGITAL

MAESTRÍA EN CIENCIAS EN SISTEMAS DIGITALES

THESIS

MACHINE LEARNING LIBRARY TO SUPPORT APPLICATIONS WITH EMBEDDED SYSTEMS AND PARALLEL COMPUTING

TO OBTAIN THE DEGREE IN

MAESTRÍA EN CIENCIAS EN SISTEMAS DIGITALES

PRESENTS

CÉSAR MIRANDA MEZA

UNDER THE DIRECTION OF

DR. JUAN JOSÉ TAPIA ARMENTA

MARCH 2022 TIJUANA, B.C., MEXICO

SIP-13
REP 2017

INSTITUTO POLITÉCNICO NACIONAL
SECRETARIA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REGISTRO DE TEMA DE TESIS
Y DESIGNACIÓN DE DIRECTOR DE TESIS

Ciudad de México, a 28 de enero del 2022

El Colegio de Profesores de Posgrado del Centro de Investigación y Desarrollo de Tecnología Digital en su Sesión
(Unidad Académica)

Extraordinaria No. 02 celebrada el día 28 del mes de enero de 2022, conoció la solicitud presentada por el (la) alumno (a):

Apellido Paterno:	Miranda	Apellido Materno:	Meza	Nombre (s):	César

Número de registro: B 2 0 0 5 6 6

del Programa Académico de Posgrado: Maestría en Ciencias en Sistemas Digitales

Referente al registro de su tema de tesis; acordando lo siguiente:

1.- Se designa al aspirante el tema de tesis titulado:

Machine learning library to support applications with embedded systems and parallel computing.

Objetivo general del trabajo de tesis:

To provide a library, in C language, with machine learning algorithms that supports embedded systems and parallel computing to be used in research projects and industrial applications.

2.- Se designa como Directores de Tesis a los profesores:

Director: Dr. Juan José Tapia Armenta 2° Director:

No aplica: x

3.- El Trabajo de investigación base para el desarrollo de la tesis será elaborado por el alumno en:

El Centro de Investigación y Desarrollo de Tecnología Digital.

que cuenta con los recursos e infraestructura necesarios.

4.- El interesado deberá asistir a los seminarios desarrollados en el área de adscripción del trabajo desde la fecha en que se suscribe la presente, hasta la aprobación de la versión completa de la tesis por parte de la Comisión Revisora correspondiente.

Director(a) de Tesis

Dr. Juan José Tapia Armenta

Aspirante

Ing. César Miranda Meza

Presidente de Colegio

Dr. Julio César Rolón Garrido

SEP
INSTITUTO POLITÉCNICO NACIONAL
CENTRO DE INVESTIGACIÓN Y DESARROLLO
DE TECNOLOGÍA DIGITAL
DIRECCIÓN

SIP-14
REP 2017

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de Tijuana B.C. siendo las 13:30 horas del día 30 del mes de marzo del 2022 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Posgrado del: Centro de Investigación y Desarrollo de Tecnología Digital

para examinar la tesis titulada: Machine learning library to support applications with embedded systems and parallel computing.

del (la) alumno (a):

Apellido Paterno:	MIRANDA	Apellido Materno:	MEZA	Nombre (s):	CÉSAR

Número de registro: B 2 0 0 5 6 6

Aspirante del Programa Académico de Posgrado: MAESTRÍA EN CIENCIAS EN SISTEMAS DIGITALES

Una vez que se realizó un análisis de similitud de texto, utilizando el software antiplagio, se encontró que el trabajo de tesis tiene 12 % de similitud. **Se adjunta reporte de software utilizado.**

Después que esta Comisión revisó exhaustivamente el contenido, estructura, intención y ubicación de los textos de la tesis identificados como coincidentes con otros documentos, concluyó que en el presente trabajo **SI** ☐ **NO** ■ **SE CONSTITUYE UN POSIBLE PLAGIO.**

JUSTIFICACIÓN DE LA CONCLUSIÓN: *(Por ejemplo, el % de similitud se localiza en metodologías adecuadamente referidas a fuente original)*

El porcentaje de similitud sin aplicar filtros es del 12 % que corresponde a palabras o frases de uso común en el tema y en fuentes citadas adecuadamente. El documento de tesis presentado corresponde en su totalidad a la autoría del alumno.

****Es responsabilidad del alumno como autor de la tesis la verificación antiplagio, y del Director o Directores de tesis el análisis del % de similitud para establecer el riesgo o la existencia de un posible plagio.**

Finalmente y posterior a la lectura, revisión individual, así como el análisis e intercambio de opiniones, los miembros de la Comisión manifestaron **APROBAR** ■ **SUSPENDER** ☐ **NO APROBAR** ☐ la tesis por **UNANIMIDAD** ■ o **MAYORÍA** ☐ en virtud de los motivos siguientes:

Con base en los resultados teóricos y experimentales presentados, la comisión revisora determina que se cumplió con el objetivo general de la tesis y se determina que el trabajo es original.

COMISIÓN REVISORA DE TESIS

Dr. Juan José Tapia Armenta
Director de Tesis
Nombre completo y firma

Dr. Oscar Humberto Montiel Ross
Nombre completo y firma

Dra. Isaura González Rubio Acosta
Nombre completo y firma

M. en C. Luis Miguel Zamudio Fuentes
Nombre completo y firma

Dr. Leonardo Trujillo Reyes
Nombre completo y firma

Dr. Julio César Rolón Garrido
Nombre completo y firma
PRESIDENTE DEL COLEGIO DE PROFESORES

S E P
INSTITUTO POLITÉCNICO NACIONAL
CENTRO DE INVESTIGACIÓN Y DESARROLLO DE TECNOLOGÍA DIGITAL
DIRECCIÓN

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de Tijuana, Baja California. el día 29 del mes de Marzo del año 2022, el que suscribe César Miranda Meza, alumno del Programa de MAESTRÍA EN CIENCIAS EN SISTEMAS DIGITALES, con número de registro B200566, adscrito(a) al CENTRO DE INVESTIGACIÓN Y DESARROLLO DE TECNOLOGÍA DIGITAL, manifiesta que es el autor intelectual del presente trabajo de Tesis bajo la dirección de **Dr. Juan José Tapia Armenta** y cede los derechos del trabajo titulado **Machine learning library to support applications with embedded systems and parallel computing**, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del (de la) autor(a) y/o director(es) del trabajo. Este puede ser obtenido escribiendo a las siguientes direcciones Av. Instituto Politécnico Nacional No. 1310 Col Nueva Tijuana, Tijuana, Baja California, México, correo electrónico de contacto: posgrado@citedi.mx. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

César Miranda Meza

Nombre y apellidos del alumno

Acknowledgements

To God for arranging all the circumstances in my life that pushed me to be here today and to have completed my master's thesis, which allowed me to grow both personally and professionally.

To my family for supporting me and for believing in me, where I give special thanks to my mother: Minerva Antonieta Meza Burgara, my syster: Liviere Miranda Meza and my grandmother: Epigmenia Burgara Cazarez.

To my fiancée: Daniela Contreras Arroyo, for believing in me and being very understanding when I had a lot of work to do. I also give special thanks to her for mentoring and helping me to greatly increase my writing skills.

To "Consejo Nacional de Ciencia y Tecnología" (CONACYT) and "Centro de Investigación y Desarrollo de Tecnología Digital" (CITEDI) of the "Instituto Politécnico Nacional" (IPN), for the financial support they provided me during my postgraduate studies.

To my thesis director, Dr. Juan José Tapia Armenta, for allowing me to work with him and for believing in my potential as an engineer and as a researcher.

To Dr. Leonardo Trujillo Reyes, Dr. Oscar Humberto Montiel Ross, Dra. Isaura González Rubio Acosta, Dra. Jessica Beltrán Márquez and M. C. Luis Miguel Zamudio Fuentes, for all the valuable feedback that they gave to me and that helped me to improve my work.

To my fellow master's degree students, with whom I was able to live enriching and joyful moments during my postgraduate studies. In this respect, I would especially like to thank Marco Antonio Moran Armenta for supporting me in some academic activities.

To my friends for being very understanding when I had a lot of work and for supporting me to continue giving my all to my thesis. In particular, I give special thanks to Daniela Prieto Medina for her mentoring in writing during my thesis development.

BIBLIOTECA DE FUNCIONES DE APRENDIZAJE AUTOMÁTICO PARA ASISTIR A APLICACIONES CON SISTEMAS EMBEBIDOS Y COMPUTACIÓN PARALELA

Resumen

Las bibliotecas de funciones de aprendizaje automático actualmente disponibles han abordado fuertemente el aprendizaje profundo y la computación paralela, pero han dejado de lado los métodos tradicionales de aprendizaje automático y el soporte a los sistemas embebidos en comparación. Por ello, en esta tesis se ha desarrollado una nueva biblioteca de aprendizaje automático con un total de 53 funciones que aporta con 6 nuevos métodos tradicionales de aprendizaje automático, a la vez que soporta la computación paralela y los sistemas embebidos. Para tener una referencia con la que validar y comparar la biblioteca desarrollada, se eligieron las biblioteca Dlib, PyTorch, scikit-learn y TensorFlow como principales bibliotecas de comparación. Durante el proceso de prueba y validación se desarrollaron varios algoritmos en modo secuencial: 6 para métodos estadísticos; 6 para métodos de escalado de datos; 9 para métodos de métricas de evaluación; 12 para métodos de regresión; 12 para métodos de clasificación; y 2 para métodos de aprendizaje profundo. Además, se desarrollaron algoritmos adicionales para paralelizar los algoritmos de aprendizaje profundo en CPU, GPU única y GPU múltiple. Sin embargo, sólo 36 de los algoritmos secuenciales fueron comparados con algoritmos equivalentes y los resultados indican que alrededor del 83,33% de las funciones de la biblioteca de esta tesis fueron más rápidas; el 8,33% fueron igualmente rápidas; y otro 8,33% fueron más lentas. Por último, se realizaron algunas implementaciones en un Arduino UNO y un microprocesador STM32F446RE para probar el soporte a sistemas embebidos en la biblioteca desarrollada.

Palabras clave. Aprendizaje automático, biblioteca de funciones, sistemas embebidos, cómputo paralelo.

MACHINE LEARNING LIBRARY TO SUPPORT APPLICATIONS WITH EMBEDDED SYSTEMS AND PARALLEL COMPUTING

Abstract

The currently available machine learning libraries have strongly addressed deep learning and parallel computing, but have neglected traditional machine learning methods and support for embedded systems in comparison. Therefore, in this thesis, a new machine learning library with a total of 53 functions has been developed and contributes with 6 new traditional machine learning methods, while supporting parallel computing and embedded systems. For a reference against which to validate and benchmark the developed library, the Dlib, PyTorch, scikit-learn and TensorFlow libraries were chosen as the main comparators. During the testing and validation process, several algorithms were developed in sequential mode: 6 for statistical methods; 6 for feature scaling methods; 9 for evaluation metric methods; 12 for regression methods; 12 for classification methods; and 2 for deep learning methods. Furthermore, additional algorithms were developed in order to parallelize the deep learning algorithms in CPU, single GPU and multiple GPU. However, only 36 of the sequential algorithms were compared with equivalent algorithms and the results indicate that about 83.33% of the library functions of this thesis were faster; 8.33% were equally fast; and another 8.33% were slower. Finally, some implementations were made on an Arduino UNO and STM32F446RE microprocessor to test the support of embedded systems with the developed library.

Keywords. Machine learning, library, embedded systems, parallel computing.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Mathematics has been one of the main tools for the evolution of mankind over the years. From the moment it was used for transactions of goods, like when coins were exchanged for food, to the realization of big infrastructure such as pyramids, castles and so many houses. The counting and measuring of things was of great importance at the moment, but over time, a huge technological gap was overcome. This allowed humankind to finally learn about electricity and, above all, to learn how to manipulate it. Subsequently, this enabled the construction of intelligent devices that would change our way of living forever. From being able to now have small portable devices on which we can talk to whomever we want, no matter the distance, up to personal computers with the capacity to process massive amounts of information in the blink of an eye. These particular systems have contributed greatly to many of the technological advancements of today. As a result, a powerful mathematical tool known as machine learning was born and consists of some artificial intelligence algorithms that have the ability to learn the desired behavior of a given system [1, 2].

Although there have been various forms of artificial intelligence that were created before machine learning, they suffered from a major limitation. In comparison to machine learning, they had a very straightforward way of determining an outcome because they lacked a real sense of learning [2]. Hence, with the introduction of greater computing power and machine learning, artificial intelligence has had a great impact on both industrial and research fields. This is due to its broad application possibilities and its potential to predict the behavior of complex systems. For instance: biology, medicine, marketing, environmental energies, robotics, entertainment and social behavior are just some of the potential application fields. Within these, software programs for speech recognition [3], robot controllers [4], human emotion detectors [5], disease detectors [6], targeting customers [7], trend analysis [7], sales forecasting [7], big data handling [8], cyberattack prevention [9], energy efficiency optimizations [10] and hotel demand modeling [11] are some of the many applications being addressed with machine learning.

Despite the progress achieved in these areas, many applications of this field have become more complex and require more computational resources. This has led developers to strive for greater optimizations in machine learning programs. At the same time, this eventually created the need for a device with superior capacity: the high performance computing systems known as servers. These have significantly contributed to many of the important technological advancements of today, especially through what is known as parallel computing. This powerful tool allows us to process information many times faster than the traditional computational way, which is known as sequential computing.

In this sense, it is easy to think that parallel computing should have a strong trend, but as a matter of fact it does not. While, machine learning has had an exponentially increasing popularity since 2016 across multiple sectors [12, 13], but which has stagnated in recent years overall [13]; parallel computing has followed the opposite tendency over the past 15 years in general [14] and has been slowly increasing only within the scientific community [12]. In reality, this can be explained by many different issues, such as perhaps the lack of documentation friendly to beginners regarding the implementation of parallel computing, its high complexity and its steep learning curve.

From these possible factors, complexity is something that is very pronounced in parallel computing applications and it is even more noticeable depending on the type of parallelization. This is usually through: a Central Processing Unit (CPU) and a Graphics Processing Unit (GPU). Although GPUs have much more computing power than CPUs [15], one of the main problems is that parallel programming is even more complicated on a GPU than on a CPU. This is due to the limitations of writing a code for each of them, where what happens is that the GPU usually has many more cores than the CPU, in terms of thousands of cores on a GPU versus a few on a CPU [15].

In spite of this, several researchers and organizations have strongly introduced parallel programming in machine learning over the last few years. This has occurred either by adding this feature to an existing Application Programming Interface (API) or by implementing it from scratch. As a result, it is easy to find today several libraries for machine learning applications with parallel computing support, where some of the most well known are Caffe [16], scikit-learn [17] and TensorFlow [18]. Not only that, but these libraries are reliable and well equipped with several algorithms that are constantly being improved and can be used in a wide range of computers. However, based on the available literature, there have been few efforts to create and provide libraries for lower profile computing systems, such as embedded systems (e.g. FPGAs), in contrast to computers or servers and specially when referring to low profile embedded systems [19, 20, 21].

Therefore, this thesis aims to enable beginners to understand the basics of machine learning

and parallel computing through a literature review. At the same time, it is intended to provide a new machine learning library in C language that is competitive, transparent and friendly to use. For this reason, some reliability validations and execution time comparisons will be made with respect to commercially available versions. In addition, this software tool will be open source and will be usable in sequential or parallel mode and also in low profile embedded systems accordingly. As a consequence, this work will contribute with a new and competitive machine learning library that addresses the need of supporting embedded systems.

1.1 Problem statement

Several machine learning libraries have provided a wide range of algorithms that are reliable and are improved over time, such as Caffe [16], TensorFlow [18], Pytorch [22] and scikit-learn [17]. However, there have been few efforts in developing machine learning libraries that support low profile embedded systems in comparison [19, 20, 21]. In addition, several publications report difficulties when working with big data and machine learning, like managing that type of data and increasing the performance of machine learning algorithms under such circumstances [23, 24, 25]. Consequently, this work proposes a competitive library to address the following issues that could be part of this problem: 1) parallel programming is very complex to use; 2) the existing documentation for parallel programming is difficult to understand by the average programmer; and 3) there are still some tools that have not been fully exploited to increase the performance of machine learning implementations (e.g., while few offer an interface with C language, most of them provide it with python instead, which has less performance).

1.2 Research questions

This work has been delimited to certain research problems, which are the driving force behind the motivation to work on this thesis. Therefore, the following is a list of these questions:

- What machine learning libraries offer parallel computing support?
- What machine learning libraries provide support to embedded systems?
- What are the most representative machine learning libraries and what are their main methods?
- What are the main contributions that have been made in the regression and classification methods?

1.3 Justification

According to the data from the job market report of Glassdoor conducted in 2020 on the USA, there were around 500'000 open job positions for the tech industry, ranking it as the third industrial sector with the highest demand [26]. In addition, artificial intelligence and data science jobs were considered to be in the rise by the report of LinkedIn from 2021 [27]. This trend in employment demand has had a similar effect in several other countries like Mexico [28]. For these reasons, it is not surprising that both the scientific community [12] and large companies such as Amazon, Facebook, Apple, IBM and Google, have invested heavily in machine learning applications and research [1]. Therefore and without a doubt, there is an existing demand and niche of opportunities for jobs that require artificial intelligence in North America, where this project aims to impact.

In spite of this, there are several areas of opportunities that require improvement on some machine learning applications. As mentioned before, there are management issues with big data [23, 24, 25] and there have been few efforts in developing machine learning libraries that support embedded systems in contrast to high performance machines [19, 20, 21]. In addition, there is also enormous potential for further innovation with new features, specialized applications and techniques. Moreover, most machine learning libraries heavily focus on providing deep learning methods and neglect all others in comparison. Conversely, the study of kaggle shows that some traditional regression and classification methods are used more than deep learning methods by currently employed professionals with the job title of "data scientists" [29].

Therefore, this thesis proposes the development of a library in C language that will be made available with machine learning algorithms. For this purpose, a portable code will be provided that will not only be usable in low profile embedded systems with microcontrollers or microprocessors, but that will also offer parallel computing for high performance systems in order to be competitive. Furthermore, the resulting algorithms will be properly validated and benchmarked to ensure reliability and a competitive execution time with respect to existing commercial alternatives. On the other hand, to compensate for the fact that C programming is more complex than other popular programming languages, transparency and a literature review is proposed to facilitate the understanding of the basics of machine learning. The way this information is to be presented will be directly related to the way the library of this thesis will work, thereby facilitating its understanding and use. Consequently, this work will contribute with a new machine learning tool that gives a new vision and more possibilities for machine learning applications in the industrial and scientific sector.

1.4 Hypothesis

- **H0**. The developed library will not have reliable results and will not have a faster execution time than the existing machine learning libraries with respect to their sequential processing version.
- **H1**. If the library is developed, then it will have reliable results and will have a faster execution time with respect to the current machine learning libraries in their sequential processing version.

1.5 General objective

To provide a library, in C language, with machine learning algorithms that supports embedded systems and parallel computing to be used in research projects and industrial applications.

1.5.1 Specific objectives

- Develop a general background for the identification and selection of the machine learning libraries and algorithms that will serve as a reference for the library to be developed in this thesis.
- Develop the chosen machine learning algorithms in their sequential CPU version to add them to the library to be developed.
- Benchmark the machine learning algorithms developed in their sequential CPU version to show their execution time with respect to the reference libraries.
- Improve the deep learning algorithms developed for the library of this thesis by implementing CPU, single and multiple GPU parallelism to provide a means of obtaining faster results with such type of algorithms.
- Identify whether the library developed in this thesis can operate in an Arduino UNO and STM32F446RE development board to determine its ability to function on these two low profile embedded systems.
- Demonstrate the credibility of the algorithms developed for the library of this thesis by creating and implementing a validation mechanism to promise reliability to its users.

1.6 Organization of the thesis

The rest of the document is organized as follows: Chapter 2 sets out the theoretical framework concerning the central concepts of machine learning; the background of machine learning libraries and some detailed information on the most representative of them. Chapter 3 gives the mathematical formulation of the methods that were implemented in the library made in this thesis. Chapter 4 describes the methodology that has been followed during the development of this work and also provides some machine learning metrics that are available in the library that was developed in this thesis. Chapter 5 shows the results of this thesis, which were obtained in an objective and organized way but without any interpretation of the data to give a clear idea of what has been found in this work. Chapter 6 discusses the interpretation given to the results obtained in the previous Chapter. Finally, Chapter 6 presents the conclusions of this thesis.

Chapter 2

Theoretical framework

According to the research conducted in this thesis, it is difficult to say when artificial intelligence originated through the efforts of mankind. However, a good startup point lies in the first time the word "robot" came to be introduced, which was through a play called Rossum's Universal Robots (RUR), written by Karel Čapek in 1920 [30]. At that time, the concept referred to an artificial living organism that has the ability to think and act on its own, but that was created to do the work of humans. Despite the way the word "robot" was described there, it actually resembled the current concept of androids, as they were made with living organic matter in the image and likeness of humans.

As a consequence of giving birth to this word, there were several explorations of the concept in the literature, plays and the world of cinema. Subsequently, having inspired people all over the world, several scientists began to make attempts to create robots. For example, when Dr. Makoto Nishimura created the first friendly robot (Gakutensoku) in Japan in 1929, inspired by the play RUR [31]. Another example is the influential article "A logical calculus of the ideas immanent in nervous activity", published in 1943. This paper laid the foundation for artificial neural networks, which would later be the inspiration for what we know today as deep learning [32]. Later, in 1955, the paper "A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence" was presented and laid one of the first groundwork for the term "artificial intelligence" [33].

2.1 Artificial intelligence

Despite that humankind has been creating and using artificial intelligence for more than 50 years, an interesting fact is that there is still no unanimous definition for it. Among other factors, it is possible that the evolution of artificial intelligence over the years has something to do with it. Nevertheless, some of the several important contributions that have been made to

this concept will be shown, in order to extrapolate a definition:

- In 1955, John McCarthy and other scientists banded together to give meaning to the term "artificial intelligence":
  - "Every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it." [33].
- In 1978, Bellman contributed with the following:
  - "[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning ..." [34].
- In 1985, Charniak and McDermott provided their own definition as follows:
  - "The study of mental faculties through the use of computational models." [34].
- In 1992, Winston presented:
  - "The study of the computations that make it possible to perceive, reason, and act." [34].

For the following concept to be developed, the ideas that will be taken into account are: 1) that any sense of intelligence should be able to be simulated by a computational system, inspired by the definitions of McCarthy, Charniak, McDermott and Winston and 2) to use models inspired by living beings in general, as interpretable from Winston's definition, instead of only human beings as Bellman states. Conclusively, this thesis contributes with the proposal of the following new definition:

Artificial intelligence is the field that studies, develops and implements computer programs to perform certain tasks and that are capable of simulating the full or partial faculty or capacity of cognitive functions.

2.2 Machine learning

Originally, the term "machine learning" was coined by Arthur Samuel in 1959, when he reported on a self learning checkers computer program [35]. But instead of developing a definition for this term in his paper, it was used considering its meaning as intuitive. Consequently, although there were several subsequent contributions on the subject, hardly any effort was made to develop its concept. This led to machine learning being initially conflated with the concept of

artificial intelligence. However, in recent years there have been several individuals who have provided reliable definitions of this term, such that a representative definition may be the following:

Machine learning is a branch of artificial intelligence that studies, develops and implements computer self-learning algorithms to model and predict the behavior of a determined system [1, 36, 37].

2.2.1 Machine learning algorithms

Machine learning algorithms have great potential because together they can cover a wide range of applications. The reason for this lies in their objective, which is to learn and predict the behavior of a given system. In regards to this, all these algorithms strictly require going through a training process. Subsequently, the mathematical model that best predicts the behavior of the system under study, will be generated but within the limitations of the chosen method. This occurs because there are several categories of methods in machine learning that generate models under different circumstances and that serve different purposes. Therefore, the following are some of the distinct categories into which machine learning algorithms can be divided:

- **Supervised learning**: It comprises all those machine learning algorithms that are distinguished by the fact that their learning process requires the use of a historical record of samples. That is, they need to learn from samples that have their corresponding specified result [36, 37] before being able to make predictions.
  - Regression algorithms: Refers to those algorithms that learn to obtain a model whose output represents or attempts to provide a continuous outcome [1, 37]. Usually, these algorithms achieve this by figuring out the mathematical function that generates the best fitting trace/plot for a certain group of scattered data (e.g., some market analysts may use it to predict the revenues that a given company will earn over a later period of time).
  - Classification algorithms: This type of algorithms learn to obtain a model whose output predicts the belonging group of a given set of features after having been trained on a certain data set [1, 37] (e.g., some individuals have been identifying and determining what type of emotion did a certain human felt through the use of this type of algorithms) [5].
- **Unsupervised learning**: It includes all those machine learning algorithms that are characterized by learning through the detection of similarities in the features of the system

under study. In addition, during their training stage they learn from samples that do not have any labeled result [36, 37].

  - Clustering algorithms: It is composed of those algorithms that attempt to identify classification groups based on significative characteristics [1, 2]. These also differ from the classification algorithms mainly because the former are trained only with unlabeled data sets and the latter are not (e.g., some people have developed a computerized smart agent that prevents and predicts cyberattacks by learning to identify different clusters/groups of computer users and then taking a certain decision on those groups that perform actions uncommon to authorized users) [9].
  - Association rule learning algorithms: These type of algorithms are data mining methods that are used to find out the probability of consequent actions or rules (e.g., some market analysts use it to figure out the probability that a customer will buy a certain product, but after he/she has already bought a specific one) [38].

- **Reinforcement learning**: These are all those machine learning algorithms that learn by trial and error and are distinguished by rewarding the algorithm after it succeeds in making a certain order of decisions in accordance with the established rules [37, 36]. This means that these algorithms need to know whether the decision/prediction they have made is correct or not after they have taken action (e.g., a drone that gets a software reward when it figures out the best trajectory to perform a certain task and is penalized by the software if it fails to do so. Therefore, after it has had enough time to learn about the best trajectories, it will potentially choose the best trajectory from there. [4]).

- **Special cases**: The following can be applied to any of the described types of machine learning algorithms (supervised, reinforced and unsupervised learning).

  - Deep learning algorithms: These are those algorithms that are distinguished by their attempt to mimic the functionality of the neuron when trying to learn the behavior/model of a certain system under study [1]. These algorithms can be used to solve any of the previously mentioned types of machine learning algorithms.

2.2.2 Machine learning libraries

Just after the term "artificial intelligence" was coined by the group of scientists of which John McCarthy was a member [33], the design of the LISP programming language began. This took place in 1958, when McCarthy became an assistant professor at the Massachusetts Institute of Technology (MIT), where he initiated that project [39]. The importance of this development

was great because LISP became one of the first means of programming artificial intelligence [40]. However, just as McCarthy had expected, it was just a matter of time that LISP became a forgotten tool due to several inefficiencies in it [39]. Consequently, between this first attempt to program artificial intelligence until today, several improved developments were made.

Among all the contributions to program artificial intelligence, there were also those intended solely for machine learning. As a result, many software libraries have been introduced for this field. Yet, despite these endeavors, several of them have been discontinued over time, even within the most recent ones. Therefore, in an attempt to understand their evolution, the following Table shows the history of some of them from 2008 to the present day. There, this thesis was delimited to only libraries that have been archived and that still exist in a code repository in GitHub [41].

Table 2.1: Background summary of machine learning libraries in GitHub from 2008 to 2022.

Library name	Release date [1]	Actively developed [2]	Interface [3]	Embedded system support	CPU parallelism support	GPU support	multi GPU support
Dlib	2008	√	C++, Python		√	√	√
scikit-learn	2010	√	Python		√		
Torch	2012		Lua		√	√	√
Caffe	2013		C++, Python		√	√	√
MicrosoftCognitive Toolkit (CNTK)	2014		Python, C++, C#/.NET, Java		√	√	√
Keras	2015	√	Python		√	√	√
Chainer	2015		Python		√	√	√
Apache SINGA	2015		Python		√	√	√
Apache MXNet	2015	√	Python, Java, C++, R, Scala, Clojure, Go, Javascript, Perl, Julia		√	√	√
TensorFlow	2015	√	Python, C++	√	√	√	√
ELL	2015		Python	√			
PyTorch	2016	√	Python, C++	√	√	√	√
Flux	2016	√	Julia			√	
BigDL	2017	√	Python, Scala		√		
PlaidML	2017	√	Python			√	
Seq2SeqSharp	2018	√	C#		√	√	√
Deeplearning4j	2019		Scala, Java		√	√	√
TinyML	2021		Python	√			

[1] The release date defined for each library is based on the first commit of its corresponding repository.

[2] A library will be considered actively developed if it has code development at least once every three months for six months prior to January 2022. If this is not met or if the owner has stated that there will be no further developments, then the library will not be considered actively developed.

[3] Only those programming languages declared as stable by the developers will be listed.

The impression gained from inspecting each of the libraries in Table 2.1 is as follows. The documentation for older libraries is often more comprehensive and user friendly than for newer libraries. Also, the latest machine learning libraries tend to be actively developed relative to older ones. In addition, no matter how old the library is, if it is actively developed, it is likely to be updated with respect to new machine learning algorithms and have more efficient code. Furthermore, only four libraries were identified to support embedded systems, of which just two of them are actively developed but they only support high profile embedded systems (e.g., FPGAs, NVIDIA Jetson Nano, and other embedded systems capable of running an operating system). Lastly, most machine learning libraries from 2008 to today are likely to have CPU parallelism support and single and multiple GPU support. Therefore, as a consequence of these observations, there are a wide range of machine learning libraries that could potentially be used as a comparative reference for this thesis.

In addition to the observations made, the following will be taken into account for the selection of the representative libraries to be used as a reference. Libraries: 1) with an older release date in GitHub that are still under active development as long as they have complete documentation; 2) that have several machine learning algorithms; 3) that together have different application purposes (e.g., for big data or embedded systems) or methods (e.g., neural networks); and 4) that together provide several of the features that can be found today as described in Table 2.1. Therefore, the libraries that will be considered as representative to be used as a comparative reference for this thesis will be the following:

- Dlib [42]
- scikit-learn [43]
- TensorFlow [44]
- Pytorch [22]

These selected machine learning libraries cover a wide range of methods together, but focus more on deep learning in general. This can be demonstrated through the next table which details most of the methods they provide, according to what was identified during the literature review conducted for this thesis. There, it can be seen that the ones who specialize only on deep learning methods are PyTorch and TensorFlow. On the other hand, Dlib and scikit-learn provide solutions for many other types of machine learning methods. However, despite the contrast in the number of methods that these libraries have, they are all reliable as they are based on formal scientific publications and are constantly being improved.

Table 2.2: Main machine learning methods provided by the selected libraries to be used as a reference.

Machine learning methods		Reference machine learning libraries			
		Dlib	PyTorch	scikit-learn	TensorFlow
Regression	Decision tree regression			√	
	Kernel recursive least squares	√			
	Kernel ridge regression	√		√	
	Linear ridge regression	√		√	
	Gaussian process regression			√	
	Linear regression			√	
	Linear support vector regression	√		√	
	Logistic regression			√	
	Nearest neighbors regression			√	
	Random forest regression	√		√	
Classification	Naive bayes classifier			√	
	Linear logistic classifier			√	
	Decision tree classifier			√	
	Gaussian process classifier			√	
	Linear support vector classifier	√		√	
	Nearest neighbors classifier			√	
	One vs one classifier	√		√	
	One vs rest/all classifier	√		√	
	Random forest classifier			√	
	Relevance vector classifier	√			
	Kernel support vector classifier	√		√	
Reinforced learning	Least squares policy iteration	√			
Clustering	Affinity propagation			√	
	Agglomerative clustering	√		√	
	BIRCH algorithm			√	
	Chinese whispers	√			
	DBSCAN algorithm			√	
	K means	√		√	
	Mean shift			√	
	Newman clustering algorithm	√			
	OPTICS algorithm			√	
	Sammon projection	√			
	Spectral clustering	√		√	
	Hierarchical clustering			√	
Deep learning	Multilayer perceptron	√	√	√	√
	Convolutional neural networks	√	√		√
	Recurrent neural networks		√		√
	Restricted boltzmann machine		√	√	
	Autoencoders				√

2.2.3 General pipeline of machine learning applications

Machine learning pipelines differ depending on whether their application is in research or in production, of which there are several variants, making the generalization of these pipelines complex [45]. However, although this thesis suggests to review other papers for more details on these type of pipelines, such as reference [45], a simple and general pipeline for machine learning libraries will be described below, taking into account that a new library will be developed in this work:

1. **Import of the data sets:** "This process consists in only importing the data sets that will be required for the machine learning application to solve" [46].

2. **Preprocessing of the data:** "On this stage, the imported data will be processed so that it can be properly used by the machine learning algorithms, but only if needed. Data preprocessing may include the removal or treatment of missing data in the data set to work with. In addition, it may also include encoding of categorical data, which consists of using dummy variables in categorical input values that were not labeled/defined numerically" [46].

3. **Data splitting:** "Although this step is not strictly necessary from a functional point of view, it is highly recommended to consider it in order to avoid biases in the model that could be generated. Thus, data would be divided into a training and test set or into a training, test and cross-validation set, when a more reliable model is desired" [46].

4. **Feature scaling:** "Despite also being an optional phase, the feature scaling (also known as data scaling) can have a great impact on the results of the modeling stage. This is because this tool standardizes/normalizes the input data that will be used to train the chosen algorithm. Therefore, allowing some machine learning algorithms to perform better in cases of a mixture of slow and abrupt changes in largely scattered input data" [46].

5. **Data modeling:** "This procedure consists of selecting a machine learning algorithm and then to train it with the resulting training data from the previous processes." [46].

6. **Model validation:** "A validation of the model with the test and/or cross validation data can be applied to determine whether the model is good or not with the help of evaluation metrics (see subsections 4.3 and 4.4). As a consequence of this, if the model has not been able to meet the desired expectations, the data modeling step can be repeated but with a different algorithm" [46]. However, another option is to repeat the stage involving the preprocessing of the data to try to obtain different results with a different arrangement or strategy for the input data.

2.3 Parallel computing

Machine learning is a tool that has contributed and been involved in several different fields, but this is a merit that it cannot take on its own. Similarly, today it is well accepted that computers take a significant part of that credit, as they have greatly facilitated the implementation of machine learning methods. One of the reasons is that these machines are a means capable of calculating hundreds, thousands and even a larger number of operations in a matter of seconds or less. The speed at which the results of these systems can be delivered will depend strictly on the processing mode that is used. One of these options is the processing mode known as sequential computing and the other is called as parallel computing, which can be further enhanced through a high performance computing system. Therefore, considering the importance and relevance of these three concepts for this thesis, their meaning will be described bellow:

- **Sequential computing** is also known as traditional computer processing and consists of executing several instructions in a consecutive order, in other words, one instruction at a time [47].
- **Parallel computing** represents the computational process of executing several instructions simultaneously, in other words, several instructions at the same time, which allows faster computations than the traditional way [48].
- **High performance computing** is difficult to describe because its concept is always changing due to the constant evolution of computer system hardware. However, it can be generalized as any computational system that employs multiple processing units when executing a determined program with high throughput and efficiency [49].

Although many of the present day computers have the capability to run processes in parallel, there are many that run them in sequential computing. The reason may be that there are some computers that cannot have parallel functionality as they must meet some hardware criteria to do so. In this sense, one way to program a computer system in parallel is to at least have two computers. Another way is to have a computer with a CPU that has at least two cores or instead a GPU, which will always have several cores. In any of these options, the idea is to distribute the work to be done among the available computers and/or cores that your entire system has.

2.3.1 Performance analysis for parallel computing

While parallel computing offers the possibility of increasing the speed of a given program with respect to its sequential version, this improvement is not linearly proportional. This is due to

the nature of how computational systems work, which results in the need to always have at least a part of the code in sequential mode. In this sense, Eq. (2.1) represents the law of Amdahl [50], which makes it possible to predict an estimate of the expected speed increase when parallelizing a program. However, this equation does not consider the influence of the number of physical units used in the parallelization process. As a consequence, Eq. (2.2) is suggested by Gustafson [51], being a modified form of the law of Amdahl that takes this issue into account.

$$speedup = \frac{1}{1 - P_{cp}} \tag{2.1}$$

Where:

- P_{cp} = Part of the code that can be parallelized, where $0 \leq P_{cp} < 1$.

$$speedup = \frac{1}{\frac{P_{ip}}{N_p} + S} \tag{2.2}$$

- P_{ip} = Part of the code that is parallelized, where $0 \leq P_{ip} \leq 1$ and $P_{ip} + S = 1$.
- N_p = Number of physical units used for parallelization, where $N_p = 1, 2, 3, ..., \infty$.
- S = Part of the code that is sequential, where $0 \leq S \leq 1$ and $P_{ip} + S = 1$.

2.3.2 Parallel programming libraries

There have been a lot of contributions to parallel computing over the years, resulting in several reliable options for implementing it. Among them, it is possible to find libraries for applications with parallelism on CPU; GPU and for both at the same time [52]. In this sense, while for CPUs two popular choices are the POSIX Threads (Pthreads) and OpenMP libraries, for GPUs two of them are CUDA and OpenACC [52]. Likewise, the OpenCL library is an option that allows parallelization with either CPUs or GPUs [52]. Yet, it is possible to create programming applications that simultaneously use a parallel programming library dedicated to the CPU and another one for the GPU.

Irrespective of how competitive and trustworthy parallel programming libraries are, their learning curve is long compared to any other high level programming language or to most libraries in general. This is also significantly intensified when dealing with an application involving a higher level of parallelization. Moreover, this can also be exacerbated when using a high performance computer due to the higher programming complexity required for it. Therefore, after having this into consideration, along with factors like the skills acquired during the development of this thesis and the time available to complete it, the conclusion is to work with the following:

- **CPU parallel library:** POSIX Threads, being a parallel programming library for CPUs that uses the multithreaded application model; can be used in C programming language; and is widely available in UNIX platforms [52].
- **GPU parallel library:** CUDA, which is a parallel programming library for GPUs that uses the CUDA model; can be programmed in C language; and supports NVIDIA GPUs [52].

Chapter 3

Mathematical formulations

Practitioners of machine learning software have at their disposal a broad range of libraries with different specializations, features and algorithms to choose from when solving a suitable problem. The latter is normal in machine learning because there is no unanimous best algorithm as each outperform the others under a very wide variety of circumstances. Moreover, machine learning libraries are growing in number and are also constantly improving, as described in Table 2.1. This also gives an indication that there is still a large niche of opportunities in this field. For example, some efforts have been made to develop new approaches with unified philosophies, such as when Dr. Jia Tangqing (creator of Caffe) joined the TensorFlow team. Another example lies in the new characteristics that have been introduced over time through the machine learning libraries shown in Table 2.1. Consequently, to develop a competitive library in this thesis, new approaches will be formulated and all the mathematics will be done from scratch to better understand how to make the fastest implementation possible.

For all the following mathematical formulations, have in mind that since the library of this thesis is intended to be transparent, some of these formulations could be seen as overdeveloped, but they will be defined as such because that is how this library will handle the corresponding input or output data. On the other hand, the statistical methods that were considered as a complement to the library of this thesis will be described first. Secondly, the feature scaling methods that were developed in this library will be formulated. Lastly, all the machine learning methods developed in that same library will be described mathematically, starting with the traditional regression methods; then the traditional classification methods; and finally, some deep learning methods.

Before starting, unless otherwise specified, let us define a vector $\boldsymbol{x}_i$ to represent the real input values or features of the machine learning algorithm to be trained (independent variables); $\boldsymbol{y}_i$ stands for the real values of the sampled output data of the system under study (dependent variables) and $\hat{\boldsymbol{y}}_i$ corresponds to the predicted values of the trained machine learning algorithm

(dependent variables), such that for one sample:

$$\boldsymbol{x}_i = \left(x_{i,1}, x_{i,2}, \cdots, x_{i,m}\right) \in R^m \tag{3.1}$$

$$\boldsymbol{y}_i = \left(y_{i,1}, y_{i,2}, \cdots, y_{i,p}\right) \in R^p \tag{3.2}$$

$$\hat{\boldsymbol{y}}_i = \left(\hat{y}_{i,1}, \hat{y}_{i,2}, \cdots, \hat{y}_{i,p}\right) \in R^p \tag{3.3}$$

In this sense, when considering a data set with n samples, instead of Eq. (3.1) the following should be used instead:

$$\boldsymbol{X} = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{pmatrix} \tag{3.4}$$

where each column represents a different independent variable or machine learning feature that can be any value from 1 to m. Moreover, instead of using Eq. (3.2) when considering n samples, the following should be used instead:

$$\boldsymbol{Y} = \begin{pmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,p} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ y_{n,1} & y_{n,2} & \cdots & y_{n,p} \end{pmatrix} \tag{3.5}$$

and accordingly, the following should be used instead of Eq. (3.3):

$$\hat{\boldsymbol{Y}} = \begin{pmatrix} \hat{y}_{1,1} & \hat{y}_{1,2} & \cdots & \hat{y}_{1,p} \\ \hat{y}_{2,1} & \hat{y}_{2,2} & \cdots & \hat{y}_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{y}_{n,1} & \hat{y}_{n,2} & \cdots & \hat{y}_{n,p} \end{pmatrix} \tag{3.6}$$

where for Eq. (3.5) and (3.6), each row stands for a different sample of the machine learning algorithm and each column represents a different dependent variable or machine learning output, which can be up to p outputs.

3.1 Statistical equations

In the process of generating a model with machine learning, several statistical equations are commonly used. Some of them can even be applied after the generation of such model to obtain a greater margin of knowledge of how the system under study will actually behave. Hence, some statistical methods to be used within the mathematical formulations of machine learning or to improve its obtained results will be described bellow. For more details on the statistical equations to be covered, the book [53] is suggested.

3.1.1 The mean of a sample set

When analyzing a given system, it is common to obtain n samples of its behavior and then apply a certain mathematical operation to obtain information about it. One approach is to identify the central behavior $\bar{x}$ of the samples of such system, so that according to Eq. (3.4) and by considering the existence of a single independent variable, its mathematical function is given by the following:

$$\bar{x}_1 = \sum_{i=1}^{n} \frac{x_{i,1}}{n} = \frac{x_{1,1} + x_{2,1} + \cdots + x_{n,1}}{n} \tag{3.7}$$

In the same manner, when multiple independent variables are considered, according to Eq. (3.4) and Eq. (3.7), the central behavior $\bar{\boldsymbol{x}}$ is given by the following:

$$\bar{\boldsymbol{x}} = \left(\bar{x}_1, \bar{x}_2, \cdots, \bar{x}_m\right) = \left(\sum_{i=1}^{n} \frac{x_{i,1}}{n}, \sum_{i=1}^{n} \frac{x_{i,2}}{n}, \cdots, \sum_{i=1}^{n} \frac{x_{i,m}}{n}\right) \tag{3.8}$$

3.1.2 The median (second quartile) of a sample set

Another approach to obtain information from a given set of samples is to acquire its central tendency, which can be obtained with the median Q_2. To acquire it for when there is only one independent variable in Eq. (3.4), the sample values must be sorted in ascending order and then apply:

$$Q_{2_1} = \begin{cases} x_{(\frac{n+1}{2},1)} & \text{if } n \text{ is odd} \\ \dfrac{x_{(\frac{n}{2},1)} + x_{(\frac{n}{2}+1,1)}}{2} & \text{if } n \text{ is even} \end{cases} \tag{3.9}$$

Likewise, when considering multiple independent variables, according to Eq. (3.4) and Eq. (3.9), the median $\boldsymbol{Q_2}$ becomes determined by the following:

$$\boldsymbol{Q_2} = \left(Q_{2_1}, Q_{2_2}, \cdots, Q_{2_k}, \cdots, Q_{2_m}\right), \tag{3.10}$$

where Q_{2_k} is given by:

$$Q_{2_k} = \begin{cases} x_{(\frac{n+1}{2},k)} & \text{if } n \text{ is odd} \\ \dfrac{x_{(\frac{n}{2},k)} + x_{(\frac{n}{2}+1,k)}}{2} & \text{if } n \text{ is even} \end{cases} \tag{3.11}$$

3.1.3 The variance of a sample set

A different way to obtain information about a set of samples is to obtain their variance s^2 with q degrees of freedom, which is a way of measuring the dispersion among certain samples. To obtain it, the following must be applied when there is only one independent variable in Eq. (3.4):

$$s_1^2 = \sum_{i=1}^{n} \frac{(x_{i,1} - \bar{x}_1)^2}{n - q} \tag{3.12}$$

where it is suggested that $q = 1$ [53].

Correspondingly, with multiple independent variables, in accordance with Eq. (3.4) and Eq. (3.12), the variance $\boldsymbol{s^2}$ is denoted by the following:

$$\boldsymbol{s^2} = \left(s_1^2, s_2^2, \cdots, s_m^2\right) = \left(\sum_{i=1}^{n} \frac{(x_{i,1} - \bar{x}_1)^2}{n - q}, \sum_{i=1}^{n} \frac{(x_{i,2} - \bar{x}_2)^2}{n - q}, \cdots, \sum_{i=1}^{n} \frac{(x_{i,m} - \bar{x}_m)^2}{n - q}\right) \tag{3.13}$$

3.1.4 The standard deviation of a sample set

Similarly to the variance, another method to measure the dispersion among certain samples is through the standard deviation s with q degrees of freedom. To obtain it when there is only one independent variable in Eq. (3.4), the following must be applied:

$$s_1 = \sqrt{s_1^2} = \sqrt{\sum_{i=1}^{n} \frac{(x_{i,1} - \bar{x}_1)^2}{n - q}} \tag{3.14}$$

where it is suggested that $q = 1$ [53].

In the same way, whenever multiple independent variables are considered, given Eq. (3.4) and Eq. (3.14), the standard deviation $\boldsymbol{s}$ becomes determined by the following:

$$\boldsymbol{s} = \left(s_1, s_2, \cdots, s_m\right) = \left(\sqrt{\sum_{i=1}^{n} \frac{(x_{i,1} - \bar{x}_1)^2}{n - q}}, \sqrt{\sum_{i=1}^{n} \frac{(x_{i,2} - \bar{x}_2)^2}{n - q}}, \cdots, \sqrt{\sum_{i=1}^{n} \frac{(x_{i,m} - \bar{x}_m)^2}{n - q}}\right) \tag{3.15}$$

3.1.5 The mode of a sample set

A different way to obtain information from a particular sample set is to obtain its mode. For one independent variable, it will be interpreted as M_o and for multiple independent variables it will be:

$$\boldsymbol{M_o} = \left(M_{o_1}, M_{o_2}, \cdots, M_{o_m}\right) \tag{3.16}$$

in accordance to Eq. (3.4). Furthermore, the determination of the value for each M_{o_k} is made by setting the most frequently repeated value among all the available samples of that particular k-th independent variable. However, if it turns out that there are several values that fit this criteria, then all of them would represent the mode of M_{o_k}.

3.1.6 Mean intervals

The mathematical method of the mean intervals is used to estimate a parameter belonging to a given population. It is useful when the true mean μ_k of a certain parameter is unknown and it is desired estimate the range in which μ_k will be. For this purpose, this method produces two different results: 1) The lower mean interval $\hat{\theta}_L$ and 2) The upper mean interval $\hat{\theta}_U$, such that:

$$\hat{\theta}_L < \mu_k < \hat{\theta}_U \tag{3.17}$$

where:

- $\hat{\theta}_L = \bar{x}_k - z_{\alpha/2}\dfrac{\sigma_k}{\sqrt{n}}$
- $\hat{\theta}_U = \bar{x}_k + z_{\alpha/2}\dfrac{\sigma_k}{\sqrt{n}}$

and where σ_k is the true value of the standard deviation and z is the standard normal distribution, but that will not be discussed in this thesis (for more details see [53]) and the appendix table A.3 of [53] will be used instead. In addition, α plays an important role in Eq. (3.17) because it determines what is know as the trust interval, which is given by $(1-\alpha)100\%$.

Moreover, as powerful and useful as the mean intervals can be, there are two important matters to consider in order to know whether they are representative of the population under study. The first is to have a preliminary sample of size $n \geq 30$ in the case of not knowing the value of σ_k so that it is possible to obtain good results [53]. Finally, the second is that there can be a $(1-\alpha)100\%$ confidence that the error e_k of using $\bar{x}_k$ as an estimate of μ_k will not exceed a specified amount of error for when the sample size is equal or greater than η, which is given by the following [53]:

$$\eta = \left(\frac{(z_{\alpha/2})(\sigma_k)}{e_k}\right)^2 \tag{3.18}$$

Nonetheless, if σ_k is not known and an approximation of it (s_k) is used, then substituting the standard normal distribution for the t distribution in Eq. (3.17) is recommended [53]:

$$\bar{x}_k - t_{\alpha/2}\frac{s_k}{\sqrt{n}} < \mu_k < \bar{x}_k + t_{\alpha/2}\frac{s_k}{\sqrt{n}} \tag{3.19}$$

where such equation has $n-1$ degrees of freedom and the t distribution ($t_{\alpha/2}$) will be calculated through the appendix table A.4 of [53].

3.2 Feature scaling for machine learning

In machine learning it is possible to obtain outstanding results by training one of its algorithms without any form of preprocessing on the data given to it. Nonetheless, depending on the algorithm being used and how reliable and well processed the data set is, this may not always be the case [1, 2]. Thus, a good practice is to apply feature scaling $\dot{x}_{i,k}$ to the each input data of the algorithm to be trained [2, 1]. However, because these methods can outperform each other under different circumstances, this thesis provides several feature scaling methods. Those selected were chosen because they were available in the representative libraries, as described in their documentation [42, 43, 44, 22].

3.2.1 Min max normalization

When implementing feature scaling to a certain data set, an approach to do so is through a well known method such as the min max normalization [1]. The definition of how to apply this feature scaling method when there is one independent variable in Eq. (3.4), is described below:

$$\dot{x}_{i,k} = \frac{x_{i,k} - x_{min,k}}{x_{max,k} - x_{min,k}} \quad | \quad 0 \leq \dot{x}_{i,k} \leq 1 \tag{3.20}$$

where the i-th value represents the current sample number and the k-th value stands for the current independent variable that is being evaluated.

The following establishes how to perform the min max normalization when there are multiple independent variables in Eq. (3.4):

$$\dot{\boldsymbol{X}} = \begin{pmatrix} \dot{x}_{1,1} & \dot{x}_{1,2} & \cdots & \dot{x}_{1,m} \\ \dot{x}_{2,1} & \dot{x}_{2,2} & \cdots & \dot{x}_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ \dot{x}_{n,1} & \dot{x}_{n,2} & \cdots & \dot{x}_{n,m} \end{pmatrix}$$

$$\dot{\boldsymbol{X}} = \begin{pmatrix} \frac{x_{1,1} - x_{min,1}}{x_{max,1} - x_{min,1}} & \frac{x_{1,2} - x_{min,2}}{x_{max,2} - x_{min,2}} & \cdots & \frac{x_{1,m} - x_{min,m}}{x_{max,m} - x_{min,m}} \\ \\ \frac{x_{2,1} - x_{min,1}}{x_{max,1} - x_{min,1}} & \frac{x_{2,2} - x_{min,2}}{x_{max,2} - x_{min,2}} & \cdots & \frac{x_{2,m} - x_{min,m}}{x_{max,m} - x_{min,m}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{x_{n,1} - x_{min,1}}{x_{max,1} - x_{min,1}} & \frac{x_{n,2} - x_{min,2}}{x_{max,2} - x_{min,2}} & \cdots & \frac{x_{n,m} - x_{min,m}}{x_{max,m} - x_{min,m}} \end{pmatrix} \quad | \quad 0 \leq \dot{x}_{i,k} \leq 1 \tag{3.21}$$

3.2.2 L2 normalization

Another way to apply feature scaling is via the method known as L2 normalization [2], where the following defines how to apply it when there is one independent variable in Eq. (3.4):

$$\dot{x}_{i,k} = \frac{x_{i,k}}{\sqrt{\sum_{i=1}^{n} \left| x_{i,k} \right|^2}} \tag{3.22}$$

Below is defined how to apply the L2 normalization when there are multiple independent variables in Eq. (3.4):

$$\dot{\boldsymbol{X}} = \begin{pmatrix} \dot{x}_{1,1} & \dot{x}_{1,2} & \cdots & \dot{x}_{1,m} \\ \dot{x}_{2,1} & \dot{x}_{2,2} & \cdots & \dot{x}_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ \dot{x}_{n,1} & \dot{x}_{n,2} & \cdots & \dot{x}_{n,m} \end{pmatrix} = \begin{pmatrix} \frac{x_{1,1}}{\sqrt{\sum_{i=1}^{n} \left| x_{i,1} \right|^2}} & \frac{x_{1,2}}{\sqrt{\sum_{i=1}^{n} \left| x_{i,2} \right|^2}} & \cdots & \frac{x_{1,m}}{\sqrt{\sum_{i=1}^{n} \left| x_{i,m} \right|^2}} \\ \\ \frac{x_{2,1}}{\sqrt{\sum_{i=1}^{n} \left| x_{i,1} \right|^2}} & \frac{x_{2,2}}{\sqrt{\sum_{i=1}^{n} \left| x_{i,2} \right|^2}} & \cdots & \frac{x_{2,m}}{\sqrt{\sum_{i=1}^{n} \left| x_{i,m} \right|^2}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{x_{n,1}}{\sqrt{\sum_{i=1}^{n} \left| x_{i,1} \right|^2}} & \frac{x_{n,2}}{\sqrt{\sum_{i=1}^{n} \left| x_{i,2} \right|^2}} & \cdots & \frac{x_{n,m}}{\sqrt{\sum_{i=1}^{n} \left| x_{i,m} \right|^2}} \end{pmatrix} \tag{3.23}$$

3.2.3 Z score normalization (standardization)

A further way to implement feature scaling is by using the standardization method, which is also known as z score normalization [1]. However, before applying it, the mean $\bar{x}$ from Eq. (3.8) and the standard deviation σ from Eq. (3.15) must be calculated. Next, it is defined in the following how to achieve this feature scaling method when there is one independent variable in

Eq. (3.4):

$$\dot{x}_{i,k} = \frac{x_{i,k} - \bar{x_k}}{\sigma_k} \tag{3.24}$$

In the following, it is stated how to implement the standardization method when there are multiple independent variables in Eq. (3.4):

$$\dot{\boldsymbol{X}} = \begin{pmatrix} \dot{x}_{1,1} & \dot{x}_{1,2} & \cdots & \dot{x}_{1,m} \\ \dot{x}_{2,1} & \dot{x}_{2,2} & \cdots & \dot{x}_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ \dot{x}_{n,1} & \dot{x}_{n,2} & \cdots & \dot{x}_{n,m} \end{pmatrix} = \begin{pmatrix} \frac{x_{1,1} - \bar{x}_1}{\sigma_1} & \frac{x_{1,2} - \bar{x}_2}{\sigma_2} & \cdots & \frac{x_{1,m} - \bar{x}_m}{\sigma_m} \\ \frac{x_{2,1} - \bar{x}_1}{\sigma_1} & \frac{x_{2,2} - \bar{x}_2}{\sigma_2} & \cdots & \frac{x_{2,m} - \bar{x}_m}{\sigma_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{x_{n,1} - \bar{x}_1}{\sigma_1} & \frac{x_{n,2} - \bar{x}_2}{\sigma_2} & \cdots & \frac{x_{n,m} - \bar{x}_m}{\sigma_m} \end{pmatrix} \tag{3.25}$$

3.3 Formulations of machine learning algorithms

Whenever an application is solved or approached using machine learning, there will always be several possible algorithms that can be applied. For this reason, it is important to choose a library that focuses on the type of algorithms to be used or that offers multiple options for several/all of them, such as those in Table 2.2. In any case, with this in consideration, it will be possible to train and generate several models with different approaches to make an adequate validation and model selection. Therefore, to meet these expectations, the following mathematical formulations will describe several machine learning algorithms that will be made available in the library developed in this thesis.

3.3.1 Simple linear regression

Consider a particular case of a matrix $\widetilde{\boldsymbol{X}}$ to represent a transformed version of Eq. (3.4) with n samples, each of them represented in rows. As well as that, this matrix has $m = 1$ independent variable $\widetilde{x}_{(i)}$, where i represents the sample currently observed. In addition, this matrix has $m + 1 = 2$ columns where the rows of the first one are all equal to the value of 1, such that $\widetilde{x}_{(i,0)} = 1$, and the other corresponds to the single independent variable, where $\widetilde{x}_{(i,1)} = x_{(i,1)}$ from Eq. (3.4). On the other hand, consider a particular case from Eq. (3.5) and Eq. (3.6) where $\boldsymbol{Y}$ and $\hat{\boldsymbol{Y}}$ have $p = 1$ dependent variable. Moreover, let us consider the real scalars $b_0, b_1 \in K$ such that $\boldsymbol{b} = (b_0, b_1)^T$ where T represents the transpose of a matrix or vector. Finally, all these conditions are governed by the characteristic mathematical form of the simple linear equation which is given by the following:

$$\hat{\boldsymbol{Y}} = \widetilde{\boldsymbol{X}}\boldsymbol{b} \tag{3.26}$$

$$\begin{pmatrix} \hat{y}_{1,1} \\ \hat{y}_{2,1} \\ \vdots \\ \hat{y}_{n,1} \end{pmatrix} = \begin{pmatrix} \widetilde{x}_{1,0} & \widetilde{x}_{1,1} \\ \widetilde{x}_{2,0} & \widetilde{x}_{2,1} \\ \vdots & \vdots \\ \widetilde{x}_{n,0} & \widetilde{x}_{n,1} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \end{pmatrix} \tag{3.27}$$

or alternatively, with the following representation by substituting the values of $\widetilde{x}_{(i,0)} = 1$ and $\widetilde{x}_{(i,1)} = x_{(i,1)}$ from Eq. (3.4) into Eq. (3.27):

$$\begin{pmatrix} \hat{y}_{1,1} \\ \hat{y}_{2,1} \\ \vdots \\ \hat{y}_{n,1} \end{pmatrix} = \begin{pmatrix} 1 & x_{1,1} \\ 1 & x_{2,1} \\ \vdots & \vdots \\ 1 & x_{n,1} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \end{pmatrix} \tag{3.28}$$

Furthermore, this can also be represented with only one sample when Eq. (3.28) is considered:

$$\hat{y}_{i,1} = b_0 + b_1 x_{i,1} \tag{3.29}$$

Undoubtedly, the description shown in Eq. (3.28) is the most formal one to solve this problem according to how these variables have been described so far. However, due to the simplicity of this regression method and the approach that was used to solve it in this work, the former will be described through Eq. (3.29), whose expected graphical output is always a straight line as shown in Figure 3.1.

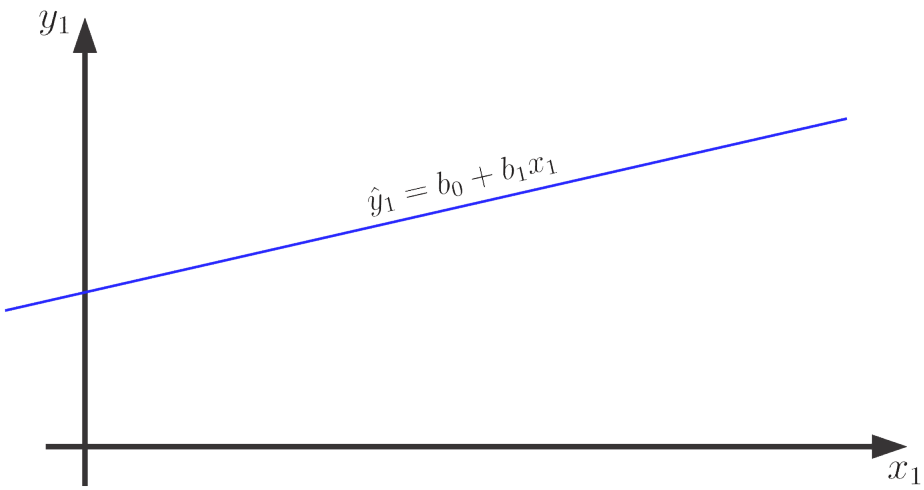


Figure 3.1: Characteristic graphical form of the simple linear equation.

In this regard, the role of the simple linear regression method arises when there is a sampled

data set from which it is desired to learn the unknown scalar coefficients of Eq. (3.29). In this sense, once the method is solved, although the equation or model resulting from the learning process will not necessarily give a perfect fitting result, it will give the best that it can provide. Consequently, when using such a model to predict the actual values of the samples taken, this will result in having errors $e_{i,1}$ as shown in Figure 3.2. Moreover, note that there are six hypothetical errors $e_{1,1}, e_{2,1}, \cdots, e_{6,1}$ for this particular illustration, one error for each output sample taken $y_{1,1}, y_{2,1}, \cdots, y_{6,1}$.

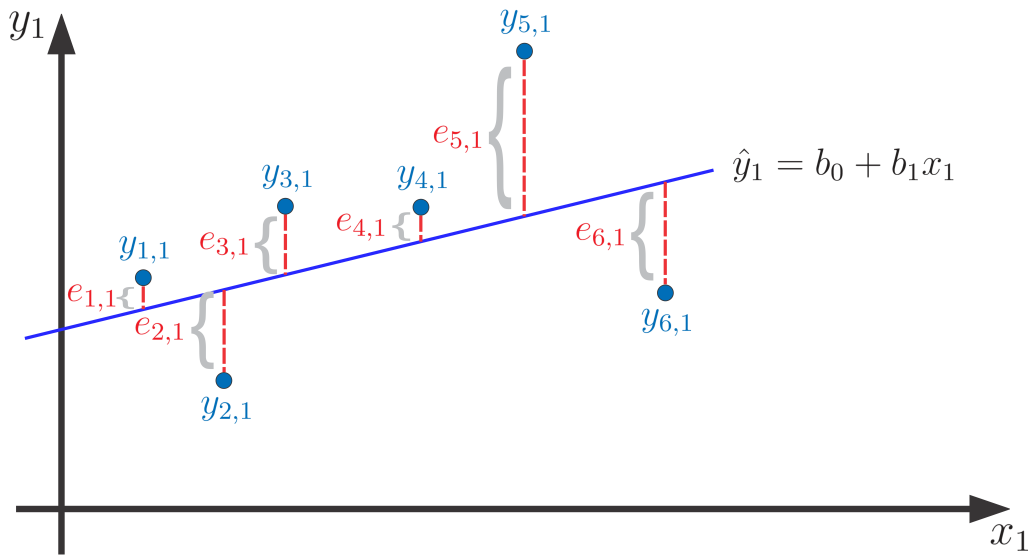


Figure 3.2: Illustration of a hypothetical data set from which the best fitting linear equation is identified.

These errors are the main key to identify the unknown scalar coefficients through the method introduced by Legendre, which is known as the least squares method [54] and is given by the following:

$$SSE = \sum_{i=1}^{n} e_i^2 \tag{3.30}$$

If we now substitute the error equivalence, as illustrated in Figure 3.2, into Eq. (3.30), we will then obtain the following:

$$SSE = \sum_{i=1}^{n} (y_{i,1} - \hat{y}_{i,1})^2 = \sum_{i=1}^{n} (y_{i,1} - b_0 - b_1 x_{i,1})^2 \tag{3.31}$$

For machine learning, the least squares are special because by deriving it with respect to the scalar coefficients obtained in Eq. (3.31) and with some additional mathematical steps, it is possible to obtain the coefficient values that will give the smallest error when substituting them in Eq. (3.29). Therefore, the following will describe such a derivation, first with respect to b_0:

$$\frac{\partial(SSE)}{\partial b_0} = \sum_{i=1}^{n}[2(y_{i,1} - b_0 - b_1 x_{i,1})(-1)]$$

$$\frac{\partial(SSE)}{\partial b_0} = 2\sum_{i=1}^{n}(-y_{i,1} + b_0 + b_1 x_{i,1}) \tag{3.32}$$

and then b_1:

$$\frac{\partial(SSE)}{\partial b_1} = \sum_{i=1}^{n}[2(y_{i,1} - b_0 - b_1 x_{i,1})(-x_{i,1})]$$

$$\frac{\partial(SSE)}{\partial b_1} = 2\sum_{i=1}^{n}(-x_{i,1}y_{i,1} + b_0 x_{i,1} + b_1 x_{i,1}^2) \tag{3.33}$$

Next, we equate the resulting derivatives to zero to obtain their critical values, which will provide the smallest error as explained before, and then rearrange them. First for Eq. (3.32):

$$\begin{aligned}
2\sum_{i=1}^{n}(-y_{i,1} + b_0 + b_1 x_{i,1}) &= 0 \\
\sum_{i=1}^{n}(-y_{i,1} + b_0 + b_1 x_{i,1}) &= 0 \\
-\sum_{i=1}^{n} y_{i,1} + b_0\sum_{i=1}^{n} + b_1\sum_{i=1}^{n} x_{i,1} &= 0 \\
\sum_{i=1}^{n} y_{i,1} = b_0\sum_{i=1}^{n} + b_1\sum_{i=1}^{n} x_{i,1} & \\
\sum_{i=1}^{n} y_{i,1} = nb_0 + b_1\sum_{i=1}^{n} x_{i,1} & \qquad (3.34)
\end{aligned}$$

and then for Eq. (3.33):

$$\begin{aligned}
2\sum_{i=1}^{n}(-x_{i,1}y_{i,1} + b_0 x_{i,1} + b_1 x_{i,1}^2) &= 0 \\
\sum_{i=1}^{n}(-x_{i,1}y_{i,1} + b_0 x_{i,1} + b_1 x_{i,1}^2) &= 0 \\
-\sum_{i=1}^{n} x_{i,1}y_{i,1} + b_0\sum_{i=1}^{n} x_{i,1} + b_1\sum_{i=1}^{n} x_{i,1}^2 &= 0 \\
\sum_{i=1}^{n} x_{i,1}y_{i,1} = b_0\sum_{i=1}^{n} x_{i,1} + b_1\sum_{i=1}^{n} x_{i,1}^2 & \qquad (3.35)
\end{aligned}$$

Now we express Eq. (3.34) and Eq. (3.35) in matrix form and then solve the linear system equation by the method of preference, for example the Gauss approach to strategically solve first for b_1:

$$\begin{bmatrix} n & \sum_{i=1}^{n} x_{i,1} \\ \sum_{i=1}^{n} x_{i,1} & \sum_{i=1}^{n} x_{i,1}^2 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y_{i,1} \\ \sum_{i=1}^{n} x_{i,1} y_{i,1} \end{bmatrix}$$

$Row1 = \dfrac{Row1}{n}$:

$$\begin{bmatrix} 1 & \sum_{i=1}^{n} \frac{x_{i,1}}{n} \\ \sum_{i=1}^{n} x_{i,1} & \sum_{i=1}^{n} x_{i,1}^2 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} \frac{y_{i,1}}{n} \\ \sum_{i=1}^{n} x_{i,1} y_{i,1} \end{bmatrix}$$

$Row2 = (Row2) - (Row1)\left(\sum_{i=1}^{n} x_{i,1}\right)$:

$$\begin{bmatrix} 1 & \sum_{i=1}^{n} \frac{x_{i,1}}{n} \\ 0 & \left(\sum_{i=1}^{n} x_{i,1}^2 - \sum_{i=1}^{n} \frac{x_{i,1}}{n} \sum_{i=1}^{n} x_{i,1}\right) \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} \frac{y_{i,1}}{n} \\ \sum_{i=1}^{n} x_{i,1} y_{i,1} - \sum_{i=1}^{n} \frac{y_{i,1}}{n} \sum_{i=1}^{n} x_{i,1} \end{bmatrix}$$

$Row2 = \dfrac{Row2}{\left(\sum_{i=1}^{n} x_{i,1}^2 - \sum_{i=1}^{n} \frac{x_{i,1}}{n} \sum_{i=1}^{n} x_{i,1}\right)}$:

$$\begin{bmatrix} 1 & \sum_{i=1}^{n} \frac{x_{i,1}}{n} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} \frac{y_{i,1}}{n} \\ \left(\dfrac{\sum_{i=1}^{n} x_{i,1} y_{i,1} - \sum_{i=1}^{n} \frac{y_{i,1}}{n} \sum_{i=1}^{n} x_{i,1}}{\sum_{i=1}^{n} x_{i,1}^2 - \sum_{i=1}^{n} \frac{x_{i,1}}{n} \sum_{i=1}^{n} x_{i,1}} \right) \end{bmatrix}$$

We then rearrange the values of the matrix:

$$\begin{bmatrix} 1 & \sum_{i=1}^{n} \frac{x_{i,1}}{n} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} \frac{y_{i,1}}{n} \\ \left(\dfrac{n \sum_{i=1}^{n} x_{i,1} y_{i,1} - \sum_{i=1}^{n} y_{i,1} \sum_{i=1}^{n} x_{i,1}}{n \sum_{i=1}^{n} x_{i,1}^2 - \sum_{i=1}^{n} x_{i,1} \sum_{i=1}^{n} x_{i,1}} \right) \end{bmatrix} \tag{3.36}$$

From the matrix representation of Eq. (3.36) we determine the following:

$$b_1 = \frac{n\sum_{i=1}^{n} x_{i,1}y_{i,1} - \sum_{i=1}^{n} y_{i,1}\sum_{i=1}^{n} x_{i,1}}{n\sum_{i=1}^{n} x_{i,1}^2 - \sum_{i=1}^{n} x_{i,1}\sum_{i=1}^{n} x_{i,1}} = \frac{n\sum_{i=1}^{n} x_{i,1}y_{i,1} - \sum_{i=1}^{n} y_{i,1}\sum_{i=1}^{n} x_{i,1}}{n\sum_{i=1}^{n} x_{i,1}^2 - \left(\sum_{i=1}^{n} x_{i,1}\right)^2} \tag{3.37}$$

Furthermore, if we substitute the solution provided for b_1 in Eq. (3.37), this means that we can identify the value of b_0 through Eq. (3.34). Consequently, we will have the following:

$$\sum_{i=1}^{n} \frac{y_{i,1}}{n} = b_0 + b_1 \sum_{i=1}^{n} \frac{x_{i,1}}{n}$$

Thus, from the principle described for the mean in Eq. (3.7), we will now have the following:

$$\bar{y}_1 = b_0 + b_1\bar{x}_1$$

Finally, we rearrange the terms:

$$b_0 = \bar{y}_1 - b_1\bar{x}_1 \tag{3.38}$$

As a result, in order to develop the well known simple linear regression, as described in several books [53], Eq. (3.37) and Eq. (3.38) give solution to the identification of the best fitting values of the unknown scalar coefficients b_0 and b_1 of Eq. (3.29). On the other hand, as a summary of all the processes that must be performed to apply the linear regression, the Pseudocode 1 lists these steps in an orderly fashion.

Algorithm 1 getSimpleLinearRegression

Input: $\boldsymbol{X}$, $\boldsymbol{Y}$
Output: $\boldsymbol{b}$

1: calculate b_1 from Eq. (3.37)
2: calculate b_0 from Eq. (3.38)
3: **return** $\boldsymbol{b}$ ▷ Return the coefficient values obtained

3.3.2 Multiple linear regression

Consider a particular case of a matrix $\widetilde{\boldsymbol{X}}$ to represent a transformed version of Eq. (3.4) with n samples, each of them represented in rows. As well as that, this matrix has m independent variables $\widetilde{x}_{(i,k)}$, where i represents the sample and k the independent variable currently observed. In addition, this matrix has $m+1$ columns where the rows of the first one are all equal to the value of 1, such that $\widetilde{x}_{(i,0)} = 1$, and the others correspond to the m independent variables, where $\widetilde{x}_{(i,k)} = x_{(i,k)}$ from Eq. (3.4). On the other hand, consider a particular case from Eq. (3.5) and

Eq. (3.6) where $\boldsymbol{Y}$ and $\hat{\boldsymbol{Y}}$ have $p = 1$ dependent variable. Moreover, let us consider the real scalars $b_0, b_1, \cdots, b_m \in K$ such that $\boldsymbol{b} = (b_0, b_1, \cdots, b_m)^T$ where T represents the transpose of a matrix or vector. Finally, all these conditions are governed by the characteristic mathematical form of the multiple linear equation which is given by the following:

$$\hat{\boldsymbol{Y}} = \widetilde{\boldsymbol{X}}\boldsymbol{b}$$

$$\begin{pmatrix} \hat{y}_{1,1} \\ \hat{y}_{2,1} \\ \vdots \\ \hat{y}_{n,1} \end{pmatrix} = \begin{pmatrix} \widetilde{x}_{1,0} & \widetilde{x}_{1,1} & \widetilde{x}_{1,2} & \cdots & \widetilde{x}_{1,m} \\ \widetilde{x}_{2,0} & \widetilde{x}_{2,1} & \widetilde{x}_{2,2} & \cdots & \widetilde{x}_{2,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \widetilde{x}_{n,0} & \widetilde{x}_{n,1} & \widetilde{x}_{n,2} & \cdots & \widetilde{x}_{n,m} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_m \end{pmatrix} \tag{3.39}$$

or alternatively, with the following representation by substituting the values of $\widetilde{x}_{(i,0)} = 1$ and $\widetilde{x}_{(i,k)} = x_{(i,k)}$ from Eq. (3.4) into Eq. (3.39):

$$\begin{pmatrix} \hat{y}_{1,1} \\ \hat{y}_{2,1} \\ \vdots \\ \hat{y}_{n,1} \end{pmatrix} = \begin{pmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_m \end{pmatrix} \tag{3.40}$$

Furthermore, this can also be represented with only one sample when Eq. (3.40) is considered:

$$\hat{y}_{i,1} = b_0 + b_1 x_{i,1} + b_2 x_{i,2} + \cdots + b_m x_{i,m} \tag{3.41}$$

Apart from this, the multiple linear regression uses the same mathematical principle as the simple linear regression. Yet, they differ in that the former uses more than one independent variable and the latter will have strictly only one. Nevertheless, in the mathematical description of this work, Eq. (3.40) will be used for the well known matrix method, as described in several books [53], instead of the analytical approach used in the simple linear regression in the subsection 3.3.1.

As with the simple linear regression in the subsection 3.3.1, the first step is to describe the error using the least squares method, which is described in Eq. (3.30) but through the matrix of Eq. (3.40) instead of Eq. (3.41). Therefore, considering that the mathematical details of matrix calculus will not be covered in this work, for more details about it for the following approach to be covered to solve the multiple linear regression, the book [55] is suggested:

$$SSE = \sum_{i=1}^{n} e_i^2$$

$$SSE = e^T e$$

$$SSE = (\boldsymbol{Y} - \hat{\boldsymbol{Y}})^T(\boldsymbol{Y} - \hat{\boldsymbol{Y}})$$

$$SSE = (\boldsymbol{Y} - \widetilde{\boldsymbol{X}}\boldsymbol{b})^T(\boldsymbol{Y} - \widetilde{\boldsymbol{X}}\boldsymbol{b})$$

$$SSE = (\boldsymbol{Y}^T - \widetilde{\boldsymbol{X}}^T\boldsymbol{b}^T)(\boldsymbol{Y} - \widetilde{\boldsymbol{X}}\boldsymbol{b})$$

$$SSE = \boldsymbol{Y}^T\boldsymbol{Y} - \boldsymbol{Y}^T\widetilde{\boldsymbol{X}}\boldsymbol{b} - \boldsymbol{b}\widetilde{\boldsymbol{X}}^T\boldsymbol{Y} + \boldsymbol{b}^T\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{X}}\boldsymbol{b} \tag{3.42}$$

With this in mind, we then derive Eq. (3.42) to later identify the values of the scalar coefficients that will provide the smallest error:

$$\frac{\partial(SSE)}{\partial \boldsymbol{b}} = \frac{\partial(\boldsymbol{Y}^T\boldsymbol{Y} - \boldsymbol{Y}^T\widetilde{\boldsymbol{X}}\boldsymbol{b} - \boldsymbol{b}\widetilde{\boldsymbol{X}}^T\boldsymbol{Y} + \boldsymbol{b}^T\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{X}}\boldsymbol{b})}{\partial \boldsymbol{b}}$$

$$\frac{\partial(SSE)}{\partial \boldsymbol{b}} = \frac{\partial(\boldsymbol{Y}^T\boldsymbol{Y})}{\partial \boldsymbol{b}} - \frac{\partial(\boldsymbol{Y}^T\widetilde{\boldsymbol{X}}\boldsymbol{b})}{\partial \boldsymbol{b}} - \frac{\partial(\boldsymbol{b}\widetilde{\boldsymbol{X}}^T\boldsymbol{Y})}{\partial \boldsymbol{b}} + \frac{\partial(\boldsymbol{b}^T\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{X}}\boldsymbol{b})}{\partial \boldsymbol{b}}$$

$$\frac{\partial(SSE)}{\partial \boldsymbol{b}} = 0 - \boldsymbol{Y}^T\widetilde{\boldsymbol{X}} - (\widetilde{\boldsymbol{X}}^T\boldsymbol{Y})^T + 2\boldsymbol{b}^T\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{X}}$$

$$\frac{\partial(SSE)}{\partial \boldsymbol{b}} = 0 - \boldsymbol{Y}^T\widetilde{\boldsymbol{X}} - \boldsymbol{Y}^T\widetilde{\boldsymbol{X}} + 2\boldsymbol{b}^T\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{X}}$$

$$\frac{\partial(SSE)}{\partial \boldsymbol{b}} = -2\boldsymbol{Y}^T\widetilde{\boldsymbol{X}} + 2\boldsymbol{b}^T\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{X}} \tag{3.43}$$

We now equal the resulting derivatives to zero to obtain the critical values and then rearrange them:

$$-2\boldsymbol{Y}^T\widetilde{\boldsymbol{X}} + 2\boldsymbol{b}^T\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{X}} = 0$$

$$2\boldsymbol{b}^T\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{X}} = 2\boldsymbol{Y}^T\widetilde{\boldsymbol{X}}$$

$$\boldsymbol{b}^T\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{X}} = \boldsymbol{Y}^T\widetilde{\boldsymbol{X}}$$

$$\boldsymbol{b}^T = \boldsymbol{Y}^T\widetilde{\boldsymbol{X}}(\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{X}})^{-1}$$

$$\boldsymbol{b} = (\widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{X}})^{-1}\widetilde{\boldsymbol{X}}^T\boldsymbol{Y} \tag{3.44}$$

As a result, Eq. (3.44) gives solution to the identification of the best fitting values of the unknown scalar coefficients $b_0, b_1, \cdots, b_m$ of Eq. (3.40) in order to develop what is known as a multiple linear regression.

To conclude, as a summary of all the processes that must be performed to apply the multiple linear regression, the Pseudocode 2 lists these steps in an orderly fashion.

Algorithm 2 getMultipleLinearRegression

Input: $\boldsymbol{X}$, $\boldsymbol{Y}$
Output: $\boldsymbol{b}$

1: Fill the matrix $\widetilde{\boldsymbol{X}}$ with the values of $\boldsymbol{X}$ as described in Eq. (3.40)
2: Obtain $\widetilde{\boldsymbol{X}}^T$. ▷ This and the next steps are made in an attempt to apply Eq. (3.44)
3: $\boldsymbol{M}_1 = \widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{X}}$.
4: We calculate the inverse matrix of $\boldsymbol{M}_1$ such that $\boldsymbol{M}_2 = (\boldsymbol{M}_1)^{-1}$.
5: $\boldsymbol{M}_3 = \boldsymbol{M}_2\widetilde{\boldsymbol{X}}^T$.
6: We now get the following multiplication $\boldsymbol{b} = \boldsymbol{M}_3\boldsymbol{Y}$
7: **return** $\boldsymbol{b}$ ▷ Return the coefficient values obtained

3.3.3 Polynomial regression

Consider a particular case of a matrix $\widetilde{\boldsymbol{X}}$ to represent a transformed version of Eq. (3.4) with n samples, each of them represented in rows. As well as that, this matrix has $m = 1$ independent variable $\widetilde{x}_{(i,1)}$, where i represents the sample currently observed. In addition, this matrix has $N+1$ columns where the rows of the first one are all equal to the value of 1, such that $\widetilde{x}_{(i,0)} = 1$, and the others correspond to the single independent variable repeated several times. However, in each of these subsequent columns, the independent variable rises exponentially from 1 to N as follows $\widetilde{x}_{(i,1)}, \widetilde{x}^2_{(i,1)}, \widetilde{x}^3_{(i,1)}, \cdots, \widetilde{x}^N_{(i,1)}$, where $\widetilde{x}_{(i,1)} = x_{(i,1)}$ from Eq. (3.4). On the other hand, consider a particular case from Eq. (3.5) and Eq. (3.6) where $\boldsymbol{Y}$ and $\hat{\boldsymbol{Y}}$ have $p = 1$ dependent variable. Moreover, let us consider the real scalars $b_0, b_1, \cdots, b_{(N)} \in K$ such that $\boldsymbol{b} = (b_0, b_1, \cdots, b_{(N)})^T$ where T represents the transpose of a matrix or vector. Finally, all these conditions are governed by the characteristic mathematical form of the polynomial equation which is given by the following:

$$\hat{\boldsymbol{Y}} = \widetilde{\boldsymbol{X}}\boldsymbol{b}$$

$$\begin{pmatrix} \hat{y}_{1,1} \\ \hat{y}_{2,1} \\ \vdots \\ \hat{y}_{n,1} \end{pmatrix} = \begin{pmatrix} \widetilde{x}_{1,0} & \widetilde{x}_{1,1} & \widetilde{x}^2_{1,1} & \widetilde{x}^3_{1,1} & \cdots & \widetilde{x}^N_{1,1} \\ \widetilde{x}_{2,0} & \widetilde{x}_{2,1} & \widetilde{x}^2_{2,1} & \widetilde{x}^3_{2,1} & \cdots & \widetilde{x}^N_{2,1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \widetilde{x}_{n,0} & \widetilde{x}_{n,1} & \widetilde{x}^2_{n,1} & \widetilde{x}^3_{n,1} & \cdots & \widetilde{x}^N_{n,1} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{(N)} \end{pmatrix} \tag{3.45}$$

or alternatively, with the following representation by substituting the values of $\widetilde{x}_{(i,0)} = 1$ and $\widetilde{x}_{(i,1)} = x_{(i,1)}$ from Eq. (3.4) into Eq. (3.45):

$$\begin{pmatrix} \hat{y}_{1,1} \\ \hat{y}_{2,1} \\ \vdots \\ \hat{y}_{n,1} \end{pmatrix} = \begin{pmatrix} 1 & x_{1,1} & x^2_{1,1} & x^3_{1,1} & \cdots & x^N_{1,1} \\ 1 & x_{2,1} & x^2_{2,1} & x^3_{2,1} & \cdots & x^N_{2,1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n,1} & x^2_{n,1} & x^3_{n,1} & \cdots & x^N_{n,1} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{(N)} \end{pmatrix} \tag{3.46}$$

Furthermore, this can also be represented with one sample when Eq. (3.46) is considered:

$$\hat{y}_{i,1} = b_0 + b_1 x_{i,k} + b_2 x_{i,k}^2 + b_3 x_{i,k}^3 + \cdots + b_{(N)} x_{i,k}^N \tag{3.47}$$

where its graphical output has the characteristic that it can give many different types of curves depending on the order of degree of the equation and its coefficient values, as illustrated in Figure 3.3.

That aside, the method used in this work for the well known polynomial regression, can be solved with the same mathematical approach as that used for the multiple linear regression in the subsection 3.3.2. As such, what makes the matrix solution provided for that regression method in Eq. (3.44) so remarkable, is that whenever the current method under study possesses the matrix form of Eq. (3.26), it can then be solved by Eq. (3.44) as well, irrespective of the number of independent variables or if the values of the independent variables are raised to an exponent or not.

To conclude, as a summary of all the processes that must be performed to apply the polynomial regression, the Pseudocode 3 lists these steps in an orderly fashion.

Algorithm 3 getPolynomialRegression

Input: $\boldsymbol{X}$, $\boldsymbol{Y}$
Output: $\boldsymbol{b}$

1: Fill the matrix $\widetilde{\boldsymbol{X}}$ with the values of $\boldsymbol{X}$ as described in Eq. (3.46)
2: Obtain $\widetilde{\boldsymbol{X}}^T$. ▷ This and the next steps are made in an attempt to apply Eq. (3.44)
3: $\boldsymbol{M}_1 = \widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{X}}$.
4: We calculate the inverse matrix of $\boldsymbol{M}_1$ such that $\boldsymbol{M}_2 = (\boldsymbol{M}_1)^{-1}$.
5: $\boldsymbol{M}_3 = \boldsymbol{M}_2 \widetilde{\boldsymbol{X}}^T$.
6: We now get the following multiplication $\boldsymbol{b} = \boldsymbol{M}_3 \boldsymbol{Y}$
7: **return** $\boldsymbol{b}$ ▷ Return the coefficient values obtained

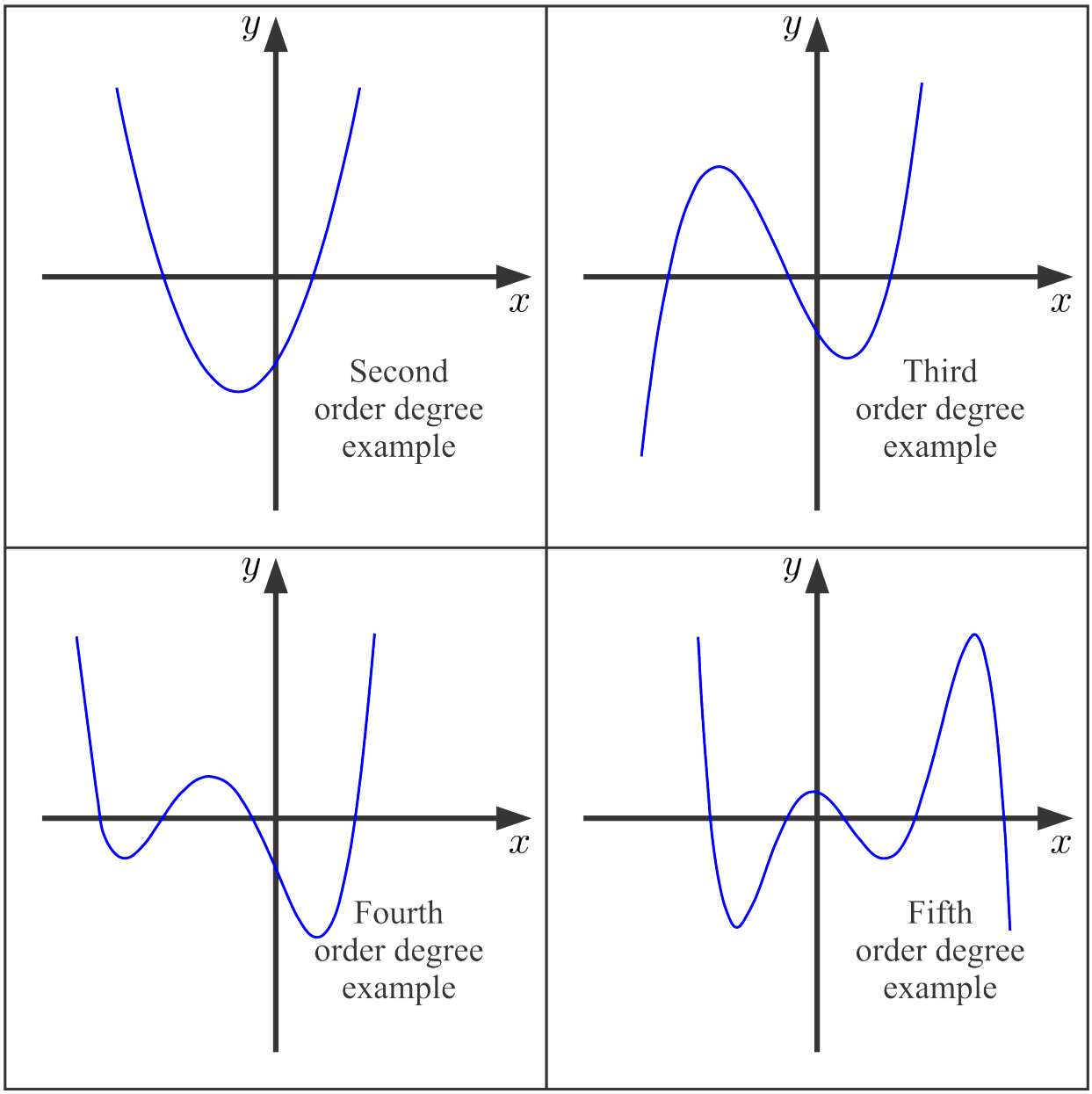


Figure 3.3: Examples of the characteristic graphical form of the polynomial equation.

3.3.4 Multiple polynomial regression (without interaction terms)

Let us consider a particular case of a matrix $\widetilde{\boldsymbol{X}}$ to represent a transformed version of Eq. (3.4) with n samples, each of them represented in rows. As well as that, this matrix $\widetilde{x}_{(i,k)}$ has m independent variables, where i represents the sample and k the independent variables currently observed. In addition, in each of these repeated columns, every independent variable increases exponentially from 1 to N in the following manner $\widetilde{x}_{(i,k)}, \widetilde{x}^2_{(i,k)}, \widetilde{x}^3_{(i,k)}, \cdots, \widetilde{x}^N_{(i,k)}$, where $\widetilde{x}_{(i,k)} = x_{(i,k)}$ from Eq. (3.4). In this sense, this matrix has $mN + 1$ columns where the rows

of the first one are all equal to the value of 1, such that $\widetilde{x}_{(i,0)} = 1$, and the others correspond to the independent variables repeated several times due to their exponential increase. On the other hand, consider a particular case from Eq. (3.5) and Eq. (3.6) where $\boldsymbol{Y}$ and $\hat{\boldsymbol{Y}}$ have $p = 1$ dependent variable. Moreover, let us consider the real scalars $b_0, b_1, \cdots, b_{(mN)} \in K$ such that $\boldsymbol{b} = (b_0, b_1, \cdots, b_{(mN)})^T$ where T represents the transpose of a matrix or vector. Finally, all these conditions are governed by the characteristic mathematical form of the multiple polynomial equation that does not consider interaction terms and is given by the following:

$$\hat{\boldsymbol{Y}} = \widetilde{\boldsymbol{X}}\boldsymbol{b}$$

$$\begin{pmatrix} \hat{y}_{1,1} \\ \hat{y}_{2,1} \\ \vdots \\ \hat{y}_{n,1} \end{pmatrix} = \begin{pmatrix} \widetilde{x}_{1,0} & \widetilde{x}^1_{1,1} & \widetilde{x}^2_{1,1} & \cdots & \widetilde{x}^N_{1,1} & \widetilde{x}^1_{1,2} & \widetilde{x}^2_{1,2} & \cdots & \widetilde{x}^N_{1,2} & \cdots & \widetilde{x}^1_{1,m} & \widetilde{x}^2_{1,m} & \cdots & \widetilde{x}^N_{1,m} \\ \widetilde{x}_{2,0} & \widetilde{x}^1_{2,1} & \widetilde{x}^2_{2,1} & \cdots & \widetilde{x}^N_{2,1} & \widetilde{x}^1_{2,2} & \widetilde{x}^2_{2,2} & \cdots & \widetilde{x}^N_{2,2} & \cdots & \widetilde{x}^1_{2,m} & \widetilde{x}^2_{2,m} & \cdots & \widetilde{x}^N_{2,m} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \widetilde{x}_{n,0} & \widetilde{x}^1_{n,1} & \widetilde{x}^2_{n,1} & \cdots & \widetilde{x}^N_{n,1} & \widetilde{x}^1_{n,2} & \widetilde{x}^2_{n,2} & \cdots & \widetilde{x}^N_{n,2} & \cdots & \widetilde{x}^1_{n,m} & \widetilde{x}^2_{n,m} & \cdots & \widetilde{x}^N_{n,m} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{(mN)} \end{pmatrix} \tag{3.48}$$

or alternatively, with the following representation by substituting the values of $\widetilde{x}_{(i,0)} = 1$ and $\widetilde{x}_{(i,k)} = x_{(i,k)}$ from Eq. (3.4) into Eq. (3.48):

$$\begin{pmatrix} \hat{y}_{1,1} \\ \hat{y}_{2,1} \\ \vdots \\ \hat{y}_{n,1} \end{pmatrix} = \begin{pmatrix} 1 & x^1_{1,1} & x^2_{1,1} & \cdots & x^N_{1,1} & x^1_{1,2} & x^2_{1,2} & \cdots & x^N_{1,2} & \cdots & x^1_{1,m} & x^2_{1,m} & \cdots & x^N_{1,m} \\ 1 & x^1_{2,1} & x^2_{2,1} & \cdots & x^N_{2,1} & x^1_{2,2} & x^2_{2,2} & \cdots & x^N_{2,2} & \cdots & x^1_{2,m} & x^2_{2,m} & \cdots & x^N_{2,m} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x^1_{n,1} & x^2_{n,1} & \cdots & x^N_{n,1} & x^1_{n,2} & x^2_{n,2} & \cdots & x^N_{n,2} & \cdots & x^1_{n,m} & x^2_{n,m} & \cdots & x^N_{n,m} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_{(mN)} \end{pmatrix} \tag{3.49}$$

Furthermore, this can also be represented with one sample when Eq. (3.49) is considered:

$$\begin{aligned} \hat{y}_{i,1} = b_0 + b_1 x^1_{i,1} + b_2 x^2_{i,1} + \cdots + b_{(N)} x^N_{i,1} + b_{(N+1)} x^1_{i,2} + b_{(N+2)} x^2_{i,2} + \\ \cdots + b_{(2N)} x^N_{i,2} + \cdots + b_{(2N+1)} x^1_{i,m} + b_{(2N+2)} x^2_{i,m} + \cdots + b_{(mN)} x^N_{i,m} \end{aligned} \tag{3.50}$$

Apart from this, the method used in this work for the multiple polynomial regression uses the same matrix mathematical approach as the one used for the multiple linear regression in the subsection 3.3.2. Therefore, Eq. (3.49) can be solved by Eq. (3.44) to address this problem just like for the multiple linear regression. However, although it seems natural to use Eq. (3.44) to solve this particular problem, during the literature review conducted for this thesis, no evidence has been found for an existing method such as the multiple polynomial regression without interaction terms as described in this work, which was inspired by the way in how the multiple linear regression is formulated in this thesis and in the way in which the polynomial regression was explained in the subsection 3.3.3.

To conclude, as a summary of all the processes that must be performed to apply the multiple polynomial regression, the Pseudocode 4 lists these steps in an orderly fashion.

Algorithm 4 getMultiplePolynomialRegression

Input: $\boldsymbol{X}$, $\boldsymbol{Y}$
Output: $\boldsymbol{b}$

1: Fill the matrix $\widetilde{\boldsymbol{X}}$ with the values of $\boldsymbol{X}$ as described in Eq. (3.49)
2: Obtain $\widetilde{\boldsymbol{X}}^T$. ▷ This and the next steps are made in an attempt to apply Eq. (3.44)
3: $\boldsymbol{M}_1 = \widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{X}}$.
4: We calculate the inverse matrix of $\boldsymbol{M}_1$ such that $\boldsymbol{M}_2 = (\boldsymbol{M}_1)^{-1}$.
5: $\boldsymbol{M}_3 = \boldsymbol{M}_2\widetilde{\boldsymbol{X}}^T$.
6: We now get the following multiplication $\boldsymbol{b} = \boldsymbol{M}_3\boldsymbol{Y}$
7: **return** $\boldsymbol{b}$ ▷ Return the coefficient values obtained

3.3.5 Logistic regression

The traditional logistic function, introduced by the mathematician Verhulst [56], is one of the many types of sigmoid functions that exist. In spite of how it was originally introduced, in this thesis it will be strategically considered in the following way with the intention of reusing previously formulated mathematical solutions to solve this problem:

$$\hat{y}_{i,1} = \frac{1}{1 + e^{-(b_0 + b_1 x_{i,1} + b_2 x_{i,2} + \cdots + b_m x_{i,m})}} \tag{3.51}$$

where $\hat{y}_{i,1}$ will represent the values contained in Eq. (3.6) when it has $p = 1$ dependent variable. Also, $x_{i,k}$ represents the values contained in Eq. (3.4) where the k-th element represents the current independent variable that can be any value from 1 up to a total of m. For both $\hat{y}_{i,1}$ and $x_{i,k}$, i represents the sample currently observed from a total of n samples. Finally, the unknown real scalar coefficients are $b_0, b_1, \cdots, b_m \in K$ such that $\boldsymbol{b} = (b_0, b_1, \cdots, b_m)^T$ where T represents the transpose of a matrix or vector.

The logistic equation has a particular output that can be used for several different application purposes and is shown in Figure 3.4 but without bias and with only one independent variable for simpler illustrative reasons. For example, although this equation was originally presented to model the growth of populations, it can also serve other purposes. Among them, it can be used as a cumulative function in statistics, from 0 to 1; to model the probability P of a certain event occurring in relation to some variables or; to introduce/create nonlinearity in an artificial neuron.

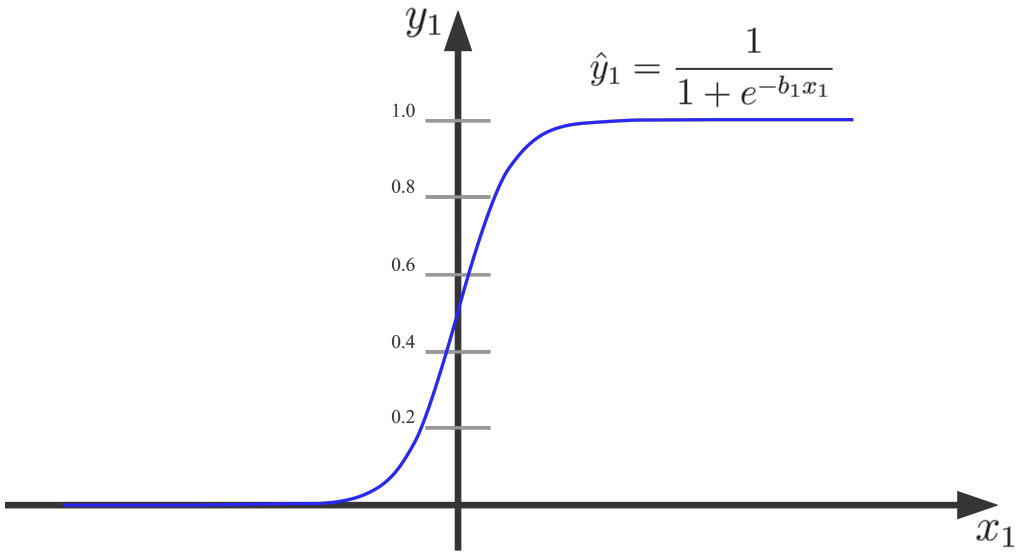


Figure 3.4: Characteristic graphical form of the logistic equation when there is only one independent and one dependent variable and does not have a bias value.

Moreover, Eq. (3.51) has a difficult arrangement of terms to be used in an attempt to obtain the best fitting unknown scalar coefficient values. Therefore, certain algebraic procedures will be made to facilitate its solution but with respect to $y_{i,1}$, since it must be used for the training process, instead of $\hat{y}_{i,1}$:

$$y_{i,1} = \frac{1}{1+e^{-(b_0+b_1x_{i,1}+b_2x_{i,2}+\cdots+b_mx_{i,m})}}$$

$$1+e^{-(b_0+b_1x_{i,1}+b_2x_{i,2}+\cdots+b_mx_{i,m})} = \frac{1}{y_{i,1}}$$

$$e^{-(b_0+b_1x_{i,1}+b_2x_{i,2}+\cdots+b_mx_{i,m})} = \frac{1}{y_{i,1}} - 1$$

$$\ln e^{-(b_0+b_1x_{i,1}+b_2x_{i,2}+\cdots+b_mx_{i,m})} = \ln \frac{1-y_{i,1}}{y_{i,1}}$$

$$-(b_0+b_1x_{i,1}+b_2x_{i,2}+\cdots+b_mx_{i,m}) = -\ln \frac{y_{i,1}}{1-y_{i,1}}$$

$$\ln \frac{y_{i,1}}{1-y_{i,1}} = b_0+b_1x_{i,1}+b_2x_{i,2}+\cdots+b_mx_{i,m}$$

$$\widetilde{y}_{i,1} = b_0+b_1x_{i,1}+b_2x_{i,2}+\cdots+b_mx_{i,m} \tag{3.52}$$

where $\widetilde{y}_{i,1} = \ln \dfrac{y_{i,1}}{1-y_{i,1}} \mid 0 < y_{i,1} < 1$.

With the variable change introduced in Eq. (3.52), it is possible to give solution to this problem by simply interpreting such equation as a multiple linear equation and applying the

multiple linear regression method as in subsection 3.3.2. Therefore, in order to solve it this way, we have to define Eq. (3.52) in its matrix form:

$$\begin{pmatrix} \widetilde{y}_{1,1} \\ \widetilde{y}_{2,1} \\ \vdots \\ \widetilde{y}_{n,1} \end{pmatrix} = \begin{pmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_m \end{pmatrix} \tag{3.53}$$

and without the change of variable $\widetilde{y}_{i,1}$ from Eq. (3.52), Eq. (3.53) is represented as:

$$\begin{pmatrix} \ln \dfrac{y_{1,1}}{1-y_{1,1}} \\ \ln \dfrac{y_{2,1}}{1-y_{2,1}} \\ \vdots \\ \ln \dfrac{y_{n,1}}{1-y_{n,1}} \end{pmatrix} = \begin{pmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_m \end{pmatrix} \quad | \ 0 < y_{i,1} < 1 \tag{3.54}$$

Finally, after applying the multiple linear regression on Eq. (3.53) and learning the best fitting values for the unknown scalar coefficients $b_0, b_1, \cdots, b_m$, these are substituted in Eq. (3.51) and with it, we would give solution to the main problem and apply what is known as a logistic regression.

To conclude, as a summary of all the processes that must be performed to apply the logistic regression, the Pseudocode 5 lists these steps in an orderly fashion.

Algorithm 5 getLogisticRegression

Input: $\boldsymbol{X}$, $\boldsymbol{Y}$
Output: $\boldsymbol{b}$

1: Fill the matrix $\widetilde{\boldsymbol{X}}$ with the values of $\boldsymbol{X}$ as described in Eq. (3.53)
2: Transform all the values of the matrix $\boldsymbol{Y}$ into $\widetilde{\boldsymbol{Y}}$ as dictated in Eq. (3.52), which states that $\widetilde{y}_{i,1} = \ln \dfrac{y_{i,1}}{1-y_{i,1}}$ for all samples.
3: Obtain $\widetilde{\boldsymbol{X}}^T$. ▷ This and the next steps are made in an attempt to apply Eq. (3.44)
4: $\boldsymbol{M}_1 = \widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{X}}$.
5: We calculate the inverse matrix of $\boldsymbol{M}_1$ such that $\boldsymbol{M}_2 = (\boldsymbol{M}_1)^{-1}$.
6: $\boldsymbol{M}_3 = \boldsymbol{M}_2 \widetilde{\boldsymbol{X}}^T$.
7: We now get the following multiplication $\boldsymbol{b} = \boldsymbol{M}_3 \widetilde{\boldsymbol{Y}}$
8: **return** $\boldsymbol{b}$ ▷ Return the coefficient values obtained

3.3.6 Gaussian regression

The Gaussian function is a very popular mathematical equation that can be easily found in many books on probability and statistics (eg. book [53]). It is used to study and predict the probability of occurrence of a certain event and its widespread use is due to the fact that its behavior appears in several phenomena that has been studied in those fields [53]. Moreover, its equation is described below when having a unitary amplitude and for illustrative purposes, its typical output is shown in Figure 3.5 when only one independent variable of this equation is considered:

$$\hat{y}_{i,1} = e^{-\left(\frac{(x_{i,1}-\bar{x}_1)^2}{2\sigma_1^2} + \frac{(x_{i,2}-\bar{x}_2)^2}{2\sigma_2^2} + \cdots + \frac{(x_{i,m}-\bar{x}_m)^2}{2\sigma_m^2}\right)} \quad | \quad 0 < \hat{y}_{i,1} \leq 1 \tag{3.55}$$

where $\hat{y}_{i,1}$ would represent the vector form of Eq. (3.6) when it has $p = 1$ dependent variable. In addition, $x_{i,k}$ represents Eq. (3.4) and for both, i denotes the currently observed sample out of a total of n samples. For $x_{i,k}$, the k-th element represents the current independent variable, that can be any value from 1 up to a total of m. Finally, for of all the n existing samples, there are two unknown real scalar coefficients for each independent variable available, where $\bar{x}_k$ represents the mean and σ_k^2 stands for the variance.

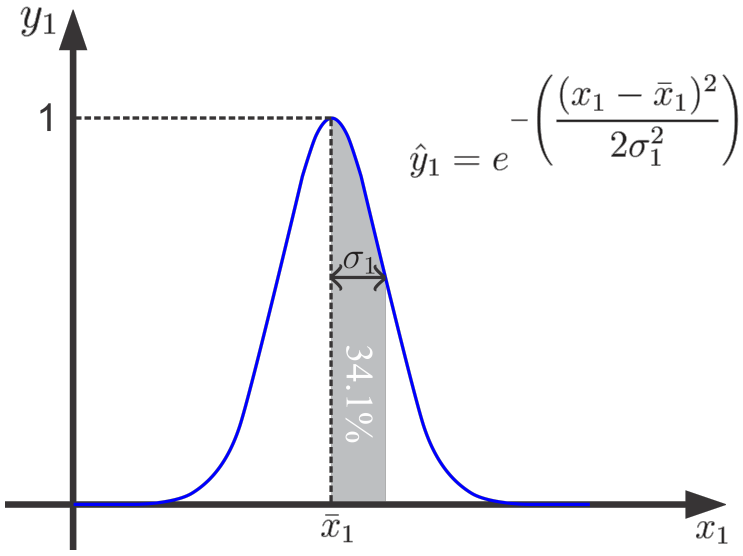


Figure 3.5: Characteristic graphical form of the Gaussian equation when there is only one independent variable.

The form of Eq. (3.55) proves to be very difficult to manage so that the unknown elements of it can be solved. Therefore, certain algebraic procedures will be made to facilitate its solution but with respect to $y_{i,1}$, since it must be used for the training process, instead of $\hat{y}_{i,1}$:

$$y_{i,1} = e^{-\left(\frac{(x_{i,1}-\bar{x}_1)^2}{2\sigma_1^2} + \frac{(x_{i,2}-\bar{x}_2)^2}{2\sigma_2^2} + \cdots + \frac{(x_{i,m}-\bar{x}_m)^2}{2\sigma_m^2}\right)}$$

$$\ln y_{i,1} = \ln e^{-\left(\frac{(x_{i,1}-\bar{x}_1)^2}{2\sigma_1^2} + \frac{(x_{i,2}-\bar{x}_2)^2}{2\sigma_2^2} + \cdots + \frac{(x_{i,m}-\bar{x}_m)^2}{2\sigma_m^2}\right)}$$

$$\ln y_{i,1} = -\left(\frac{(x_{i,1}-\bar{x}_1)^2}{2\sigma_1^2} + \frac{(x_{i,2}-\bar{x}_2)^2}{2\sigma_2^2} + \cdots + \frac{(x_{i,m}-\bar{x}_m)^2}{2\sigma_m^2}\right)$$

$$\ln y_{i,1} = -\left(\frac{x_{i,1}^2 - 2x_{i,1}\bar{x}_1 + \bar{x}_1^2}{2\sigma_1^2} + \frac{x_{i,2}^2 - 2x_{i,2}\bar{x}_2 + \bar{x}_2^2}{2\sigma_2^2} + \cdots + \frac{x_{i,m}^2 - 2x_{i,m}\bar{x}_m + \bar{x}_m^2}{2\sigma_m^2}\right)$$

$$\ln y_{i,1} = -\left(\frac{x_{i,1}^2}{2\sigma_1^2} - \frac{2\bar{x}_1 x_{i,1}}{2\sigma_1^2} + \frac{\bar{x}_1^2}{2\sigma_1^2} + \frac{x_{i,2}^2}{2\sigma_2^2} - \frac{2\bar{x}_2 x_{i,2}}{2\sigma_2^2} + \frac{\bar{x}_2^2}{2\sigma_2^2} + \cdots + \frac{x_{i,m}^2}{2\sigma_m^2} - \frac{2\bar{x}_m x_{i,m}}{2\sigma_m^2} + \frac{\bar{x}_m^2}{2\sigma_m^2}\right)$$

$$\ln y_{i,1} = -\frac{x_{i,1}^2}{2\sigma_1^2} + \frac{\bar{x}_1 x_{i,1}}{\sigma_1^2} - \frac{\bar{x}_1^2}{2\sigma_1^2} - \frac{x_{i,2}^2}{2\sigma_2^2} + \frac{\bar{x}_2 x_{i,2}}{\sigma_2^2} - \frac{\bar{x}_2^2}{2\sigma_2^2} - \cdots - \frac{x_{i,m}^2}{2\sigma_m^2} + \frac{\bar{x}_m x_{i,m}}{\sigma_m^2} - \frac{\bar{x}_m^2}{2\sigma_m^2}$$

$$\ln y_{i,1} = \left(-\frac{\bar{x}_1^2}{2\sigma_1^2} - \frac{\bar{x}_2^2}{2\sigma_2^2} - \cdots - \frac{\bar{x}_m^2}{2\sigma_m^2}\right) + \left(\frac{\bar{x}_1}{\sigma_1^2}\right) x_{i,1} + \left(-\frac{1}{2\sigma_1^2}\right) x_{i,1}^2 + \left(\frac{\bar{x}_2}{\sigma_2^2}\right) x_{i,2} + \left(-\frac{1}{2\sigma_2^2}\right) x_{i,2}^2 + \cdots + \left(\frac{\bar{x}_m}{\sigma_m^2}\right) x_{i,m} + \left(-\frac{1}{2\sigma_m^2}\right) x_{i,m}^2$$

$$\widetilde{y}_{i,1} = \widetilde{b}_{0,1} + \widetilde{b}_{1,1} x_{i,1} + \widetilde{b}_{2,1} x_{i,1}^2 + \widetilde{b}_{3,1} x_{i,2} + \widetilde{b}_{4,1} x_{i,2}^2 + \cdots + \widetilde{b}_{(2m-1,1)} x_{i,m} + \widetilde{b}_{(2m,1)} x_{i,m}^2 \qquad (3.56)$$

where:

- $\widetilde{y}_{i,1} = \ln y_{i,1} \quad | \quad y_{i,1} > 0$
- $\widetilde{b}_{0,1} = -\frac{\bar{x}_1^2}{2\sigma_1^2} - \frac{\bar{x}_2^2}{2\sigma_2^2} - \cdots - \frac{\bar{x}_m^2}{2\sigma_m^2}$
- $\widetilde{b}_{1,1} = \frac{\bar{x}_1}{\sigma_1^2}$
- $\widetilde{b}_{2,1} = -\frac{1}{2\sigma_1^2}$
- $\widetilde{b}_{3,1} = \frac{\bar{x}_2}{\sigma_2^2}$
- $\widetilde{b}_{4,1} = -\frac{1}{2\sigma_2^2}$
- $\cdots$

- $\widetilde{b}_{(2m-1,1)} = \dfrac{\bar{x}_m}{\sigma_m^2}$
- $\widetilde{b}_{(2m,1)} = -\dfrac{1}{2\sigma_m^2}$

Thanks to the mathematical form obtained in Eq. (3.56), it is possible to apply the multiple polynomial regression to it, as described in the subsection 3.3.4. This will allow to learn the best fitting values for its unknown scalar coefficients $\widetilde{b}_{0,1}, \widetilde{b}_{1,1}, \widetilde{b}_{2,1}, \widetilde{b}_{3,1}, \widetilde{b}_{4,1}, \cdots, \widetilde{b}_{(2m-1,1)}, \widetilde{b}_{(2m,1)}$. Yet, even though this does not conclude the solution of the main problem, it happens that the unknown scalar coefficients $\bar{x}_k, \sigma_k^2$ of the Gaussian function can be obtained through the values of the now identified coefficients $\widetilde{b}_{0,1}, \widetilde{b}_{1,1}, \widetilde{b}_{2,1}, \widetilde{b}_{3,1}, \widetilde{b}_{4,1}, \cdots, , \widetilde{b}_{(2m-1,1)}, \widetilde{b}_{(2m,1)}$. Subsequently, Eq. (3.55) is completely solved after substituting them into this equation.

Finally, Eq. (3.56) poses a problem when used in/for classification (as an example, see more details in subsection 3.3.9), due to the restriction of Eq. (3.56), where $y_{i,1} > 0$. This precludes that the training values from $y_{i,1}$ cannot have exactly the value of 0 and, therefore, the value of 0.1 is suggested instead. However, it is also possible to try to tune this approximation to 0 with a different value, as long as it is smaller than 1, to try to get a better result for your particular data set. Nonetheless, this approximation suggestion is not recommended for regression applications, meaning that the true values should be used instead of this suggestion for regression applications.

To conclude, as a summary of all the processes that must be performed to apply the Gaussian regression, the Pseudocode 6 lists these steps in an orderly fashion.

Algorithm 6 getGaussianRegression

Input: $\boldsymbol{X}$, $\boldsymbol{Y}$
Output: $\boldsymbol{b}$

1: Fill the matrix $\widetilde{\boldsymbol{X}}$ with the values of $\boldsymbol{X}$ as described in Eq. (3.49)
2: Transform all the values of the matrix $\boldsymbol{Y}$ into $\widetilde{\boldsymbol{Y}}$ as dictated in Eq. (3.56), which states that $\widetilde{y}_{i,1} = \ln y_{i,1}$ for all samples.
3: Obtain $\widetilde{\boldsymbol{X}}^T$. ▷ This and the next steps are made in an attempt to apply Eq. (3.44)
4: $\boldsymbol{M}_1 = \widetilde{\boldsymbol{X}}^T\widetilde{\boldsymbol{X}}$.
5: We calculate the inverse matrix of $\boldsymbol{M}_1$ such that $\boldsymbol{M}_2 = (\boldsymbol{M}_1)^{-1}$.
6: $\boldsymbol{M}_3 = \boldsymbol{M}_2\widetilde{\boldsymbol{X}}^T$.
7: We now get the following multiplication $\boldsymbol{b} = \boldsymbol{M}_3\widetilde{\boldsymbol{Y}}$
8: **return** $\boldsymbol{b}$ ▷ Return the coefficient values obtained

3.3.7 Linear logistic classification

The linear logistic classification method has the same characteristic equation as in the logistic regression, which is Eq. (3.51). It also seeks to learn the best fitting values for the unknown

coefficients in that equation. However, the intent of solving that equation here is to determine the probability that a data point belongs to a certain group with respect to two different possible ones, to then make a prediction of it, as described in the following:

$$\hat{y}_{i,1} = \begin{cases} \text{group 1} & \text{if } \dfrac{1}{1+e^{-(b_0+b_1x_{i,1}+b_2x_{i,2}+\cdots+b_mx_{i,m})}} > \text{threshold} \\ \text{group 2} & \text{if } \dfrac{1}{1+e^{-(b_0+b_1x_{i,1}+b_2x_{i,2}+\cdots+b_mx_{i,m})}} \leq \text{threshold} \end{cases} \quad | \quad 0 < \text{threshold} < 1 \quad (3.57)$$

where it will be essential to take into account that the real values $y_{i,1}$ should be arranged in a specific way before the training process begins. Those belonging to group 1 must have a value as close as possible to 1 and if they belong to group 2, they must have a value as near as possible to 0. Regardless of the case, all these must never be exactly the values of 0 or 1 due to the restriction defined in Eq. (3.54).

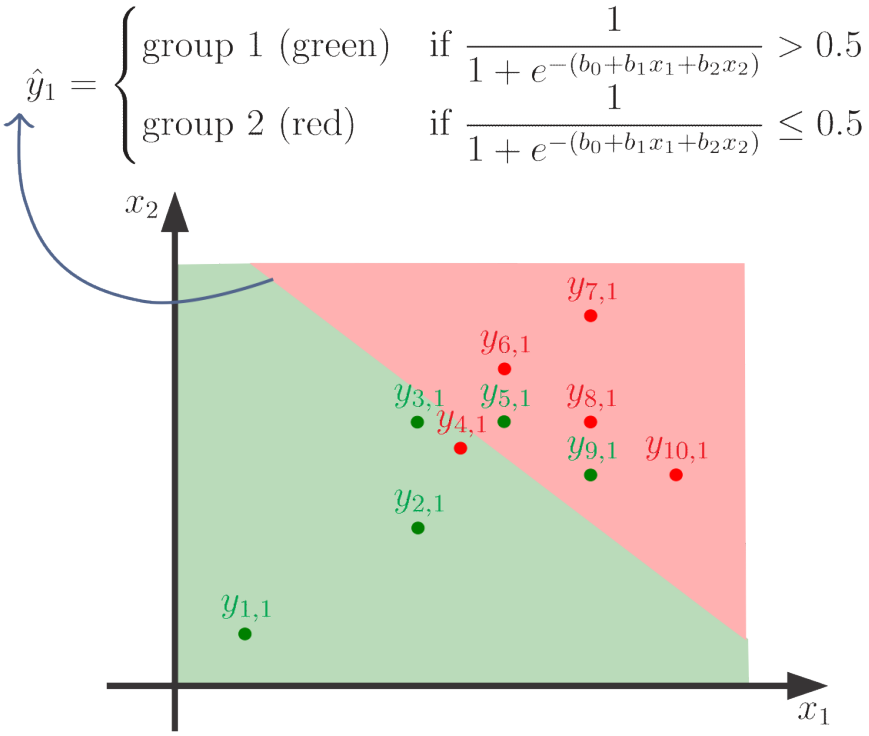


Figure 3.6: Illustrative example of applying the linear logistic classification method with a hypothetical data set.

As an illustrative example, Figure 3.6 describes a hypothetical data set $y_{1,1}, y_{2,1}, \cdots, y_{10,1}$ and has two independent variables x_1, x_2. Some of these data points belong to a hypothetical group 1 (colored in green) and the others to a hypothetical group 2 (colored in red). In addition, the predictions resulting from a linear logistic classification model that was obtained by Eq. (3.57) is shown through the colored background of this Figure. There, this model was defined with a

threshold of 0.5, representing a 50% probability that the given data point belongs to group 1. Conclusively, this figure depicts a best hypothetical attempt to model that data set, but as in most real life cases, errors in the predictions are expected (eg. data points $y_{4,1}, y_{5,1}, y_{9,1}$ were not accurately predicted).

To conclude, as a summary of all the processes that must be performed to apply the linear logistic classification, the Pseudocode 7 lists these steps in an orderly fashion.

Algorithm 7 getLinearLogisticClassification

Input: $\boldsymbol{X}$, $\boldsymbol{Y}$, theshold
Output: $\boldsymbol{b}$

1: Implement the Pseudocode 5 to apply the logistic regression with respect to the input data $\boldsymbol{X}$ and $\boldsymbol{Y}$ in order to obtain the values of the coefficients $\boldsymbol{b}$ of Eq. (3.53).
2: For the desired predicted data $\hat{\boldsymbol{Y}}$, use the value of "threshold" and apply Eq. (3.57) to determine to what group does such prediction belongs to.

3.3.8 Simple linear machine classification

The method that will be introduced as the simple linear machine classification is inspired in the simple linear regression, the linear Support Vector Machine (SVM) and the perceptrons. In this thesis, no publication was found that presented the same method as it will be described for the simple linear machine classification, despite it was exhaustively researched for it. Therefore, before explaining this method, it is desired in this work to clearly identify the differentiating factors among the inspired classifier methods and the simple linear machine classification. As a result, it will be possible to identify the different features that all of these methods offer. This will also permit the practitioner to know when it is best to use the simple linear machine classification with respect to the other two classifiers.

First, the following will give an understanding of the linear SVM classification method since it will not be provided in the framework of this thesis. Therefore, let us first describe a hypothetical data set $y_{1,1}, y_{2,1}, \cdots, y_{10,1}$ as illustrated in Figure 3.7. From among these, we will consider that all data points colored in green will be positive samples, representing group 1. On the other hand, all data points colored in red will be negative samples, representing group 2. Furthermore, the goal of this method is to provide a means to be able to predict to which group any possible data point $\hat{\boldsymbol{y}}_i$ belongs. To do this, we will define a vector $\boldsymbol{x}_i$ that will contain the m independent variables, interpreting them as the coordinates of the current data point. Also, we will define a line called hyperplane, which is a perpendicular plane (seen as a straight line in Figure 3.7) whose purpose is to try to find the best possible centered position with respect to the lines known as support vectors. These support vectors are the two perpendicular planes that are parallel to the hyperplane and that lie at the intersection of the nearest data point on

both sides of the hyperplane. Finally, we will also define a vector $\boldsymbol{\omega}$, also called weight vector, which will be perpendicular to this hyperplane.

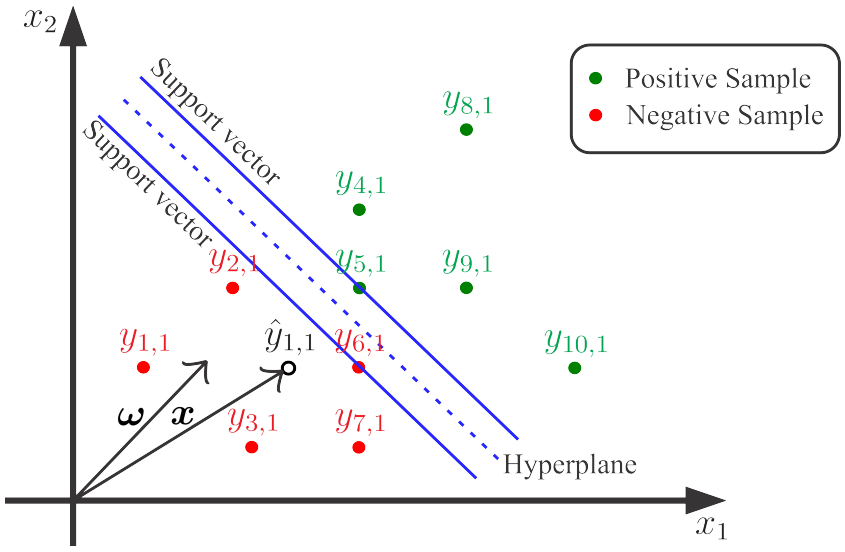


Figure 3.7: Illustrative example of applying the linear support vector machine classification method with a hypothetical data set.

To conclude with the explanation of the linear SVM, it is of interest to determine how to predict to which group the current data point belongs. Therefore, the model generated by this method [57] is presented below, but adapted to how this thesis has interpreted the variable definitions up until now:

$$\boldsymbol{x}_i \in \begin{cases} \text{group 1} & \text{if } \boldsymbol{\omega}.\boldsymbol{x}_i^T + b_0 > 0 \\ \text{group 2} & \text{if } \boldsymbol{\omega}.\boldsymbol{x}_i^T + b_0 \leq 0 \end{cases} \tag{3.58}$$

where T is the transpose of a vector or matrix and where both $\boldsymbol{\omega}$ and b_0 are the unknown parameters of this equation that must be further obtained in order to get the best fitting model for the data set under study. Moreover, Eq. (3.58) and the explanation already given for Figure 3.7 are complemented by the following:

$$\boldsymbol{y}_i = \boldsymbol{\omega}.\boldsymbol{x}_i^T + b_0 \tag{3.59}$$

where:

$$\boldsymbol{y}_i = \begin{cases} 1 & \text{if } \boldsymbol{x}_i \in \text{group 1} \\ -1 & \text{if } \boldsymbol{x}_i \in \text{group 2} \end{cases}$$

On the other hand, leaving aside the linear SVM classifier, a perceptron is an artificial neuron that is used in particular for classification problems [58]. The characteristic equation of

its generated model turns out to be Eq. (3.58), the same as in the linear SVM classifier. Since the linear SVM model is inspired by the perceptron, as stated in its original publication article [57], this is not a surprise. However, the perceptron uses a different philosophy and methodology to obtain the best fitting model, despite being the same equation. In this case, the model is trained by trying to obtain a hyperplane that best splits the data, but with respect to the data dispersion rather than the support vectors [58]. To do this, the nature of how a neuron learns is mimicked, where several iterative learning processes are performed to learn more at each one of them (epoch) on how to better fit the model [58]. Yet, the neuron must be manually configured by the machine learning practitioner with some tuning parameters known as hyperparameters [58]. As a consequence, the entire learning process of this method may have to be repeated several times by the practitioner to achieve a good result (see subsection 3.3.10 for more details about artificial neurons).

With this in mind, lets us now begin to develop the expression of Eq. (3.59) to obtain the solution of the simple linear classifier method to be introduced:

$$\boldsymbol{y}_i = \boldsymbol{\omega}.\boldsymbol{x}_i^T + b_0$$

$$\boldsymbol{y}_i = \begin{pmatrix} \omega_1 & \omega_2 & \cdots & \omega_m \end{pmatrix} \begin{pmatrix} x_{i,1} \\ x_{i,2} \\ \vdots \\ x_{i,m} \end{pmatrix} + b_0$$

$$\boldsymbol{y}_i = b_0 + \omega_1 x_{i,1} + \omega_2 x_{i,2} + \cdots + \omega_m x_{i,m} \tag{3.60}$$

Eq. (3.60) considers only one sample, while for a certain case of multiple samples it would be interpreted as follows:

$$y_{i,1} = b_0 + \omega_1 x_{i,1} + \omega_2 x_{i,2} + \cdots + \omega_m x_{i,m} \tag{3.61}$$

The Eq. (3.61) represents a critical element of the approach to be followed in this thesis, particularly because it has the same mathematical form as Eq. (3.41). This means that the weight vector $\boldsymbol{\omega}$ and b_0 can be obtained by applying the multiple linear regression method on Eq. (3.61). Then, these values must be substituted into Eq. (3.58) and then this equation will be able to predict the group to which the current data point belongs.

As a result, all the processes that must be performed to apply the simple linear machine classification are listed in an orderly fashion in the Pseudocode 8.

Algorithm 8 getSimpleLinearMachineClassification

Input: $\boldsymbol{X}$, $\boldsymbol{Y}$

Output: $\boldsymbol{b}$

1: Implement the Pseudocode 2 to apply the multiple linear regression with respect to the input data $\boldsymbol{X}$ and $\boldsymbol{Y}$ in order to obtain the values of the coefficients $b_0, \omega_1, \omega_2, ..., \omega_m$ of Eq. (3.61).

2: Substitute the previously obtained coefficients $b_0, \omega_1, \omega_2, ..., \omega_m$ into Eq. (3.59) in order to solve the model of the simple linear machine classification.

3: For the desired predicted data $\hat{\boldsymbol{Y}}$, apply Eq. (3.58) to determine to what group does such prediction belongs to.

In conclusion, the linear SVM approach does not solve this by using some regression method, but rather gives more complex mathematical descriptions and even uses a Lagrangian transformation [57]. As a result, it is sometimes able to give a more centered hyperplane with respect to the vector supports than in the simple linear classifier method. However, as the simple linear classifier approach takes into account the data dispersion, it can sometimes obtain more unbiased results than the linear SVM method. Furthermore, the linear SVM approach poses an inconvenience when processing data during the training of its model. Because it processes the magnitude of vectors in its learning process, it has sensitivity problems when a machine learning feature (independent variable) has considerably larger values than another feature. Yet, although there are preprocessing methods known as feature scaling that help reduce this problem, the simple linear machine classification does not require them because the magnitude of a vector is never used in this approach. Consequently, this will allow the possibility of obtaining better training results under restricted circumstances where feature scaling is not desired, a good or possible idea, especially when using low profile embedded systems.

In contrast, the perceptron method solves this through an iterative learning process and requires the use of hyperparameters, that can unstable the neuron if the practitioner adjusts the wrong values. Conversely, the right hyperparameters can also cause the learning process to be considerably faster under certain and very specific circumstances. However, the perceptron usually takes more time to obtain results than the simple linear classifier due to this iterative learning process. Nonetheless, the simple linear classifier will allow the possibility of obtaining more reliable results and in faster times when the correct hyperparameters for the perceptron are not know and, in particular, when using big data.

3.3.9 Kernel machine classification

Before starting with the explanation of this method, let us define a hypothetical data set $y_{1,1}, y_{2,1}, \cdots, y_{8,1}$ as illustrated in Figure 3.8. Among them, we will consider all data points

colored in green to be positive samples, representing group 1 with a value of $y_{i,1} = 1$. On the other hand, all data points colored in red will be negative samples, representing group 2 with a value of $y_{i,1} = -1$. Furthermore, this data set poses a situation that cannot be adequately modeled with the previous classification methods because they are all linear classifiers. This means that they either classify by separating the data with a line or a perpendicular plane and there is simply no possible line/plane that can properly classify this. However, there is a possibility to solve non linear problems if the simple linear machine classification method from the subsection 3.3.8 is used, but with a mathematical procedure known as the Kernel trick [57].

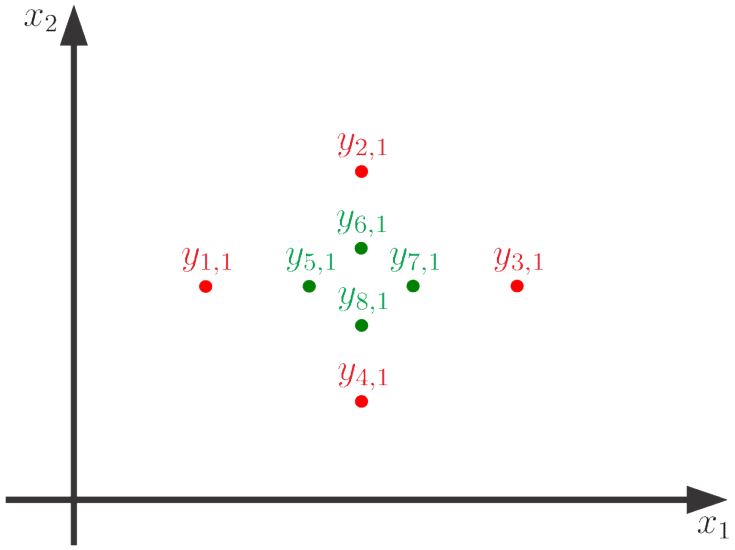


Figure 3.8: Example of a hypothetical data set for a nonlinear classification application.

For this purpose, let us redefine Eq. (3.58)

$$\boldsymbol{x}_i \in \begin{cases} \text{group 1} & \text{if } \boldsymbol{\omega}.\boldsymbol{x}_i^T + b_0 > 0 \\ \text{group 2} & \text{if } \boldsymbol{\omega}.\boldsymbol{x}_i^T + b_0 \leq 0 \end{cases}$$

into the following:

$$\boldsymbol{x}_i \in \begin{cases} \text{group 1} & \text{if } \beta\kappa(\boldsymbol{x}_i) + b_0 > 0 \\ \text{group 2} & \text{if } \beta\kappa(\boldsymbol{x}_i) + b_0 \leq 0 \end{cases} \tag{3.62}$$

where β and b_0 are the unknown real scalar coefficients of the model to be solved and $\kappa(\boldsymbol{x}_i)$ is a function called Kernel.

In addition, let us also redefine Eq. (3.59)

$$\boldsymbol{y}_i = \boldsymbol{\omega}.\boldsymbol{x}_i^T + b_0$$

accordingly into the following:

$$\boldsymbol{y}_i = \beta\kappa(\boldsymbol{x}_i) + b_0 \tag{3.63}$$

where:

$$\boldsymbol{y}_i = \begin{cases} 1 & \text{if } \boldsymbol{x}_i \in \text{group 1} \\ -1 & \text{if } \boldsymbol{x}_i \in \text{group 2} \end{cases}$$

The purpose of using the Kernel function $\kappa(\boldsymbol{x}_i)$ in these mathematical expressions is to apply a domain transformation with respect to $\boldsymbol{x}_i$. This can be interpreted as a means of attempting to "deform" the space in order to spread the data set across a new dimension. This will aim to make it possible to properly split the data with the linear plane generated with Eq. (3.62). Also, what makes the Kernel trick so useful in this approach is that any function that comes to mind can be used, as long as it is possible for the practitioner to generate its corresponding regression method.

To better explain how the Kernel trick works, the same hypothetical example from Figure 3.8 will be used and a Gaussian function for the Kernel will be proposed. Therefore, the first step would be to apply the Gaussian regression, from the subsection 3.3.6. There, we will feed into the training process of the model all the training output values $y_{i,1}$ and all its input values $x_{i,1}, x_{i,2}, \cdots, x_{i,m}$ up to the m independent variables that the data set has (two in this example). The Gaussian model generated will be the equivalence of the Kernel function $\kappa(\boldsymbol{x}_i)$ and will be used to create a new dimension κ, as illustrated in Figure 3.9. The next step is to apply a simple linear regression to Eq. (3.63) since the output values of the Kernel function are one dimensional. This will permit to know the best fitting values of the unknown scalar coefficients β and b_0. Finally, after substituting them into Eq. (3.62), our model will intersect a plane into the transformed data set, as illustrated in Figure 3.10. This will be ultimately be seen as in Figure 3.11 when evaluating Eq. (3.62) with respect to the independent variables $\boldsymbol{x}_i$.

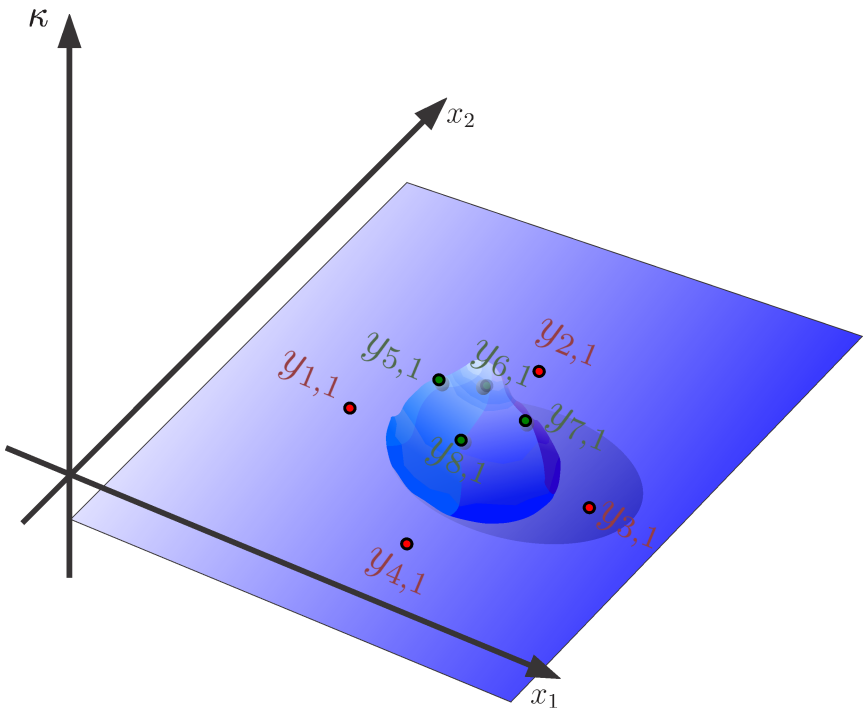


Figure 3.9: Illustrative example of applying a Gaussian function with the Kernel trick.

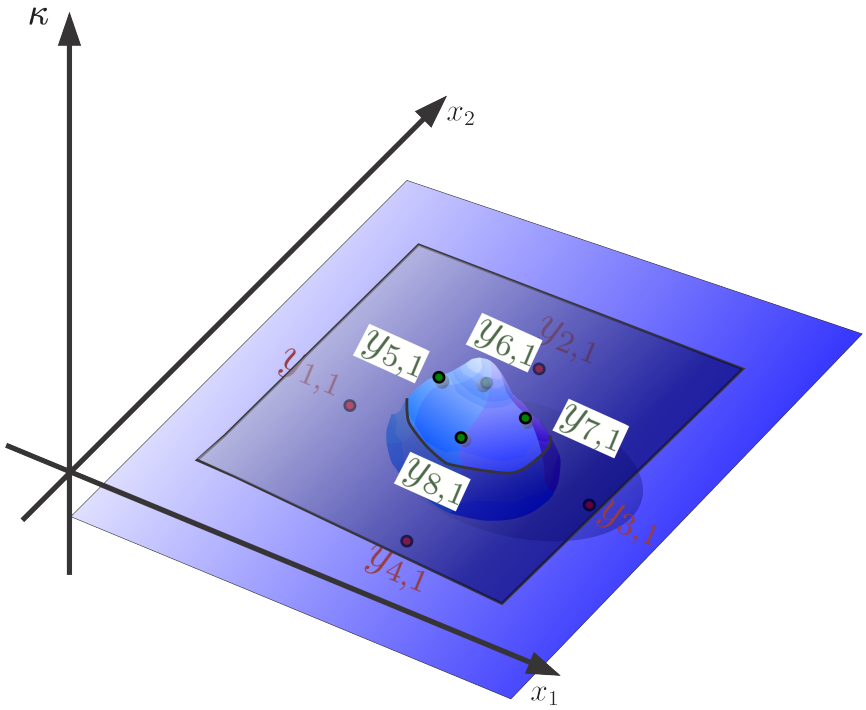


Figure 3.10: Illustrative example of the intersection of the plane from the Kernel machine classifier into a hypothetical data set.

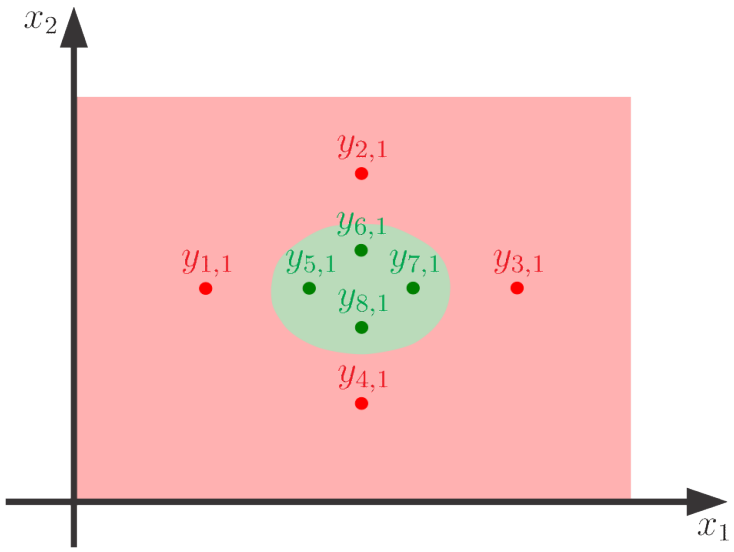


Figure 3.11: Illustrative example of the predicted results of the Kernel machine classification model, when using a Gaussian function for the Kernel, in a hypothetical dataset.

As a summary, all the processes that must be performed to apply the Kernel machine classification are listed in order in the Pseudocode 9.

Algorithm 9 getKernelMachineClassification

Input: $\boldsymbol{X}$, $\boldsymbol{Y}$
Output: $\boldsymbol{b}$

1: In order to obtain the coefficient values of the chosen Kernel function with respect to the input data $\boldsymbol{X}$ and $\boldsymbol{Y}$, implement the Pseudocode 2 for the case of the linear Kernel; Pseudocode 4 for the case of the polynomial Kernel; Pseudocode 5 for the case of the logistic Kernel; or Pseudocode 6 for the case of the Gaussian Kernel.
2: For each sample of $\boldsymbol{X}$, obtain the predicted outputs $\kappa(\boldsymbol{X})$ with the model that was previously generated for the chosen Kernel function.
3: Implement the Pseudocode 2 to apply the multiple linear regression with respect to the input data $\boldsymbol{X}$ and $\kappa(\boldsymbol{X})$ in order to obtain the values of the coefficients β and b_0 of Eq. (3.63).
4: Substitute the previously obtained coefficients β and b_0 into Eq. (3.62) in order to solve the model of the Kernel machine classification.
5: For the desired predicted data $\hat{\boldsymbol{Y}}$, apply Eq. (3.62) to determine to what group does such prediction belongs to.

In conclusion, it is important to note that the Kernel machine classification method is inspired by the Kernel SVM method [57] and that they are not the same. This is because the Kernel SVM method solves this problem by attempting the best centered hyperplane with respect to the support vectors. In contrast, the Kernel machine classification solves it by trying to predict the best centered hyperplane with respect to the data dispersion. Furthermore, the

Kernel machine classification method allows the possibility of obtaining better training results under restricted circumstances where feature scaling is not desired; a good or possible idea, especially when using low profile embedded systems. In addition, it is also important to distinguish that this method does not have sensitivity problems when having a feature (independent variable) that has large values with respect to another feature, as in the Kernel SVM method.

3.3.10 Single neuron in Deep Neural Network

The concept of a neuron, defined as a nerve cell that is part of the brain which is responsible for the cognitive functions of the body, was well known since the 1800s [34]. However, at that time, it was never demonstrated until there was a means to confirm it, thanks to the work of Camilo Golgi in 1873 [34]. There he developed a staining technique that would later be used by Santiago Ramon y Cajal to finally demonstrate its existence [34]. Finally, after several more contributions, Warren McCulloch and Walter Pitts published in 1943 the first mathematical model of a neural network [59]. As a result of all these efforts, today it is easy to find a reliable source of information that explains the mathematical model of a single artificial neuron and even an artificial neural network [2, 1, 34].

Since it is of interest to develop the algorithm of an artificial neuron, let us first detail the most relevant parts of a neuron for its mathematical modeling. As illustrated in Figure 3.12, every known neuron has a cell body that contains a cell nucleus and several fibers called dentrites, that branch from the cell body [34]. In addition, the cell body has a particularly long fiber known as the axon, that extends further as shown in Figure 3.12. The axon can reach a meter in length, but it is usually 1cm long and at its end, it has an axonal arborization that contains junctions called synapses [34]. By attaching the synapses of a particular neuron to the dentrite of another neuron, a connection will be formed between the two neurons [34]. Additionally, a neuron can normally make from 10 to 100'000 connections with other neurons, allowing the propagation of signals through them, by a complicated electrochemical reaction [34].

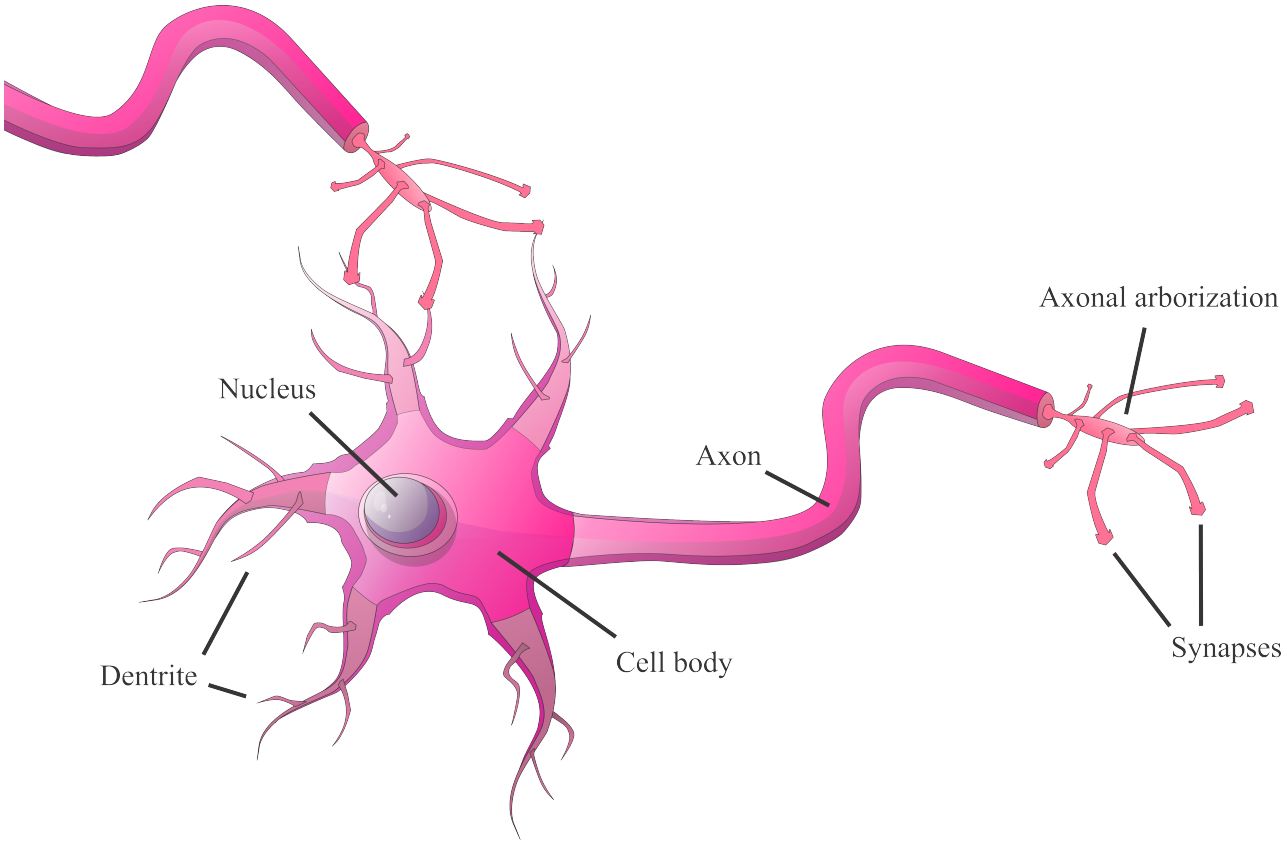


Figure 3.12: Didactic illustration of the parts of a neuron that are relevant for its mathematical modeling.

On the other hand, although the information described so far helps to make it possible to mathematically mimic the behavior of a neuron, there are also some initial mathematical definitions to be made. Therefore, let us consider a particular case of a matrix $\widetilde{\boldsymbol{X}}$ to represent a transformed version of Eq. (3.4) with n samples, each of them represented in rows. As well as that, this matrix has m independent variables $\widetilde{x}_{(i,k)}$, where i represents the sample and k the independent variable currently observed. In addition, this matrix has $m+1$ columns where the rows of the first one are all equal to the value of 1, such that $\widetilde{x}_{(i,0)} = 1$, and the others correspond to the m independent variables, where $\widetilde{x}_{(i,k)} = x_{(i,k)}$ from Eq. (3.4). Moreover, consider a particular case from Eq. (3.5) and Eq. (3.6) where $\boldsymbol{Y}$ and $\hat{\boldsymbol{Y}}$ have $p = 1$ dependent variable. Also, let us consider the real scalars $\omega_0, \omega_1, \cdots, \omega_m \in K$ such that $\boldsymbol{\omega} = (\omega_0, \omega_1, \cdots, \omega_m)^T$ where T represents the transpose of a matrix or vector and where $\boldsymbol{\omega}$ will be called weight vector. Finally, all the terms described so far are related in the following way:

$$\hat{\boldsymbol{Y}} = \widetilde{\boldsymbol{X}}\boldsymbol{b}$$

$$\begin{pmatrix} \hat{y}_{1,1} \\ \hat{y}_{2,1} \\ \vdots \\ \hat{y}_{n,1} \end{pmatrix} = \begin{pmatrix} \widetilde{x}_{1,0} & \widetilde{x}_{1,1} & \widetilde{x}_{1,2} & \cdots & \widetilde{x}_{1,m} \\ \widetilde{x}_{2,0} & \widetilde{x}_{2,1} & \widetilde{x}_{2,2} & \cdots & \widetilde{x}_{2,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \widetilde{x}_{n,0} & \widetilde{x}_{n,1} & \widetilde{x}_{n,2} & \cdots & \widetilde{x}_{n,m} \end{pmatrix} \begin{pmatrix} \omega_0 \\ \omega_1 \\ \vdots \\ \omega_m \end{pmatrix} \tag{3.64}$$

or alternatively, with the following representation by substituting the values of $\widetilde{x}_{(i,0)} = 1$ and $\widetilde{x}_{(i,k)} = x_{(i,k)}$ from Eq. (3.4) into Eq. (3.64):

$$\begin{pmatrix} \hat{y}_{1,1} \\ \hat{y}_{2,1} \\ \vdots \\ \hat{y}_{n,1} \end{pmatrix} = \begin{pmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{pmatrix} \begin{pmatrix} \omega_0 \\ \omega_1 \\ \vdots \\ \omega_m \end{pmatrix} \tag{3.65}$$

Subsequently, by using the artificial neuron model illustrated in Figure 3.13, it is determined that for the current sample i, we will have the following:

$$f(\widetilde{\boldsymbol{x}}_i) = \sum_{k=0}^{m} \omega_k \widetilde{x}_{i,k} \tag{3.66}$$

also representable as:

$$f(\widetilde{\boldsymbol{x}}_i) = \omega_0 + \omega_1 x_{i,1} + \omega_2 x_{i,2} + \cdots + \omega_m x_{i,m} \tag{3.67}$$

where $\widetilde{\boldsymbol{x}}_i = \left(\widetilde{x}_{i,0}, \widetilde{x}_{i,1}, \cdots, \widetilde{x}_{i,m}\right)$ and $f(\widetilde{\boldsymbol{x}}_i)$ would represent a summing function of all the input values of $\widetilde{\boldsymbol{x}}_i$ given to the artificial neuron. In addition, this summing function also includes a bias value ω_0 and the multiplier coefficients $\omega_1, \omega_2, \cdots, \omega_m$ of $\widetilde{\boldsymbol{x}}_i$. All together, the input values $\widetilde{\boldsymbol{x}}_i$; its multipliers $\omega_1, \omega_2, \cdots, \omega_m$; and the bias ω_0, represent the process of receiving the input signals into the cell body from the dentrites. Finally, from the elements described, $\omega_0, \omega_1, \cdots, \omega_m$ are the unknown real scalar values that must be identified to solve the model of an artificial neuron.

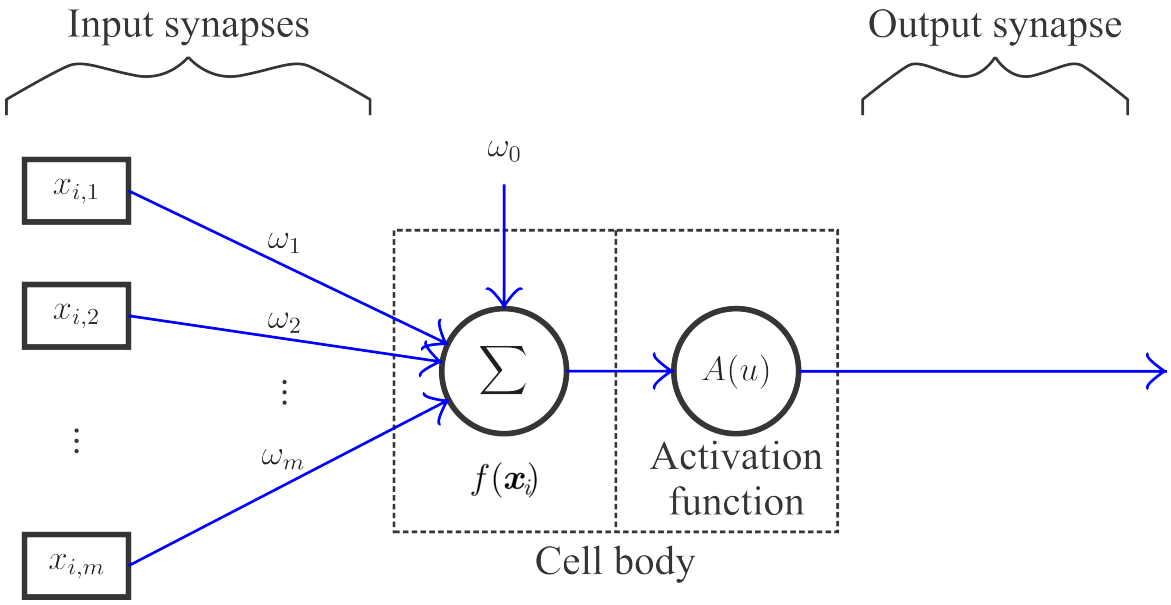


Figure 3.13: Model of a single neuron in Deep Neural Network.

Moreover, Figure 3.13 describes in the cell body, a transformation function $A(u)$ commonly known as activation function. This activation function is applied on the output value $f(\widetilde{\boldsymbol{x}}_i)$ that is obtained from Eq. (3.66). As a result, this will apply a domain transformation similar to the Kernel trick, that was explained in subsection 3.3.9. However, the reasons for which this concept is used here are completely different, mainly because of the training methodology used. Nonetheless, before explaining such a process, it is important to note that there are a growing number of activation functions available, of which the following are some of them:

- Rectified Linear Units (ReLU): This function is commonly used to force an output to have the same value as its input but that will give zero whenever the input value is negative, as shown in the following.

$$A(u) = \begin{cases} u & \text{if } u > 0 \\ 0 & \text{if } u \leq 0 \end{cases} \tag{3.68}$$

  whose first derivative is as follows:

$$\frac{dA(u)}{du} = \begin{cases} 1 & \text{if } u > 0 \\ 0 & \text{if } u \leq 0 \end{cases} \tag{3.69}$$

- Hyperbolic tangent (tanh): This sigmoid function is commonly used to force an output between -1 and 1 through the following equation.

$$A(u) = tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}} \quad | \quad -1 < A(u) < 1 \tag{3.70}$$

whose first derivative is as follows:

$$\frac{dA(u)}{du} = 1 - tanh^2(u)$$

$$\frac{dA(u)}{du} = 1 - [A(u)]^2 \tag{3.71}$$

- Logistic: This sigmoid function is commonly used to force an output between 0 and 1 through the following equation.

$$A(u) = \frac{1}{1 + e^{-u}} \quad | \quad 0 < A(u) < 1 \tag{3.72}$$

whose first derivative is as follows:

$$\frac{dA(u)}{du} = \frac{e^{-u}}{(1 + e^{-u})^2}$$

$$\frac{dA(u)}{du} = \frac{1}{1 + e^{-u}} \left(1 - \frac{1}{1 + e^{-u}}\right)$$

$$\frac{dA(u)}{du} = A(u)[1 - A(u)] \tag{3.73}$$

- Raise to the 1^{st} power: Also known as the identity activation function, it is commonly used as a non modifying equation because it will deliver in its output the exact same result as in its input.

$$A(u) = u \tag{3.74}$$

whose first derivative is as follows:

$$\frac{dA(u)}{du} = 1 \tag{3.75}$$

- Raise to the 2^{nd} power: As the name implies, this function is used to raise the input value u to the second power.

$$A(u) = u^2 \tag{3.76}$$

whose first derivative is as follows:

$$\frac{dA(u)}{du} = 2u \tag{3.77}$$

- Raise to the 3^{rd} power: As the name implies, this function is used to raise the input value u to the third power.

$$A(u) = u^3 \tag{3.78}$$

whose first derivative is as follows:

$$\frac{dA(u)}{du} = 3u^2 \tag{3.79}$$

- Raise to the 4^{th} power: As the name implies, this function is used to raise the input value u to the fourth power.

$$A(u) = u^4 \tag{3.80}$$

whose first derivative is as follows:

$$\frac{dA(u)}{du} = 4u^3 \tag{3.81}$$

- Raise to the 5^{th} power: As the name implies, this function is used to raise the input value u to the fifth power.

$$A(u) = u^5 \tag{3.82}$$

whose first derivative is as follows:

$$\frac{dA(u)}{du} = 5u^4 \tag{3.83}$$

- Raise to the 6^{th} power: As the name implies, this function is used to raise the input value u to the sixth power.

$$A(u) = u^6 \tag{3.84}$$

whose first derivative is as follows:

$$\frac{dA(u)}{du} = 6u^5 \tag{3.85}$$

- 1^{st} order degree exponential: As the name implies, this function is used to raise the input value u to a first order exponential.

$$A(u) = e^u \tag{3.86}$$

whose first derivative is as follows:

$$\frac{dA(u)}{du} = e^u$$

$$\frac{dA(u)}{du} = A(u) \tag{3.87}$$

- 2^{nd} order degree exponential: As the name implies, this function is used to raise the input value u to a second order exponential.

$$A(u) = e^{(u^2)} \tag{3.88}$$

whose first derivative is as follows:

$$\frac{dA(u)}{du} = 2ue^{(u^2)}$$

$$\frac{dA(u)}{du} = 2uA(u) \tag{3.89}$$

where for all these activation functions:

$$u = f(\widetilde{\boldsymbol{x}}_i) \tag{3.90}$$

Now, as Figure 3.13 suggests, to model an artificial neuron it is necessary to apply the desired/selected activation function $A(u)$ to the resulting value of $f(\widetilde{\boldsymbol{x}}_i)$ from Eq. (3.66). As a result, the output value of the activation function $A(u)$ would represent the predicted value of the artificial neuron, which can also be denoted as $\hat{y}_{i,1}$ for the current predicted sample. However, for this prediction to be consistent with the training data used, it is necessary to explain the expected training process of the artificial neuron. Therefore, let us begin by developing the corresponding expression of the error function for this case. Hence, a modified version of the least squares method of Eq. (3.30) will be used, as it is mathematically convenient for the particular training process to be described:

$$MSSE = \frac{1}{2}\sum_{i=1}^{n} e_i^2 \tag{3.91}$$

where $MSSE$ stands for the modified least squares method and where such equation can also be described as follows:

$$MSSE = \frac{1}{2}\sum_{i=1}^{n}(y_{i,1} - \hat{y}_{i,1})^2 \tag{3.92}$$

$$MSSE = \frac{1}{2}\sum_{i=1}^{n}\Big(y_{i,1} - A(u)\Big)^2 \tag{3.93}$$

Before continuing, lets state that the objective of the training process of the artificial neuron is to learn the most appropriate values for the unknown coefficients $\boldsymbol{\omega}$ little by little. Therefore, the training of this algorithm will be performed using the gradient descent method, attributed to Cauchy since 1847 [60]. In addition, we will have to establish that the values of the unknown scalar coefficients $\omega_0, \omega_1, \cdots, \omega_m$ are initialized with random values that will lie between -1 and 1. The reasoning behind this motive is that it is an expectation of the gradient descent method to initialize them with small random numbers. Furthermore, when running this algorithm with the gradient descent method, it will find a better fitting value at each of the next occurring iterations (also known as epochs), as illustrated in Figure 3.14. Moreover, from a mathematical perspective, it can be clearly seen in this figure that, the goal of the training process is to reach the critical value of $MSSE$ with respect to ω_k.

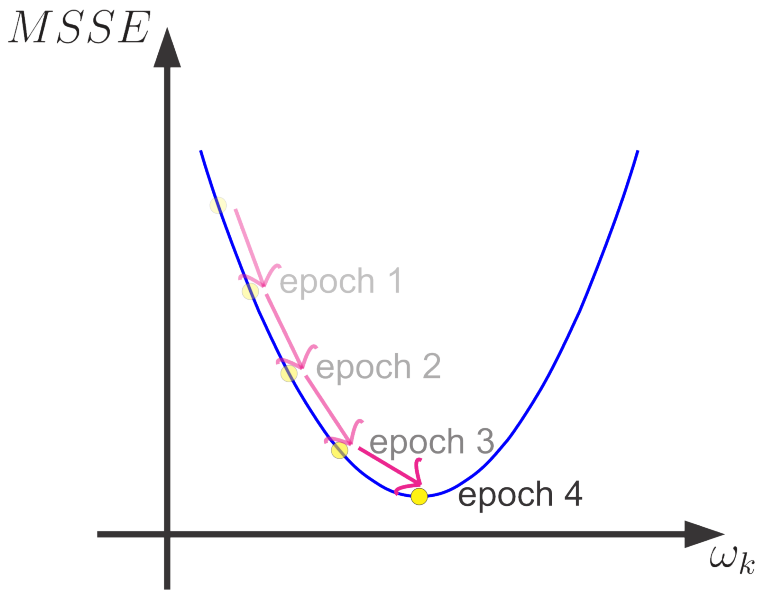


Figure 3.14: Process of updating the value of a randomly initialized weight through several iterations/epochs in a hypothetical $MSSE$ function.

As a result of the analysis of Figure 3.14, it is possible to define the equation that will update the values for each of the coefficients ω_k, such that:

$$\omega_{k_{(new)}} = \omega_{k_{(old)}} + \Delta\omega_k \tag{3.94}$$

where $\omega_{k_{(old)}}$ represents the current weight value, $\omega_{k_{(new)}}$ stands for the updated value and $\Delta\omega_k$ is the differentiation of the position of the new updated value $\omega_{k_{(new)}}$ with respect to the current one $\omega_{k_{(old)}}$.

Nevertheless, Eq. (3.94) and the analysis done so far, cannot tell us yet what will be the equivalence of $\Delta\omega_k$, but we know that it has to be equal to the update value of the current

coefficient ω_k. Fortunately, it happens that when we derive a function, it will always be true that the critical value of the nearest valley will be in the direction of where the slope is negative. Therefore, as illustrated in Figure 3.15, we can determine that $\Delta\omega_k$ has to be proportional to the negative of the partial derivative of $MSSE$ with respect to the current weight ω_k:

$$\Delta\omega_k \propto -\frac{\partial(MSSE)}{\partial\omega_k} \tag{3.95}$$

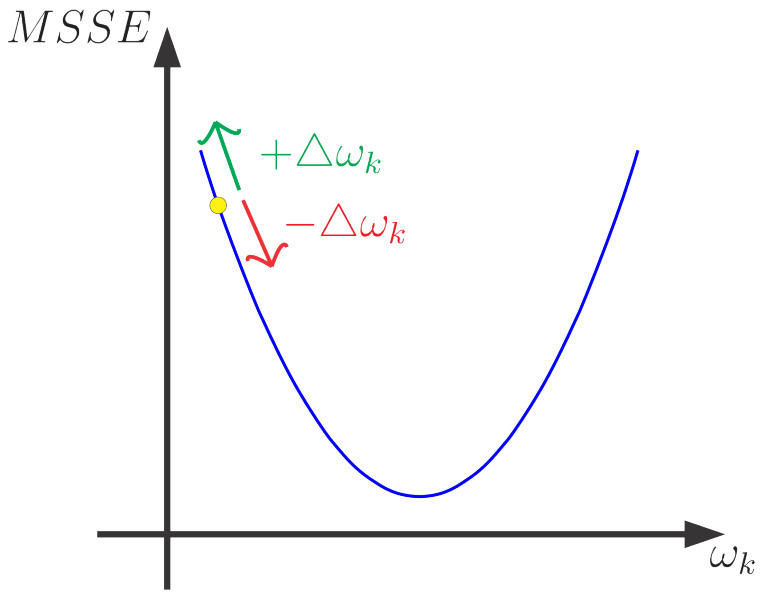


Figure 3.15: Relationship of the sign of the slope with respect to the direction of a randomly initialized weight in a hypothetical $MSSE$ function.

However, although Eq. (3.95) gives us an idea of what $\Delta\omega_k$ has to be proportional to, that equation is not an equivalence. At the same time, there is no definite way to know what the correct equivalence is going to be, because the shape of the function $MSSE$ can have a wide variety of forms (it will not always have a parabolic shape). Nevertheless, a commonly used strategy is to multiply Eq. (3.95) by an arbitrary scaling parameter, which means that this is a value that will have to be randomly defined and tuned by the machine learning practitioner. As a result, Eq. (3.95) will become the following:

$$\Delta\omega_k = -\lambda\frac{\partial(MSSE)}{\partial\omega_k} \tag{3.96}$$

where λ is this arbitrary scaling parameter (all arbitrary scaling parameters are also called hyperparameters in machine learning), also known as "learning rate". The reason for such a name is because this learning rate will literally define the length of the steps by which each

epoch will obtain a new weight value, as illustrated in Figure 3.14.

The Eq. (3.96) is key to being able to solve the artificial neuron model $A(u)$, so that the algorithm mimics the behavior of a neuron as expected. However, it is not yet defined what is the equivalence of $\frac{\partial(MSSE)}{\partial\omega_k}$, but it can be obtained by developing the partial derivative of Eq. (3.93):

$$MSSE = \frac{1}{2}\sum_{i=1}^{n}\Big(y_{i,1} - A(u)\Big)^2$$

$$\frac{\partial(MSSE)}{\partial\omega_k} = \frac{1}{2}\sum_{i=1}^{n}\left((2)\Big(y_{i,1} - A(u)\Big)\frac{\partial}{\partial\omega_k}\Big(y_{i,1} - A(u)\Big)\right)$$

$$\frac{\partial(MSSE)}{\partial\omega_k} = \frac{2}{2}\sum_{i=1}^{n}\left(\Big(y_{i,1} - A(u)\Big)\left(\frac{\partial y_{i,1}}{\partial\omega_k} - \frac{\partial A(u)}{\partial\omega_k}\right)\right)$$

$$\frac{\partial(MSSE)}{\partial\omega_k} = \sum_{i=1}^{n}\left(\Big(y_{i,1} - A(u)\Big)\left((0) - \frac{\partial A(u)}{\partial\omega_k}\right)\right)$$

$$\frac{\partial(MSSE)}{\partial\omega_k} = -\sum_{i=1}^{n}\left(\Big(y_{i,1} - A(u)\Big)\left(\frac{\partial A(u)}{\partial\omega_k}\right)\right)$$

To continue developing this mathematical expression, we apply the chain rule to solve the partial derivative $\frac{\partial A(u)}{\partial\omega_k}$ contained in it:

$$\frac{\partial(MSSE)}{\partial\omega_k} = -\sum_{i=1}^{n}\left(\Big(y_{i,1} - A(u)\Big)\left(\left(\frac{dA(u)}{du}\right)\left(\frac{\partial(u)}{\partial\omega_k}\right)\right)\right)$$

Now, given Eq. (3.90), this will allow us to obtain the following:

$$\frac{\partial(MSSE)}{\partial\omega_k} = -\sum_{i=1}^{n}\left(\Big(y_{i,1} - A(u)\Big)\left(\frac{dA(u)}{du}\right)\left(\frac{\partial\Big(f(\widetilde{\boldsymbol{x}}_i)\Big)}{\partial\omega_k}\right)\right)$$

Next, we substitute the equivalence of $f(\widetilde{\boldsymbol{x}}_i)$, from Eq. (3.66), into the currently developed equation such that:

$$\frac{\partial(MSSE)}{\partial\omega_k} = -\sum_{i=1}^{n}\left(\Big(y_{i,1} - A(u)\Big)\left(\frac{dA(u)}{du}\right)\left(\frac{\partial\left(\sum_{k=0}^{m}\omega_k\widetilde{x}_{i,k}\right)}{\partial\omega_k}\right)\right)$$

$$\frac{\partial(MSSE)}{\partial\omega_k} = -\sum_{i=1}^{n}\left(\left(y_{i,1} - A(u)\right)\left(\frac{dA(u)}{du}\right)\sum_{k=0}^{m}\left(\frac{\partial\left(\omega_k\widetilde{x}_{i,k}\right)}{\partial\omega_k}\right)\right)$$

Here, note in the term $\sum_{k=0}^{m}\left(\frac{\partial\left(\omega_k\widetilde{x}_{i,k}\right)}{\partial\omega_k}\right)$ that because we are deriving with respect to the current weight, that means that all the other weights will give a value of zero, which will give us the following expression:

$$\frac{\partial(MSSE)}{\partial\omega_k} = -\sum_{i=1}^{n}\left(\left(y_{i,1} - A(u)\right)\left(\frac{dA(u)}{du}\right)\left((1)\widetilde{x}_{i,k}\right)\right)$$

$$\frac{\partial(MSSE)}{\partial\omega_k} = -\sum_{i=1}^{n}\left(\left(y_{i,1} - A(u)\right)\left(\frac{dA(u)}{du}\right)\left(\widetilde{x}_{i,k}\right)\right) \tag{3.97}$$

$$\frac{\partial(MSSE)}{\partial\omega_k} = -\sum_{i=1}^{n}\left(\varepsilon\widetilde{x}_{i,k}\right) \tag{3.98}$$

where $\varepsilon = \left(y_{i,1} - A(u)\right)\left(\frac{dA(u)}{du}\right)$ and is known as the "error term".

Thanks to the obtained representation, we can now know the equivalence of $\Delta\omega_k$ by substituting the term from Eq. (3.98) into Eq. (3.96):

$$\Delta\omega_k = -\lambda\frac{\partial(MSSE)}{\partial\omega_k}$$

$$\Delta\omega_k = -\lambda\left(-\sum_{i=1}^{n}\left(\varepsilon\widetilde{x}_{i,k}\right)\right)$$

$$\Delta\omega_k = \lambda\sum_{i=1}^{n}\left(\varepsilon\widetilde{x}_{i,k}\right) \tag{3.99}$$

and then we substitute Eq. (3.99) into Eq. (3.94) to finally obtain the equation to be used to update the value of the current weight of the artificial neuron:

$$\omega_{k_{(new)}} = \omega_{k_{(old)}} + \Delta\omega_k$$

$$\omega_{k_{(new)}} = \omega_{k_{(old)}} + \lambda\sum_{i=1}^{n}\left(\varepsilon\widetilde{x}_{i,k}\right) \tag{3.100}$$

or alternatively in its developed form, when using Eq. (3.97) over Eq. (3.98) to substitute it into Eq. (3.96), whose resulting term is substituted into Eq. (3.94):

$$\omega_{k_{(new)}} = \omega_{k_{(old)}} + \lambda\sum_{i=1}^{n}\left(\left(y_{i,1} - A(u)\right)\left(\frac{dA(u)}{du}\right)\left(\widetilde{x}_{i,k}\right)\right) \tag{3.101}$$

or in its other alternative representation, when it is desired to obtain all the updated weight values:

$$\boldsymbol{\omega}_{(new)} = \boldsymbol{\omega}_{(old)} + \lambda \begin{pmatrix} \Big(y_{1,1} - A(u)\Big)\left(\dfrac{dA(u)}{du}\right) \\ \Big(y_{2,1} - A(u)\Big)\left(\dfrac{dA(u)}{du}\right) \\ \vdots \\ \Big(y_{n,1} - A(u)\Big)\left(\dfrac{dA(u)}{du}\right) \end{pmatrix}^{T} . \left(\widetilde{\boldsymbol{X}}\right) \tag{3.102}$$

where it is indispensable for Eq. (3.100); Eq. (3.101) and Eq. (3.102) to recall that $u = f(\widetilde{\boldsymbol{x}}_i) = \sum_{k=0}^{m} \omega_k \widetilde{x}_{i,k}$, according to Eq. (3.90) and Eq. (3.66). Moreover, it is important to point out that it is up to the machine learning practitioner to define how many epochs it is desired for the artificial neuron to be trained. In addition, a "stop condition" can be simultaneously implemented to immediately stop the algorithm if it finds some weight values that cause the function $A(u)$ to have a specified accuracy or more than that.

To conclude, as a summary of all the processes that must be performed to train an artificial neuron, the Pseudocode 10 lists these steps in an orderly fashion.

Algorithm 10 getSingleNeuronInDNN

Input: $activationFuntionToBeUsed$, $maxEpoch$, λ, $desiredAccuracy$, $\boldsymbol{X}$, $\boldsymbol{Y}$, $isCustomW$, $custom\_w$

Output: $\boldsymbol{\omega}_{new}$

Fill the matrix $\widetilde{\boldsymbol{X}}$ with the values of $\boldsymbol{X}$ as described in Eq. (3.65)
if $isCustomW =$ false **then**
    Initialize $\boldsymbol{\omega}_{new}$ with random values from -1 to 1
else
    $\boldsymbol{\omega}_{new} = custom\_w$
end if
Calculate $f(\widetilde{\boldsymbol{x}}_i)$ for all samples available through Eq. (3.66) using $\boldsymbol{\omega}_{new}$
Determine what activation function $A(u)$ has to be used with $activationFuntionToBeUsed$
Calculate $A(u)$ and $\frac{dA(u)}{du}$ from Eq. (3.68) to Eq. (3.89) correspondingly
$\hat{\boldsymbol{Y}} = A(u)$
calculate $currentAccuracy$ of $\hat{\boldsymbol{Y}}$ using your preferred accuracy method
if $currentAccuracy > desiredAccuracy$ **then**
    return $\boldsymbol{\omega}_{new}$ ▷ Return the latest weight values obtained
end if
for $currentEpoch = 0, 1, \cdots, maxEpoch$ **do** ▷ Start the artificial neuron training
    $\boldsymbol{\omega}_{old} = \boldsymbol{\omega}_{new}$
    Recalculate $\boldsymbol{\omega}_{new}$ with Eq. (3.100); Eq. (3.101) or Eq. (3.102)
    Recalculate $f(\widetilde{\boldsymbol{x}}_i)$ from Eq. (3.66) using $\boldsymbol{\omega}_{new}$
    Recalculate $A(u)$ and $\frac{dA(u)}{du}$ from Eq. (3.68) to Eq. (3.89) correspondingly
    $\hat{\boldsymbol{Y}} = A(u)$
    Calculate $currentAccuracy$ of $\hat{\boldsymbol{Y}}$ using your preferred accuracy method
    if $currentAccuracy > desiredAccuracy$ **then**
        break
    end if
end for
return $\boldsymbol{\omega}_{new}$ ▷ Return the latest weight values obtained

3.3.11 Deep Neural Network with a single output

A deep neural network consists of exactly the same principles described for the single artificial neuron model as explained in the subsection 3.3.10, but with only one difference. This is the fact of having a model with more than one artificial neuron working simultaneously and where several layers exist, as illustrated in Figure 3.16. There, it is shown that a deep neural network has three types of layers, where the first layer among all others is the input layer. This layer is represented with the identifier 0 and contains the values of all the features of the model for the current sample taken. The last layer, labeled with the identifier L, is known as output layer and

usually contains all the output artificial neurons of the model. However, due to the interests of this thesis and to make the mathematical development simpler and more convenient, a single artificial neuron for the output layer will be proposed. Lastly, all layers located between the input and output layers are called hidden layers and they can have the identifier from 1 to $(L-1)$. As for their purpose, the hidden and output layers work together to process all the input data of the model.

In addition, Figure 3.16 describes several artificial neurons N_e distributed along all the hidden and output layers. These neurons are arranged in a determined positioned and are labeled as $N_{e(r,c)}$, where r represents the current row position and c denotes the current column of the neuron under observation or whose output is to be calculated. The proposal in this work is that the maximum value of r depends on the total number of neurons created for the current layer. This means that there can be different numbers of rows for each layer and they will be labeled with the constant values $m_0, m_1, \cdots, m_{(L-1)}, m_L$. Here, m_0 will represent the machine learning features or independent variables of the model (for previous models m was used instead). On the other hand, the values of m_1 up to $m_{(L-1)}$ are proposed by the machine learning practitioner. In contrast, $m_L = 1$ because as mentioned earlier, this model will describe a single output neuron. Lastly, for the total number of layers proposed (L), c can be any value from 1 to L.

Furthermore, by having several weight values for several artificial neurons contained in such an elaborated distribution, a matrix for the weight values will be used instead of a vector as in previous methods. Therefore, the current weight value $\omega_{(r,c,k)}$ will be identified by the values of r, c and through k, which will represent a specific weight of that neuron. Moreover, the mathematical considerations and definitions made for the input values $\widetilde{\boldsymbol{X}}$ and the output values $\boldsymbol{Y}, \hat{\boldsymbol{Y}}$ will remain the same as those mentioned in the description of Eq. (3.64) and Eq. (3.65).

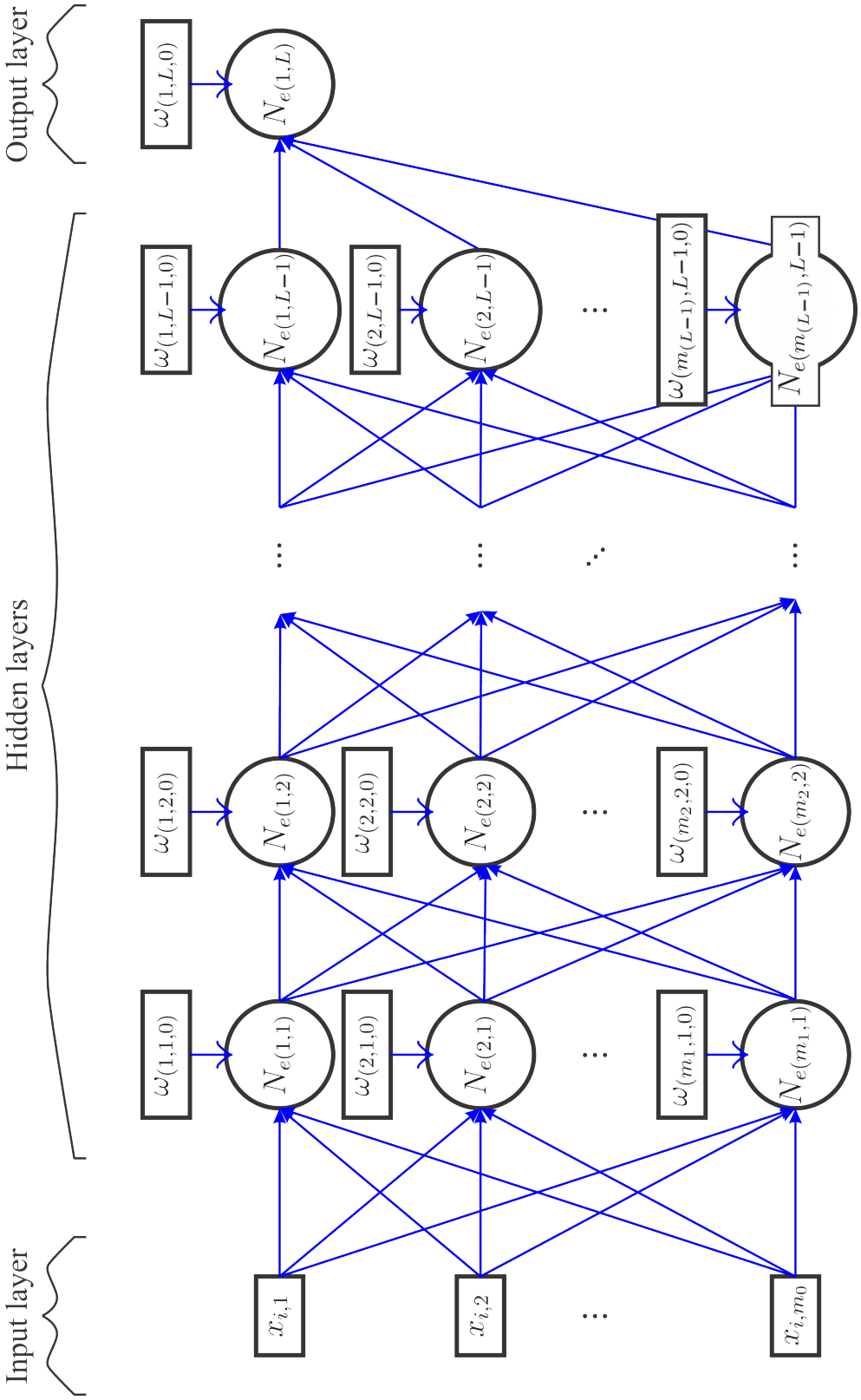


Figure 3.16: Model of a deep neural network.

Moreover, given Eq. (3.90) and Eq. (3.66), it is known that $u = f(\widetilde{\boldsymbol{x}}_i) = \sum_{k=0}^{m} \omega_k \widetilde{x}_{i,k}$ for a single artificial neuron model. But after interpreting for an deep neural network, this will mean that all the input values towards the current neuron will be multiplied by their corresponding weight values and then summed. Therefore, by applying this concept, let us redefine such expression into the following:

$$u_{(r,c)} = \begin{cases} \sum_{k=0}^{m_0} \omega_{(r,c,k)} \widetilde{x}_{(i,k)} & \text{if } c = 1 \\ \sum_{k=0}^{m_{(c-1)}} \omega_{(r,c,k)} A(u_{(k,c-1)}) & \text{if } c > 1 \end{cases} \tag{3.103}$$

where $u_{(r,c)}$ will represent the particular output value for the neuron $N_{e(r,c)}$ in a deep neural network, but without an activation function applied to it.

Also, let us establish that each neuron will have its own activation function and will be defined by the machine learning practitioner. In this sense, all neurons can have different or equal activation functions and will be represented as $A(u_{(r,c)})$. Additionally, the neurons contained in all layers, except layer 1, will individually process the output values of the neurons contained in the previous layer by using them as their input values. On the other hand, the neurons located in layer 1 will process the samples of the machine learning features by using them as their input values. Consequently, because it will be mathematically convenient, the following will be defined to make it clear how to interpret the inputs of the current neuron whose output is to be calculated:

$$A(u_{(k,c-1)}) = \begin{cases} 1 & \text{if } k = 0 \quad | \quad c > 1 \\ A(u_{(k,c-1)}) & \text{if } k > 0 \quad | \quad c > 1 \end{cases} \tag{3.104}$$

$$A^{'}(u_{(k,c-1)}) = \begin{cases} 0 & \text{if } k = 0 \quad | \quad c > 1 \\ A^{'}(u_{(k,c-1)}) & \text{if } k > 0 \quad | \quad c > 1 \end{cases} \tag{3.105}$$

where $A(u_{(r,c)})$ and $A^{'}(u_{(r,c)})$ will stand for the output of the neuron $N_{e(r,c)}$ that is to be calculated and $A(u_{(k,c-1)})$ and $A^{'}(u_{(k,c-1)})$ for a certain neuron that is to be used as input of $N_{e(r,c)}$.

To complement this, Eq. (3.104) and Eq. (3.105) are supported by an observation made for all cases where there are neurons in the layer prior to $N_{e(r,c)}$. When considering $A(u_{(k,c-1)})$ and $A^{'}(u_{(k,c-1)})$, the row position of these input neurons (with respect to $N_{e(r,c)}$), will be identified with k and can be any number from 1 to $m_{(c-1)}$. What this tries to describe, is that no neuron will exist when $k = 0$ for all cases and for convenience they will be interpreted as the input value

containing a unit value, which will be used for the bias of the neuron $N_{e(r,c)}$ under observation. As for the input layer, this will not apply there because no neurons are present in that layer at all, which means that it is impossible to apply an activation function there.

With all these considerations in mind, as in the single artificial neuron model, it is intended to initialize all the weights with random values ranging between -1 and 1. Next, it is necessary to have the mathematical model to train these weight values in a similar manner to how a real neuron or, in this case, a deep neural network would. To do this, we will redefine Eq. (3.94) slightly, just so that it can be applied for $N_{e(r,c)}$, but by retaining the definitions already given so far for a deep neural network model, such that:

$$\omega_{(r,c,k)_{(new)}} = \omega_{(r,c,k)_{(old)}} + \Delta\omega_{(r,c,k)} \tag{3.106}$$

where $\Delta\omega_{(r,c,k)}$ can be intuitively deduced from Eq. (3.96) as follows:

$$\Delta\omega_{(r,c,k)} = -\lambda\frac{\partial(MSSE)}{\partial\omega_{(r,c,k)}} \tag{3.107}$$

In order to further solve Eq. (3.106), it will be necessary to determine the equivalence of $\frac{\partial(MSSE)}{\partial\omega_{(r,c,k)}}$ from Eq. (3.107). However, it happens that its equality will be different depending on the layer in which the neuron to be updated is located. Therefore, the particular solution for when such layer is L, $(L-1)$ and $(L-2)$ will be presented below, so that sufficient information is provided to give the means to solve any case of a deep neural network:

- First, with the help of Eq. (3.92), the following will give solution to $\frac{\partial(MSSE)}{\partial\omega_{(r,c,k)}}$ for the particular case where the current neuron to be updated is in the output layer, which will be for the neuron $N_{e(1,L)}$:

$$MSSE = \frac{1}{2}\sum_{i=1}^{n}(y_{i,1} - \hat{y}_{i,1})^2$$

$$\frac{\partial(MSSE)}{\partial\omega_{(1,L,k)}} = \frac{1}{2}\sum_{i=1}^{n}\left((2)\left(y_{i,1} - \hat{y}_{i,1}\right)\frac{\partial}{\partial\omega_{(1,L,k)}}\left(y_{i,1} - \hat{y}_{i,1}\right)\right)$$

  Now, if it is taken into account that the activation function of the neuron $N_{e(1,L)}$ is $A(u_{(1,L)})$, this will mean that $\hat{y}_{i,1} = A(u_{(1,L)})$. Consequently, this equivalence can be conveniently used as follows:

$$\frac{\partial(MSSE)}{\partial\omega_{(1,L,k)}} = \frac{2}{2}\sum_{i=1}^{n}\left(y_{i,1} - \hat{y}_{i,1}\right)\frac{\partial}{\partial\omega_{(1,L,k)}}\left(y_{i,1} - A(u_{(1,L)})\right)$$

$$\frac{\partial(MSSE)}{\partial\omega_{(1,L,k)}} = \sum_{i=1}^{n}\left(y_{i,1} - \hat{y}_{i,1}\right)\left(\frac{\partial y_{i,1}}{\partial\omega_{(1,L,k)}} - \frac{\partial A(u_{(1,L)})}{\partial\omega_{(1,L,k)}}\right)$$

$$\frac{\partial(MSSE)}{\partial\omega_{(1,L,k)}} = \sum_{i=1}^{n}\left(y_{i,1} - \hat{y}_{i,1}\right)\left((0) - \frac{\partial A(u_{(1,L)})}{\partial\omega_{(1,L,k)}}\right)$$

$$\frac{\partial(MSSE)}{\partial\omega_{(1,L,k)}} = -\sum_{i=1}^{n}\left(y_{i,1} - \hat{y}_{i,1}\right)\left(\frac{\partial A(u_{(1,L)})}{\partial\omega_{(1,L,k)}}\right)$$

To continue developing this mathematical expression, we apply the chain rule to solve the partial derivative $\frac{\partial A(u_{(1,L)})}{\partial\omega_{(1,L,k)}}$ contained in it:

$$\frac{\partial(MSSE)}{\partial\omega_{(1,L,k)}} = -\sum_{i=1}^{n}\left(y_{i,1} - \hat{y}_{i,1}\right)\left(\frac{dA(u_{(1,L)})}{du_{(1,L)}}\right)\left(\frac{\partial du_{(1,L)}}{\partial\omega_{(1,L,k)}}\right)$$

Now, given Eq. (3.103), it will be possible to obtain the following if the equivalence of $u_{(1,L)}$ is substituted in the current mathematical expression:

$$\frac{\partial(MSSE)}{\partial\omega_{(1,L,k)}} = -\sum_{i=1}^{n}\left(y_{i,1} - \hat{y}_{i,1}\right)\left(A^{'}(u_{(1,L)})\right)\left(\frac{\partial\left(\sum_{k=0}^{m_{(L-1)}}\omega_{(1,L,k)}A(u_{(k,L-1)})\right)}{\partial\omega_{(1,L,k)}}\right) \tag{3.108}$$

$$\frac{\partial(MSSE)}{\partial\omega_{(1,L,k)}} = -\sum_{i=1}^{n}\left(y_{i,1} - \hat{y}_{i,1}\right)A^{'}(u_{(1,L)})A(u_{(k,L-1)}) \tag{3.109}$$

- Secondly, again with the help of Eq. (3.92), the following will give solution to $\frac{\partial(MSSE)}{\partial\omega_{(r,c,k)}}$ for the case where the current neuron to be updated is in a hidden layer, particularly for the neuron $N_{e(r,L-1)}$. However, for practical reasons, the development of such mathematical process will be continued from Eq. (3.108) but having the partial derivative with respect to $\omega_{(r,L-1,k)}$ instead of $\omega_{(1,L,k)}$:

$$\frac{\partial(MSSE)}{\partial\omega_{(r,L-1,k)}} = -\sum_{i=1}^{n}\left(y_{i,1} - \hat{y}_{i,1}\right)\left(A^{'}(u_{(1,L)})\right)\left(\frac{\partial\left(\sum_{\breve{w}_L=0}^{m_{(L-1)}}\omega_{(1,L,\breve{w}_L)}A(u_{(\breve{w}_L,L-1)})\right)}{\partial\omega_{(r,L-1,k)}}\right)$$

Now, it is indispensable to observe on this mathematical expression that we are deriving the term $u_{(1,L)}$ with respect to a weight contained in a neuron that is inside of it. This

means that according to Eq. (3.104) and Eq. (3.105) and the case for when $\breve{w}_L = 0$, this will make the derivation to result in zero because the bias of $u_{(1,L)}$ is independent of the current weight to be updated. Therefore, this mathematical expression will now result in the following:

$$\frac{\partial(MSSE)}{\partial\omega_{(r,L-1,k)}} = -\sum_{i=1}^{n}\Big(y_{i,1} - \hat{y}_{i,1}\Big)\Big(A^{'}(u_{(1,L)})\Big)\left(\frac{\partial\Big(\sum\limits_{\breve{w}_L=1}^{m_{(L-1)}}\omega_{(1,L,\breve{w}_L)}A(u_{(\breve{w}_L,L-1)})\Big)}{\partial\omega_{(r,L-1,k)}}\right)$$

$$\frac{\partial(MSSE)}{\partial\omega_{(r,L-1,k)}} = -\sum_{i=1}^{n}\Big(y_{i,1} - \hat{y}_{i,1}\Big)A^{'}(u_{(1,L)})\sum_{\breve{w}_L=1}^{m_{(L-1)}}\omega_{(1,L,\breve{w}_L)}\frac{\partial}{\partial\omega_{(r,L-1,k)}}\Big(A(u_{(\breve{w}_L,L-1)})\Big)$$

To continue developing this mathematical expression, we apply the chain rule to solve the partial derivative $\dfrac{\partial A(u_{(\breve{w}_L,L-1)})}{\partial\omega_{(1,L,k)}}$ contained in it:

$$\frac{\partial(MSSE)}{\partial\omega_{(r,L-1,k)}} = -\sum_{i=1}^{n}\Big(y_{i,1} - \hat{y}_{i,1}\Big)A^{'}(u_{(1,L)})\sum_{\breve{w}_L=1}^{m_{(L-1)}}\omega_{(1,L,\breve{w}_L)}\left(\frac{dA(u_{(\breve{w}_L,L-1)})}{du_{(\breve{w}_L,L-1)}}\right)\left(\frac{\partial du_{(\breve{w}_L,L-1)}}{\partial\omega_{(r,L-1,k)}}\right)$$

Now, given Eq. (3.103), it will be possible to obtain the following if the equivalence of $u_{(\breve{w}_L,L-1)}$ is substituted in the current mathematical expression:

$$\frac{\partial(MSSE)}{\partial\omega_{(r,L-1,k)}} = -\sum_{i=1}^{n}\Big(y_{i,1} - \hat{y}_{i,1}\Big)A^{'}(u_{(1,L)})\sum_{\breve{w}_L=1}^{m_{(L-1)}}\omega_{(1,L,\breve{w}_L)}\Big(A^{'}(u_{(\breve{w}_L,L-1)})\Big)\left(\frac{\partial\Big(\sum\limits_{\breve{w}_{(L-1)}=0}^{m_{(L-2)}}\omega_{(\breve{w}_L,L-1,\breve{w}_{(L-1)})}A(u_{(\breve{w}_{(L-1)},L-2)})\Big)}{\partial\omega_{(r,L-1,k)}}\right) \tag{3.110}$$

Next, because we are deriving with respect to the currently specified neuron, all the derivations with respect to non specified neurons, through $\breve{w}_L$, are zero. Subsequently, the same process will occur for all the unspecified weight values, through $\breve{w}_{(L-1)}$ and as a result, Eq. (3.110) will turn into the following:

$$\frac{\partial(MSSE)}{\partial\omega_{(r,L-1,k)}} = -\sum_{i=1}^{n}\Big(y_{i,1} - \hat{y}_{i,1}\Big)A^{'}(u_{(1,L)})\omega_{(1,L,r)}A^{'}(u_{(r,L-1)})\Big(A(u_{(k,L-2)})\Big) \tag{3.111}$$

- Thirdly, again with the help of Eq. (3.92), the following will give solution to $\dfrac{\partial(MSSE)}{\partial\omega_{(r,c,k)}}$ for the case where the current neuron to be updated is in a hidden layer, particularly for the

neuron $N_{e(r,L-2)}$. However, for practical reasons, the development of such mathematical process will be continued from Eq. (3.110) but having the partial derivative with respect to $\omega_{(r,L-2,k)}$ instead of $\omega_{(1,L-1,k)}$:

$$\frac{\partial(MSSE)}{\partial\omega_{(r,L-2,k)}} = -\sum_{i=1}^{n}\Big(y_{i,1}-\hat{y}_{i,1}\Big)A^{'}(u_{(1,L)})\sum_{\breve{w}_L=1}^{m_{(L-1)}}\omega_{(1,L,\breve{w}_L)}\Big(A^{'}(u_{(\breve{w}_L,L-1)})\Big)$$

$$\left(\frac{\partial\Big(\sum_{\breve{w}_{(L-1)}=0}^{m_{(L-2)}}\omega_{(\breve{w}_L,L-1,\breve{w}_{(L-1)})}A(u_{(\breve{w}_{(L-1)},L-2)})\Big)}{\partial\omega_{(r,L-2,k)}}\right)$$

Now, it is indispensable to observe on this mathematical expression that we are deriving the term $u_{(\breve{w}_L,L-1)}$ with respect to a weight contained in a neuron that is inside of it. This means that according to Eq. (3.104) and Eq. (3.105) and the case for when $\breve{w}_{(L-1)}=0$, this will make the derivation to result in zero because the biases of all the possibles $u_{(\breve{w}_L,L-1)}$ are independent of the current weight to be updated. Therefore, this mathematical expression will now result in the following:

$$\frac{\partial(MSSE)}{\partial\omega_{(r,L-2,k)}} = -\sum_{i=1}^{n}\Big(y_{i,1}-\hat{y}_{i,1}\Big)A^{'}(u_{(1,L)})\sum_{\breve{w}_L=1}^{m_{(L-1)}}\omega_{(1,L,\breve{w}_L)}A^{'}(u_{(\breve{w}_L,L-1)})$$

$$\left(\frac{\partial\Big(\sum_{\breve{w}_{(L-1)}=1}^{m_{(L-2)}}\omega_{(\breve{w}_L,L-1,\breve{w}_{(L-1)})}A(u_{(\breve{w}_{(L-1)},L-2)})\Big)}{\partial\omega_{(r,L-2,k)}}\right)$$

$$\frac{\partial(MSSE)}{\partial\omega_{(r,L-2,k)}} = -\sum_{i=1}^{n}\Big(y_{i,1}-\hat{y}_{i,1}\Big)A^{'}(u_{(1,L)})\sum_{\breve{w}_L=1}^{m_{(L-1)}}\omega_{(1,L,\breve{w}_L)}A^{'}(u_{(\breve{w}_L,L-1)})$$

$$\sum_{\breve{w}_{(L-1)}=1}^{m_{(L-2)}}\omega_{(\breve{w}_L,L-1,\breve{w}_{(L-1)})}\frac{\partial}{\partial\omega_{(r,L-2,k)}}\Big(A(u_{(\breve{w}_{(L-1)},L-2)})\Big)$$

To continue developing this mathematical expression, we apply the chain rule to solve the partial derivative $\dfrac{\partial A(u_{(\breve{w}_{(L-1)},L-2)})}{\partial\omega_{(r,L-2,k)}}$ contained in it:

$$\frac{\partial(MSSE)}{\partial\omega_{(r,L-2,k)}} =$$

$$-\sum_{i=1}^{n}\Big(y_{i,1}-\hat{y}_{i,1}\Big)A^{'}(u_{(1,L)})\sum_{\breve{w}_L=1}^{m_{(L-1)}}\omega_{(1,L,\breve{w}_L)}A^{'}(u_{(\breve{w}_L,L-1)})\sum_{\breve{w}_{(L-1)}=1}^{m_{(L-2)}}\omega_{(\breve{w}_L,L-1,\breve{w}_{(L-1)})}$$

$$\left(\frac{dA(u_{(\breve{w}_{(L-1)},L-2)})}{du_{(\breve{w}_{(L-1)},L-2)}}\right)\left(\frac{\partial du_{(\breve{w}_{(L-1)},L-2)}}{\partial \omega_{(r,L-2,k)}}\right)$$

Now, if the equivalence of $u_{(\breve{w}_{(L-1)},L-2)}$ is substituted given Eq. (3.103), it will be possible to obtain the following:

$$\frac{\partial(MSSE)}{\partial \omega_{(r,L-2,k)}} = -\sum_{i=1}^{n}\left(y_{i,1}-\hat{y}_{i,1}\right)A^{'}(u_{(1,L)})\sum_{\breve{w}_L=1}^{m_{(L-1)}}\omega_{(1,L,\breve{w}_L)}A^{'}(u_{(\breve{w}_L,L-1)})\sum_{\breve{w}_{(L-1)}=1}^{m_{(L-2)}}\omega_{(\breve{w}_L,L-1,\breve{w}_{(L-1)})}$$
$$\left(A^{'}(u_{(\breve{w}_{(L-1)},L-2)})\right)\left(\frac{\partial\left(\sum_{\breve{w}_{(L-2)}=0}^{m_{(L-3)}}\omega_{(\breve{w}_{(L-1)},L-2,\breve{w}_{(L-2)})}A(u_{(\breve{w}_{(L-2)},L-3)})\right)}{\partial \omega_{(r,L-2,k)}}\right) \quad (3.112)$$

Finally, because we are deriving with respect to the currently specified neuron, all the derivations with respect to non specified neurons, through $\breve{w}_{L-1}$, are zero. Subsequently, the same process will occur for all the unspecified weight values, through $\breve{w}_{(L-2)}$ and as a result, Eq. (3.112) will turn into the following:

$$\frac{\partial(MSSE)}{\partial \omega_{(r,L-2,k)}} = -\sum_{i=1}^{n}\left(y_{i,1}-\hat{y}_{i,1}\right)A^{'}(u_{(1,L)})\sum_{\breve{w}_L=1}^{m_{(L-1)}}\omega_{(1,L,\breve{w}_L)}A^{'}(u_{(\breve{w}_L,L-1)})\omega_{(\breve{w}_L,L-1,r)}$$
$$A^{'}(u_{(r,L-2)})\left(A(u_{(k,L-3)})\right) \quad (3.113)$$

Next, by inspecting the tendency of the mathematical expressions of Eq. (3.109); Eq. (3.111) and Eq. (3.113), it is possible to intuitively infer the pattern of the solution for the subsequent layers. Therefore, to solve the derivative of $MSSE$ with respect to an r-th weight located in the c-th layer, we will then have the following:

$$\frac{\partial(MSSE)}{\partial \omega_{(r,c,k)}} = -\sum_{i=1}^{n}\left(y_{i,1}-\hat{y}_{i,1}\right)A^{'}(u_{(1,L)})\sum_{\breve{w}_L=1}^{m_{(L-1)}}\omega_{(1,L,\breve{w}_L)}A^{'}(u_{(\breve{w}_L,L-1)})\sum_{\breve{w}_{(L-1)}=1}^{m_{(L-2)}}\omega_{(\breve{w}_L,L-1,\breve{w}_{(L-1)})}$$
$$A^{'}(u_{(\breve{w}_{(L-1)},L-2)})\cdots\sum_{\breve{w}_{(c+2)}=1}^{m_{(c+1)}}\omega_{(\breve{w}_{(c+2)},c+2,\breve{w}_{(c+2)})}A^{'}(u_{(\breve{w}_{(c+2)},c+1)})\omega_{(\breve{w}_{(c+2)},c+1,r)}A^{'}(u_{(r,c)})A(u_{(k,c-1)}) \quad (3.114)$$

where $1 < c < (L-2)$.

In the same way, to obtain the derivative of $MSSE$ with respect to an r-th weight located

specifically in the first layer, we will then have the following:

$$\frac{\partial(MSSE)}{\partial\omega_{(r,1,k)}} = -\sum_{i=1}^{n}\Big(y_{i,1}-\hat{y}_{i,1}\Big)A^{'}(u_{(1,L)})\sum_{\breve{w}_L=1}^{m_{(L-1)}}\omega_{(1,L,\breve{w}_L)}A^{'}(u_{(\breve{w}_L,L-1)})\sum_{\breve{w}_{(L-1)}=1}^{m_{(L-2)}}\omega_{(\breve{w}_L,L-1,\breve{w}_{(L-1)})}$$
$$A^{'}(u_{(\breve{w}_{(L-1)},L-2)})\cdots\sum_{\breve{w}_3=1}^{m_{(2)}}\omega_{(\breve{w}_4,3,\breve{w}_3)}A^{'}(u_{(\breve{w}_3,2)})\omega_{(\breve{w}_3,2,r)}A^{'}(u_{(r,1)})\widetilde{x}_{(i,k)} \quad (3.115)$$

where $c = 1$.

Since all possible mathematical expressions for the solution of $\frac{\partial(MSSE)}{\partial\omega_{(r,c,k)}}$ are now known, Eq. (3.109); Eq. (3.111) Eq. (3.113); Eq. (3.114) and Eq. (3.115) have to be substituted into Eq. (3.107) correspondingly. Once that is done, the resulting expression obtained for $\Delta\omega_{(r,c,k)}$ has to be substituted into Eq. (3.106) and with it, the current weight k of the neuron located in the row r and column c will be updated. However, these mathematical expressions possess a significant performance problem that can be easily grasped in Eq. (3.115) and that becomes worse the larger L is. This is due to the fact that for each additional whole number added into the value of L, this will contribute with another summing component, that in programming is seen as a for-loop. Consequently, in this thesis a proposal will be made to reduce the impact of the problem described for the L value, but first the effects of Eq. (3.109); Eq. (3.111) Eq. (3.113); Eq. (3.114) and Eq. (3.115) will be explained.

In this regard, Figure 3.17 illustrates a hypothetical example of a specific deep neural network model. This will be used to analyze the interaction of the mathematics obtained so far with the artificial neurons of this model when deriving the MSSE with respect to a certain weight. To begin with, the analysis of what Eq. (3.109) does in the model can be seen in Figure 3.18 for when we want to update the output neuron. The Figure 3.19 illustrates this interaction but when the first neuron of the penultimate layer is to be updated and Figure 3.20 does the same but for the second neuron of that layer. Finally, Figure 3.21 explains the interaction that occurs between the neurons when the neuron to be updated is contained in the antepenultimate layer.

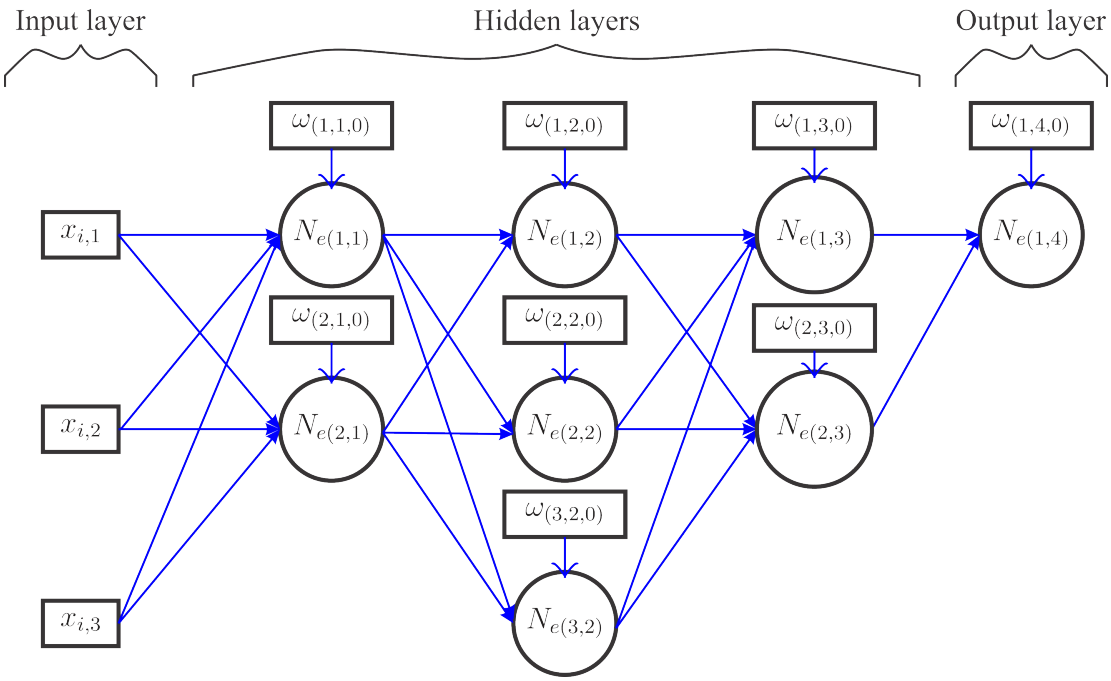


Figure 3.17: Hypothetical example of a deep neural network.

$$\frac{\partial(MSSE)}{\partial\omega_{(1,L,k)}} = -\sum_{i=1}^{n}\Big(y_{i,1} - \hat{y}_{i,1}\Big)A^{'}(u_{(1,L)})A(u_{(k,L-1)})$$

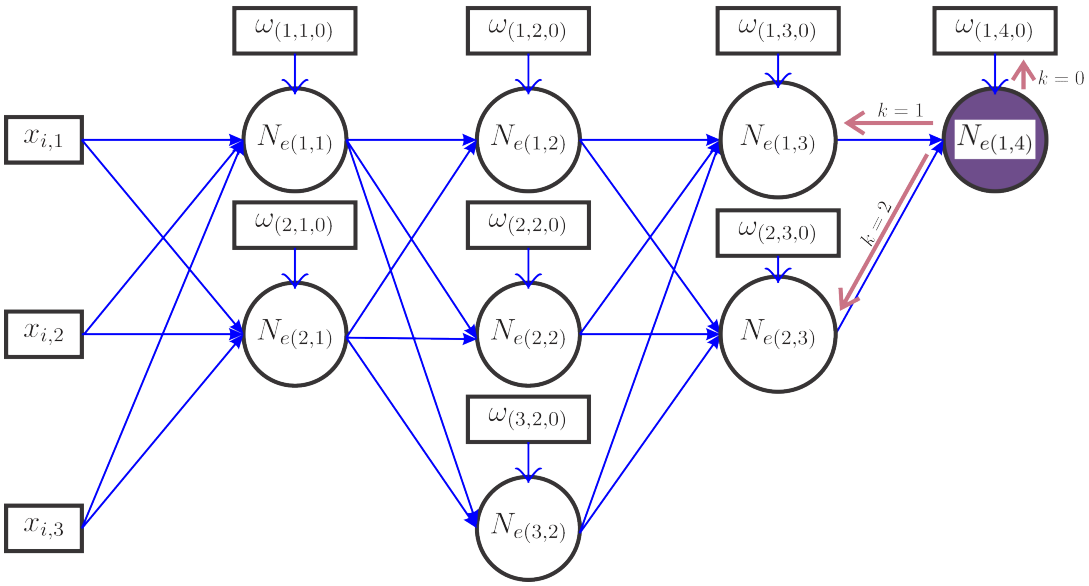


Figure 3.18: Interaction of artificial neurons in a hypothetical model when deriving the MSSE with respect to a particular weight in the last layer.

$$\frac{\partial(MSSE)}{\partial\omega_{(r,L-1,k)}} = -\sum_{i=1}^{n}\Big(y_{i,1}-\hat{y}_{i,1}\Big)A^{'}(u_{(1,L)})\omega_{(1,L,r)}A^{'}(u_{(r,L-1)})A(u_{(k,L-2)})$$

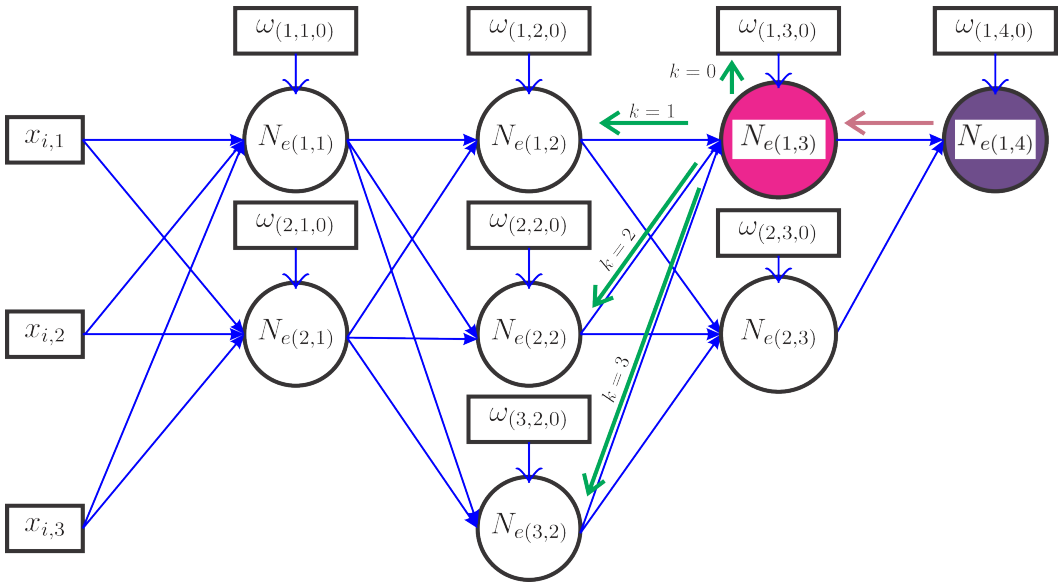


Figure 3.19: Case 1 of interaction of artificial neurons in a hypothetical model when deriving the MSSE with respect to a particular weight in the penultimate layer.

$$\frac{\partial(MSSE)}{\partial\omega_{(r,L-1,k)}} = -\sum_{i=1}^{n}\Big(y_{i,1}-\hat{y}_{i,1}\Big)A^{'}(u_{(1,L)})\omega_{(1,L,r)}A^{'}(u_{(r,L-1)})A(u_{(k,L-2)})$$

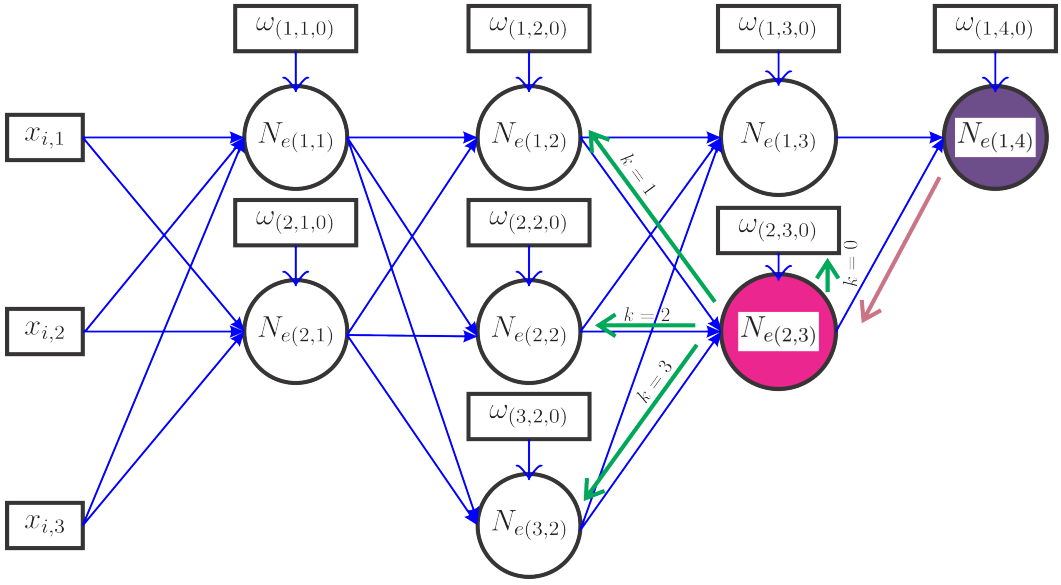


Figure 3.20: Case 2 of interaction of artificial neurons in a hypothetical model when deriving the MSSE with respect to a particular weight in the penultimate layer.

$$\frac{\partial(MSSE)}{\partial\omega_{(r,L-2,k)}} = -\sum_{i=1}^{n}\left(y_{i,1}-\hat{y}_{i,1}\right)$$

$$\mathrm{A}^{'}(u_{(1,L)})\sum_{\breve{w}_L=1}^{m_{(L-1)}}\omega_{(1,L,\breve{w}_L)}A^{'}(u_{(\breve{w}_L,L-1)})\omega_{(\breve{w}_L,L-1,r)}A^{'}(u_{(r,L-2)})A(u_{(k,L-3)})$$

$\breve{w}_L = 1$

$\breve{w}_L = 2$

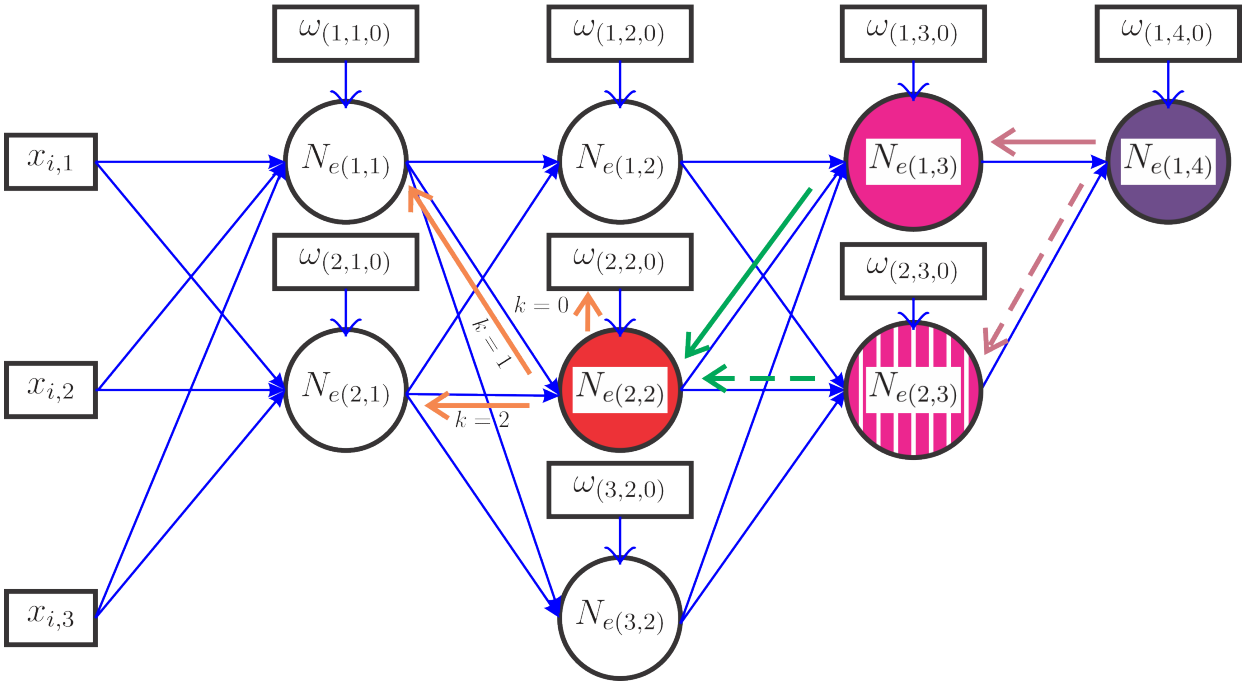


Figure 3.21: Interaction of artificial neurons in a hypothetical model when deriving the MSSE with respect to a particular weight in the antepenultimate layer.

As a result, these observations lead to the conclusion that the error difference is required to be multiplied with other key components. These are the derivative of all the neurons and their inputs that form a path from the output neuron to the where the one that is to be updated is located ($N_{e(r,c)}$). Then, it will be necessary to further multiply this with the corresponding input that is directly connected to the weight to be updated from the neuron $N_{e(r,c)}$. As a consequence of this conclusion, it is now obvious that is is not convenient to follow literally all the summations specified each time a weight is updated. Rather, the proposal of this thesis is to solve Eq. (3.106) from the last layer to the first one by accumulating in the software memory all the necessary multiplications of the paths formed in the subsequent layers with respect to the current one being updated. This way, it will be possible to reduce the impact of having higher values of L by increasing the computer performance through this strategy.

To conclude, as a summary of all the processes that must be performed to train a deep neural network, the Pseudocode 11 lists these steps in an orderly fashion.

Algorithm 11 getDeepNeuralNetwork

Input: $maxEpoch$, λ, $activationFunctionsList$, $desiredAccuracy$, $\boldsymbol{X}$, $\boldsymbol{Y}$, n, $\boldsymbol{m}$, L, $isCustomW$, $customW$

Output: $\boldsymbol{W}_{new}$

fill the matrix $\widetilde{\boldsymbol{X}}$ with the values of $\boldsymbol{X}$ as described in Eq. (3.65)
if $isCustomW =$ false **then**
    initialize $\boldsymbol{W}_{new}$ with random values from -1 to 1.
else
    $\boldsymbol{W}_{new} = customW$
end if
calculate $u_{(r,c)}$ from Eq. (3.103) for all the available neurons by using $\boldsymbol{W}_{new}$
determine what activation function $A(u_{(r,c)})$ has to be used with $activationFunctionsList$ for each neuron
calculate $A(u_{(r,c)})$ and $\dfrac{dA(u_{(r,c)})}{du}$ by using Eq. (3.68) to Eq. (3.89) correspondingly for each neuron
$\hat{\boldsymbol{Y}} = A(u_{(1,L)})$
calculate $currentAccuracy$ of $\hat{\boldsymbol{Y}}$ using your preferred accuracy method
if $currentAccuracy > desiredAccuracy$ **then**
    return $\boldsymbol{W}_{new}$ ▷ Return the latest weight values obtained through a matrix
end if
for $currentEpoch = 1, 2, \cdots, maxEpoch$ **do** ▷ Start the deep neural network training
    $\boldsymbol{W}_{old} = \boldsymbol{W}_{new}$
    recalculate $\boldsymbol{W}_{new}$ with Eq. (3.106) by calculating $\dfrac{\partial(MSSE)}{\partial\omega_{(r,c,k)}}$ for each neuron with Eq. (3.109); Eq. (3.111) Eq. (3.113); Eq. (3.114) and Eq. (3.115) correspondingly.
    recalculate $u_{(r,c)}$ from Eq. (3.103) for all the available neurons by using $\boldsymbol{W}_{new}$
    recalculate $A(u_{(r,c)})$ and $\dfrac{dA(u_{(r,c)})}{du}$ by using Eq. (3.68) to Eq. (3.89) correspondingly for each neuron
    $\hat{\boldsymbol{Y}} = A(u_{(1,L)})$
    calculate $currentAccuracy$ of $\hat{\boldsymbol{Y}}$ using your preferred accuracy method
    if $currentAccuracy > desiredAccuracy$ **then**
        break
    end if
end for
return $\boldsymbol{W}_{new}$ ▷ Return the latest weight values obtained through a matrix

Chapter 4

Methodology

The following content will describe the methodology followed during this thesis project and will also describe the evaluation metrics that were used. In this regard, this process consisted of first performing a literature review to then apply a certain criteria of selection. This allowed the definition of the representative machine learning libraries and some other complementary libraries to be used as a reference. Then, several algorithms were chosen to be developed in the library that was made in this thesis. The decision of which algorithms to select, was based on the algorithms that the representative machine learning libraries have, but also on any areas of opportunity identified in the literature review. Subsequently, these chosen algorithms were described mathematically and implemented in the software using that exact mathematical description. Next, all the developed algorithms were evaluated to have coherent results and then a comparison of their execution times was made with the representative and complementary libraries. Finally, the results of three different application examples of the use of this thesis library in embedded systems are shown as a proof of concept.

4.1 Methods

An analysis of a literature review of machine learning was performed to be able to identify the availability and usefulness of its existing tools. During this research stage, most of the investigations for this matter were conducted from January 2021 to May 2021. In this regard, the first step was to conduct a research to identify and analyze the state of the art contributions that have been made in machine learning. However, this led to the need of studying its background in order to understand several aspects in depth and bring a differentiating value to this thesis. Subsequently, from the data collected, a number of representative libraries were identified and studied to later serve as a comparative reference for the product of this thesis. This last study allowed the identification of the most representative machine learning algorithms, thus helping

in the later determination of which algorithms to choose for the library to be developed. For this identification process, the following list was used as a criteria of selection for the libraries identified, which are shown in Table 2.1:

1. Libraries with an older release date in GitHub that are still under active development as long as they have complete documentation.

2. Libraries that have several machine learning algorithms.

3. Libraries that together have different application purposes (e.g., for big data or embedded systems) or methods (e.g., neural networks).

4. Libraries that together provide several of the features that can be found today as described in Table 2.1.

Note that because the main contributions intended with this thesis are with respect to the machine learning algorithms, no criteria of selection was applied to the selected libraries used in the validation of the statistics; feature scaling and machine learning evaluation algorithms. Instead, the representative machine learning libraries were used to validate the results of those algorithms, if possible. For the cases where there was no representative library to compare with, then some popular and well known Python libraries were randomly selected instead (NumPy, statistics and statsmodels).

Consequently, the stage of investigating and describing the mathematical representation of the selected machine learning algorithms and other useful statistical tools took place from June 2021 to September 2021. This research was based on a machine learning library that lacked scientific formality and that was made by César Miranda Meza before starting this thesis project [46]. However, the methods of that library that were used in this thesis, along with the additional ones provided here, were given due scientific formality during their research, validation and writing process in this document. Finally, the methods described mathematically in the chapter 3; subsection 4.3 and subsection 4.4 were the ones studied and selected to be developed in the library made in this thesis.

Moreover, 8 main databases were created in python during September 2021, where each of them had a different characteristic equation. These were created with the consideration that together were sufficient to provide the expected output pattern for each of the machine learning methods formulated in subsection 3.3. All of the databases were additionally duplicated to have different samples: $n = 10, 100, \cdots, 1000000$ for all models with one independent variable and $n = 100, 1000, \cdots, 1000000$ for all models with two of them. For all these cases, the highest n was proposed for the representative tests that were conducted for the results presented in this thesis, as long as the training process did not take more than 11 seconds. However, this will not apply

to the deep learning algorithms since they were all were strategically tested with $n = 1000000$ because some of them have parallel computing applied. In addition, these databases were all duplicated to have two different versions of the same characteristic equation: 1) a random bias value added to each generated sample and 2) without a random bias value. This allowed all experiments to focus on evaluating the performance and coherence of each of the developed algorithms with respect to the representative machine learning libraries. In conclusion, a total of 88 databases were created for testing any of the machine learning algorithms of this thesis. To obtain either these databases; their documentation; or the codes that generated their data, download the files from the reference [61] in its release version 1.0.0.0.

Subsequently, the development of all the selected algorithms in several processing modes took place from September 2021 to January 2022. To this end, an individual file was created to represent each processing mode: 1) CPU in sequential mode; 2) CPU in parallel mode; 3) Single GPU mode and 4) Multiple GPU mode. The original proposal was to attempt to develop all the chosen algorithms in each of the files of those four processing modes. However, only the file designated for the CPU in sequential mode was concluded with all of those algorithms. In contrast, the other files containing other processing modes have only the algorithms for the single neuron in Deep Neural Network method. As for the steps that were followed for the development of each algorithm in all of these four files, the following illustrates such process:

1. **Development of the chosen algorithm**: The chosen algorithm was developed by giving the highest priority to read and write the data in an aligned and coalesced manner. The second priority was to write a code that had the least amount of assembly instructions. Finally, the third priority was to develop a code that was as lightweight possible.

2. **Validation of the chosen algorithm**: The developed algorithm of step 1 was verified to give consistent results with respect to one of the databases created for this thesis, with and without its random bias component. Then, the results obtained were compared with all the representative libraries that also had the equivalent method. If no representative library was found to also have such equivalent algorithm, the code of this thesis was executed and evaluated with a database that had been made with a strategic characteristic equation, but without a random bias component. In contrast, for the statistics; feature scaling and machine learning evaluation algorithms, they were verified to give consistent results by giving an exact match with an allowable margin of error. Where possible, these comparisons were first made with the representative machine learning libraries and, if not, then with the popular Python libraries that were previously selected instead.

   An additional criteria for selecting an algorithm from a particular library, to use as a comparative reference, is that such algorithm was selected only if that library provided

a programming function for calculating/processing it. In addition, it was also taken into account whether that library had such a method documented and, if not, no algorithm was chosen. This is because, in some cases, it was identified that the online developer community has attempted to contribute by externally providing additional code that allows users to have added functionality that the actual library does not have. Those cases were not considered because it is difficult to guarantee that they will be efficient and follow the programming philosophies of the actual developers of the reference libraries.

3. **Sampling of the execution times of the chosen algorithm**: 30 tests were performed with the chosen database in order to register the value of the execution time of the selected algorithm from step 1. In addition, the mean of those execution times were calculated, as well as their variance and their mean intervals with a confidence value of 99% and a degree of freedom of "tests − 1".

4. **Sampling of the execution times of the reference algorithms**: In sequential processing mode, Step 3 was applied but on the reference libraries that were used in step 2 (NumPy [62], statistics [63], statsmodels [64], BigDL [65], Dlib [42], PyTorch [22], scikit-learn [43] and/or Tensorflow [44]) to validate the algorithm of step 1 and only if they possessed the equivalent algorithm of the first step.

5. **Determining whether or not the chosen algorithm was good enough**: It was determined whether the results of step 3 were good enough with respect to those of step 4, if applicable. In the case that the results were not good enough, the algorithm of step 1 was improved if possible and then step 2 and 3 were repeated.

6. **Obtaining a conclusion for the chosen algorithm**: From the results obtained, it was concluded whether the algorithm of step 1 had worse, equivalent or better execution times with respect to the algorithms of step 4 through the use of the reference [66] as a guideline for it and by comparing the algorithms in their sequential processing mode.

   Note: The validation codes that were made can be found in the reference [67] in its release version 1.0.0.0. There, it is important to observe that the best attempt was made to make the finest implementation with the reference libraries. This was done for fair comparison purposes and, in order to have more reliability in the results obtained, the recommendations of their documentations were followed and the best algorithms identified for such comparisons were selected. On the other hand, this thesis does not describe the code functions or structure of the library that was developed. However, all those details; all the files for this library; and the documentation for the software that was developed,

can be obtained by downloading the files from the reference [68] in its release version 1.0.0.0.

Finally, three different application examples of the use of this thesis library in embedded systems were developed during February 2022. The first of these softwares [69] was developed for an Arduino UNO board and was intended to train and validate a simple linear regression made with this thesis library. This program used the data contained in one of the databases generated in this thesis, which contained 10 samples and was then replicated several times. The purpose of this was to execute the tests with the highest number of samples as possible in order to provide that limitation in this work. On the other hand, the other two softwares [70] were made for a development board with the STM32F446RE microprocessor. In one of them, the same application example that was made for the Arduino UNO board was developed for the STM microprocessor but with 65530 samples. In the other, a single neuron in Deep Neural Network was trained with 1990 samples, where for the latter two cases, the number of samples used was the maximum possible in that device. To conclude, for comparison purposes, these three algorithms with the same number of samples were also implemented in the server in which the library of this thesis was developed.

4.2 Materials

In order to have consistent results, it was determined to run all the evaluation and validation tests under the same conditions. This implied executing all the codes made for this thesis in the same computer system and under the same versions of operative system, external libraries and packages used.

4.2.1 Hardware used

All the algorithms made for this thesis product, were executed on a dedicated computer system with Ubuntu OS v20.04.3 LTS; gcc v9.3.0 and nvcc v11.4, with the following hardware:

1. Motherboard: 1 x HUANANZHI X99Dual-F8D [71].
2. CPU: 2 x Intel(R) Xeon(R) E5-2699V4 @ 2.10GHz [72].
3. RAM: 1 x SAMSUNG M386A4G40EM2-CRC [73].
4. Storage device: 1 x Samsung SSD 970 EVO plus [74].
5. GPU: 1 x GeForce GTX 1660 SUPER [75] (connected through Timack 20cm riser PCIe extension cable model B08BR7NB3W [76]).

6. GPU: 4 x Tesla K80 [77] (two of the four physical GPUs are contained in a single enclosure that has one PCI-E connector. In other words, two Tesla K80 GPUs where each one of them has two GPUs inside, making a total of four).

where the CUDA files were executed only in the Tesla K80 GPUs, which were entirely dedicated to this thesis.

Note: Subsection 8.1 of the Annexes contains additional technical information of the enlisted hardware.

4.2.2 External libraries and packages used

The following list will detail the version of the external libraries and packages that were used:

- **For Python:**
  - pip version 21.2.4
  - TensorFlow version 2.7.0
  - scikit-learn version 1.0.1
  - numpy version 1.21.4
  - matplotlib version 3.4.3
  - pandas version 1.3.3
  - Dlib version 19.22
  - statsmodels version 0.13.1
  - spyder version 5.2.0
  - cmake version 3.16.3
- **For C:**
  - pbPlots version 0.1.9.0

4.3 Evaluation metrics for regression problems

In machine learning, a very important issue after generating a model is to measure how well does it fits the training or test data. However, this is somewhat complicated because there are several ways to do this through different metrics, whose best one is difficult to determine. Consequently, this leaves the option to the practitioner to use the ones he/she considers best

for his/her particular application. Therefore, the following metrics provide several options for the machine learning practitioner to evaluate regression models and that are also available in the developed library for this thesis.

4.3.1 Mean squared error

The Mean Squared Error (MSE) represents a means of measuring the average of the squared errors of the predicted values from a given machine learning method with respect to the real values, and is given by the following [53]:

$$MSE = \frac{SSE}{n - q} \tag{4.1}$$

where q represents the degrees of freedom of this equation (it is suggested that $q = 2$ [53]) and $SEE = \sum_{i=1}^{n}(y_{i,1} - \hat{y}_{i,1})^2$, according to the Eq. (3.30) and given that $p = 1$.

In other words, this technique has conveniently been used as a way to measure how much error does a certain regression model have. However, the Eq. (4.1) is valid only when it is desired to measure the error of a system that has only one feature or independent variable. Therefore, when the phenomenon being studied has several features, the following should be used instead [53]:

$$MSE = \frac{SSE}{n - m - q} \tag{4.2}$$

where it is suggested that $q = 1$ [53] and $SEE = \sum_{i=1}^{n}(y_{i,1} - \hat{y}_{i,1})^2$, according to the Eq. (3.30) and given that $p = 1$.

4.3.2 Coefficient of determination

The coefficient of determination, also denoted as R^2, is a means of measuring how well a machine learning model fits with respect to the variance of a certain data set, and it is given by the following [53]:

$$R^2 = 1 - \frac{SSE}{SST} \quad | \quad R^2 \leq 1 \tag{4.3}$$

where, according to the Eq. (3.30) and given that $p = 1$, $SSE = \sum_{i=1}^{n}(y_{i,1} - \hat{y}_{i,1})^2$ and the Corrected Sum of Squares (SST) $= \sum_{i=1}^{n}(y_{i,1} - \bar{y}_1)^2$. In addition, the coefficient of determination has the convenient advantage that its output values will normally range from 0 to 1 if the model

can actually explain the data set, unlike the MSE method of the Eq. (4.1) and Eq. (4.2). However, this method is appropriate for when only one feature or independent variable of the system under study is available.

4.3.3 Adjusted coefficient of determination

The adjusted coefficient of determination, also denoted as R^2_{adj}, does exactly the same as R^2 from the Eq. (4.3) but with the difference that R^2_{adj} penalizes the model when there are several independent variables in it, and is given by the following [53]:

$$R^2_{adj} = 1 - \frac{SSE/(n-m-q)}{SST/(n-q)} \quad | \quad R^2_{adj} \leq 1 \tag{4.4}$$

where, it is suggested that $q = 1$ [53]; according to the Eq. (3.30) and given that $p = 1$, $SSE = \sum_{i=1}^{n}(y_{i,1} - \hat{y}_{i,1})^2$ and $SST = \sum_{i=1}^{n}(y_{i,1} - \bar{y}_1)^2$. In addition, this method should be preferred over the Eq. (4.3) when evaluating various machine learning features.

4.4 Evaluation metrics for classification problems

The following metrics provide several options for the machine learning practitioner to evaluate classification models, which are also available in the developed library for this thesis.

4.4.1 Cross entropy error function

The cross entropy error function, also denoted as NLL, is a means of measuring the error of a machine learning model in a manner similar to MSE, from the Eq. (4.1) and Eq. (4.2), and is given by the following [78]:

$$NLL = -\sum_{i=1}^{n}\Big(y_{i,1}\ln\hat{y}_{i,1} + (1-y_{i,1})\ln(1-\hat{y}_{i,1})\Big) \tag{4.5}$$

where it is expected that $p = 1$ and both $y_{i,1}$ and $\hat{y}_{i,1}$ must never be exactly 0 or 1 to avoid the invalid mathematical computation of $\ln 0$.

Although both the MSE and the NLL are intended to measure the error of a model, they differ from each other because the NLL penalizes more heavily when a binary output data is not well categorized. Therefore, this method is usually applied when measuring the error of a classification machine learning model, rather than the MSE. However, it is important not to

misunderstand that this does not necessarily mean that the MSE cannot give good results for classification problems.

4.4.2 Confusion matrix

Since there are many real life situations where databases will have a lot of data, it will be difficult to know how many times the model is being correct in its predictions. Fortunately, there is something known as a confusion matrix that allows the machine learning practitioner to obtain this information in a very simple way, as seen in the following [78]:

Table 4.1: Confusion matrix data distribution.

		Truth	
		1	0
Prediction	1	TP	FP
	0	FN	TN

where TP = "True Positives"; FP = "False Positives"; FN = "False Negatives" and TN="True Negatives". In more detail, TP will be equal to the number of times the model correctly predicted the 1 binary value and FP when the prediction of 1 was incorrect. Furthermore, TN will equal to the number of times the model correctly predicted the 0 binary value and FN when the prediction of 0 was incorrect.

4.4.3 Accuracy

The generic accuracy of a model is a mathematical method that explains the proportion between all the predicted values that were correct with respect to the total number of predicted values, and it can be determined as follows [2]:

$$A = \frac{TP + TN}{TP + FP + FN + TN} = \frac{TP + TN}{n} \tag{4.6}$$

4.4.4 Precision

The mathematical process used to find out how many of the predicted binary output values of 1 were correct with respect to all the predicted values of 1, can be obtained with the precision

P method and is given by the following [78]:

$$P = \frac{\sum_{i=1}^{n} \Big((y_{i,1})(\hat{y}_{i,1}) \Big)}{\sum_{i=1}^{n} \hat{y}_{i,1}} \tag{4.7}$$

4.4.5 Recall

The mathematical process used to find out how many of the predicted binary output values of 1 were correct with respect to the real values of 1, can be obtained with the recall R method and is given by the following [78]:

$$R = \frac{\sum_{i=1}^{n} \Big((y_{i,1})(\hat{y}_{i,1}) \Big)}{\sum_{i=1}^{n} y_{i,1}} \tag{4.8}$$

4.4.6 F1 score

Besides the measurements possible with the Eq. (4.7) and Eq. (4.8), there is another alternative widely used in information retrieval systems. This method combines the precision and recall methods of those equations into a single statistic through their harmonic mean, which is known as F1 score (F_1) and is given by the following [78]:

$$F_1 = \frac{2PR}{P + R} \tag{4.9}$$

4.5 Evaluation of underfitting and overfitting in machine learning models

Although several evaluation metrics have already been discussed, there is a possibility that these give very good or perfect scores, even when such models are not good in reality. Such cases can occur when the models exhibit a behavior known as overfitting during their training process [2, 1], as illustrated in the Figure 4.1. Here, the model has the characteristic of fitting extremely well to the training set, but when testing it with other datasets, its performance is poor and it does not generalize the test data well [2, 1]. On the other hand, when a model is poorly trained; does not have enough features; or does not have enough model complexity, it will manifest a

phenomenon called underfitting [2, 1]. Finally, a good fitting is identified when the model is able to capture the pattern of the training set so that there is a slight difference between the training error and the test error when comparing underfit and overfit situations. Nonetheless, in order to better detect underfitting; good fitting; or overfitting, a visual assessment by the machine learning practitioner and a consideration of the symptoms described in the Figure 4.1 will be necessary [2, 1].

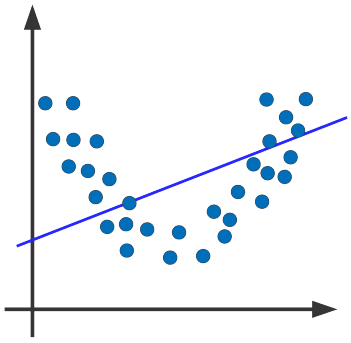
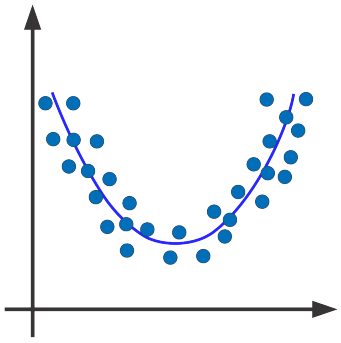
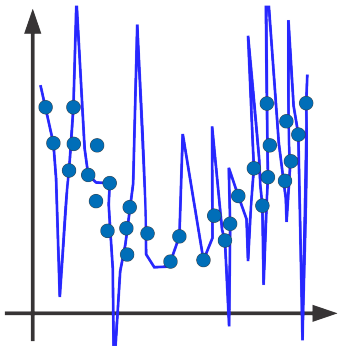
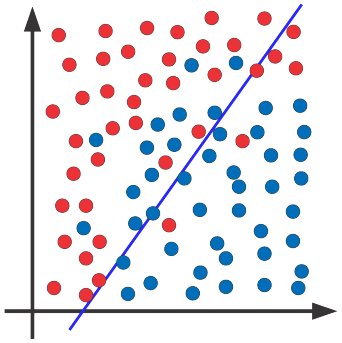
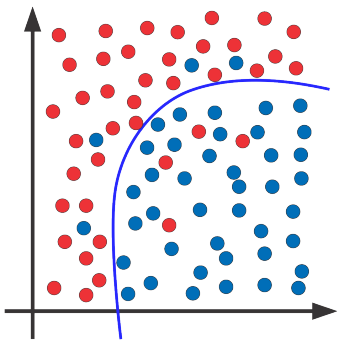
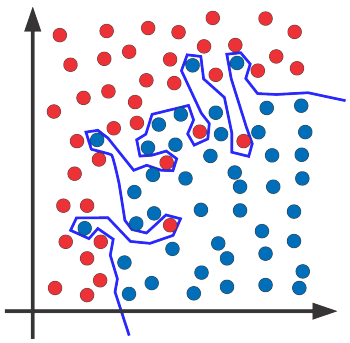
	Underfitting	Good fitting	Overfitting
Symptoms	• High error with training data. • Model curve seems limited to capture the pattern of the training set. • High bias.	• The training error is slightly different than the test error.	• Very low training error but does not generalize the test data well. • High variance.
Regression examples			
Classification examples			
Possible solutions	• Train the model more time. • Add more complexity to the model. • Add more features to the model.		• Increase data samples. • Reduce the complexity of the model. • Reduce the number of features.

Figure 4.1: Hypothetical examples of regression and classification illustrating cases of underfitting; good fitting and overfitting.

Chapter 5

Results

The theoretical framework of this work in Chapter 2 provided answers to the four research questions of this thesis. The first two questions were answered with Table 2.1 wich not only outlines the libraries that support parallel computing but also embedded systems. On the other hand, Table 2.2 lists and details the most representative machine learning libraries identified in this work and thus answers the third research question. Regarding the last research question, it has been determined that the main contributions that have been made in the regression and classification methods are listed in Table 2.2 with respect to the representative libraries. Hence, all this responds to the research questions proposed in this thesis and, in the following, the hypothesis of this work will be answered through the results obtained.

In this regard, this Chapter will focus on addressing the results obtained with respect to four types of algorithms that were developed: 1) Statistical functions; 2) Feature scaling algorithms; 3) Machine learning evaluation metric functions; and 4) Machine learning algorithms. Then, it will conclude by showing the results obtained for three different application examples of the use of embedded systems with the library that has been developed for this thesis. Regarding the results of the library developed, detailed information about the execution times and validation results of all its algorithms will be shown, together with comparison results with the chosen reference libraries. In addition, for all their figures and tables, the mathematical symbols used were described in Chapter 3 and the error bars will stand for the mean intervals which have a confidence value of 99% and a degree of freedom of "tests-1", where "tests $= 30$". Furthermore, all data boxes marked with † will mean that the method of interest was not investigated to be in existence and "—" will indicate that such data box is not applicable. Conversely, if a data box is left blank, this will represent that no method was identified for it. Finally, all the improvements registered were calculated by dividing the fastest execution time of the comparison libraries with the implementation carried out in this thesis correspondingly.

5.1 Statistical functions

The statistical algorithms that were developed and implemented as software functions in the library of that was created for this thesis are the following:

- The mean.
- The median.
- The mode.
- A sort function.
- The standard deviation.
- The variance.

whose mean execution times and mean intervals, both for the library of this thesis and for the comparison libraries, are shown in the bar charts of Figures 5.1 and 5.2. These two Figures contain the results of all the statistical algorithms together, but were separated to better visualize the differences in the execution times obtained. If more data is needed, Tables 5.1 and 5.2 present the information of the bar charts with more detail.

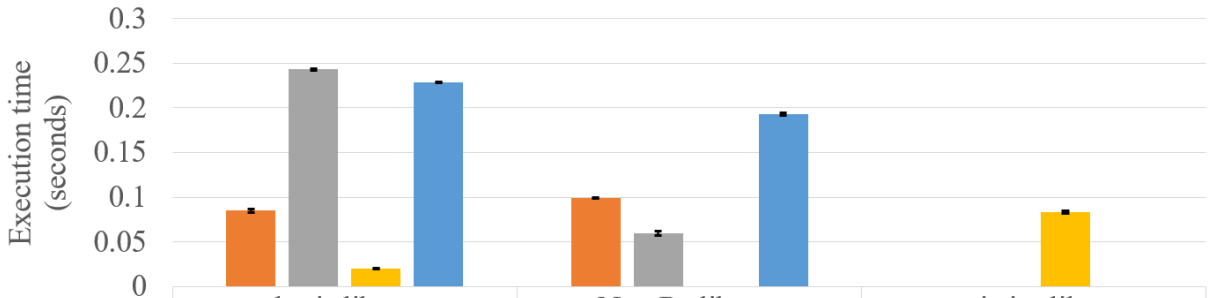


Figure 5.1: Bar chart of the execution times for the mean; median; mode and quick sort algorithms that were developed in sequential processing mode and compared with the NumPy and statistics library.

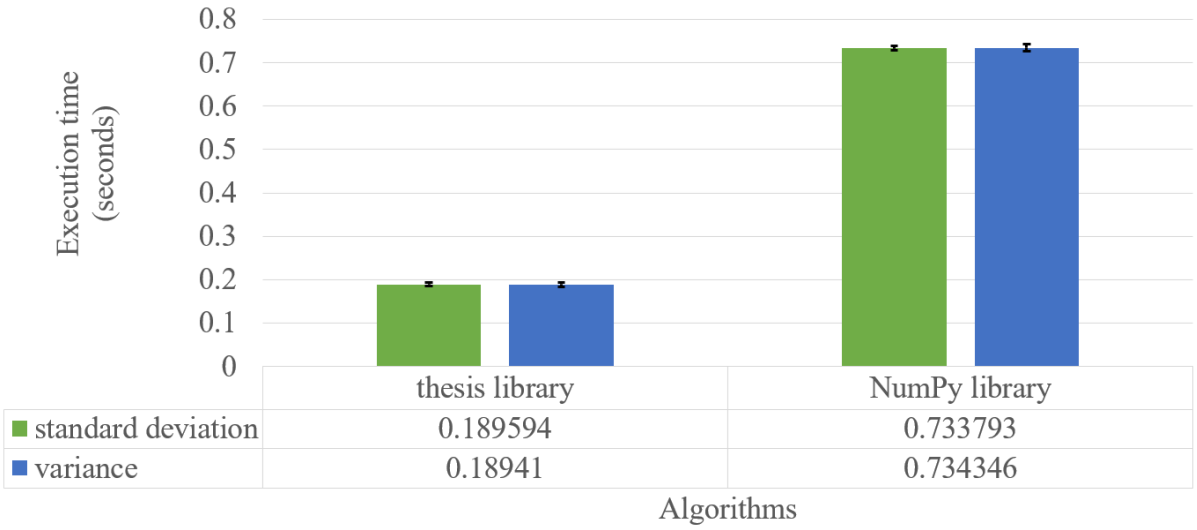


Figure 5.2: Bar chart of the execution times for the standard deviation and variance algorithms that were developed in sequential processing mode and compared with the NumPy library.

Table 5.1: Mean and data dispersion of the execution times measured for the statistical algorithms that were made and used in this thesis.

Algorithm	**Execution time (seconds)**			
	$\hat{\theta}_L$	$\bar{x}$	$\hat{\theta}_U$	s
mean_thesis	0.08278	0.0849	0.087019	0.004506
mean_Numpy	0.098231	0.098777	0.099322	0.001161
mean_statistics	†	†	†	†
median_thesis	0.241792	0.242978	0.244164	0.002522
median_Numpy	0.057626	0.059894	0.062162	0.004822
median_statistics	†	†	†	†
quickMode_thesis	0.019183	0.019946	0.020709	0.001622
quickMode_Numpy				
mode_statistics	0.081761	0.083061	0.08436	0.002763
quickSort_thesis	0.227577	0.228444	0.229311	0.001843
quickSort_Numpy	0.191227	0.192728	0.194229	0.003192
sort_statistics	†	†	†	†
standardDeviation_thesis	0.18494	0.189594	0.194248	0.009895
standardDeviation_Numpy	0.728195	0.733793	0.739391	0.011904
standardDeviation_statistics	†	†	†	†
variance_thesis	0.184335	0.18941	0.194485	0.010791
variance_Numpy	0.726142	0.734346	0.74255	0.017443
variance_statistics	†	†	†	†
95meanIntervals_thesis	0.013521	0.013837	0.014153	0.000672
99meanIntervals_thesis	0.013185	0.013543	0.013902	0.000763
99_9meanIntervals_thesis	0.013211	0.013578	0.013945	0.00078

The results obtained for all the statistical algorithms that were developed concluded being reliable because they matched the results obtained with the comparison libraries whose margin of errors allowed are indicated in Table 5.2. Moreover, Table 5.3 summarizes the improvements achieved for each algorithm with respect to the mean execution times obtained. In it, the mean; the mode; the standard deviation and the variance were found to have higher performance than the comparison libraries that were taken into account. However, the median and the sort algorithms that were developed in this thesis did not provide with faster computation results. This somewhat contradicts the initial hypothesis but only in the aspect of having increased the performance of every single algorithm.

Table 5.2: Validation and reliability results obtained for the statistical algorithms that were made and used in this thesis.

Algorithm	n	$\eta_{(tests)}$ $(e_k = \bar{x} - \hat{\theta}_L)$	**Comparison of results**	
			Did they matched?	**Error allowed**
mean_thesis	100000000	30	YES	9.20E-08
mean_Numpy	100000000	31	YES	9.20E-08
mean_statistics	†	†	†	†
median_thesis	5000000	31	YES	3.01E-07
median_NumPy	5000000	30	YES	3.01E-07
median_statistics	†	†	†	†
quickMode_thesis	500000	30	YES	0
quickMode_NumPy				
mode_statistics	500000	30	YES	0
quickSort_thesis	4000000	30	YES	1.00E-06
quickSort_NumPy	4000000	31	YES	1.00E-06
sort_statistics	†	†	†	†
standardDeviation_thesis	100000000	30	YES	4.99E-07
standardDeviation_NumPy	100000000	31	YES	4.99E-07
standardDeviation_statistics	†	†	†	†
variance_thesis	100000000	31	YES	2.51E-07
variance_NumPy	100000000	30	YES	2.51E-07
variance_statistics	†	†	†	†
95meanIntervals_thesis[1]	5000000	31	YES	9.50E-03
99meanIntervals_thesis[1]	5000000	31	YES	1.60E-01
99_9meanIntervals_thesis[1]	5000000	30	YES	4.80E-02

[1]This algorithm was validated with an equivalent function applied with the "Data Analysis" tool provided by Microsoft Excel because no library with such a method was identified within the available research time.

Table 5.3: Improvements obtained for all the statistical algorithms developed with respect to the mean execution times obtained.

Algorithm	**This thesis (seconds)**	**Comparison library (seconds)**	**Improvement**
Mean	0.0849	0.098777	1.163451
Median	0.242978	0.059894	0.2465
Mode	0.019946	0.083061	4.164294
Sort	0.228444	0.192728	0.843655
Standard deviation	0.189594	0.733793	3.870339
Variance	0.18941	0.734346	3.877018
95% mean intervals	0.013837		—
99% mean intervals	0.013543		—
99.9% mean intervals	0.013578		—

5.2 Feature scaling functions for machine learning

The feature scaling algorithms that were developed with their get and reverse functionalities as software functions, in the library that was created for this thesis, are the following:

- The L2 normalization.
- The min max normalization.
- The Z score normalization (standardization).

of which the bar chart of Figure 5.3 shows the mean execution times and mean intervals for all these algorithms and for ones that were compared with. Should more data be needed, Tables 5.4 and 5.5 give the information of the bar chart with more detail.

Regarding the results obtained, the developed feature scaling algorithms were reliable because the validation results coincided with the comparison library used, of which the permitted margin of errors are indicated in Table 5.5. On the other hand, Table 5.6 outlines the improvements achieved for each developed algorithm with respect to the mean execution times that were registered. In it, the get of the L2 normalization; the min max normalization and the Z score normalization turned out to have superior performance to the comparison library taken into account. Consequently, all these algorithms fully confirmed the initial hypothesis, as they were reliable and also increased the performance with respect to the reference library used.

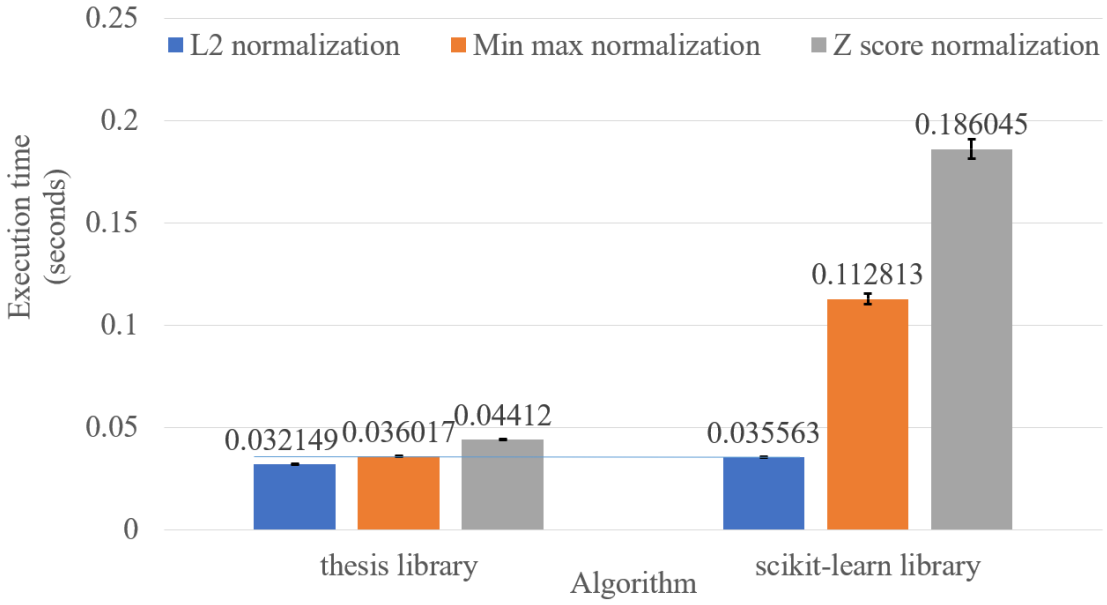


Figure 5.3: Bar chart of the execution times for the L2 normalization; min max normalization and Z score normalization algorithms that were developed in sequential processing mode and compared with the scikit-learn library.

Table 5.4: Mean and data dispersion of the execution times measured for the feature scaling algorithms that were made and used in this thesis.

Algorithm	**Execution time (seconds)**			
	$\hat{\theta}_L$	$\bar{x}$	$\hat{\theta}_U$	s
getL2Normalization_thesis	0.031843	0.032149	0.032456	0.000651
getL2Normalization_scikitLearn	0.035193	0.035563	0.035933	0.000787
getMinMaxNormalization_thesis	0.035721	0.036017	0.036313	0.00063
getMinMaxNormalization_scikitLearn	0.110194	0.112813	0.115432	0.005569
getZscoreNormalization_thesis	0.043794	0.04412	0.044446	0.000694
getZscoreNormalization_scikitLearn	0.181195	0.186045	0.190895	0.010313
reverseL2Normalization_thesis	0.024132	0.024345	0.024559	0.000454
reverseMinMaxNormalization_thesis	0.023504	0.023771	0.024038	0.000568
reverseZscoreNormalization_thesis	0.024738	0.025018	0.025298	0.000595

Table 5.5: Validation and reliability results obtained for the feature scaling algorithms that were made and used in this thesis.

Algorithm	n	$\eta_{(tests)}$ $(e_k = \bar{x} - \hat{\theta}_L)$	**Comparison of results**	
			Did they matched?	**Error allowed**
getL2Normalization_thesis	5000000	31	YES	5.00E-07
getL2Normalization_scikitLearn	5000000	31	YES	5.00E-07
getMinMaxNormalization_thesis	5000000	31	YES	5.00E-07
getMinMaxNormalization_scikitLearn	5000000	31	YES	5.00E-07
getZscoreNormalization_thesis	5000000	31	YES	5.00E-07
getZscoreNormalization_scikitLearn	5000000	31	YES	5.00E-07
reverseL2Normalization_thesis	5000000	31	YES	1.00E-09
reverseMinMaxNormalization_thesis	5000000	31	YES	1.00E-10
reverseZscoreNormalization_thesis	5000000	30	YES	1.00E-10

Table 5.6: Improvements obtained for all the feature scaling algorithms developed with respect to the mean execution times obtained.

Algorithm	**This thesis (seconds)**	**Comparison library (seconds)**	**Improvement**
getL2Normalization	0.032149	0.035563	1.106193
getMinMaxNormalization	0.036017	0.112813	3.132215
getZscoreNormalization	0.04412	0.186045	4.216795
reverseL2Normalization	0.024345	†	—
reverseMinMaxNormalization	0.023771	†	—
reverseZscoreNormalization	0.025018	†	—

5.3 Evaluation metric functions for machine learning algorithms

The evaluation metric algorithms that were developed as software functions for regression and classification models in the library that was created for this thesis are the following:

- For regression models:
  - R-squared.
  - Adjusted R-squared.
  - Mean squared error.
- For classification models:
  - Accuracy.

- Confusion matrix.
- Cross entropy error.
- F1 score.
- Precision.
- Recall.

for which the mean execution times and mean intervals are shown in the bar charts of Figures 5.4 and 5.5 for both the library of this thesis and for the comparison libraries used. The first figure separates the evaluation metrics used for regression models and the second for classification models to better visualize the differences in the obtained execution times. Complementarily, Tables 5.7 and 5.8 provide more details of the tests that were made for the evaluation and validation of the implementation of such algorithms.

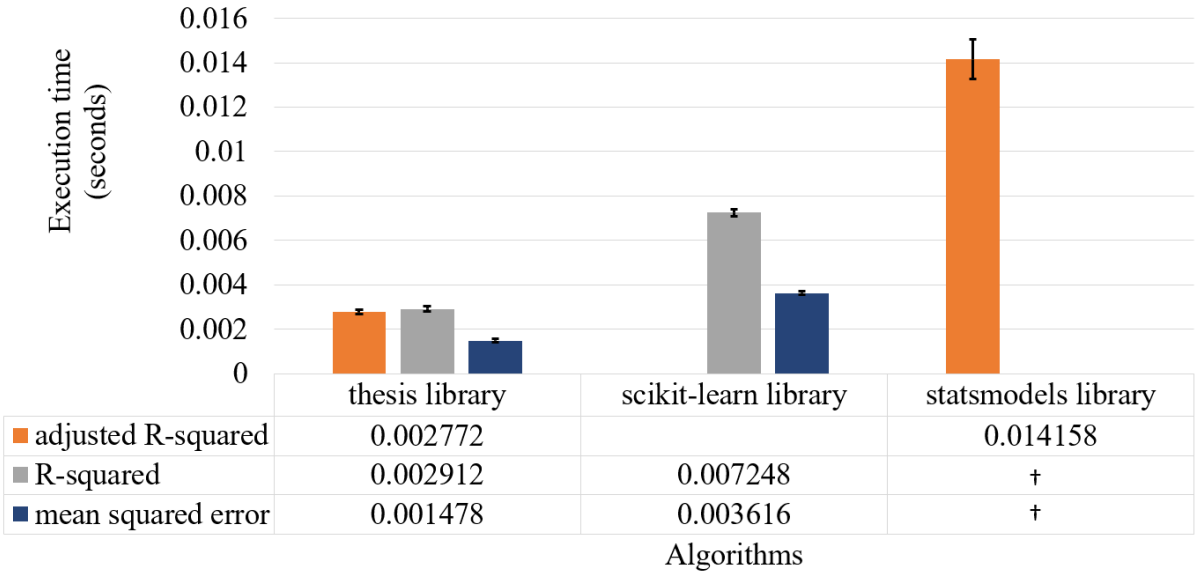


Figure 5.4: Bar chart of the execution times for the algorithms of the regression evaluation metrics that were developed in sequential processing mode and compared with the scikit-learn and statsmodels libraries.

On the other hand, the results obtained for all the evaluation metric algorithms that were developed, concluded being reliable because they matched the validation results obtained with the comparison libraries whose margin of errors allowed are indicated in Table 5.8. Moreover, Table 5.9 summarizes the improvements achieved for each algorithm with respect to their registered mean execution times. In it, all the evaluation metric algorithms developed in this thesis were found to have higher performance than the comparison libraries that were taken into account. Therefore, the results obtained with all these algorithms confirmed the initial hypothesis.

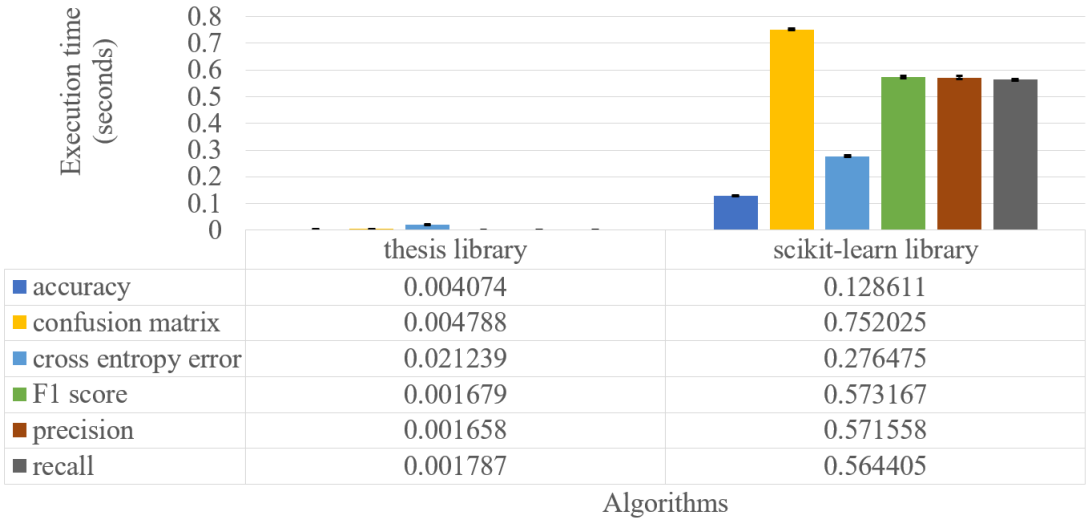


Figure 5.5: Bar chart of the execution times for the algorithms of the classification evaluation metrics that were developed in sequential processing mode and compared with the scikit-learn library.

Table 5.7: Mean and data dispersion of the execution times measured for the machine learning evaluation metric algorithms that were made and used in this thesis.

Algorithm	**Execution time (seconds)**			
	$\hat{\theta}_L$	$\bar{x}$	$\hat{\theta}_U$	s
accuracy_thesis	0.004004	0.004074	0.004145	0.000151
accuracy_scikitLearn	0.126995	0.128611	0.130228	0.003437
statsmodels	†	†	†	†
adjRsquared_thesis	0.002675	0.002772	0.00287	0.000207
scikitLearn				
adjRsquared_statsmodels	0.013263	0.014158	0.015054	0.001904
Rsquared_thesis	0.002802	0.002912	0.003021	0.000233
Rsquared_scikitLearn	0.007098	0.007248	0.007398	0.000319
statsmodels	†	†	†	†
confusionMatrix_thesis	0.004728	0.004788	0.004848	0.000127
confusionMatrix_scikitLearn	0.749411	0.752025	0.754639	0.005558
statsmodels	†	†	†	†
crossEntropyError_thesis	0.021064	0.021239	0.021414	0.000372
crossEntropyError_scikitLearn	0.273542	0.276475	0.279408	0.006236
statsmodels	†	†	†	†
F1score_thesis	0.001617	0.001679	0.001741	0.000132
F1score_scikitLearn	0.568123	0.573167	0.578211	0.010725
statsmodels	†	†	†	†
meanSquaredError_thesis	0.001397	0.001478	0.001559	0.000172
meanSquaredError_scikitLearn	0.003535	0.003616	0.003697	0.000172
statsmodels	†	†	†	†
precision_thesis	0.001574	0.001658	0.001742	0.000178
precision_scikitLearn	0.565762	0.571558	0.577354	0.012324
statsmodels	†	†	†	†
recall_thesis	0.001683	0.001787	0.001891	0.000221
recall_scikitLearn	0.561185	0.564405	0.567624	0.006845
statsmodels	†	†	†	†

Table 5.8: Validation and reliability results obtained for the machine learning evaluation metric algorithms that were made and used in this thesis.

Algorithm	n	$\eta_{(tests)}$ $(e_k = \bar{x} - \hat{\theta}_L)$	**Comparison of results**	
			Did they matched?	**Error allowed**
accuracy_thesis	1000000	31	YES	0
accuracy_scikitLearn	1000000	31	YES	0
statsmodels	†	†	†	†
adjRsquared_thesis	1000000	31	YES	1.00E-14
scikitlearn				
adjRsquared_statsmodels	1000000	31	YES	1.00E-14
Rsquared_thesis	1000000	30	YES	3.45E-08
Rsquared_scikitLearn	1000000	31	YES	3.45E-08
statsmodels	†	†	†	†
confusionMatrix_thesis	1000000	30	YES	0
confusionMatrix_scikitLearn	1000000	30	YES	0
statsmodels	†	†	†	†
crossEntropyError_thesis	1000000	30	YES	4.04E-06
crossEntropyError_scikitLearn	1000000	30	YES	4.04E-06
statsmodels	†	†	†	†
F1score_thesis	1000000	31	YES	3.88E-07
F1score_scikitLearn	1000000	31	YES	3.88E-07
statsmodels	†	†	†	†
meanSquaredError_thesis	1000000	30	YES	3.66E-07
meanSquaredError_scikitLearn	1000000	30	YES	3.66E-07
statsmodels	†	†	†	†
precision_thesis	1000000	30	YES	3.17E-07
precision_scikitLearn	1000000	31	YES	3.17E-07
statsmodels	†	†	†	†
recall_thesis	1000000	30	YES	1.13E-07
recall_scikitLearn	1000000	30	YES	1.13E-07
statsmodels	†	†	†	†

Table 5.9: Improvements obtained for all the machine learning evaluation metric algorithms developed with respect to the mean execution times obtained.

Algorithm	**This thesis (seconds)**	**Comparison library (seconds)**	**Improvement**
Accuracy	0.004074	0.128611	31.56873
Adjusted R-squared	0.002772	0.014158	5.107504
R-squared	0.002912	0.007248	2.489011
Confusion matrix	0.004788	0.752025	157.0645
Cross entropy error	0.021239	0.276475	13.01733
F1 score	0.001679	0.573167	341.374
Mean squared error	0.001478	0.003616	2.446549
Precision	0.001658	0.571558	344.7274
Recall	0.001787	0.564405	315.8394

5.4 Machine learning functions

Up to now, all the results shown describe the complementary tools that were made in this thesis and that are being used in machine learning. In what follows, the results obtained for the developed machine learning algorithms will be presented, as well as their performance with respect to the currently available options. In this regard, the first results achieved corresponded to all the traditional regression algorithms that were developed. Subsequently, some traditional classification algorithms were made and subjected to a testing process. Finally, this work concluded by doing the same for the deep learning algorithms that were considered.

5.4.1 Regression algorithms

The regression algorithms that were developed with both their get and predict functionalities as software functions in the library that was created for this thesis are the following:

- Simple linear regression.
- Multiple linear regression.
- Polynomial regression.
- Multiple polynomial regression.
- Logistic regression.
- Gaussian regression.

for which the mean execution times and mean intervals are shown in the bar charts of Figures 5.6, 5.7 and 5.8 for both the library of this thesis and for the comparison library used. These Figures are representative only of the algorithms developed in this thesis and that were compared to an equivalent algorithm. Those algorithms are the simple linear regression, the multiple linear regression and the Gaussian regression. As for the ones that were not shown in bar graphs, it was simply because an equivalent algorithm was not found with respect to the available research time. Furthermore, apart from not having identified such algorithms within the representative machine learning libraries, no explicit description of them was found in their documentations. Moreover, Tables 5.10 and 5.11 provide more details of the tests that were made for the evaluation and validation of the implementation of all the algorithms considered.

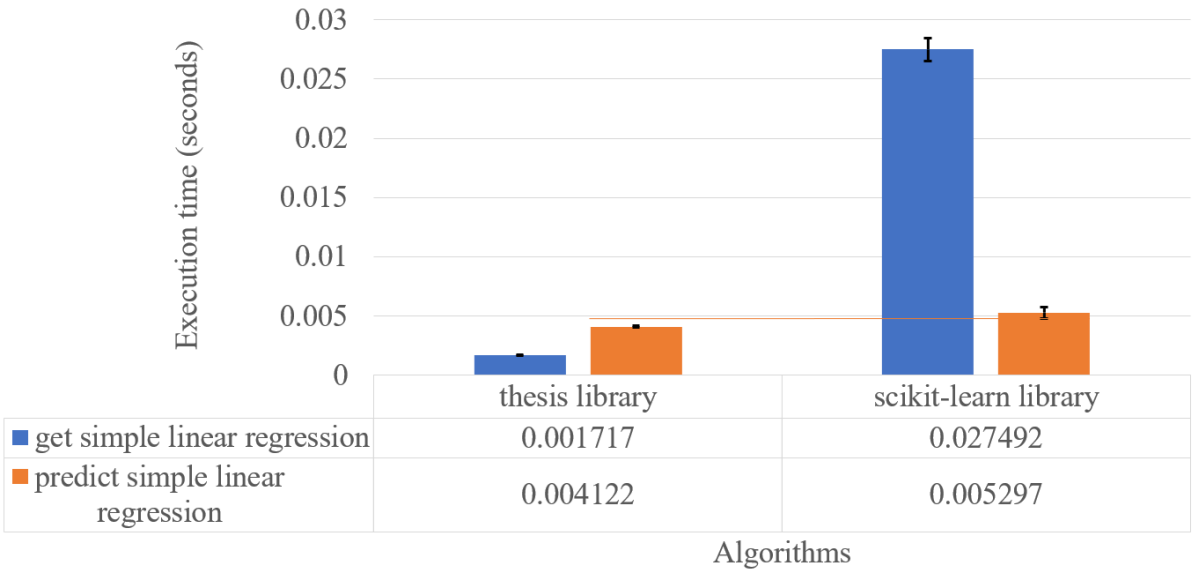


Figure 5.6: Bar chart of the execution times for the get and predict algorithms of the simple linear regression that was developed in sequential processing mode and compared with the scikit-learn library.

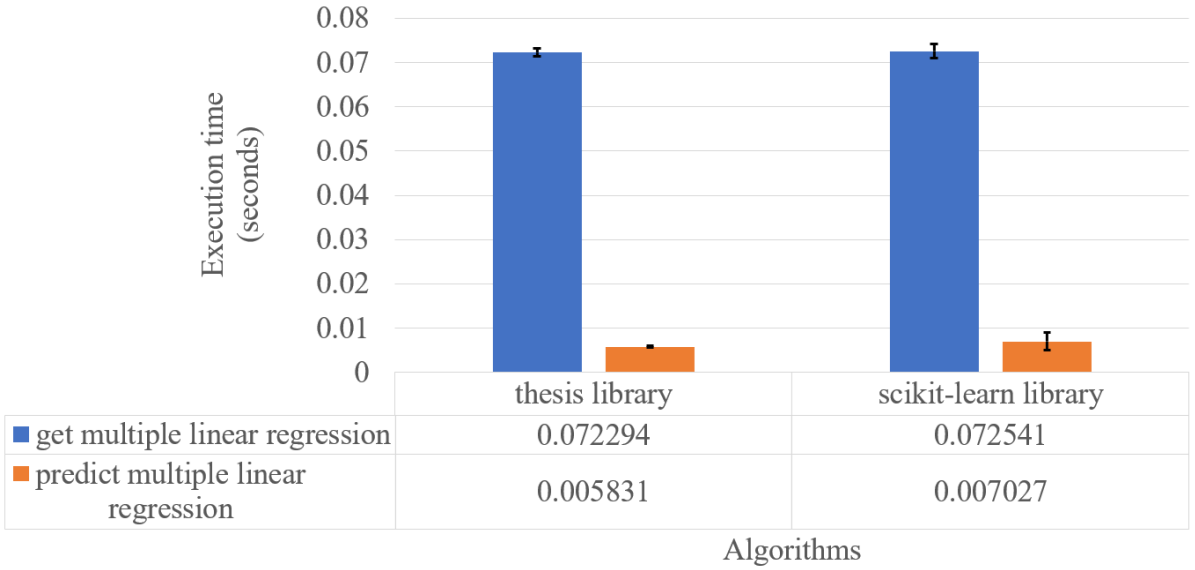


Figure 5.7: Bar chart of the execution times for the get and predict algorithms of the multiple linear regression that was developed in sequential processing mode and compared with the scikit-learn library.

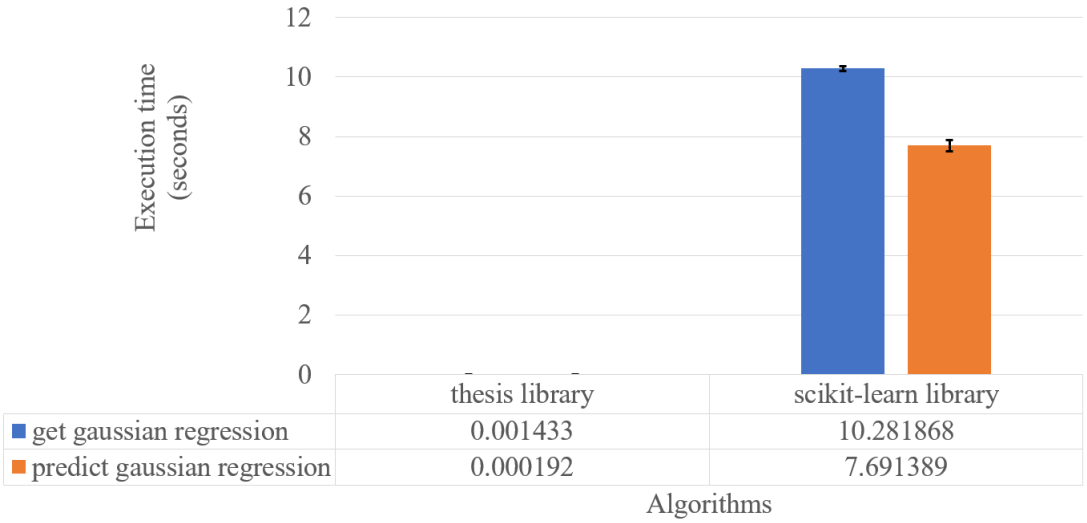


Figure 5.8: Bar chart of the execution times for the get and predict algorithms of the Gaussian regression that was developed in sequential processing mode and compared with the scikit-learn library.

Table 5.10: Mean and data dispersion of the execution times measured for the traditional regression algorithms that were made and used in this thesis.

Algorithm	**Execution time (seconds)**			
	$\hat{\theta}_L$	$\bar{x}$	$\hat{\theta}_U$	s
getSimpleLinearRegression_thesis	0.001685	0.001717	0.00175	0.00007
getSimpleLinearRegression_scikitLearn	0.026546	0.027492	0.028437	0.002011
predictSimpleLinearRegression_thesis	0.00407	0.004122	0.004175	0.000111
predictSimpleLinearRegression_scikitLearn	0.004804	0.005297	0.005789	0.001048
getMultipleLinearRegression_thesis	0.071457	0.072294	0.073131	0.001779
getMultipleLinearRegression_scikitLearn	0.070936	0.072541	0.074147	0.003414
predictMultipleLinearRegression_thesis	0.005704	0.005831	0.005958	0.00027
predictMultipleLinearRegression_scikitLearn	0.005028	0.007027	0.009025	0.00425
getPolynomialRegression_thesis	0.143255	0.144388	0.145522	0.00241
scikit-learn				
predictPolynomialRegression_thesis	0.009417	0.009489	0.00956	0.000152
scikit-learn				
getMultiplePolynomialRegression_thesis	0.142862	0.144487	0.146112	0.003454
scikit-learn				
predictMultiplePolynomialRegression_thesis	0.008631	0.008719	0.008807	0.000188
scikit-learn				
getLogisticRegression_thesis	0.060967	0.061417	0.061868	0.000958
scikit-learn				
predictLogisticRegression_thesis	0.013574	0.013715	0.013856	0.0003
scikit-learn				
getGaussianRegression_thesis	0.001186	0.001433	0.001681	0.000526
getGaussianRegression_scikitLearn	10.20998	10.28187	10.35376	0.152851
predictGaussianRegression_thesis	0.000168	0.000192	0.000217	0.000051
predictGaussianRegression_scikitLearn	7.514389	7.691389	7.868388	0.376346

Regarding the validation results obtained, all the regression algorithms were reliable because they did not have any errors with the model they obtained to predict the data that was intentionally generated to expect that outcome (see Table 5.11). Moreover, Table 5.12 outlines the improvements achieved for all the developed algorithms with respect to the mean of the measured execution times, if applicable. In addition, as indicated in that table, all the regression algorithms developed in this thesis that were compared with a reference library turned out to perform faster. Therefore, the results obtained with all these algorithms confirmed the initial hypothesis.

Table 5.11: Validation and reliability results obtained for the traditional regression algorithms that were made and used in this thesis.

Algorithm	n	$\eta_{(tests)}$ $(e_k = \bar{x} - \hat{\theta}_L)$	**Execution time (seconds)**		
			MSE	R^2	R_adj^2
getSimpleLinearRegression_thesis	1000000	32	0	1	1
getSimpleLinearRegression_scikitLearn	1000000	30	8.08E-28	1	
predictSimpleLinearRegression_thesis	1000000	31	—	—	—
predictSimpleLinearRegression_scikitLearn	1000000	30	—	—	—
getMultipleLinearRegression_thesis	1000000	30	0	1	1
getMultipleLinearRegression_scikitLearn	1000000	31	1.65E-28	1	
predictMultipleLinearRegression_thesis	1000000	30	—	—	—
predictMultipleLinearRegression_scikitLearn	1000000	30	—	—	—
getPolynomialRegression_thesis	1000000	31	0	1	1
scikit-learn					
predictPolynomialRegression_thesis	1000000	30	—	—	—
scikit-learn			—	—	—
getMultiplePolynomialRegression_thesis	1000000	30	0	1	1
scikit-learn					
predictMultiplePolynomialRegression_thesis	1000000	31	—	—	—
scikit-learn			—	—	—
getLogisticRegression_thesis	1000000	31	0	1	1
scikit-learn					
predictLogisticRegression_thesis	1000000	31	—	—	—
scikit-learn			—	—	—
getGaussianRegression_thesis	10000	31	0	1	1
getGaussianRegression_scikitLearn	10000	30	8.16E-14	1	
predictGaussianRegression_thesis	10000	30	—	—	—
predictGaussianRegression_scikitLearn	10000	30	—	—	—

Table 5.12: Improvements obtained for all the traditional regression algorithms developed with respect to the mean execution times obtained.

Algorithm	**This thesis (seconds)**	**Comparison library (seconds)**	**Improvement**
getSimpleLinearRegression	0.001717	0.027492	16.01165
predictSimpleLinearRegression	0.004122	0.005297	1.285056
getMultipleLinearRegression	0.072294	0.072541	1.003417
predictMultipleLinearRegression	0.005831	0.007027	1.205111
getPolynomialRegression	0.144388		—
predictPolynomialRegression	0.009489		—
getMultiplePolynomialRegression	0.144487		—
predictMultiplePolynomialRegression	0.008719		—
getLogisticRegression	0.061417		—
predictLogisticRegression	0.013715		—
getGaussianRegression	0.001433	10.28187	7175.066
predictGaussianRegression	0.000192	7.691389	40059.32

5.4.2 Classification algorithms

The classification algorithms that were developed with both their get and predict functionalities as software functions in the library that was created for this thesis are the following:

- Linear logistic classification.
- Simple linear machine classification.
- Kernel machine classification.
  - Linear Kernel.
  - Polynomial Kernel.
  - Logistic Kernel.
  - Gaussian Kernel.

for which the mean execution times and mean intervals are shown in the bar charts of Figures 5.9, 5.10, 5.11 and 5.12 for both the library of this thesis and for the comparison libraries used. These Figures are representative only of the algorithms developed in this thesis and that were compared to an equivalent algorithm. Nonetheless, the simple machine classification and Kernel machine classification were compared with the linear/Kernel support vector machine despite being different algorithms. This is because, although they address the same type of problems with a different approach, they solve exactly the same model. On the other hand, those algorithms that were not shown in bar graphs were due to the fact that no equivalent

algorithm was found with respect to the available research time. Moreover, Tables 5.13 and 5.14 provide more details of the tests that were made for the evaluation and validation of all the algorithms considered.

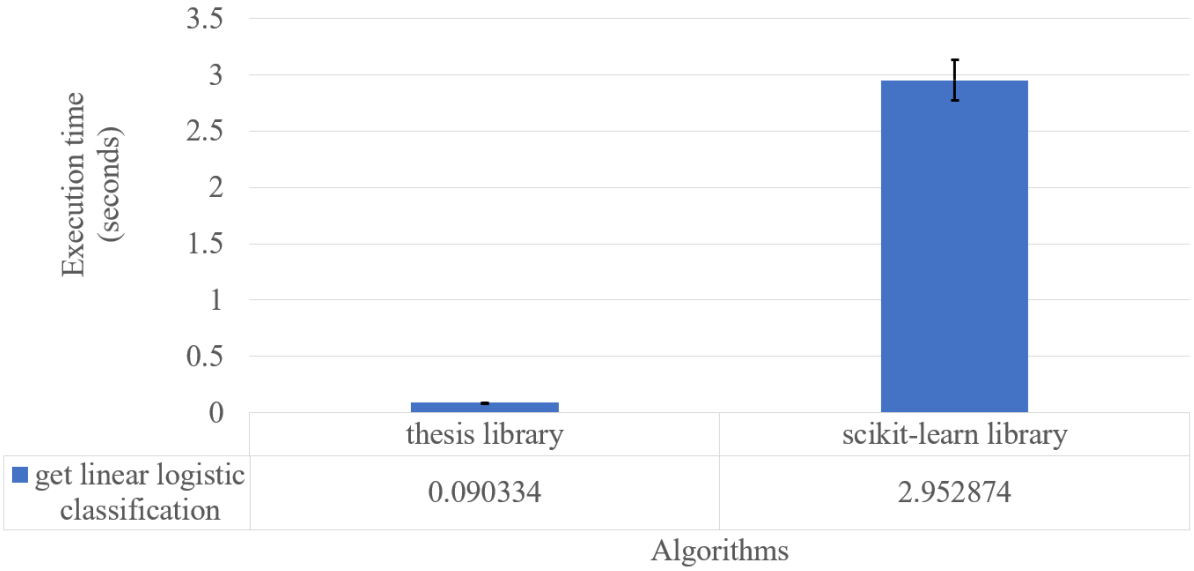


Figure 5.9: Bar chart of the execution times for the get algorithm of the linear logistic classification that was developed in sequential processing mode and compared with the scikit-learn library.

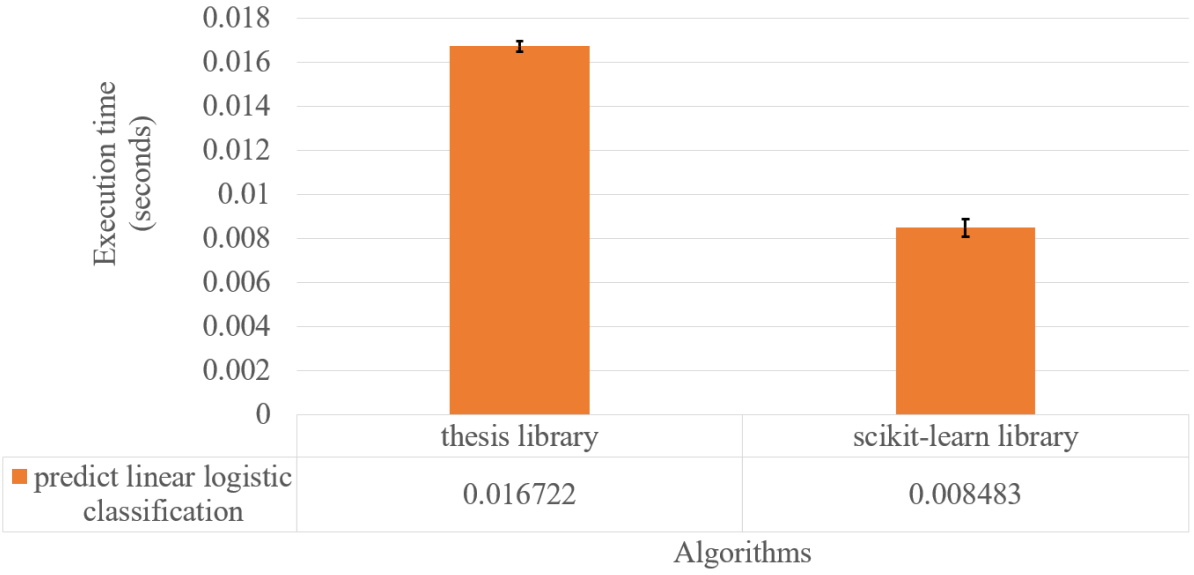
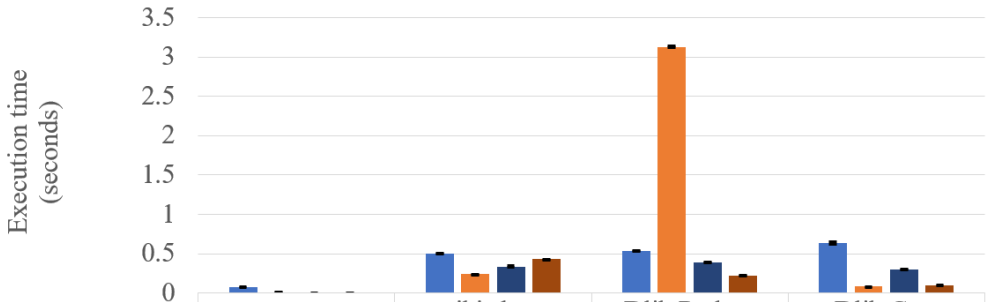


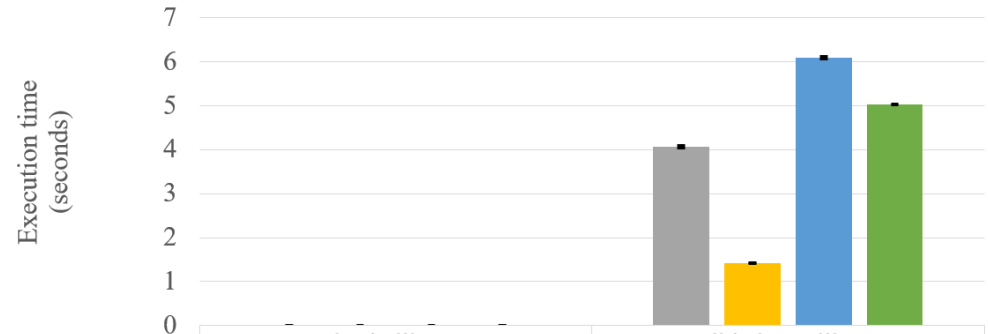
Figure 5.10: Bar chart of the execution times for the predict algorithm of the linear logistic classification that was developed in sequential processing mode and compared with the scikit-learn library.



	thesis library	scikit-learn library	Dlib Python library	Dlib C++ library
get linear Kernel MC/SVM	0.075372	0.496705	0.530455	0.637149
predict linear Kernel MC/SVM	0.006305	0.238902	3.130361	0.083765
get gaussian/rbf Kernel MC/SVM	0.002787	0.33706	0.391016	0.296757
predict gaussian/rbf Kernel MC/SVM	0.000299	0.428953	0.217079	0.099565

Algorithms

Figure 5.11: Bar chart of the execution times for the get and predict algorithms of the linear Kernel machine classification and the Gaussian Kernel machine classification that were developed in sequential processing mode and compared with the linear support vector machine and the radial basis function Kernel support vector machine of the scikit-learn library.



	thesis library	scikit-learn library
get polynomial Kernel MC/SVM	0.002513	4.068899
predict polynomial Kernel MC/SVM	0.000177	1.411839
get Sigmoid Kernel MC/SVM	0.001506	6.085327
predict Sigmoid Kernel MC/SVM	0.000307	5.030099

Algorithms

Figure 5.12: Bar chart of the execution times for the get and predict algorithms of the polynomial Kernel machine classification and the logistic Kernel machine classification that were developed in sequential processing mode and compared with the polynomial Kernel support vector machine and the sigmoid Kernel support vector machine of the scikit-learn library.

Table 5.13: Mean and data dispersion of the execution times measured for the traditional classification algorithms that were made and used in this thesis.

Algorithm	n	$\eta_{(tests)}$ $(e_k = \bar{x} - \hat{\theta}_L)$	**Execution time (seconds)**			
			$\hat{\theta}_L$	$\bar{x}$	$\hat{\theta}_U$	s
getLinearLogisticClassification_thesis	1000000	30	0.089639	0.090334	0.091028	0.001476
getLinearLogisticClassif_scikitLearn	1000000	30	2.77422	2.952874	3.131528	0.379863
predLinearLogisticClassification_thesis	1000000	30	0.016471	0.016722	0.016973	0.000533
predLinearLogisticClassif_scikitLearn	1000000	31	0.008093	0.008483	0.008873	0.00083
getSimpleLinMachineClassif_thesis	1000000	31	0.074443	0.075372	0.076301	0.001976
getLinearSvmClassification_scikitLearn	1000000	31	0.493122	0.496705	0.500288	0.007619
getLinearSvmClassification_PythonDlib	1000000	30	0.524251	0.530455	0.536659	0.013191
getLinearSvmClassification_C++Dlib	1000000	30	0.617719	0.637149	0.656578	0.041312
predSimpleLinMachineClassif_thesis	1000000	30	0.006159	0.006305	0.00645	0.000309
predLinearSvmClassification_scikitLearn	1000000	30	0.237476	0.238902	0.240328	0.003032
predLinearSvmClassification_PythonDlib	1000000	31	3.119252	3.130361	3.14147	0.023621
predLinearSvmClassification_C++Dlib	1000000	31	0.08351	0.083765	0.084021	0.000543
getPolyKernelMachineClassif_thesis	10000	31	0.002331	0.002513	0.002696	0.000389
getPolyKernelSvmClassif_scikitLearn PythonDlib C++Dlib	10000	31	4.037252	4.068899	4.100546	0.06729
predictPolyKernelMachineClassif_thesis	10000	32	0.00016	0.000177	0.000194	0.000037
predictPolyKernelSvmClassif_scikitLearn PythonDlib C++Dlib	10000	30	1.403426	1.411839	1.420252	0.017888
getLogisticKernelMachineClassif_thesis	10000	30	0.001297	0.001506	0.001714	0.000443
getSigmoidKernelSvmClassif_scikitLearn PythonDlib C++Dlib	10000	30	6.058173	6.085327	6.112481	0.057736
predLogisticKernelMachineClassif_thesis	10000	30	0.000262	0.000307	0.000352	0.000095
predSigmoidKsvmClassif_scikitLearn PythonDlib C++Dlib	10000	30	5.02025	5.030099	5.039947	0.020941
getGaussianKernelMachineClassif_thesis	10000	31	0.00255	0.002787	0.003024	0.000504
getRbfKernelSvmClassif_scikitLearn	10000	31	0.330494	0.33706	0.343626	0.013961
getRbfKernelSvmClassif_PythonDlib	10000	31	0.383916	0.391016	0.398117	0.015098
getRbfKernelSvmClassif_C++Dlib	10000	31	0.290135	0.296757	0.30338	0.014081
predGaussianKmachineClassif_thesis	10000	32	0.000262	0.000299	0.000337	0.00008
predRbfKernelSvmClassif_scikitLearn	10000	30	0.425479	0.428953	0.432426	0.007385
predRbfSvmMachineClassif_PythonDlib	10000	31	0.21106	0.217079	0.223098	0.012798
predRbfSvmMachineClassif_C++Dlib	10000	30	0.099445	0.099565	0.099684	0.000255

Table 5.14: Validation and reliability results obtained for the traditional classification algorithms that were made and used in this thesis.

Algorithm	**Execution time (seconds)**								
	NLL	TP	FP	FN	TN	A	P	R	F_1
getLinearLogisticClassification_thesis	0	505000	0	0	495000	1	1	1	1
getLinearLogisticClassif_scikitLearn	9.99E-10	505000	0	0	495000	1	1	1	1
getSimpleLinMachineClassif_thesis	0	505000	0	0	495000	1	1	1	1
getLinearSvmClassification_scikitLearn	9.99E-10	505000	0	0	495000	1	1	1	1
getLinearSvmClassification_PythonDlib	341941.8	505000	9900	0	485100	0.9901	0.980773	1	0.990293
getLinearSvmClassification_C++Dlib	0	505000	0	0	495000	1	1	1	1
getPolyKernelMachineClassif_thesis	12503.32	4393	362	0	5245	0.9638	0.92387	1	0.960429
getPolyKernelSvmClassif_scikitLearn	83066.93	3458	1470	935	4137	0.7595	0.701705	0.787161	0.74198
PythonDlib									
C++Dlib									
getLogisticKernelMachineClassif_thesis	0	5500	0	0	4500	1	1	1	1
getSigmoidKernelSvmClassif_scikitLearn	310852.6	1000	4500	4500	0	0.1	0.181818	0.181818	0.181818
PythonDlib									
C++Dlib									
getGaussianKernelMachineClassif_thesis	10396.17	4092	0	301	5607	0.9699	1	0.931482	0.964526
getRbfKernelSvmClassif_scikitLearn	1105.241	4361	0	32	5607	0.9968	1	0.992716	0.996345
getRbfKernelSvmClassif_PythonDlib	552.6204	4377	0	16	5607	0.9984	1	0.996358	0.998176
getRbfKernelSvmClassif_C++Dlib	552.6204	4377	0	16	5607	0.9984	1	0.996358	0.998176

Table 5.15: Improvements obtained for all the traditional classification algorithms developed with respect to the mean execution times obtained.

Algorithm	**This thesis (seconds)**	**Comparison library (seconds)**	**Improvement**
getLinearLogisticClassification	0.090334	2.952874	32.6884
predictLinearLogisticClassification	0.016722	0.008483	0.507296
getSimpleLinearMachineClassification or getLinearSvmClassification	0.075372	0.496705	6.590047
predictSimpleLinearMachineClassification or predictLinearSvmClassification	0.006305	0.083765	13.28549
getPolynomialKernelMachineClassification or getPolynomialKernelSvmClassification	0.002513	4.068899	1619.14
predictPolynomialKernelMachineClassification or predictPolynomialKernelSvmClassification	0.000177	1.411839	7976.492
getLogisticKernelMachineClassification or getSigmoidKernelSvmClassification	0.001506	6.085327	4040.722
predictLogisticKernelMachineClassification or predictSigmoidKernelSvmClassification	0.000307	5.030099	16384.69
getGaussianKernelMachineClassification or getRbfKernelSvmClassification	0.002787	0.296757	106.479
predictGaussianKernelMachineClassification or predictRbfKernelSvmClassification	0.000299	0.099565	332.9933

The results obtained for all the classification algorithms were reliable because most of them had no errors with the model they obtained to predict the data from the databases that were intentionally generated to expect a specific outcome (see Table 5.14). In addition, the ones that had some error were small where the lowest F1 score obtained was 0.960429 for the polynomial Kernel machine classification algorithm. Moreover, Table 5.15 presents the mean execution times of the classification algorithms during the testing process and also shows the improvement achieved for each of them. In it, all the classification algorithms developed in this thesis and that were compared with another library resulted to be faster than the comparison libraries that were taken into account, except for the predict of the linear logistic classification. Therefore, the results obtained with all these algorithms somewhat contradicts the initial hypothesis, but only for the predict of the linear logistic classification.

5.4.3 Deep learning algorithms

The deep learning methods that were developed with both their get and predict functionalities as software functions in the library that was created for this thesis are the following:

- Single artificial neuron.

for which the mean execution times and mean intervals are shown in the bar charts of Figures 5.13 and 5.14 for both the library of this thesis and for the comparison libraries. Should more data be needed, Tables 5.16 and 5.17 give the information of the bar charts with more detail. Moreover, consider for such results that it was strategically decided to have all the algorithms solve a linear regression problem and have them train for 30863 epochs.

On the other hand, the results obtained for the developed deep learning algorithm was reliable because it was able to get a fair model with respect to the stop function it was given. Such stop function was assigned an adjusted R-squared of 0.99 for the data that was intentionally generated to expect a specific linear outcome (see Table 5.17). Moreover, Table 5.18 presents the improvements achieved for the developed algorithm with respect to the mean execution times of its get and predict functions. In that Table, the get of the deep learning algorithm developed in this thesis resulted to be faster than the comparison libraries that were taken into account. Conversely, the predict of such algorithm was in fact slower than the reference libraries that were considered. Therefore, the results obtained with the developed deep learning algorithm somewhat contradicts the initial hypothesis, but only for the predict function.

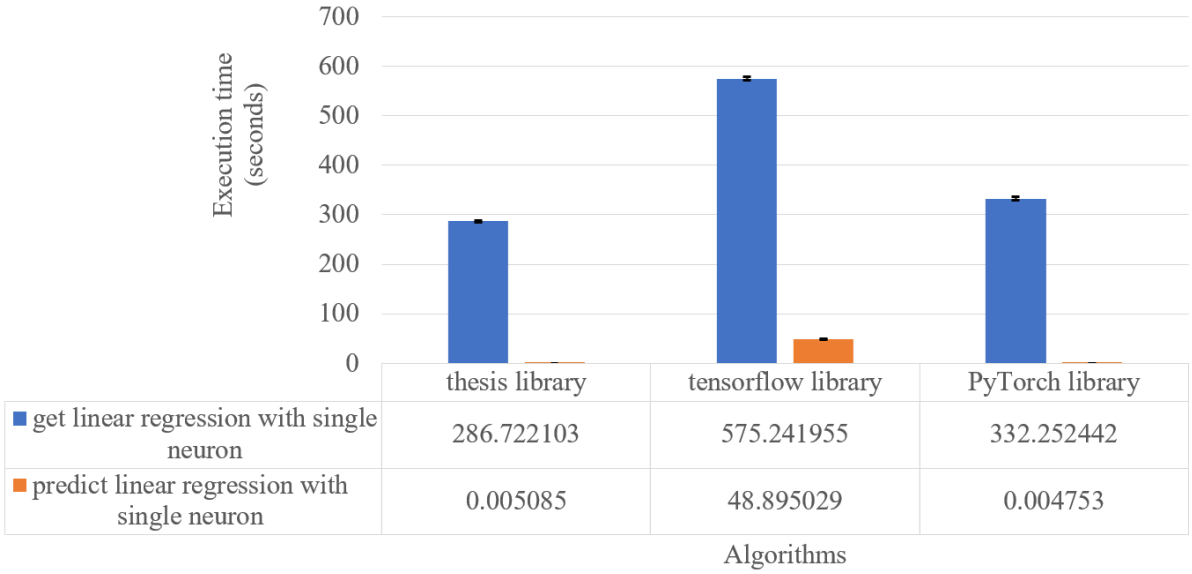


Figure 5.13: Bar chart of the execution times for the get and predict algorithms of the single artificial neuron that was developed in sequential processing mode and compared with the TensorFlow and PyTorch library with respect to a linear equation system to be solved.

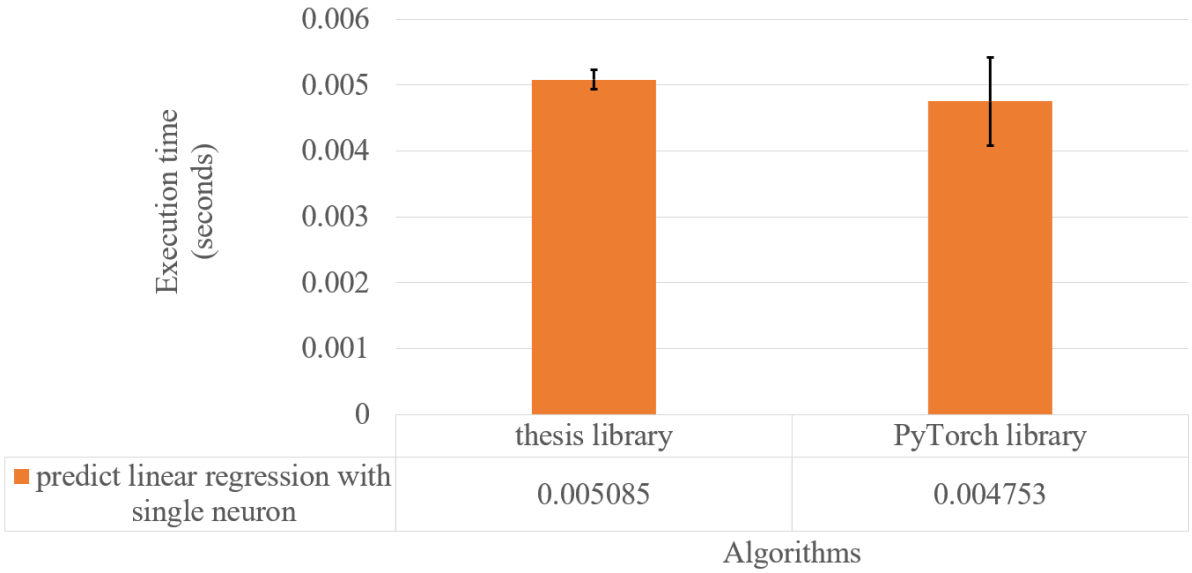


Figure 5.14: Bar chart of the execution times for the predict algorithms of the single artificial neuron that was developed in sequential processing mode and compared with the PyTorch library with respect to a linear equation system to be solved.

Table 5.16: Mean and data dispersion of the execution times measured for the deep learning algorithms that were made and used in this thesis.

Algorithm	**Execution time (seconds)**			
	$\hat{\theta}_L$	$\bar{x}$	$\hat{\theta}_U$	s
getLinearRegressionSingleNeuron_thesis	284.9777	286.7221	288.4665	3.708969
getLinearRegressionSingleNeuron_TensorFlow	572.4671	575.242	578.0168	5.900059
getLinearRegressionSingleNeuron_Pytorch	328.5965	332.2524	335.9084	7.773505
predictLinearRegressionSingleNeuron_thesis	0.004943	0.005085	0.005227	0.000302
predictLinearRegressionSingleNeuron_TensorFlow	48.5086	48.89503	49.28146	0.821646
predictLinearRegressionSingleNeuron_PyTorch	0.004083	0.004753	0.005423	0.001425

Table 5.17: Validation and reliability results obtained for the deep learning algorithms that were made and used in this thesis.

Algorithm	n	$\eta_{(tests)}$ $(e_k = \bar{x} - \hat{\theta}_L)$	**Execution time (seconds)**		
			MSE	R^2	R_adj^2
getLinearRegressionSingleNeuron_thesis	1000000	30	5.333279	0.99	0.99
getLinearRegressionSingleNeuron_TensorFlow	1000000	30	1.142789		
getLinearRegressionSingleNeuron_PyTorch	1000000	31	20.50723	†	†
predictLinearRegressionSingleNeuron_thesis	1000000	31	—	—	—
predictLinearRegressionSingleNeuron_TensorFlow	1000000	30	—	—	—
predictLinearRegressionSingleNeuron_PyTorch	1000000	31	—	—	—

Table 5.18: Improvements obtained for all the deep learning algorithms developed with respect to the mean execution times obtained.

Algorithm	**This thesis (seconds)**	**Comparison library (seconds)**	**Improvement**
getLinearRegressionWithSingleNeuron	286.7221	332.2524	1.158796
predictLinearRegressionWithSingleNeuron	0.005085	0.004753	0.93471

5.4.4 Parallel computing in deep learning algorithms

This thesis addresses parallelism support only for the deep learning algorithm that was developed in its sequential version, mainly due to the time available to complete this project. Nevertheless, it was considered that this type of algorithm was the one that required it the most, in consideration of the drastic difference in execution times obtained with respect to all the algorithms evaluated. Therefore, a great effort was made to parallelize not only with the CPU but also with one and multiple GPUs. However, the main goal of this parallelization was not to compare with the reference libraries but with the sequential version that was made during this thesis. This is because the get function of that algorithm had already been found to

be faster than the reference libraries and because that is the algorithm in which there is more interest to parallelize due to the reasons already explained.

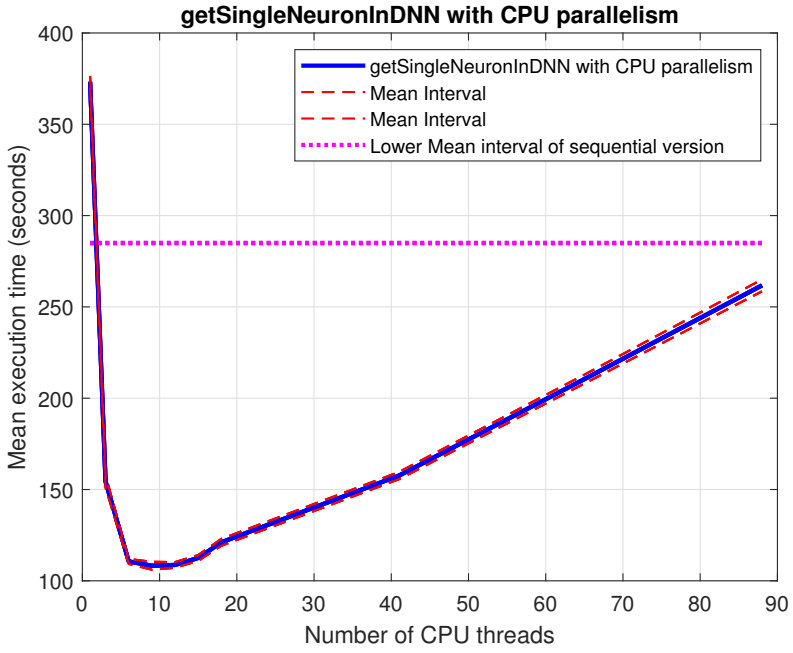


Figure 5.15: Graph of the execution times for the get algorithm of the single artificial neuron that was developed in CPU parallel processing mode to solve a linear equation system.

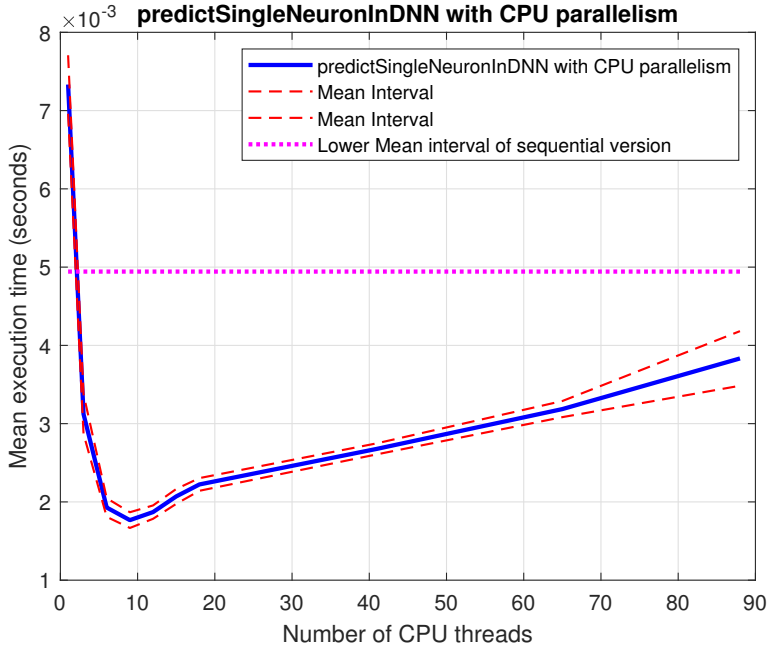


Figure 5.16: Graph of the execution times for the predict algorithm of the single artificial neuron that was developed in CPU parallel processing mode to solve a linear equation system.

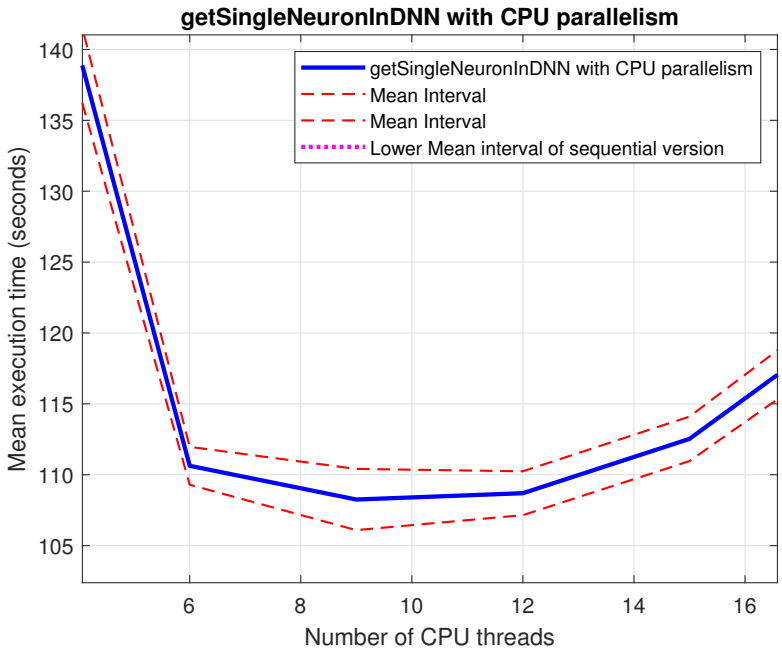


Figure 5.17: Graph of the CPU threads that gave the fastest execution times for the get algorithm of the single artificial neuron that was developed in CPU parallel processing mode to solve a linear equation system.

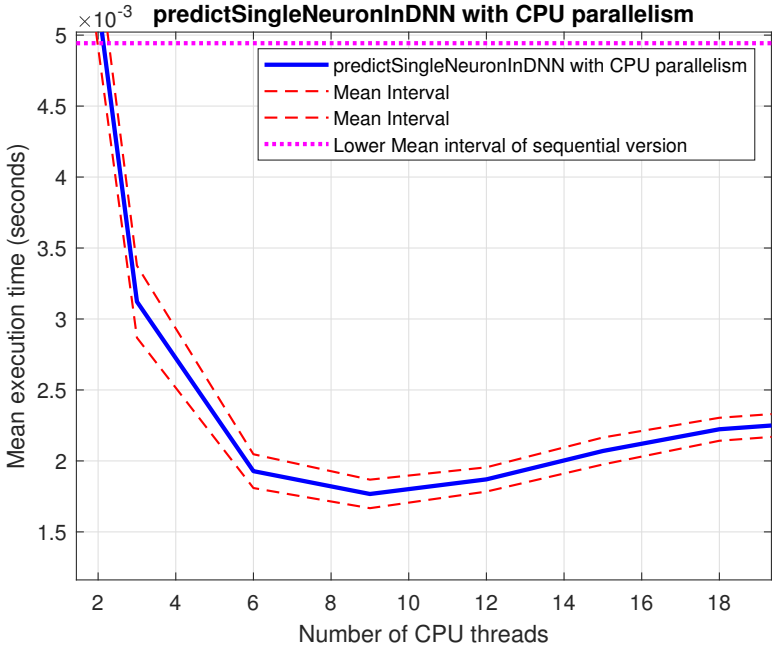


Figure 5.18: Graph of the CPU threads that gave the fastest execution times for the predict algorithm of the single artificial neuron that was developed in CPU parallel processing mode to solve a linear equation system.

Table 5.19: Improvements obtained for the deep learning algorithm developed in CPU parallel mode with respect to the mean execution times obtained.

Algorithm	**Sequential version of this thesis (seconds)**	**CPU parallel version of this thesis**		**Improvement**
		threads	**seconds**	
get linear regression with single neuron in DNN	286.7221	1	373.2712	0.768133
		3	154.4304	1.856643
		6	110.6309	2.5917
		9	108.2513	2.648671
		12	108.6965	2.637823
		15	112.531	2.547939
		18	121.0896	2.367851
		41	157.618	1.819095
		65	210.4336	1.36253
		88	261.7748	1.095301
predict linear regression with single neuron in DNN	0.005085	1	0.007333	0.693441
		3	0.003122	1.628764
		6	0.001928	2.637448
		9	0.001767	2.877759
		12	0.00187	2.719251
		15	0.00207	2.456522
		18	0.002223	2.287449
		41	0.002678	1.898805
		65	0.003186	1.596045
		88	0.003833	1.326637

Regarding the results obtained when parallelizing with the CPU, the mean execution times and mean intervals are shown in the graphs of Figures 5.15 and 5.16. In contrast, Figures 5.17 and 5.18 display the same information of the previous two Figures but showing only the results of the CPU threads that achieved the fastest execution times. On the other hand, Table 5.19 presents the improvements achieved for the developed parallelized algorithm with respect to the mean execution times of its get and predict functions. In that Table, the get of the CPU parallelized version of the deep learning algorithm was found to be about 2.65 times faster than the sequential version at the most, when using 9 CPU threads. In contrast, its predict function achieved an improvement of around 2.88 times at the most over the sequential version when 9 CPU threads are employed. Moreover, not only faster execution times were obtained but, as expected, the parallelized version of its predict function turned out to be faster than the fastest sequential version of the reference libraries (Table 5.18).

Table 5.20: Improvements obtained for the deep learning algorithm developed in multiple GPU mode with respect to the mean execution times obtained.

Algorithm	**Sequential version of this thesis (seconds)**	**Multi GPU parallel version of this thesis (Tesla K80)**		**Improvement**
		GPUs	**seconds**	
get linear regression with single neuron in DNN	286.7221	1	45.14065	6.351749
		2	25.5258	11.23264
		3	19.74193	14.52351
		4	17.17618	16.69301
predict linear regression with single neuron in DNN	0.005085	1	0.009806	0.51856
		2	0.013077	0.388851
		3	0.015724	0.323391
		4	0.019549	0.260116

As for the deep learning algorithm developed in its multiple GPU version, the mean execution times and improvements obtained are shown in Table 5.20. There, for the get function, it can be determined that the higher the number of GPUs, the faster the results obtained, unlike what happened with the CPU parallel version (see Table 5.19). Conversely, the predict function is slower at all times when GPU parallelism is used and gets worse the higher the number of GPUs employed. However, the results obtained for both cases were as expected: 1) For the get function because GPUs are more adept at parallelism than CPUs and 2) For the predict function because GPUs are particularly better when the number of processes needed for the task is larger and cannot give any benefit when such processes are few. Conclusively, the best result obtained with the multiple GPU version, achieved an improvement of around 16.69 times faster execution times than the sequential version when using 4 GPUs.

5.5 Results obtained in embedded systems

During this thesis, it was decided to demonstrate the use of the developed library in low profile embedded systems. For this purpose, the Arduino UNO and the STM32F446RE development board were selected for these tests, whose developed application examples can be obtained through the references [69] and [70]. On the Arduino software, a simple linear regression was trained and, on the STM32F446RE microprocessor, both a simple linear regression and a single neuron in Deep Neural Network were trained. The reason on why that deep learning method was not used in the Arduino UNO board was because more memory in that device was required for that. On the other hand, all the algorithms executed in these embedded systems were also

tested in the server, whose hardware is listed in the subsection 4.2.1, for comparison purposes. Moreover, because the objective of these tests was to only proof that the developed library supports embedded systems, one test was performed for each case evaluated and its results are shown in Table 5.21. Finally, the value of n that was used for each case was limited to the maximum number of samples with which each case was able to work correctly.

Table 5.21: Execution times obtained for the Arduino UNO and STM32F446RE low profile embedded systems, when using the library of this thesis in them and in the server that was used to validate the software developed for this thesis.

Algorithm	**Computer System**	n	**Execution Time (seconds)**
getSimpleLinearRegression	Arduino UNO	115	0.007
	Server	115	0.000001
	STM32F446RE	65530	0.129
	Server	65530	0.000236
getLinearRegressionWithSingleNeuron	STM32F446RE	1990	241.904
	Server	1990	0.275809

Chapter 6

Discussions

Machine learning is a very popular tool that has contributed to solve several problems in many different research fields in which even parallel computing has been heavily involved. Because of this, there are several such libraries that are open source and can be used by anyone who wishes to do so. However, most of these machine learning libraries primarily address deep learning algorithms and have few contributions to traditional machine learning algorithms. At the same time, while these libraries have strived to improve the performance of their algorithms through parallel programming, solutions for embedded systems have been neglected in comparison. In contrast, the results in Chapter 5 indicate that this thesis contributes with a new machine learning library that supports parallel computing on deep learning algorithms and that also supports embedded systems. In addition, several new non deep learning algorithms were introduced that benefit greatly in execution time over the existing algorithms without having the implementation of parallel computing in them.

6.1 Statistical functions

The statistical functions proposed for this thesis are relatively simple from a mathematical and software implementation point of view. In contrast, this is not the case when attempting to develop them efficiently in software so that they can compete with the reference libraries that were selected. A great deal of knowledge of software development and even computer hardware is required to be successful. In this regard, there is confidence in the overall work done, except for the quick median and quick sort implementations that were made because the C pointers used in their code was not fully exploited as in the other algorithms. Nonetheless, in this thesis it is considered that the measurements for the quick median and quick sort algorithms were not done fairly for neither of the compared libraries due to the nature of how these work. This is because both use a pivot whose value is defined by the developer and it is difficult to know what

value was defined for the comparison library used (NumPy). Therefore, future work could be based on a fair evaluation of them by permuting the position of the input data as many times as possible to test each of them.

Regarding the results obtained for the execution times, the mean intervals of Figures 5.1 and 5.2 are going to be considered key in defining what method is genuinely faster. But first, let us state that these mean intervals have all a confidence value of 99% and that they were not all sufficiently tested according to η in Table 5.2. However, the worst case scenario required 31 samples over the 30 that were actually made, which will only mean that there is a 99% confidence that the error of considering the $\bar{x}$ achieved as μ will not exceed the entire range of the mean intervals obtained plus an additional 3.31% approximately. Because this additional value is negligible in consideration of the analysis to be applied with the mean intervals, only the full range of those obtained for them in Table 5.1 will be taken into account.

As a result, because of the high significance of the results obtained with the mean intervals of the methods in Figures 5.1 and 5.2, it is considered that the mean; mode; standard deviation and variance are faster in the thesis library than in the comparison libraries. In contrast, due to the same reasons and despite the non fair evaluations previously mentioned, it is considered in this thesis that the median and sort methods are faster in the comparison library used than in the thesis library. These determinations are assumed because the mean intervals of each of the compared algorithms do not overlap, meaning that those with a lower mean execution time are highly likely to be faster with respect to their corresponding comparators. Nonetheless, future work could contribute by increasing the number of samples obtained to meet the requested value of η for each algorithm in order to give more strength to the results obtained. However, since the mean intervals obtained in the results are considerably separated, the fastest algorithms defined are not expected to change in spite of this. Finally, for all the statistical algorithms and any other developed in this thesis, future work could provide a better insight of the circumstances at which each algorithm is best by testing at different values of n and m.

6.2 Feature scaling functions for machine learning

The feature scaling methods that were considered in this thesis are a total of 3, of which 6 algorithms were made because each of them has its application function of that feature scaling method and its reverse function. There, only the application or get functions were compared with a reference library, where it was found that all these algorithms are highly likely to be faster than their comparators as shown in Figure 5.3. This is due to the fact that their corresponding mean intervals do not overlap and that the expected error margins determined by η are negligible. This means that even if future work were to increase the number of samples, it

is not expected in this thesis that such conclusions would change.

On the other hand, each of the reverse functions were only validated by comparing them with the original data given to their corresponding get function, instead of comparing them with the reference libraries. This was due to the time available to complete this project and because the machine learning algorithms were the priority of this work. Therefore, future work could build on this research by comparing these 3 reverse functions with the reference libraries. Nevertheless, all the validation results obtained in Table 5.5 indicate that all algorithms are reliable.

6.3 Evaluation metric functions for machine learning algorithms

The mean intervals of all the evaluation metric functions for machine learning algorithms that were developed also have a negligible error as foreseen with η. Therefore, as for the previous algorithms, only the full range obtained for the mean intervals in Figures 5.4 and 5.5 will be considered. Furthermore, since none of the corresponding compared algorithms have their mean intervals overlapping and due to their obtained mean execution times, it is highly likely that all the evaluation metric functions developed in this thesis are faster than the comparison libraries. In addition, considering the low values of the permitted errors that were registered in Table 5.8, it is determined that all these algorithms developed in this thesis are reliable. Nonetheless, future work could enrich the robustness of the currently obtained results by increasing the number of tests performed to meet the minimum value of η for each algorithm. However, because all the mean intervals for each corresponding comparison are considerably separated, it is not expected by this thesis to determine that one of these type of algorithms that were made here will not be faster.

6.4 Machine learning functions

Several machine learning algorithms were developed in the library of this thesis, some of which are available in other reference libraries, but others are completely new contributions of this work (see subsection 3.3 or Chapter 7 for more details). However, because some of these new methods happened to solve the same mathematical model as other well known approaches, it was decided to compare them. In this regard, the results obtained indicated that these new methods are considerably faster than the comparison libraries and that although they do not have parallel computing implemented, they may feel as if they do because of the very drastic

difference. This occurs because the new methods of this thesis give a direct solution and the reference libraries provide it by other means that require an iteration process. Therefore, for the following discussions, know that all algorithms will not have parallel computing unless otherwise mentioned.

6.4.1 Regression algorithms

In this thesis, a total of 6 different traditional regression methods have been developed, each with their get and predict functions (a total of 12 algorithms). From these methods, the simple linear regression and multiple linear regression were the only ones identified in external works and reference libraries. In contrast, the polynomial regression; multiple polynomial regression; logistic regression and Gaussian regression were not found as mathematically formulated in this thesis. Therefore, from these last four methods mentioned, only one equivalent algorithm for the Gaussian regression was found within the reference libraries: in scikit-learn. Because of this, only such method; the simple linear regression and multiple linear regression were compared with a reference library.

As for the Gaussian regression, after reviewing the documentation of scikit-learn [43], it was suspected in this work that their method is not intended to solve regression problems as proposed in this thesis, yet it was capable of doing so at the same time. Consequently, since no other equivalent method was found to be able to apply the traditional Gaussian regression method, it was decided to compare them. Outstandingly, the results were incredibly better with the proposal made in this thesis as can be seen in Table 5.12. Nonetheless, because it is suspected that the scikit-learn method is meant to address different problems, it is not entirely fair to compare them, but at the same time no other option was found in the literature review for when a practitioner would required such a regression method.

On the other hand, the comparison made with the simple linear regression method indicates that both the get and predict functions developed in this thesis are highly likely to be faster than the comparison libraries. The reason for this is that the mean intervals in Figure 5.6 are considerably separated and the expected range of the error of the mean intervals is negligible for this comparison. For this, consideration is being given to requiring 32 tests over the 30 performed as described in Table 5.11, which means an increase in the expected error margin of approximately 6.56% instead of 3.31% for when $\eta = 31$, but that is not enough to compromise the given interpretation.

In contrast to the above, although faster mean execution times were obtained for the get and predict functions of the multiple linear regression method of this thesis according to Table 5.12 , this is not considered to be true after analyzing Figure 5.7. This is because the mean intervals completely overlap with each of their corresponding comparators. Therefore, this is

interpreted as defining that these algorithms are highly likely neither faster nor slower, rather they are considered to be equally fast. Finally, as with the previous algorithms, future work could strengthen the interpreted results of all the regression algorithms discussed so far and, in particular, for the simple linear regression. However, since the mean intervals of the latter algorithm are very tight, it is not expected in this thesis that the given interpretation for it will change.

6.4.2 Classification algorithms

The classification methods that were developed in the library of this thesis are considered to be faster than all the comparison libraries, except in the case of the predict function of the linear logistic classification. This interpretation was given in spite of the values η in Table 5.13 because the error margins of the mean intervals were considered negligible. Furthermore, the full range of those mean intervals were considerably separated for all their corresponding comparators in Figures 5.9, 5.10, 5.11 and 5.12. Nonetheless, in order to give more reliability to the mean intervals, future work could increase the number of tests performed to meet the value of η. Despite this, due to the large difference between those intervals, it is not expected in this thesis that such work will compromise the interpretations currently given.

On the other hand and as a reminder, the results defined in Table 5.15 are not considering parallel computing despite having such outstanding results overall. With this being mentioned, it is considered in this work that those results were not even fair to the methods developed in this thesis. The reason is that, unlike the simple linear machine classification method of this thesis, the support vector machine method of the comparison libraries require adjustable parameters (hyperparameters) in order to obtain a reliable training result. This means that the time invested to manually find out the best values of such parameters is not shown in the results and in some cases it was necessary to spend several minutes to figure them out. Consequently, this gives even more value to the simple linear/Kernel machine classification method contributed in this thesis.

6.4.3 Deep learning algorithms

Although two different types of deep learning methods were mathematically formulated in this thesis, only one of them was developed and tested: the single neuron in Deep Neural Network. This is mainly because this method is the least complex form of a neural network, which was considered the most favorable way to measure the fastest possible results for all the deep algorithms to test and have the least amount of bias during that. As a consequence, this gives the opportunity for future work to review; develop and test the mathematical formulation done

in this thesis for the Deep Neural Network with a single output.

Moreover, since the deep learning algorithms have very different mechanisms for training their neurons, the way it was decided to evaluate their execution times was to have them train for 30863 epochs, which was randomly proposed to make them train for at least a couple of minutes. This was decided in an attempt to reduce any bias in the results obtained because it was determined that basically the fastest algorithm was the one in which the best hyperparameter values were defined and an interesting observation is that they were different for each library. A further factor that favored this form of evaluation was that there was no end to finding better values for the hyperparameters and if this had been considered in the results to determine which algorithm was the fastest, then the results of this thesis would have been easily biased. In addition, it should be noted that the scikit learn library obtained outstanding results only when using some its unique features to modulate the learning rate during its training process, something that was not found in any other library. However, because it took too many hours to train endlessly when forced to train for the same proposed number of epochs, that library was discarded from testing.

On the other hand and regarding the results obtained, it was determined that it is highly likely that the get function of the single neuron method was faster in this thesis than with the comparison libraries considered for testing. This was supported with Figure 5.13 because the mean intervals are far apart with respect to those in this thesis; the expected error margin for the intervals of the PyTorch library are negligible (according to the required η) and because it had the fastest mean execution time. Conversely, it was determined that it is highly likely that the predict function of such method was equally fast for both the PyTorch library and this thesis. This was decided because the mean intervals of both overlap completely and both have the fastest execution times among all the compared libraries, according to Figure 5.14.

Concerning the validation results obtained in Table 5.17, it should be noted that they were not really prioritized as long as they were able to give an output that attempted to consistently predict the output of the system under study. Therefore, it is not suggested to use this data to determine which library gets the highest quality results and, instead, use it to simply validate that they work correctly. However, this also means that future work could contribute to this thesis by conducting tests to specifically validate and determine the library that obtains the best quality results among the others.

6.4.4 Parallel computing in deep learning algorithms

The results obtained by applying CPU parallelism to the single neuron in Deep Neural Network algorithms were as expected. Figures 5.15; 5.16; 5.17 and 5.18 illustrate the results obtained for such parallelization applied on the get and predict functions and there, it can be determined

that the fastest results were obtained with the measured 6; 9 and 12 CPU threads. These were considered highly likely to be equally fast because their mean intervals overlap. However, considering that the higher the number of CPU threads, the higher the electrical power consumed, it is suggested to use 6 CPU threads when using such an algorithm to obtain the best cost effective results. In addition, these Figures mentioned illustrate that for both the get and predict functions in their CPU parallel version, they will be slower than the sequential version if less than 3 threads are used.

On the other hand, the results of Table 5.20 indicate that there is an improvement when parallelizing the get function with GPUs but not on the predict function, at least for the number of samples with which the tests were conducted. Furthermore, at least for the number of GPUs tested, it appears that the more GPUs used, the greater the improvement achieved. Therefore, future work could strengthen this observation or even set it aside by testing with a larger number of multiple GPUs. In addition, another possible future work is to remake these GPU tests but with a modern high profile GPU, since the one used is considered very slow in this thesis compared to the modern GPUs of today. Moreover, one hypothesis as to why there was no improvement with the predict function is because of the low number of processes that such an algorithm requires from the GPU, which makes other processes in the GPU (e.g. communication between host and device) take more time than processing the algorithm itself.

In comparison to all the other machine learning methods, the tested get function of the deep learning method has several disappointing aspects. First, when there is an intentional input data where it is expected to have a perfect fitting model, most other machine learning methods are able to output exactly that, but this is not true for deep learning, as it is always learning endlessly to improve the model and never really gets to that "perfect" point. Secondly, even if better hyperparameter values are given, the traditional machine learning algorithms with equivalent models take considerably less time to conclude their training process (especially for the library of this thesis) with respect to the tested deep learning method. Thirdly, even if multiple GPU parallelism is applied to the deep learning method, it is still slower than the other equivalent machine learning methods. Consequently, in this work, it is hard to understand why deep learning was so popular in the literature review conducted after knowing these aspects, which raises the following questions that were not address because they were beyond the scope of this thesis: "Do other researchers know about this?", "Are there any other papers that agree with these observations?". Therefore, these questions raise future work that could contribute by providing an answer to them.

6.5 Results obtained in embedded systems

The last part of this work was to demonstrate that the library developed in this thesis can be used in embedded systems and the results obtained in Table 5.21 satisfy exactly that. Regarding the results obtained, there is one important matter that is wanted to be pointed out in this thesis and that is, that although it is possible to use deep learning algorithms in low profile embedded systems, it is suggested to use instead the traditional machine learning algorithms that were contributed. Nonetheless, it is further suggested to train the models on a high profile computational system and try to make implementations on low profile embedded systems by delivering the resulting trained model to them. As a consequence, it is expected in this thesis to open up a whole new world of possible applications that can be addressed with machine learning and low profile embedded systems. However, it must be accepted that although the aforementioned goal was met, few algorithms and few tests were made. Therefore, future work could strengthen this thesis by extending the number of tests performed per algorithm and to also determine the limitations of the other algorithms of this thesis in such embedded systems. Other future work could also contribute by considering a larger number of embedded systems in the tests conducted.

Chapter 7

Conclusions

The tools that are available for machine learning are being used both in research and the industry to solve a wide diversity of problems across several different fields today. Because of this, there are various machine learning libraries available that are open source, according to the literature review performed in this thesis. From among them, it was concluded that the most representative ones are: 1) Dlib, 2) PyTorch, 3) scikit-learn and 4) TensorFlow. However, it was also identified that most machine learning libraries tend to address deep learning more notably than the traditional methods. At the same time, something similar occurs when it comes to identifying machine learning libraries that support parallel computing with respect to embedded systems, of which the latter are being neglected in comparison. As a consequence, this thesis presents a new and competitive machine learning library named CenyML that is programmed in C; supports both parallel computing and embedded systems; and contributes with a several algorithms.

As for the sequential methods that have been developed in the CenyML library, several different options have been made available that, together, complement each other and aim to provide the main tools that a machine learning practitioner will need:

- Statistical methods:
  - Mean.
  - Median.
  - Variance.
  - Standard deviation.
  - Mode.
  - Mean intervals.
- Feature scaling methods:

    - Min max normalization (get and reverse functions).
    - L2 normalization (get and reverse functions).
    - Z score normalization (get and reverse functions).
- Machine learning evaluation metric methods:
    - For regression problems:
        * Mean squared error.
        * Coefficient of determination (R-squared).
        * Adjusted coefficient of determination (adjusted R-squared).
    - For classification problems:
        * Cross entropy error function.
        * Confusion matrix.
        * Accuracy.
        * Precision.
        * Recall.
        * F1 score.
- Machine learning methods:
    - Traditional Regression methods:
        * Simple linear regression (get and predict functions).
        * Multiple linear regression (get and predict functions).
        * Polynomial regression (get and predict functions).
        * Multiple polynomial regression (get and predict functions).
        * Logistic regression (get and predict functions).
        * Gaussian regression (get and predict functions).
    - Traditional Classification methods:
        * Linear logistic classification (get and predict functions).
        * Simple linear machine classification (get and predict functions).
        * Kernel machine classification (get and predict functions).
            · Linear Kernel.
            · Polynomial Kernel.
            · Logistic Kernel.

        - Gaussian Kernel.
    - Deep learning methods:
        - Single artificial neuron (get and predict functions).

of which the single artificial neuron method has the get and predict functions also available to apply CPU, single GPU and multiple GPU parallelism. As a consequence, a total of 53 functions were developed in the CenyML library, of which 18 belong to 6 machine learning methods whose mathematical approach is a completely new contribution of this thesis, according to the literature review conducted and because the representative libraries do not directly provide a software function for them:

- Contributed machine learning methods:
    - Traditional Regression methods:
        - Polynomial regression (get and predict functions).
        - Multiple polynomial regression (get and predict functions).
        - Logistic regression (get and predict functions).
        - Gaussian regression (get and predict functions).
    - Traditional Classification methods:
        - Simple linear machine classification (get and predict functions).
        - Kernel machine classification (get and predict functions).

All the CenyML algorithms were tested several times to validate them with respect to their execution times and to determine whether their results were reliable or not. At the same time, most of them were compared with the representative machine learning libraries and, when not possible, with some other complementary libraries. In this regard, 36 of the 53 algorithms in the CenyML library were benchmarked against the competition. There, only 3 of the feature scaling algorithms (the reverse functions) were not benchmarked due to time considerations and because they were not the priority of this work; 6 of them had nothing to compare with, since they are new contributions; 2 of them were the get and predict functions of an equivalent method already benchmarked and the other 6 were the parallel computing algorithms, whose benchmarking was out of the scope of this thesis.

Based on the results obtained, it can be concluded that 30 out of 36 sequential algorithms benchmarked were faster in the CenyML library with respect to the reference libraries. On the other hand, 3 of those other 6 algorithms were equally fast and the remaining 3 were the only ones where the CenyML library was not faster. Therefore, around 83.33% of the CenyML

algorithms were faster, 8.33% were equally fast and 8.33% were slower. Furthermore, a 100% of the entire algorithms were successful in validating their reliability, meaning that it is concluded that the hypothesis of this thesis is fulfilled. In other words, the CenyML library that was developed had reliable results and has overall faster execution times with respect to the current machine learning libraries in their sequential version. Not only that, but the CenyML library was also shown to be compatible with low profile embedded systems, such as Arduino UNO and the STM32F446RE development board.

Finally, as for the future work to be continued after this thesis, it is expected to complete the benchmarking of the remaining algorithms that were not compared due to time constraints. In addition, some additional parallelization strategies will be evaluated with respect to those currently applied in order to attempt to improve the results already obtained. Furthermore, it is intended to develop the parallel version for all the other algorithms in which such outstanding tool was not applied. Also, there is a work in progress of some additional new machine learning methods that have already been mathematically formulated. Lastly, funding is expected to be sought to acquire modern high profile GPUs and a wide diversity of embedded systems to ensure that this library will be compatible with modern GPUs and a wide diversity of embedded systems.

Chapter 8

Annexes

8.1 Detailed specifications of the computer system used

The technical specifications of the hardware in the computational system used for this thesis are detailed below:

1. Motherboard: 1 x HUANANZHI X99Dual-F8D
    - Memory Standard: Four channels DDR4 1866/2133/2400MHz (max. memory capacity is 8x32G)
    - Expansion slots: 3xPCI express 3.0 x 16, 1xMx2 NVME, 1xM.2 NGFF interface
    - Network interface: RTL8111H gigabit ethernet card
2. CPU: 2 x Intel(R) Xeon(R) E5-2699V4 @ 2.10GHz
    - Number of cores: 22
    - Number of threads: 44
    - Processor base frequency: 2.20 GHz
    - Max turbo frequency: 3.60 GHz
    - Bus Speed: 9.6 GT/s
    - Thermal design power: 145 W
    - Case temperature: 79 ^{o}C
3. RAM: 1 x SAMSUNG M386A4G40EM2-CRC
    - DDR: DDR4

- Density: 32 GB
- Speed: 2400 Mbps
- Number of Pins: 288
- Dimm type: LRDIMM
- Operating temperature: 0 - 85 ^{o}C

4. Storage device: 1 x Samsung SSD 970 EVO plus
    - Form factor: M.2
    - Interface: PCIe gen 3.0 x4, NVMe
    - Sequential read: Up to 3'500 MB/s
    - Sequential write: Up to 3'300 MB/s
    - Random read (4KB, QD32): Up to 600'000 IOPS
    - Random write (4KB, QD32): Up to 550'000 IOPS
    - Random read (4KB, QD1): Up to 19'000 IOPS
    - Random write (4KB, QD1): Up to 60'000 IOPS
    - Operating temperature: 0 - 70 ^{o}C
    - Average power consumption: 6.0 W
5. GPU: 1 x GeForce GTX 1660 SUPER (connected through Timack 20cm riser PCIe extension cable B08BR7NB3W)
    - CUDA driver version / runtime version: 11.2/11.2
    - CUDA capability major/minor version number: 7.5
    - Total amount of global memory: 5944 MBytes (6233260032 bytes)
    - (22) multiprocessors, (64) CUDA cores/MP: 1408 CUDA cores
    - GPU max clock rate: 1785 MHz (1.78 GHz)
    - Memory clock rate: 7001 Mhz
    - Memory bus width: 192-bit
    - L2 cache size: 1572864 bytes
    - Maximum texture dimension size (x,y,z) 1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)

- Maximum layered 1D texture size, (num) layers 1D=(32768), 2048 layers
- Maximum layered 2D texture size, (num) layers 2D=(32768, 32768), 2048 layers
- Total amount of constant memory: 65536 bytes
- Total amount of shared memory per block: 49152 bytes
- Total shared memory per multiprocessor: 65536 bytes
- Total number of registers available per block: 65536
- Warp size: 32
- Maximum number of threads per multiprocessor: 1024
- Maximum number of threads per block: 1024
- Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
- Max dimension size of a grid size (x,y,z): (2147483647, 65535, 65535)
- Maximum memory pitch: 2147483647 bytes
- Texture alignment: 512 bytes
- Maximum operating temperature: 93 ^{o}C
- Power Consumption: 125 W
- Concurrent copy and kernel execution: Yes with 3 copy engine(s)
- Run time limit on kernels: Yes
- Integrated GPU sharing host memory: No
- Support host page-locked memory mapping: Yes
- Alignment requirement for surfaces: Yes
- Device has ECC support: Disabled
- Device supports unified addressing (UVA): Yes
- Device supports managed memory: Yes
- Device supports compute preemption: Yes
- Supports cooperative kernel launch: Yes
- Supports multiDevice co-op kernel launch: Yes
- Device PCI domain ID / bus ID / location ID: 0 / 2 / 0

6. GPU: 4 x Tesla K80

- CUDA driver version / runtime version: 11.2/11.2

- CUDA capability major/minor version number: 3.7
- Total amount of global memory: 11441 MBytes (11996954624 bytes)
- (13) multiprocessors, (192) CUDA cores/MP: 2496 CUDA cores
- GPU max clock rate: 824 MHz (0.82 GHz)
- Memory clock rate: 2505 Mhz
- Memory bus Width: 384-bit
- L2 Cache Size: 1572864 bytes
- Maximum texture dimension size (x,y,z): 1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
- Maximum layered 1D texture size, (num) layers 1D=(16384), 2048 layers
- Maximum layered 2D texture size, (num) layers 2D=(16384, 16384), 2048 layers
- Total amount of constant memory: 65536 bytes
- Total amount of shared memory per block: 49152 bytes
- Total shared memory per multiprocessor: 114688 bytes
- Total number of registers available per block: 65536
- Warp size: 32
- Maximum number of threads per multiprocessor: 2048
- Maximum number of threads per block: 1024
- Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
- Max dimension size of a grid size (x,y,z): (2147483647, 65535, 65535)
- Maximum memory pitch: 2147483647 bytes
- Texture alignment: 512 bytes
- Maximum operating temperature: 45 ^{o}C
- Power consumption: 150 W
- Concurrent copy and kernel execution: Yes with 2 copy engine(s)
- Run time limit on kernels: Yes
- Integrated GPU sharing host memory: No
- Support host page-locked memory mapping: Yes
- Alignment requirement for surfaces: Yes

- Device has ECC support: Enabled
- Device supports unified addressing (UVA): Yes
- Device supports managed memory: Yes
- Device supports compute preemption: No
- Supports cooperative kernel launch: No
- Supports multiDevice co-op kernel launch: No
- Device PCI domain ID / bus ID / location ID: 0 / 5 / 0

References

[1] S. Raschka and V. Mirjalili, *Python machine learning: Machine learning and deep learning with python, scikit-learn, and tensorflow 2*. Packt Publishing, 3rd ed., 2019.

[2] G. Bonaccorso, *Machine learning algorithms*. Packt Publishing, 2017.

[3] Z. Li, Y. Ming, L. Yang, and J.-H. Xue, "Mutual-learning sequence-level knowledge distillation for automatic speech recognition," *Neurocompting*, vol. 428, pp. 259–267, 2021.

[4] C. Piciarelli and G. L. Foresti, "Drone patrolling with reinforcement learning," in *Proceedings of the 13th International Conference on Distributed Smart Cameras*, ICDSC 2019, (New York, NY, USA), Association for Computing Machinery, 2019.

[5] A. F. Bulagang, J. Mountstephens, and J. Teo, "Multiclass emotion prediction using heart rate and virtual reality stimuli," *Journal of Big Data*, vol. 8, no. 1, 2021.

[6] M. Zitnik, F. Nguyen, B. Wang, J. Leskovec, A. Goldenberg, and M. M. Hoffman, "Machine learning for integrating data in biology and medicine: Principles, practice, and opportunities," *Information Fusion*, vol. 50, pp. 71–91, 2019.

[7] N. Syam and A. Sharma, "Waiting for a sales renaissance in the fourth industrial revolution: Machine learning and artificial intelligence in sales research and practice," *Industrial Marketing Management*, vol. 69, pp. 135–146, 2018.

[8] M. Salama, H. A. Kader, and A. Abdelwahab, "An analytic framework for enhancing the performance of big heterogeneous data analysis," *International Journal of Engineering Bussiness Management*, vol. 13, 2021.

[9] N. El Kamel, M. Eddabbah, Y. Lmoumen, and R. Touahni, "A smart agent design for cyber security based on honeypot and machine learning," *Security and Communications Networks*, vol. 2020, 2020.

[10] T. Ahmad, D. Zhang, C. Huang, H. Zhang, N. Dai, Y. Song, and H. Chen, "Artificial intelligence in sustainable energy industry: Status Quo, challenges and opportunities," *Journal of Cleaner Production*, vol. 289, 2021.

[11] A. Ampountolas and M. P. Legg, "A segmented machine learning modeling approach of social media for predicting occupancy," *International Journal of Contemporary Hospitality Management*, 2021.

[12] Clarivate, "Web of science," *January 30, 2022 (recovery date), of Web of Science website: https://bit.ly/35feTtV.*

[13] Google, "machine learning," *January 30, 2022 (recovery date), of Google Trends website: https://bit.ly/3lW6BuT.*

[14] Google, "parallel computing," *January 30, 2022 (recovery date), of Google Trends website: https://bit.ly/3lZaKyf.*

[15] J. Han and B. Sharma, *Learn CUDA programming: A beginner's guide to GPU programming and parallel computing with CUDA 10.x and C/C++*. Packt Publishing, 2019.

[16] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM International Conference on Multimedia*, MM '14, (New York, NY, USA), p. 675–678, Association for Computing Machinery, 2014.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[18] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, (USA), p. 265–283, USENIX Association, 2016.

[19] T. S. Ajani, A. L. Imoize, and A. A. Atayero, "An overview of machine learning within embedded and mobile devices–optimizations and applications," *Sensors*, vol. 21, no. 13, 2021.

[20] Y. Chen, B. Zheng, Z. Zhang, Q. Wang, C. Shen, and Q. Zhang, "Deep learning on mobile and embedded devices: State-of-the-art, challenges, and future directions," *ACM Comput. Surv.*, vol. 53, no. 4, 2020.

[21] S. Branco, A. G. Ferreira, and J. Cabral, "Machine learning in resource-scarce embedded systems, fpgas, and end-devices: A survey," *Electronics*, vol. 8, no. 11, 2019.

[22] PyTorch, "Pytorch documentation," *January 30, 2022 (recovery date), of PyTorch website: https://bit.ly/2Rd9Cfz.*

[23] H. Chu, S. Kim, J.-Y. Lee, and Y.-K. Suh, "Empirical evaluation across multiple gpu-accelerated dbmses," in *Proceedings of the 16th International Workshop on Data Management on New Hardware*, DaMoN '20, (New York, NY, USA), Association for Computing Machinery, 2020.

[24] T. B. Rolinger, T. A. Simon, and C. D. Krieger, "An empirical evaluation of allgatherv on multi-gpu systems," in *Proceedings of the 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGrid '18, p. 123–132, IEEE Press, 2018.

[25] R. H. Hariri, E. M. Fredericks, and K. M. Bowers, "Uncertainty in big data analytics: survey, opportunities, and challenges," *Journal of Big Data*, vol. 6, no. 1, 2019.

[26] Glassdoor, "Glassdoor job market report - November 2020," *January 30, 2022 (recovery date), of Glassdoor website: https://bit.ly/3soYlpM*, 2020.

[27] LinkedIn, "Jobs on the rise in 2021," *January 30, 2022 (recovery date), of LinkedIn website: https://bit.ly/3Hh1xuW*, 2021.

[28] LinkedIn, "México empleos en auge," *January 30, 2022 (recovery date), of LinkedIn website: https://bit.ly/3s9WAxY*, 2021.

[29] Kaggle, "State of data science and machine learning 2021," *January 30, 2022 (recovery date), of Kaggle website: https://bit.ly/3EdKg3x*, 2021.

[30] K. Čapek, *R.U.R. - Rossum's Universal Robots*. White Press, 2014.

[31] Y. Frumer, "The short, strange life of the first friendly robot: Japan's Gakutensoku was a giant pneumatic automaton that toured through Asia-until it mysteriously disappeared," *IEEE Spectrum*, vol. 57, no. 6, pp. 42–48, 2020.

[32] W. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[33] J. McCarthy, M. L. Minsky, and C. E. Shannon, "A proposal for the Dartmouth summer research project on artificial intelligence - August 31, 1955," *AI Magazine*, vol. 27, pp. 12–14, WIN 2006.

[34] S. J. Russell and P. Norvig, *Artificial intelligence: A modern approach*. Upper Saddle River: Prentice Hall, 3rd ed., 2010.

[35] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, 1959.

[36] T. G. Dietterich, *Machine learning*, p. 1056–1059. GBR: John Wiley and Sons Ltd., 2003.

[37] Q. Bi, K. E. Goodman, J. Kaminsky, and J. Lessler, "What is machine learning? A primer for the epidemiologist," *American Journal of Epidemiology*, vol. 188, no. 12, pp. 2222–2239, 2019.

[38] Z. Yu, A. Bedig, F. Montalto, and M. Quigley, "Automated detection of unusual soil moisture probe response patterns with association rule learning," *Environmental Modelling and Software*, vol. 105, pp. 257–269, 2018.

[39] J. McCarthy, "History of lisp," *SIGPLAN Not.*, vol. 13, no. 8, p. 217–223, 1978.

[40] P. Norvig, *Paradigms of artificial intelligence programming: case studies in common LISP*. San Francisco, Calif: Morgan Kaufman Publishers, 1992.

[41] GitHub, "Where the world builds software," *January 30, 2022 (recovery date), of GitHub website: https://github.com/.*

[42] Dlib C++ Library, "The library," *January 30, 2022 (recovery date), of Dlib C++ Library website: https://bit.ly/3g0gdCv.*

[43] scikit-learn, "Api reference," *January 30, 2022 (recovery date), of scikit-learn website: https://bit.ly/3uHxmGm.*

[44] TensorFlow, "Tensorflow core v2.7.0," *January 30, 2022 (recovery date), of TensorFlow website: https://bit.ly/3cahCFb.*

[45] D. Xin, H. Miao, A. G. Parameswaran, and N. Polyzotis, "Production machine learning pipelines: Empirical analysis and optimization opportunities," *CoRR*, vol. abs/2103.16007, 2021.

[46] C. Miranda Meza, "Mortrack_ML_Library," *January 30, 2022 (recovery date), of GitHub website: https://bit.ly/3zrP42M*, 2021.

[47] F. Botella, A. Peñalver, M. Quesada-Martínez, F. Bermejo, and F. Borrás, "Teaching the sequential programming concept using a robotic arm in an interactive museum," in *Proceedings of the XX International Conference on Human Computer Interaction*, Interacción '19, (New York, NY, USA), Association for Computing Machinery, 2019.

[48] M. Amoretti, "Review of elements of parallel computing," *SIGACT News*, vol. 51, no. 3, p. 10–13, 2020.

[49] C. John, G. Max, and M. Ty, *Professional CUDA C programming*. GBR: Wrox Press Ltd., 1st ed., 2014.

[50] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), (New York, NY, USA), p. 483–485, Association for Computing Machinery, 1967.

[51] J. L. Gustafson, "Reevaluating amdahl's law," *Commun. ACM*, vol. 31, no. 5, p. 532–533, 1988.

[52] P. Czarnul, J. Proficz, and K. Drypczewski, "Survey of methodologies, approaches, and challenges in parallel programming using high-performance computing systems," *Scientific Programming*, vol. 2020, 2020.

[53] R. Walpole, *Probabilidad y estadistica para ingenieria y ciencias*. Pearson Educación, 8th. ed., 2007.

[54] A. M. Legendre, *Nouvelles methodes pour la determination des orbites des cometes [microform] / par A.M. Legendre*. F. Didot Paris, 1805.

[55] A. Graham, *Kronecker products and matrix calculus: with applications*. Horwood ; Halsted Press Chichester : New York, 1981.

[56] N. Bacaër, *Verhulst and the logistic equation (1838)*, pp. 35–39. London: Springer London, 2011.

[57] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pp. 144–152, ACM Press, 1992.

[58] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Cornell Aeronautical Laboratory. Report no. VG-1196-G-8, Spartan Books, 1962.

[59] W. Mcculloch and W. Pitts, "A logical calculus of ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 127–147, 1943.

[60] A. Cauchy, "Methode generale pour la resolution des systemes d'equations simultanees," *Académie des sciences*, vol. 25, pp. 536–538, 1847.

[61] C. Miranda Meza, "CenyML/databases/," *January 30, 2022 (recovery date), of GitHub website: https://github.com/Mortrack/CenyML/tree/master/databases*, 2022.

[62] NumPy, "Numpy 1.22.0 released," *February 01, 2022 (recovery date), of NumPy website: https://numpy.org/.*

[63] Python, "statistics - mathematical statistics functions," *February 01, 2022 (recovery date), of Python website: https://bit.ly/3IU24n5.*

[64] statsmodels, "statsmodels," *February 01, 2022 (recovery date), of statsmodels website: https://bit.ly/3HkZ0Ql.*

[65] BigDL, "Bigdl project," *January 30, 2022 (recovery date), of BigDL website: https://bit.ly/2S29qQD.*

[66] G. Cumming, F. Fidler, and D. L. Vaux, "Error bars in experimental biology ," *Journal of Cell Biology*, vol. 177, no. 1, pp. 7–11, 2007.

[67] C. Miranda Meza, "CenyML/Validation_of_CenyML/," *January 30, 2022 (recovery date), of GitHub website: https://github.com/Mortrack/CenyML/tree/master/Validation_of_CenyML*, 2022.

[68] C. Miranda Meza, "CenyML," *February 25, 2022 (recovery date), of GitHub website: https://github.com/Mortrack/CenyML*, 2022.

[69] C. Miranda Meza, "CenyML/Arduino_Examples/," *February 07, 2022 (recovery date), of GitHub website: https://github.com/Mortrack/CenyML/tree/master/Arduino_Examples*, 2022.

[70] C. Miranda Meza, "CenyML/STM32F446RE_Examples/," *February 07, 2022 (recovery date), of GitHub website: https://github.com/Mortrack/CenyML/tree/master/STM32F446RE_Examples*, 2022.

[71] Huananzhi, "X99dual-f8d motherboard," *March 02, 2022 (recovery date), of Huananzhi website: https://bit.ly/3MlwL7o*.

[72] Intel, "Intel xeon processor e5-2699 v4," *March 02, 2022 (recovery date), of Intel website: https://intel.ly/3MjoksY*.

[73] Samsung, "M386a4g40em2-crc," *March 02, 2022 (recovery date), of Samsung website: https://bit.ly/3HBIMlo*.

[74] Samsung, "970 evo plus," *March 02, 2022 (recovery date), of Samsung website: https://bit.ly/3hzcCwc*.

[75] Nvidia, "The 16 super series geforce gtx 1660 super," *March 02, 2022 (recovery date), of Nvidia website: https://bit.ly/3K5lS7w*.

[76] T. store, "Pcie extension cable pcie riser cable 16x pci-e 16x flexible extension cable (pcie extension cable 16x 90 degree cable 20cm)," *March 02, 2022 (recovery date), of Amazon website: https://amzn.to/3MhkK2F*.

[77] Nvidia, "Nvidia tesla k80," *March 02, 2022 (recovery date), of Nvidia website: https://bit.ly/3vCwo27*.

[78] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.