POLITECNICO DI MILANO

School of Industrial and Information Engineering
Department of Aerospace Science and Technology

Master of Science in

AERONAUTICAL ENGINEERING

A NEURAL NETWORK INVESTIGATION FOR AN AERODYNAMIC PREDICTION

Academic supervisor: Professor Maurizio QUADRIO
Institute supervisor: Dr.-Ing. habil. Markus RÜTTEN

Candidate: Andrea SPINI
ID: 899465

Deutsche Zentrum für Luft- und Raumfahrt (DLR)
Institute of Aerodynamics and Flow Technology

Academic Year 2019-2020

Acknowledgments

I would, first, like to express my deep gratitude to Dr.-Ing. habil. Markus Rütten for the opportunity to work at the German Aerospace Center (DLR), for his supervision, for his availability and for his constant support throughout my stay in Göttingen.

I would like, also, to express my gratitude to professor Maurizio Quadrio for giving me, in collaboration with Dr. Markus Rütten, the chance to work on this project and to develop it at DLR. I am also grateful for his availability and support.

A gratitude must be expressed to the DLR institute of Aerodynamics and Flow Technology. Especially, I am grateful to all the employees of the High Speed Configurations department, who have always been available and supportive. In particular, I would like to thank the following PhD students who never failed to offer me their experiences and support: Francesco Tocci, Paul Hoffman, Michael Werner and Simon Pascal.

This work comes at the end of my career at *Politecnico di Milano*, where I have studied for the last five years. I consider myself blessed for the opportunity to share my course of study with amazing people, mates and especially, friends. Thus, I have to thank: Ylenia, Giulia, Nicole, Riccardo, Simone, Emanuele, Andrea, Artemio, Tommaso e Matteo. Their friendship, that goes beyond the boundaries of Bovisa's campus, is an unexpected gift from *Politecnico di Milano*. I also have to thank all the other mates that I met met at *Politecnico di Milano*, especially with whom I shared the first year of bachelor.

Furthermore, I have to say thank you to my friends from Göttingen. Your faces, your names, your stories will always be part of me. I cannot name everyone but I would like to mention "*the fantastic four*", I could add pages about you, the "*Roma-Siena-Bari-Treviso*"group, the "*Ratatouille*", "*the fellowship of the ring*"and the friends of "*Warum nicht*".

A special thank goes to my friends from Lecco: to the old friends from "*costa la fa posta* "and to the ones from "*capitone di Natale* ". I would also like to thank Gioele, Sarah and Noemi for their friendship and their constant support.

Through these years I had one constant, my family. I would like to thank my parents. They gave me the possibility to follow my dreams and they are an example to follow. I am also grateful to my sister Elena for being stronger than she thinks and to my brother Alessandro, a guide, my valedictorian and a friend.

Finally, a special thought has to be spared to Elena R. .

Contents

List of Figures

List of Tables

Abstract

In the present thesis a Machine Learning approach in Fluid Mechanics field was investigated. In particular Artificial Neural Networks were used to predict lift and drag coefficient of NACA 4 digit airfoils.
In the last years the application of Artificial Intelligence and in particular of Machine Learning to scientific disciplines increased substantially. Machine Learning offers techniques to extract information and knowledge from data and it provides the possibility to handle with massive quantitative of data.
The ojective of the related work was to investigate how Machine Learning is working, in particular Neural Networks, and how it has to be applied in order to make a prediction. The preliminary phase of the work was to create the data-set necessary for the secondary phase, the Neural Network analysis. The generation of the dataset involved CFD simulations. Those were performed with DLR-TAU code, a finite volume method for RANS equations. A tool as a code for RANS equations were used because of its ability to capture the aerodynamic coefficients of interest in the related work. The preliminary phase includes also all the steps that a CFD simulation concerns: CAD generation performed with Geocreate, mesh generation with Centaur and numerical simulation with DLR-TAU code. The related theoretical background is given in chapter 1, instead in chapter 3 numerical simulations are presented. In the context of neural network approach the software package Google Tensorflow 2 via Python3 interface was used. Therefore, in the context of this thesis, artificial neural networks were used to manage lift and drag coefficient generated from CFD simulations. This work presents in chapter 2 an overview of what is Machine Learning and a detailed introduction to artificial neural networks. Final results and considerations are shown in chapter 4.

Sommario

Nella presente tesi è stato investigato l'utilizzo dell'apprendimento automatico (noto anche come Machine Learning) in campo fluidodinamico. In particolare, le reti neurali artificiali sono state considerate per predire il coefficiente di portanza e il coefficiente di resistenza per profili appartenenti alla serie NACA 4-digit.
Negli ultimi anni l'utilizzo dell'apprendimento automatico e in particolare delle reti neurali artificiali è cresciuto notevolmente. L'apprendimento automatico prevede approcci che forniscono la possibilità di estrarre informazioni dai dati di interesse e permette anche la gestione di elevate quantità di dati.
L'obiettivo di questo elaborato era quello di investigare i principi fondamentali dell'apprendimento automatico, in particolare delle reti neurali artificiali, e come debbano essere applicate per generare una previsione. La fase preliminare ha compreso la generazione dei dati necessari per la seconda fase, l'analisi tramite le reti neurali artificiali. Le generazione dei dati necessari ha coinvolto simulazioni CFD. Le ultime state eseguite tramite il codice DLR-TAU, un metodo numerico ai volumi finiti per le equazioni RANS. Uno strumento come un codice per le RANS è stato preso in considerazione per le la sua abilità nel "catturare" i coefficienti aerodinamici di interesse in questo progetto. La fase preliminare include anche tutti i preparativi per una simulazione CFD: la generazione del modello, il software Geocreate è stato utilizzato a tale scopo; la generazione della griglia, effettuata con Centaur ed infine la simulazione numerica, svolta con il codice DLR-TAU. Le relative conoscenze teoriche sono fornite nel capitolo 1, invece nel capitolo 3 le simulazioni numeriche sono presentate. Riguardo all'approccio con le reti neurali è stato utilizzata la libreria software Tensorflow di Google tramite interfaccia Python3. Dunque, in questo progetto di tesi le reti neurali sono utilizzate per gestire i coefficienti di portanza e di resistenza generati tramite le simulazioni CFD. Nel capitolo 2 sono introdotti un riepilogo dell'apprendimento automatico e una dettagliata descrizione delle reti neurali artificiali. I risultati e le considerazioni finali sono presentati nel capitolo 4.

Introduction

Machine learning (ML) and Artificial Intelligence (AI) are nowadays of great interest in several scientific fields. Aeronautical and aerospace engineering are two of these fields. The aerospace industry is constantly looking for effective ways to speed up development processes in order to meet the growing demand as well as deliver high-quality components. Artificial intelligence can be applied to several different sectors of the aerospace industry such as air traffic management, enhance operational efficiency, product design, customer service, pilot training and research. In the related work Artificial Intelligence is applied to computational fluid dynamics (CFD). Computational Fluid Dynamics (CFD) has become a fundamental tool in the aerospace industry. Design and evaluation of aircraft aerodynamics is executed by means of CFD. During the last decades the computational resources rose rapidly. Despite this, CFD simulations (DNS, RANS and LES) require a computational effort. Artificial intelligence and in particular machine learning can be involved to handle massive amount of data to extract knowledge from involved data. Machine Learning provides a framework that can be tailored to address many challenges in fluid mechanics, such as reduced-order modeling, experimental data processing, shape optimization, turbulence closure modeling and control. Machine learning facilitate automation of tasks and augment human domain knowledge.
The topic of the related work was the investigation of machine learning techniques and their applicability the aerodynamic problem of the prediction of drag and lift coefficient for different airfoils under different on-flow conditions. Specifically, the objective was to investigate working principles and design constraints of artificial neural networks (ANN or NN) in aerodynamic research. Precisely the work involves prediction of drag and lift coefficient for NACA 4-digits airfoils. Airfoils have been investigated under two different on-flow conditions, $Mach = 0.2$ and $Mach = 0.5$, for several different angles of attack (from -5^o to 25^o with $\Delta = 1^o$ and additionally at 30^o). Simulations were performed with the DLR-TAU code, which implements finite volume method for RANS equations. The data-set used for the neural networks analysis has been specifically designed. The preliminary part of the work take into account all the steps required for a CFD simulations and the generations of the data-set. The secondary part concerns the Neural Network analysis.
The thesis is organized in four chapters, whose main objectives and contents are summarized below.

Chapter 1 This chapter provides the theoretical fluid dynamics background required to understand the CFD simulations performed and to generate the mesh in an appropriate way. This chapter starts presenting the details of the boundary layer especially on airfoil profiles. Then the law of the wall is introduced. The governing equations are presented with the steps necessary to reach the RANS formulation. The closure problem of RANS equation is closed with turbulence

models, here Spalart-Allmaras model and $K - \omega$ model are explained. Basic concepts of finite volume methods (FVM) are presented.

Chapter 2 This chapter is mainly devoted to provide Neural Networks work principles. Parameters required for a Neural Network and introduced and then explained with more details. Furthermore, the first part of the chapter starts from Artificial Intelligence history switching then to a more specific explanation of Machine Learning field. The objective of this chapter is provide the necessary theoretical knowledge to apply NN to a case study.

Chapter 3 This chapter introduced the work done in order to generate the data-set. All the steps of the procedure are briefly introduce. The airfoils coordinates were recovered from http://airfoiltools.com/. From these coordinates using the CAD program Geocreate the airfoil 3D CAD model was generated. The discretization and the mesh generation were performed with Centaur, a mesh generator. The CFD simulations here considered are 2D simulation. Before the mesh was generated, the transition from 3D to 2D model was performed. The DLR-TAU code is introduced with dual grid approach, time discretization and spatial discretization. Lastly, the CFD results are presented with a short description.

Chapter 4 This chapter is dedicated to NN approach. At the beginning a brief overview of machine learning for fluid dynamics is given. Then the Neural Networks analysis performed is explained and presented. The analysis was executed with the software package Google Tensorflow 2 via Python3 interface. Additionally the API keras was used. It works on the top of tensorflow and it offers a consistent and simple framework. The first part is related to investigation of the hyper-parameters (what hyper-parameters are is explained in chapter 2). The second part is devoted to investigate the generalization capability of different architectures. This investigation includes also further analysis for different input data settings. A prediction example is displayed with a brief description.

Chapter 1

Foundamentals of Fluid Dynamics

This chapter deals with the introduction of fundamentals of fluid dynamics. In particular from the general view of the flow around an airfoil, the law of the wall. Then Navier-Stokes equations, RANS and turbulence models are introduced. These are the basis to understand DLR TAU-code used for CFD simulations in order to create data-set involved in related work. The presented chapter follows the structure used in [1] and [2]

1.1 Boundary layer

The details of the flow within the boundary layer are very important for many problems in aerodynamics, including wing's stall, the skin friction drag and the heat transfer that occurs in high speed flight. In CFD simulations, like the ones here involved, where the results are the aerodynamic coefficients, a good resolution of the boundary layer is necessary to reach realistic results. Here, some theoretical notions about the boundary layer are given. In chapter 3 a practical application will be shown.

As an object moves through a fluid, or as a fluid moves around an object, the molecules of the fluid are disturbed around the object. The area of the flow field close to the object is strongly influenced by the viscosity. This area is defined as boundary layer. Roughly speaking the boundary layer is a thin layer of fluid close to the surface in which the velocity profile changes from zero at the surface to the free stream value away from the surface. Further, boundary layers may be either laminar , or turbulent. Due to this is necessary to introduce the Reynolds number because observing the Reynolds number it's possible to distinguish between a laminar or turbulent boundary layer

$$Re = \frac{inertial force}{viscous force} = \frac{\rho u L}{\mu} = \frac{uL}{\nu} \quad . \tag{1.1}$$

This number helps to determine which flow regime is present: laminar or turbulent. Laminar regime is characterized by a flow with parallel layers and undisturbed flow path. Turbulent flow is a flow regime characterized by chaotic property changes, irregular fluctuation motion. Turbulence is a phenomenon not fully clarified yet and a clear definition doesn't exist. It can be add that turbulent flows are characterized by unsteady vortices. The process that lead a turbulent flow from a laminar one is known as transition. This transition, in a free-stream flow, start to occur at between $Re = 2000$. Transition can even happen at $Re = 40000$ in extreme case. When a

turbulent flow is present also the turbulence viscosity or eddy viscosity has to be taken into account, which has no exact expression. It has to be modelled as shown in the previous chapter. Figure 1.1 shows typical behaviour of laminar and turbulent boundary layers velocity profiles. The thickness of the velocity boundary layer is normally defined as the distance from the object surface to the point at which the flow velocity is 99% of the free-stream velocity. The latter is also called momentum thickness.

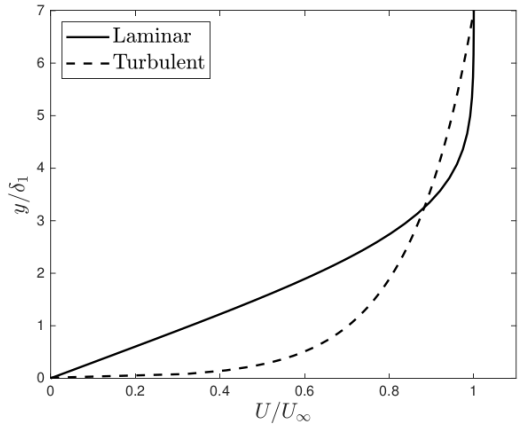


Figure 1.1: Velocity profiles for laminar and turbulent boundary layers. Moreover, y defines the wall distance, δ_1 is the boundary layer thickness, U is the free-stream velocity or inviscid flow-fled velocity and $\bar{u}$ represent the local velocity [3].

For a flat plate those expression are given:

- laminar boundary layer

$$\delta = \frac{5.2x}{\sqrt{Re_x}}, \tag{1.2}$$

- turbulent boundary layer

$$\delta = \frac{0.37x}{{Re_x}^{0.2}}, \tag{1.3}$$

where x represents the distance from the leading edge of the flat plate and Re_x is the local Reynolds number, at position x.

The boundary layer give to the considered object an "effective" shape which is slightly different from the physical shape.
The boundary layer may also separate as shown in figure 1.2. Boundary layer separation occurs when a the flow slows down and when an adverse pressure gradient occurs. Separation leads to an "effective" shape much different to the original one causing a dramatic decrease in lift and increase in drag. When this happens the profile has stalled.

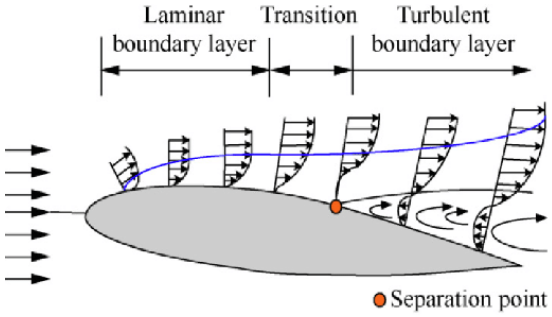


Figure 1.2: Boundary layer separation and transition over a wing profile [4].

1.2 Law of the wall

The structure of this paragraph is equal to the previous one.The theory is here introduced and it will switch to a practical choice in chapter 3 in order to have an good mesh resolution and to well solve the boundary layer. As mentioned before It's necessary to have good result in particular in terms of drag.

The law of the wall defines that the mean velocity U can be correlated in term of shear stress τ_w, the distance from the surface y, and the fluid properties ρ (density) and μ (molecular viscosity) [5]. It's derived assuming that the turbulence near the surface is function only of the flow conditions related to the wall.
It's necessary to introduce the friction velocity defined as follows:

$$u_\tau = \left(\frac{\tau_w}{\rho}\right)^{1/2} \tag{1.4}$$

and it's also worth it to introduce dimensionless wall distance y^+ with equation 1.5 and dimensionless velocity u^+ with equation 1.6.

$$y^+ = \frac{u_\tau y}{\nu} \ , \tag{1.5}$$

$$u^+ = \frac{U}{u_\tau} \ , \tag{1.6}$$

The law of the wall is a logarithmic law which has the following form:

$$u^+ = \frac{1}{K}\, ln(y^+) + C \ , \tag{1.7}$$

where K and C are empirical constants ($K \approx 0.41$ and $C \approx 5$ [5]).
In a turbulent flow the turbulent fluctuations must go to zero at the wall since the stream velocity is zero. Therefore, a very small layer next to the wall, in which the flow is essentially laminar, will always exist. Moreover in this layer :

$$\tau_w = \rho\nu\frac{\partial u}{\partial y} \ . \tag{1.8}$$

Using equation 1.6 and 1.5 the just introduced 1.8 becomes simply:

$$u^+ = y^+ \tag{1.9}$$

and it is valid in the laminar sub-layer. The fully-turbulent flow begins about $y^+ = 30$ and the laminar sub-layer extends up to about $y^+ = 5$. The intermediate region is defined as "buffer zone", see figure 1.3 as reference.

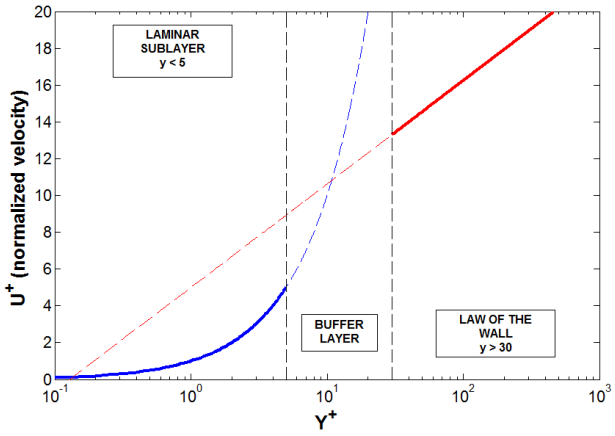


Figure 1.3: Dimensionless velocity profile, u^+, in term of dimensionless wall distance y^+ [6].

In the following part Navier-stokes equations and RANS are introduced. RANS equations are appropriate *tools* used for engineering purpose when the object of simulations is a turbulent flow.

1.3 Governing equations

DLR TAU-code is implemented for solving Reynolds-averaged Navier-Stokes (RANS) equations. Those equations are a modified version of the exact Navier-Stokes equations. The latter are the governing equations of fluid dynamics.

1.3.1 Navier-Stokes Equations

Navier-Stokes equations consist of a time dependent continuity equation for conservation of mass, three time-dependent conservation of momentum equations and a time dependent equation for energy conservation (enery equation). These equation are derived from three fundamentals physics principles: conservation of mass, Newton's second law and first law of thermodynamic.
The flow field described by NS equations is characterized by the three components of the velocity vector $\bar{u}$ (u, v and w), by the pressure p and by the temperature T . Each of those variables is a function of the three spatial coordinates and time t.
In integral conservative form Navier-Stokes equations with external forces equal to zero can be introduced as:

$$\frac{\partial}{\partial t}\iiint_V \rho dV + \iint_{\partial V} \rho \mathbf{u}\cdot\mathbf{n}dS = 0, \tag{1.10}$$

$$\frac{\partial}{\partial t}\iiint_V \rho \mathbf{u} dV + \iint_{\partial V} \rho(\mathbf{u}\otimes\mathbf{u})\cdot\mathbf{n}dS = \iint_{\partial V} \mathbf{T}\cdot\mathbf{n}dS, \tag{1.11}$$

$$\frac{\partial}{\partial t}\iiint_V \rho E dV + \iint_{\partial V} \rho H \mathbf{u}\cdot\mathbf{ndS} = \iint_{\partial \mathbf{V}} (\mathbf{T}_\mu \mathbf{u})\cdot\mathbf{ndS} - \iint_{\partial \mathbf{V}} \mathbf{q}\cdot\mathbf{ndS}. \tag{1.12}$$

Equation 1.10 is the continuity equation. It states that the convection of mass through a control volumes is equal to the change of mass in time. Respectively, in each equations, ∂V is the surface of the control volume and V is the control volume (CV) .
Equation 1.11 is the momentum equation. The equation describes that the convection of momentum through the borders of a control volume plus the change of momentum in time is equal to the sum of external force acting on a control volume (only surface force are considered). In the momentum equation appears $\mathbf{T}$. It's the stress tensor and it can be written as:

$$\mathbf{T} = -p\mathbf{I} + 2\mu\mathbf{D} + \lambda tr(\mathbf{D})\mathbf{I}\,, \tag{1.13}$$

where the first term in the right-hand side is the pressure part ($\mathbf{T}_p$) and the last two terms are the viscous part ($\mathbf{T}_\mu$).Additionally $\mathbf{D}$ is the instantaneous strain rate tensor, λ and μ are respectively the second coefficient of viscosity and the dynamic molecular viscosity.
Equation 1.12 is the energy equation: the conbftion of energy through the borders of the control volume plus the variation of the total energy in time is equal to the work of the pressure and viscous forces plus the conduction of heat through the borders.
Equation 1.12 contains specific total energy $E = e + \frac{1}{2}\mathbf{u}\cdot\mathbf{u}$, the specific total enthalpy $H = h + \frac{1}{2}\mathbf{u}\cdot\mathbf{u}$ and the heat flux vector $\mathbf{q}$. The relation between specific internal energy, e, and specific enthalpy is $h = e + \frac{p}{\rho}$.
Furthermore the system of equations 1.10-1.12 can be condensed as follows [1]:

$$\frac{\partial}{\partial t}\iiint_V \mathbf{W} dV = -\iint_{\partial V} \mathbf{F}\cdot\mathbf{n} dS\,, \tag{1.14}$$

where

$$\mathbf{W} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{pmatrix}, \tag{1.15}$$

is the vector of conserved quantities.
Equation 1.14 is the integral conservative form. It describes the change rate of the conserved quantity, here expressed by the vector $\mathbf{W}$. This change of rate over the arbitrary control volume considered is given by the flux $\mathbf{F}$ through the boundary of the control volume.
The components of $\bar{W}$ in eq. 1.14 are: the density ρ from the continuity equation; ρu, ρv and ρw from the three equation of momentum conservation and ρE, total energy, from the energy equation. In equation 1.14 $\mathbf{F}$ is the flux density tensor composed by flux vectors in the three spatial coordinates (F, G and H):

$$\mathbf{F} = (\mathbf{F_i^c} + \mathbf{F_v^c})\cdot\mathbf{e_1} + (\mathbf{G_i^c} + \mathbf{G_v^c})\cdot\mathbf{e_2} + (\mathbf{H_i^c} + \mathbf{H_v^c})\cdot\mathbf{e_3}\ . \tag{1.16}$$

In 1.16 i and v, as indices, are used to denote inviscid and viscous contributions. $\bar{F}$, $\bar{G}$, and $\bar{H}$ are the fluxes of the conservative quantity expressed in $\bar{W}$. They are defned

as follows:

$$\mathbf{F_i^c} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ \rho Hu \end{pmatrix}, \quad \mathbf{F_v^c} = - \begin{pmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xz} \\ u\tau_{xx} + v\tau_{xy} + w\tau_{xz} + k\frac{\partial T}{\partial x} \end{pmatrix}, \tag{1.17}$$

$$\mathbf{G_i^c} = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ \rho Hv \end{pmatrix}, \quad \mathbf{G_v^c} = - \begin{pmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ \tau_{yz} \\ u\tau_{xy} + v\tau_{yy} + w\tau_{yz} + k\frac{\partial T}{\partial y} \end{pmatrix}, \tag{1.18}$$

$$\mathbf{H_i^c} = \begin{pmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ \rho Hw \end{pmatrix}, \quad \mathbf{H_v^c} = - \begin{pmatrix} 0 \\ \tau_{xz} \\ \tau_{yz} \\ \tau_{xx} \\ u\tau_{xz} + v\tau_{yz} + w\tau_{zz} + k\frac{\partial T}{\partial z} \end{pmatrix}. \tag{1.19}$$

Where τ is used to introduce stress tensor components and k is the heat conductivity. These terms involved the flux of some physical quantity such as mass flux (ρu),three components of momentum flux ($\rho u\bar{u}$, $\rho v\bar{u}$ and $\rho w\bar{u}$) and the flux of the total energy ($\rho(e + \frac{u^2}{2})\bar{u}$) that it is not explicit but it's hidden inside $\rho H\bar{u}$.
The viscous part contains the component of the stress tensor and in the last term appears the dependency from the temperature for the energy equation.
The pressure is computed by the equation of state as follows:

$$p = (\gamma - 1)\rho(E - \frac{u^2 + v^2 + w^2}{2}), \tag{1.20}$$

with γ the adiabatic coefficient.
The differential form of the system of equations 1.10-1.12 is:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = \mathbf{0}, \tag{1.21}$$

$$\frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u}) = \nabla \mathbf{p} + \nabla \cdot \mathbf{T}_\mu, \tag{1.22}$$

$$\frac{\partial (\rho E)}{\partial t} + \nabla \cdot (\rho H \mathbf{u}) = \nabla \cdot (\mathbf{T}_\mu \mathbf{u}) - \nabla \mathbf{q}. \tag{1.23}$$

1.3.2 Reynolds and Favre averaging

Reynolds-averaging is an approach applied to Naviers-Stokes equations for the modeling of turbulent flows to reduce the range of scales present these flows. In the related work the aim of the CFD simulations, as already mentioned, are the lift and drag coefficients of NACA profile. Using RANS is a pratical approach in terms of accuracy result and computation demand in order to capture this characteristics. LES and DNS are more precise but they also require a definitely higher computational effort. This approach, that leads NS-equations to RANS-equations, finds his starting point in the Reynolds decomposition: flow variables are decomposed into mean and fluctuating

parts. Lets consider ϕ, a generic instantaneous flow quantity in a turbulent flow. Then the Reynolds decomposition of ϕ is :

$$\phi = \bar{\phi} + \phi^{'}, \tag{1.24}$$

where $\bar{\phi}$ is the mean, or time average, and $\phi^{'}$ is termed the "fluctuating part".
In this case Reynolds time averaging has been used:

$$\bar{\phi} = \lim_{T\to\infty} \frac{1}{T} \int_{t^{t+T}} \phi dt. \tag{1.25}$$

The decomposition used has the following two properties:

$$\bar{\bar{\phi}} = \bar{\phi}, \quad \bar{\phi}' = 0. \tag{1.26}$$

Using two generic variables ϕ and ψ it is assumed that also these properties hold[7]:

$$\begin{gathered} \overline{\phi\psi} = \bar{\phi} + \bar{\psi} + \overline{\phi'\psi'}, \quad \overline{\bar{\phi}\phi'} = \overline{\bar{\psi}\psi'} = \overline{\bar{\phi}\psi'} = \overline{\bar{\psi}\phi'}, \\ \bar{\phi}^2 = \bar{\phi}^2 + \overline{\phi'^2}, \quad \overline{\frac{\partial\phi}{\partial t}} = \frac{\partial\bar{\phi}}{\partial t}, \quad \overline{\frac{\partial\phi}{\partial x_i}} = \frac{\partial\bar{\phi}}{\partial x_i}. \end{gathered} \tag{1.27}$$

In compressible flow is useful to apply the density-weighted averaging procedure suggested by A. Favre. This overcomes the problem that arises by using only Reynolds time-averaging approach. If only the standard Reynolds time-averaging procedure is employed for the compressible conservation equations (1.21), (1.22) and (1.23), the resulting mean equations contain additional terms that have no analogs in the former laminar equations [Wilcox, 2006]. Following the density-weighted, or mass-weighted, procedure the instantaneous flow quantity can be subdivided as before in mean and fluctuating part:

$$\phi = \tilde{\phi} + \phi^{''}, \tag{1.28}$$

with

$$\tilde{\phi} = \frac{1}{\bar{\rho}} \lim_{T\to\infty} \frac{1}{T} \int_{t^{t+T}} \rho\phi dt = \frac{\overline{\rho\phi}}{\bar{\rho}}, \tag{1.29}$$

and

$$\overline{\phi''} = -\frac{\overline{\rho'\phi'}}{\bar{\rho}} \neq 0, \tag{1.30}$$

where $\bar{\rho}$ is exactly the Reynold-time average of the density and $\rho^{'}$ its fluctuation.

1.3.3 RANS equations

Here the RANS term is used also for Favre averaged compressible Navier-Stokes equations, that are RANS for compressible flows.
Using Reynolds decomposition and Favre decomposition on the flow quantities as

$$\begin{gathered} \mathbf{u} = \tilde{\mathbf{u}} + \mathbf{u}^{''}, \quad \mathbf{e} = \tilde{\mathbf{e}} + \mathbf{e}^{''}, \quad \mathbf{T} = \tilde{\mathbf{T}} + \mathbf{T}^{''}, \quad \mathbf{h} = \tilde{\mathbf{h}} + \mathbf{h}^{''}, \\ \rho = \bar{\rho} + \rho^{'}, \quad p = \bar{p} + p^{'}, \quad \mathbf{q} = \bar{\mathbf{q}} + \mathbf{q}^{'}. \end{gathered} \tag{1.31}$$

and substituting them into (1.21), (1.22) lead to the following averaged formulation:

and (1.23)
mean continuity equation

$$\frac{\partial \bar{\rho}}{\partial t} + \nabla \cdot (\bar{\rho}\tilde{\mathbf{u}}) = 0 \,, \tag{1.32}$$

mean momentum equation

$$\frac{\partial(\bar{\rho}\tilde{\mathbf{u}})}{\partial t} + \nabla \cdot (\bar{\rho}\tilde{\mathbf{u}} \otimes \tilde{\mathbf{u}}) = -\nabla \bar{p} + \nabla \cdot \left[\tilde{\mathbf{T}}_\mu - \bar{\rho}\,\tilde{\mathbf{T}}_t\right], \tag{1.33}$$

mean total energy equation

$$\frac{\partial(\bar{\rho}\tilde{E})}{\partial t} + \nabla \cdot (\bar{\rho}\tilde{H}\tilde{\mathbf{u}}) = \nabla \cdot \left[\left(\tilde{\mathbf{T}}_\mu - \bar{\rho}\,\tilde{\mathbf{T}}_t\right)\tilde{\mathbf{u}}\right] + \nabla \cdot \left[-\bar{\mathbf{q}} - \bar{\mathbf{q}}_t + \overline{\mathbf{T}_\mu\,\mathbf{u}''} - \frac{1}{2}\overline{\rho \mathbf{u}'' \cdot \mathbf{u}''\mathbf{u}''}\right] . \tag{1.34}$$

Equations 1.32, 1.33, 1.34 are the RANS equations. It is possible to notice that additional terms arise from the time averaging of non linear terms. This term is the Favre-averaged Reynold stress tensor that incorporates the effect of turbulent motion on the mean stress:

$$\bar{\rho}\tilde{\mathbf{T}}_t = -\overline{\rho \mathbf{u}'' \otimes \mathbf{u}''}. \tag{1.35}$$

Half of the trace of this vector is defined as turbulence kinetic energy:

$$\bar{\rho}k = \frac{1}{2}\overline{\rho \mathbf{u}'' \cdot \mathbf{u}''} \,. \tag{1.36}$$

In equation 1.34 $\nabla \cdot (\bar{\mathbf{q}} + \bar{\mathbf{q}}_t)$, where $\bar{\mathbf{q}}_t = \overline{\rho h'' \mathbf{u}''}$, describes the turbulent heat transport and the heat molecular diffusion. Additionally the divergence of $\tilde{\mathbf{T}}_\mu \tilde{\mathbf{u}}$ and $\tilde{\mathbf{T}}_t \tilde{\mathbf{u}}$ correspond to the work done by molecular viscous stresses and the turbulent friction, respectively. The divergence of the last two terms in equation 1.34, $\overline{\mathbf{T}_\mu\,\mathbf{u}''}$ and $\frac{1}{2}\overline{\rho \mathbf{u}'' \cdot \mathbf{u}''\mathbf{u}''}$, are the molecular diffusion and the transport of turbulence kinetic energy [2].

1.4 Turbulence models

The Reynolds-Average Navier-Stokes system of equations is not a closed system. As mentioned before, in equations 1.33 and 1.34 arise the Favre-averaged Reynolds-stress tensor. It contains six new unknown terms. This is the closure problem of RANS equations. It has his solution in models that express the unknown terms of Reynold-stress tensor as function of mean velocity and/or other variables [7].
Different classes of models have been developed:

- Linear eddy viscosity models
  - Algebraic models or zero-equation models
  - One-equations models (Prandtl model, Spalart-Allmaras)
  - Two-equation models (K-ϵ , K-ω)
- Non linear eddy viscosity models
- Reynolds stress model (RSM)

In linear eddy viscosity models Reynolds stresses are modelled by a linear constitutive relation with the mean flow.
This linear relationship is known as Boussinesq hypothesis:

$$\bar{\rho}\tilde{\mathbf{T}}_t = 2\mu_t\left[\tilde{\mathbf{D}} - \frac{1}{3}tr(\tilde{\mathbf{D}})\mathbf{I}\right] - \frac{2}{3}\bar{\rho}k\mathbf{I}\,, \tag{1.37}$$

where μ_t is the turbulent or eddy viscosity.

Zero-equation models don't require solution of any additional equations. One-equation models introduce one turbulent transport equation, It is usually written in terms of turbulent kinetic energy k. Two-equation models add one more additional transport equation. Most often one of the transport equation is written in term of the turbulent kinetic energy k.
The second transport variable depends on the models that is being used:
in $k-\epsilon$ model turbulence dissipation ϵ is used and in $k-\omega$ model the specific turbulent dissipation (mean frequency of the turbulence or the reciprocal turbulent time scale) rate is used.

The Reynold Stress Model (RSM) represent the most complete classical turbulence model. It us also defined as Second Order Closure. Components of the Reynolds stress tensor are directly computed. These models use the exact Reynolds stress transport equation for their formulation.
Spalart-Allmaras model and $k-\omega$ model are briefly introduced (notation used is base on [1]).

1.4.1 Spalart-Allmaras model

The Spalart-Allmaras (SA) model has been presented for the first time in 1992 [8], followed by [9].

All SA models involve one additional transport equation for eddy-viscosity. Eddy viscosity appears in terms of kinematic eddy viscosity in the transport equation with a variable called as Spalart-Allmaras variable or "SA-viscosity" $\tilde{\nu}$:

$$\nu_t = \frac{\mu_t}{\bar{\rho}} = \tilde{\nu} f_{\nu 1} \tag{1.38}$$

where $f_{\nu 1}$ is a damping function defined as

$$\tilde{\nu} f_{\nu 1} = \frac{\chi^3}{\chi^3 + c_{\nu 1}^3}\,, \quad \chi = \frac{\tilde{\nu}}{\nu}\,. \tag{1.39}$$

The notation here used, with ν as the apex, is the one followed in [1]. It shows explicitly the dependency of the related term on the SA-viscosity.
Transport equations for eddy-viscosity is given by:

$$\frac{\partial(\bar{\rho}\tilde{\nu})}{\partial t} + \nabla\cdot(\bar{\rho}\tilde{\nu}\tilde{\boldsymbol{u}}) = \bar{\rho}P^{(\tilde{\nu})} + \bar{\rho}D^{(\tilde{\nu})} - \bar{\rho}\Phi^{(\tilde{\nu})} + \bar{\rho}C^{(\tilde{\nu})}\,. \tag{1.40}$$

In 1.40 $P^{(\tilde{\nu})}$, $D^{(\tilde{\nu})}$, $\Phi^{(\tilde{\nu})}$ and $C^{(\tilde{\nu})}$ represent the production, diffusion, destruction and compressibility terms, respectively. They are defined as:

$$\bar{\rho}P^{(\tilde{\nu})} = c_{b1}(1 - f_{t2})\tilde{S}\bar{\rho}\tilde{\nu}, \tag{1.41}$$

$$\bar{\rho}D^{(\tilde{\nu})} = \frac{1}{\sigma}\left\{\nabla\cdot\left[\bar{\rho}(\nu+\tilde{\nu})\nabla\tilde{\nu}\right] + c_{b2}\bar{\rho}(\nabla\tilde{\nu})^2\right\}, \tag{1.42}$$

$$\bar{\rho}\Phi^{(\tilde{\nu})} = \left(c_{w1}f_w - \frac{c_{b1}}{k^2}f_{t2}\right)\bar{\rho}\left[\frac{\tilde{\nu}}{d}\right]^2, \tag{1.43}$$

$$C^{(\tilde{\nu})} = -\frac{1}{\sigma}(\nu+\tilde{\nu})\nabla\bar{\rho}\cdot\nabla\tilde{\nu}\,. \tag{1.44}$$

In the production term, $\tilde{S}$ is a measure the velocity gradient and has different definitions based on the different SA model used. In the destruction term appears d that is the distance to the closest surface and f_w :

$$f_w = g\left[\frac{1+c_{w3}^6}{g^6+c_{w3}^6}\right]^{1/6}, \quad g = r + c_{w2}(r^6 - r)\ . \tag{1.45}$$

As for $\tilde{S}$, also different versions of r are available. In the original SA model $r = \frac{\nu}{\tilde{S}k^2d^2}$. Missing functions and costants are defined as follow:

$$\begin{aligned} &c_{b1} = 0.1355,\ \ c_{b2} = 0.622,\ \ \sigma = \frac{2}{3},\ \ k = 0.41,\ c_{w1} = \frac{c_{b1}}{k^2} + \frac{1+c_{b2}}{\sigma},\\ &c_{w2} = 0.3,\ \ c_{w3} = 2,\ \ f_{t2} = c_{t3}\,exp\{-c_{t4}\chi^2\},\ \ c_{t3} = 1.2,\ c_{t4} = 0.5\ . \end{aligned} \tag{1.46}$$

The SA-standard model refers to what is called standard SA model in [10]. The scalar gradient velocity is defined according to [9] as:

$$\tilde{S} = \begin{cases} S + \bar{S}, & \bar{S} \geq -c_{\nu 2}S \\ S + \frac{S({c_{\nu 2}}^2 S + c_{\nu 3}\bar{S})}{(c_{\nu 3} - 2c_{\nu 2})S - \bar{S}}, & \bar{S} < -c_{\nu 2}S \end{cases} \tag{1.47}$$

with $\bar{S} = \frac{\tilde{\nu}}{k^2d^2}f_{\nu 2}$, $f_{\nu 2} = 1 - \frac{\chi}{1+\chi f_{\nu 1}}$, $c_{\nu 2} = 0.7$ and $c_{\nu 3} = 0.9$.
The solution of $\tilde{\nu}$ must be always positive. When negative values of ν occur in the numerical simulations the negative-SA model can be used. In region with positive $\tilde{\nu}$ the negative-Sa model in identical to SA-standard model. In region with negative $\tilde{\nu}$ the following modification to 1.41 - 1.43 are used:

$$\bar{\rho}P_n^{\tilde{\nu}} = c_{b1}(1 - c_{t3})\bar{\rho}S\tilde{\nu}, \tag{1.48}$$

$$\bar{\rho}\Phi_n^{\tilde{\nu}} = -c_{w1}\bar{\rho}\left[\frac{\tilde{\nu}}{d}\right]^2, \tag{1.49}$$

$$\bar{\rho}D_n^{\tilde{\nu}} = \frac{1}{\sigma}\left\{\nabla\cdot\left[\bar{\rho}(\nu + f_n\tilde{\nu})\nabla\tilde{\nu}\right]\right\}. \tag{1.50}$$

S is the scalar norm of the deformation tensor (vorticity), $f_n = \frac{c_{n1}+\chi^3}{c_{n1}-\chi^3}$ and $c_{n1} = 16$ in [1].

1.4.2 K-ω model

As aforementioned this model considers two transport equations: one for turbulent kinetic energy k and one for specific dissipation rate ω. Typical form of k-equation is:

$$\frac{\partial(\bar{\rho}\tilde{k})}{\partial t} + \nabla\cdot\left(\bar{\rho}\tilde{k}\tilde{\boldsymbol{u}}\right) = \bar{\rho}P^{(k)} - \bar{\rho}\epsilon + \bar{\rho}D^{(k)}. \tag{1.51}$$

In the right-hand side of 1.51 is possible to notice production, dissipation and diffusion terms.

$$\begin{aligned} \bar{\rho}P^{(k)} &= 2\mu^{(t)}\tilde{S}^{*}_{ij}\tilde{S}_{ij} - \frac{2}{3}\bar{\rho}\tilde{k}\nabla\cdot\tilde{\boldsymbol{u}} \ , \\ \bar{\rho}\epsilon &= \beta^{(\tilde{k})}\bar{\rho}\tilde{k}\omega \\ \bar{\rho}D^{(k)} &= \nabla\cdot\left[\left(\bar{\mu}+\sigma^{(\tilde{k})}\mu^{(t)}\right)\nabla\tilde{k}\right] . \end{aligned} \tag{1.52}$$

The ω-equation is:

$$\frac{\partial(\bar{\rho}\omega)}{\partial t} + \nabla\cdot\left(\bar{\rho}\omega\tilde{U}_k\right) = \bar{\rho}P^{(\omega)} - \bar{\rho}\Phi^{(\omega)} + +\bar{\rho}C_D^{(\omega)} + \bar{\rho}D^{(\omega)} \ , \tag{1.53}$$

with production term

$$\bar{\rho}P^{(\omega)} = \gamma^{(\omega)}S_c^{(\omega)}\bar{\rho}P^{(\tilde{k})} \ , \tag{1.54}$$

dissipation term

$$\bar{\rho}\Phi^{(\omega)} = \beta^{(\omega)}\bar{\rho}\omega^2 \ , \tag{1.55}$$

the cross diffusion-term

$$\bar{\rho}C_D^{(\omega)} = \sigma^d\frac{\bar{\rho}}{\omega}\nabla\tilde{k}\cdot\nabla\tilde{\omega} \ , \tag{1.56}$$

and the diffusion term

$$\bar{\rho}D^{(\omega)} = \nabla\cdot\left[(\bar{\mu}+\sigma^{(\omega)}\mu^{(t)})\nabla\omega\right] \ . \tag{1.57}$$

In the previous equations (from 1.51 to 1.57) some terms are not specified. Those assume different definition with respect $k-\omega$ model used [1] [7].

1.5 Finite Volume Method

DLR-TAU code implements a finite volume scheme in order to solve Reynolds-average Navier Stokes equations. A brief introduction of FVM follows.

Finite Volume Method, FVM, is one the most used discretization techniques in CFD. It's a technique that transforms partial differential equations representing conservation laws over differential volumes into discrete algebraic equations over finite volumes, also called elements or cells [11].
The first step in FVM is the discretization of the problem domain into control volumes (cells or elements). Then partial differential equations of the governing are integrated over each control volume. This step results in a semi-discretized set of equations that express the conservation for the variable inside the control volume. Interpolation profiles are then used to approximate the variation of the variables across adjacent cells and relate the surface values to the cells values. The final set of equations is composed of algebraic equations.
The conservation of quantities such as mass, momentum and energy is exactly satisfied for any control volume such as for the whole problem domain. FVM is strictly conservative since it is based on the conservation of fluxes through control volumes. Fluxes between adjacent control volumes are directly balanced.

1.5.1 Basic concepts

In FVM the first step is the division of the problem's domain into control volumes. The continuous domain with is transformed in a discrete one of non overlapping cells. A continuous domain can be discretize with a structured or an unstructured grid, see fig. 1.4. Structured grids are made of quadrilateral in 2D and hexahedrons in 3D, instead unstructured grids typically employ triangles in 2D and tetrahedrons in 3D. There are also different types of variables arrangements:

- Cell-Centered scheme where control volumes are identical with grid cells, flow variables are located at the centers of the grid cells and fluxes are located at element surface;
- Vertex-Centered scheme where flow variables are stored at the vertices and and elements are constructed around the variable location by using dual mesh and dual cells.

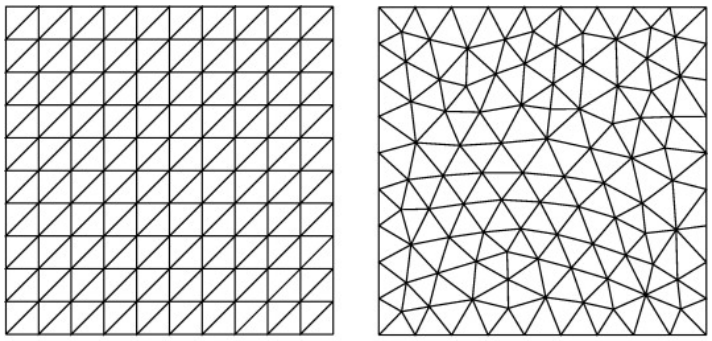


Figure 1.4: Difference between structured and unstructured grids [12]

Looking at the governing equations, the first step for them is being integrated over the control volumes in which the domain is subdivided. For simplicity it is possible to consider a single partial differential equation in conservative form:

$$\frac{\partial u}{\partial t} + \nabla \cdot \mathbf{f}(u) = 0 \tag{1.58}$$

where $f(u)$ represent the flux vector of the variable u. The latter, u, is the conserved quantity. It is possible to notice the relation between equation 1.58 and equations in the system 1.21 - 1.23 .
Integrating this equations over a control volume V lead to the integral form of the governing equation in conservative form:

$$\int_V \left(\frac{\partial u}{\partial t} + \nabla \cdot \mathbf{f}(u) \right) dV = 0 \ . \tag{1.59}$$

Using the divergence theorem equations 1.59 becomes:

$$\frac{\partial}{\partial t} \int_V u dV + \oint_{\partial V} \mathbf{f}(u) \cdot \hat{\mathbf{n}} \, dS = 0. \tag{1.60}$$

where the second integral represent the net flux across the surface of the control volume considered and $\hat{\mathbf{n}}$ represent the vector of unit outward normal. At this point a piece-wise constant cell average is introduced for each control volume:

$$u_i = \frac{1}{V_i} \int_{V_i} u \, dV, \tag{1.61}$$

where i is the index that represent the control volume considered. The computational domain is V is divided in N cell therefore $i = 1, ..., N$.
The flux at the interfaces, $\mathbf{f}(u)$ is replaced by a numerical flux $\mathbf{F}(u)$. This numerical flux has to satisfy flux conservation at adjacent control volumes and consistency. The surface integral has to be evaluated at each face. For example in a 2D problem the second integral of 1.60 is approximated as

$$\oint_{\partial V} \mathbf{f}(u) \cdot \hat{\mathbf{n}} \, dS \approx \sum_{ij \in N(i)} F_{ij}(u_i, u_j) \Delta S_{ij} \ . \tag{1.62}$$

where indexes $_{ij}$ are used to express interface between cell "i" and cell "j". Gaussian quadrature, trapezoidal rule and other choices could be taken in 3D problem. The first integral, considering always a 2D problem, of 1.60 with 1.61 can be write as:

$$\int_{\mathbf{V}} \frac{\partial u}{\partial t} = V_i \frac{\partial u_i}{\partial t} \ . \tag{1.63}$$

At this point to have a fully discrete finite volume form it's only required a time integration formula. For example using forward Euler scheme the final equation is:

$$(u_i^{n+1} - u_i^n) V_i = -\Delta t \sum_{ij \in N(i)} F_{ij}(u_i, u_j) \Delta S_{ij}, \tag{1.64}$$

where $n + 1$ and n indicate steps used for time integration.
A scheme is identified by the way the numerical flux F approximated the physical flux across each cell face or edge (edge based approach). Flux exchanges between cells is shown in chapter 3 when dual grid approach in introduced.

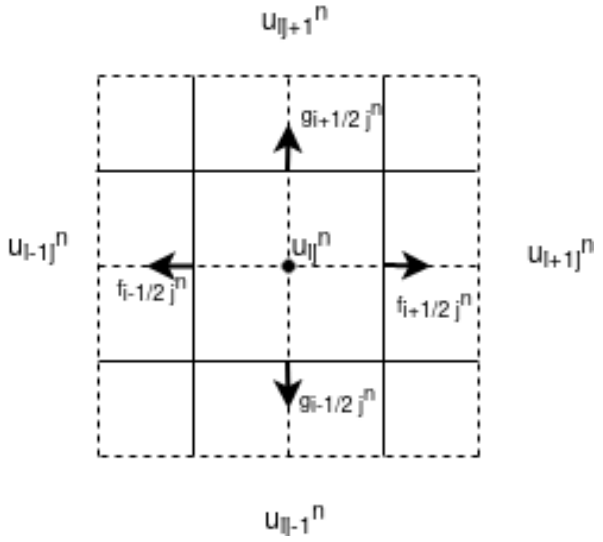


Figure 1.5: Balance of fluxes for 2D finite volume scheme:u is the stored variable; f and g are the variable's fluxes in x and y direction; "i" and "j" represent the spatial indexes and n indicates the step of time discretization.

Chapter 2

Foundamentals of Machine Learning

Machine Learning is rapidly evolving because of new computing technology. Over the last decade due to high-performance computer and due to advanced experimental techniques the quantitative of data to handle is increased. In fluid mechanics, as in other scientific disciplines, the ability to handle a massive quantity of data became mandatory. Machine learning provides a framework where it's possible to handle data and extract informations from them. From this is very clear why Machine Learning start to be a resource increasingly applied to fluid mechanics as to others topics .

In order to understand what Deep Learning means, this chapter starts from the history and the big world of Artificial Intelligence (AI) .
The Machine Learning (ML) subset is then explained. A brief overview of Machine Learning is followed by the specific introduction of Neural Networks.

2.1 From Artificial Intelligence to Neural Networks

The main goal of the chapter is to give an exhaustive description of the steps between Artificial Intelligence and Deep Learning.

1936 could be considered as the starting point of Artificial Intelligence. In this year Alan Turing applies his theories to prove that a machine could be capable of executing a cognitive process since the latter could be broken into multiple, individual steps [13]. This is the foundation of what now is called Artificial Intelligence. The term AI proposed by the programmer John McCarthy was coined in the 1956 [14]. From 1956 up to the present day the community of research made great strides. For example 1966 is the birth of the first computer program able to communicate with humans, in the 1972 AI enters the medical field helping doctors for diagnosis and treatment and in 1986 the computer is given a voice. 'Deep Blue' , a computer from IBM, beats the world chess champion in the 1997 and in 2005 a Stanford robot drove autonomously for 131 miles. Lastly, 2011 is considered the year that AI enters everyday life.

In order to clearly understand the intermediate terms between AI and Deep Learning It's necessary introduce a definition of what Artificial Intelligence is considered. This, therefore, raises two sources of ambiguity:

- a unique definition of Artificial Intelligence doesn't exist;
- Machine Learning and Artificial Intelligence terms are often used interchangeably, however there is a stark difference between them.

Artificial Intelligence(AI) could be defined as when a computer algorithm does an intelligent work. Everyday example of AI are: Google's AI-Powered Predictions (Google Maps), Ridesharing Apps (Uber).
The main problem of creating or simulating intelligence has been broken into subproblems:

- Machine Learning - ML
- Natural Language Processing- NLP
- Expert System
- Speech recognition
- Computer Vision
- Robotics
- Planning.

It has necessary to point out that Machine Leaning has a overlap region with the others subsets since it can be considered also a method used to perform all the other point.

NLP is a sub-field of computer science and artificial intelligence that enables computer system to understand and process human language.

Experts system emulate the decision making ability of human experts (f.e. suggestion for spelling error in google search box).

Speech recognition is a technology which enables a machine to understand the spoken language and translate it into a machine-readable format.

Computer Vision according to Prof. Fei-Fei Li is defined as *"a subset of mainstream artificial intelligence that deals with the science of making computers or machines visually enabled, i.e., they can analyze and understand an image"* [15].

Now It's clear that Artificial Intelligence is the super-set of Machine Learning i.e. all the Machine Learning is Artificial Intelligence but not all AI is Machine Leaning.
As aforementioned the sub-problem considered in this work is Machine learning. A common ground characterize AI and Machine Learning : a unique definition does not exist. With respect to AI the definition of ML is more consistent:
according to Oxford Dictionaries, Machine Learning can be defined as the study of computer algorithms that improve automatically through experience. This is the most general definition of Machine Learning. To be more technical and precise It is possible to introduce Tom Mitchell's widely quoted definition of Machine Learning :
" A computer program is set to learn from an experience E with respect to some task T and some performance measure P if Its performance on T as measured by P improves

with experience E" [16]. [1]
Table 1.1 further clarifies key differences between AI and ML:

Table 2.1: Key differences between AI and ML

Artificial Intelligence	**Machine Learning**
Artificial intelligence is "the theory and development of computer systems able to perform tasks normally requiring human intelligence" [17]	"machine learning is a branch of artificial intelligence and it's about construction and study of systems that can learn from data" [18]
The goal is to simulate natural intelligence to solve complex problem	The goal is to learn from data on a certain task to maximize machine's performance on that

Machine Learning in turn is categorized as it follows

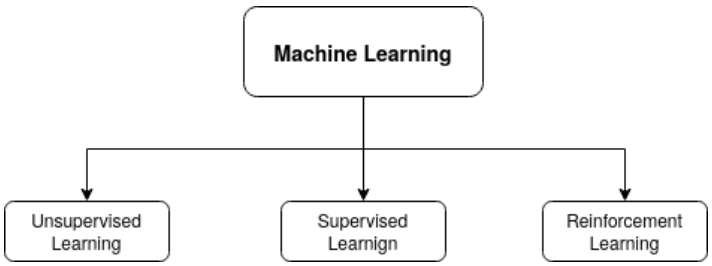


Figure 2.1: Machine Learning subsets.

With reference to figure 2.1 the subsets of Machine Learning are briefly described below:

Unsupervised Learning: It's the sub-field of Machine Learning where a computer learns without labeled data, without any supervision as guessed by the name (the *label* is the "true value" that the algorithm is trying to predict). Algorithms are fed with data which is neither labeled nor classified. Therefore, Unsupervised Learning is used if the used method does not require label data (e.g. PCA) or if labels are not available (e.g. because it's a latent variable which cannot be measured or the labelling process would be too cost/time/budget consuming). In case that labels are available, one would use supervised learning then due to better learning performance in regards of time and cost.
Unsupervised Learning is grouped into Clustering and Association.

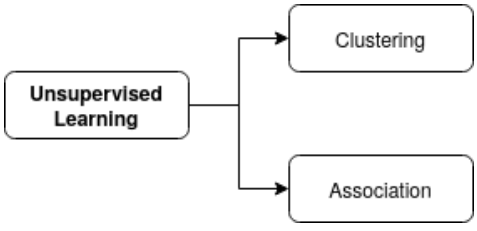


Figure 2.2: Unsupervised Learning

[1] Tom Michael Mitchell (born August 9, 1951) is an American computer scientist

Clustering mainly deals with finding a structure or pattern in a collection of uncategorized data. Clustering algorithms will process data and find natural clusters(groups) if they exist in the data. It's also possible to define how many clusters the algorithms should identify.

Association analysis attempts to find relationships between different entities. The classic example of association rules is market basket analysis. This means using a database of transactions in a supermarket to find items that are bought together.

Reinforcement Learning : Reinforcement Learning is defined as a Machine Learning method that is concerned with how software agents should take actions in an environment. A software agent is trained by some commands and after each command it gets a rewards. Time plays a crucial role in reinforcement problems. Two kinds of Reinforcement Learning are Positive Reinforcement Learning and Negative Reinforcement Learning.

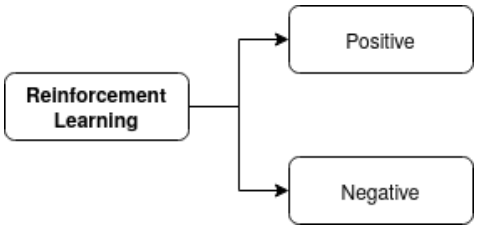


Figure 2.3: Reinforcement Learning

Positive reinforcement is adding a pleasant stimulus to enhance a behaviour.

Negative reinforcement is removing an averse stimulus to enhance a behaviour.

Supervised Learning: It deals with labeled data. Supervised learning algorithms try to model relationships and dependencies between the target output and the input features. Therefore, from those relationships learned it's possible to predict the output values for new data .
Also Supervised Learning, in turn, can be subdivided in others two categories of algorithms as Classification And Regression.

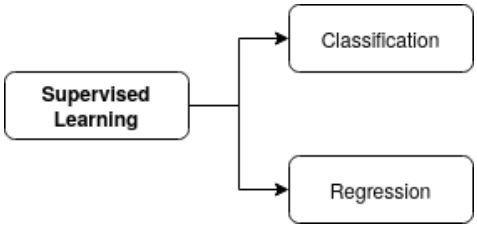


Figure 2.4: Supervised Learning

Classification algorithms are used when the output variable is categorical and it's the process of finding or discovering a model or function which helps in separating the data into multiple categorical classes i.e. discrete values. Basically it involves assigning new input variables to the class to which they most likely belong in based on a classification model, that was built from the training data that was already labeled.

Regression is the process of finding a model between input values and output values of the labeled data-set used and then predict continuous values.

Figure 2.5 shows a brief summary of the subdivisions aforementioned.

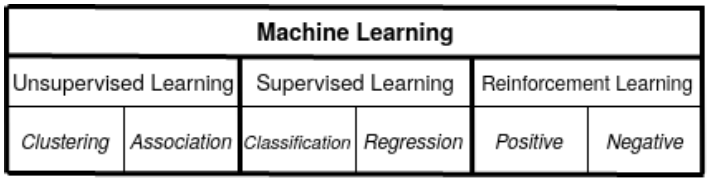


Figure 2.5: Summary of Machine Learning fields.

2.1.1 Supervised Learning

In the related work the Machine Learning field involved is Supervised Learning due to the availability of labeled data: NACA 4 digit profiles, angle of attack (AoA), Mach number, lift and drag coefficients. Here a more detailed explanation of Supervised Learning is presented.

It's easy to think about an application: a learning Machine whose outputs are labels of the training data. Three main steps of this application can be identified:

1. feeding data to Learning Machine(LM): It's necessary to provide to the Learning Machine labeled data and labels.

2. training phase: Learning Machine tries to extract a relation between the labeled data and labels.

3. prediction: after the training phase, data that LM has not seen before are used to generate predictions.

Different algorithms are used to implement Supervised Learning, the most common are:

- Support Vector Machine
- Linear regression
- Logistic regression
- decision tree
- K-nearest neighbor (K-NN)
- Artificial Neural Network (ANN).

Support vector Machine or SVM is one of the most popular Supervised Learning algorithms. This algorithm is used to generate the best line or decision boundary that can enclose n-dimensional space into classes in order to place new data in the correct category. The decision boundary, or hyperplane, is the one that maximize the margins from the labels of the target values. Most of the time this particular algorithm is used related to classification problem.

Linear regression is the easiest algorithm related to Supervised learning: Linear regression discovers and shows the linear relation between a dependent and independent variable. Instead Logistic regression is used to predict "categorical dependent variable" so It can produce an output that is only 1 or 0: 1 means that the data belongs to that particular class and 0 means that it does not. The latter one occurs in classification problems and the former one in regression problems.

Decision Tree is mostly preferred for classification problem. The algorithm is structured as a tree. Nodes represent attributes of the involved data-set and branches are the decision rule. Using decision tree means deciding on which features to choose.

K-nearest neighbor (K-NN) is based on the assumption that similar data are close to each other. The term close in this case can be interpreted as distance, proximity or closeness. Here it is another example of terminology ambiguity. K-NN classifies data in based on the similarity. It's often used for classification problem.

Artficial Neural Network (ANN) consist in a large number of simulated neurons, as in a human brain, and connection between them.

The investigated method are Artificial Neuronal Networks.

2.1.2 Artificial Neural Networks - ANN

Artificial Neural Network (ANN), also called neural network is the most famous application of machine learning.
A neural network is an algorithms that aims at identifying potentially hidden coherences in a set of data. It's a model of computation inspired by the architecture of neural network in human brain [19]. Such a system "learns" to perform tasks by analyzing examples. Basically they are computational algorithms used for non linear function approximation. Easiest example of neural network is the perceptron (fig. 2.6), a single layer neural network that operates as a binary classifier. Binary classification is the task of classisying the elements of a given set into two groups. Its definition arises in 1958 at the Cornell Aeronautical Laboratory by Frank Rosenblatt [20].

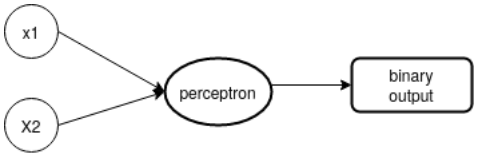


Figure 2.6: perceptron structure

If the result of the addition of the inputs is larger then a threshold, then the output is set to one otherwise it's set to 0. The original application of Rosenblatt perceptron shown in figure 2.6 was the one of a binary classifier. However, it can be used for regression but just linear one.

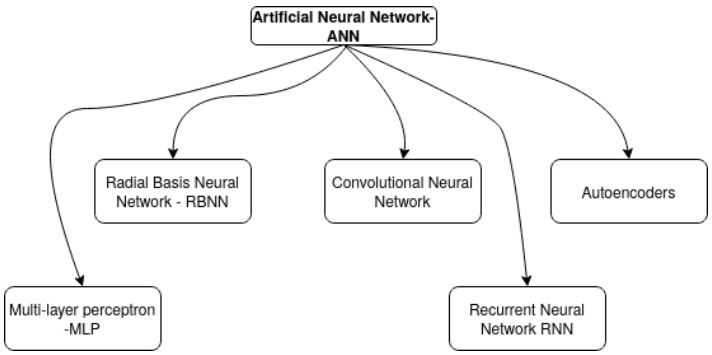


Figure 2.7: Most popular Artificial Neural Networks

Different types of neural network could be taken into account before to address a particular task. Each one has points of strength and use different principles. Most common types of Neural Network are:

- Multi-layer perceptron (MLP) is a classical feed forward neural network(FFNN) with each layer's neurons being connected to every neuron of the previous layer. Although sometimes FFNN is used as a synonym, it isn't actually: feed forward (FF) just describes the movement of data through the network, thus also a CNN can be FF as long as it doesn't have any recurrent layers in it.
- Radial Basis Neural Network (RBNN or RBFN): it performs classification by measuring the input's similarity to examples from the training set.
- Convolutional Neural Network (CNN) : the main characteristic is the presence of a convolutional operation on the input layer. Due to this, (CNN) manifests good results in image and video recognition.
- Recurrent Neural Network (RNN): in a (RNN) the output of a layer, previously selected, is saved and fed back. It used for sequential data processing e.g. NLP, speech recognition, stock market analysis etc.
- Autoencoders: special NN that aim to reproduce the input after its conversion in a different representation.

  Since the area of NN is very broad not all types can be presented here. The reader is advised to refer to [21].

In terms of layers a neural network is made of the input layer, hidden layers that could be more than one and output layer. Figure 2.8 shows a general structure of a neural network. One can clearly identify the presence of an input layer with four neuron, two hidden layers and the output layer with 2 neurons.

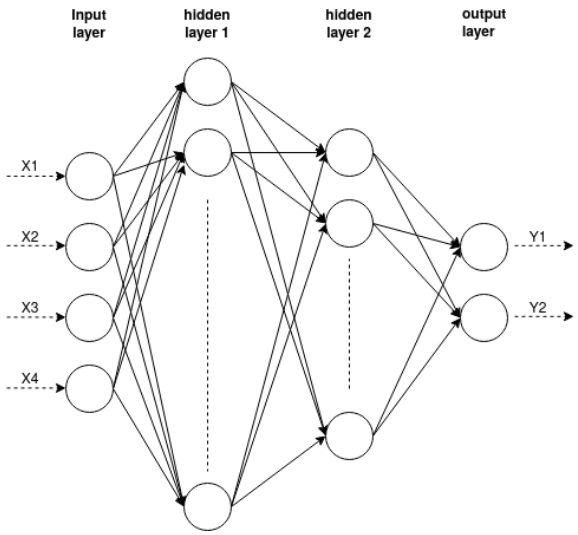


Figure 2.8: Scheme of a general neural network

2.2 Specifics of a Neural Network

Building a neural network involves several choices. The first one is to figure out which type of NN is the most suitable for the problem considered. In the related work, the a MLP is used. This particular NN has been chosen because is suited for predictions where a real value quantity is predicted given a set of input variables and because it is the most used NN and the easiest to set..Afterward it's necessary to choose model hyper-parameters and model parameters. There is an essential difference between them:

- model hyper-parameters: it's a prior specification. It's a value that cannot be learned within the estimator directly, in this case the estimator is a NN.
- model parameters: they are properties of the training data that are learned during the training phase.

It's possible to encounter ambiguity around these two specific of a NN. There is a main difference between them: the programmer is able to define in advance the model hyper-parameters but not the model parameters. Model parameters could be only initialized before the training, after that NN has the control on them.

Model hyper-parameters are the following:

- number of hidden layers
- number of units in each layer
- activation function
- optimizer
- loss function
- learning rate

- batch size

Model parameters:

- weights: they defined how much influence the input of a neuron will have on the output (see as refernce 2.9)
- biases : they are additional input used to guarantee that even when the inputs are zero there will be an activation in the neuron.

Hyper-parameters will be explained in the next paragraph. Another significant point for NN is how to handle the data-set. The data-set employed needs to be subdivided in three subset: training data-set, validation data-set and test data-set. Furthermore the normalization of the data is very common in NNs. In Chapter 4 an example of training without normalization will show why it is necessary. It's worth to point out that the presence or not of normalization is due to the characteristics of the data. A crucial point to understand how a NN works is the back-propagation procedure that arise in the training phase. It will be presented in paragraph 2.2.4 .

2.2.1 Model Hyper-parameter

Here a brief introduction on the hyper-parameter is given. It's necessary to have clear the role of each one of them before to generate a NN for any purpose.
How they interact in the training process will be clearer where back-propagation procedure is presented in paragraph 2.2.4.

Number of layer and Number of Neurons Number of layers and number of neurons are here introduced together because they define the architecture of a NN. Taken for granted that the presence of an input and output layer is essential for a NN the presence of an input and output layer, it's up to the developer the choice of hidden layers. Concerning this, it's useful to know that one might find a not fully consistent nomenclature in literature regarding the number of layers. If nothing is specified when referring to a "two-layer NN", this means that a NN with one hidden layer is considered. Only the hidden layer and the output layer are counted in the name.
Selecting the number of layers and number of neurons for each layer could be seen as the first choice to do, but a peculiarity of this choice is that there is not a general rule of thumb to follow in order to have the best results. Since the beginning of the application of NN, not only in fluid mechanics but in each field, different methods were presented to find the best architecture as the Taguchi method [22] or the trial-and error approach here followed. Commonly the background process on which architecture will be involved is a random test research.

Activation Function, Loss function and Optimizer Crucial decision for the final result, in terms of accuracy and computational efficiency, are activation function, loss function and optimizer. These three hyper-parameters represent the mathematical section of the Neural NN.

Activation function is the mathematical function that defines the output of each neuron and t defines also the output of the neural network.

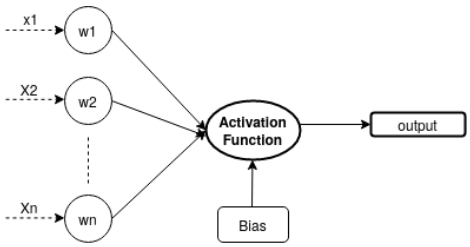


Figure 2.9: Neuron activation function

In figure 2.9 a single neuron is represented. The the neuron receives a series of inputs (x) with their weights and a bias. For each neuron only one bias is given. The inputs can be either the data fed to the network or outputs of other layers.

The Loss function describes how to calculate the loss,the difference between target values and outputs. The loss function is the calculated loss "plotted" over the parameter space. The loss is a result of the chosen Loss function. Loss function, that can be defined also Objective function or cost function, is a method of evaluating how well the algorithm models the data-set.

The optimizer is an optimization algorithm that helps to minimize (or maximize) an objective function which is a simple mathematical function dependent on the model's internal learnable parameters (model parameters) which are used to compute the target value. A more detailed explanation is given in paragraph 2.5.

Learning Rate, Number of Epochs and Batch size The learning rate is the hyper-parameter that controls model's update step size with respect to the accuracy achieved with the present model. It's an important parameter.A value too small may result in a slow training and danger of getting stuck in a local. Instead a large value may result in risk of non converting to any minimum, since the optimizer jumps out a minimum rather than descending to it.

Number of epochs refers to the number of times that the whole data-set arranged for the training phase is feed into the model. This means that if a number of epochs is equal to 10, during the training phase the model "sees" 10 times the whole data-set. With this parameter it's possible to partially overcome the size of the data-set.

Batch size define the number of data sample seen by the model before updating the model parameters to more optimal values for improved predictions.

2.2.2 Model parameters

Model-parameters as aforementioned can only be initialized by the NN developer. Weights initialization is not mandatory but sometimes it improves the learning process especially in the first period (see Chapter 4).

Weights and Biases It's clear at this point that the developer doesn't have control on model parameters . As mentioned before it's only possible to initialize them. In Chapter 4 this will be addressed more clearly. Weights and biases play an important role in the definition of the output of each neuron and for the whole NN.

The weight is a characteristic that defines how much the source nod influences the receptor node.
Bias is another parameter added to the object of the activation function. The presence of the bias allows to translate the activation function left or right in his own graph.

2.2.3 Data-set

The data-set involved play a fundamental role in achieving the goal. Three sub-sets that differ in their involvement are extracted from the whole data-set:

- Training Data-set: it consists of data samples used to fit the model
- Validation Data-set: subset used to provide an unbiased evaluation of a model, fit on the training set while tuning model hyper-parameters,
- Test Data-set: it's the last division used. These data samples are used to provide unbiased evaluation of a final model fit on the training data-set.

See figure 2.11 as reference.

2.2.4 Back-propagation procedure

Up to this point only definitions necessary to understand how a NN works have been introduced. In this section, the back propagation procedure will be discussed. More generally this describes how a NN works. A more detailed explanation is given in section 2.5. A NN has one input layer, some hidden layers and one output layer. The input layer brings data inside the network and these data "travel" towards the output layer going through hidden layers.
Therefore, by the presence of connections layer-layer and connection neuron-neuron data fed in the model are manipulated thanks to weights, biases and activation function. That's where batch size, loss function, learning rate, optimizer and number of epochs come into the process. The order used for mentioning the hyper-parameter is not random.

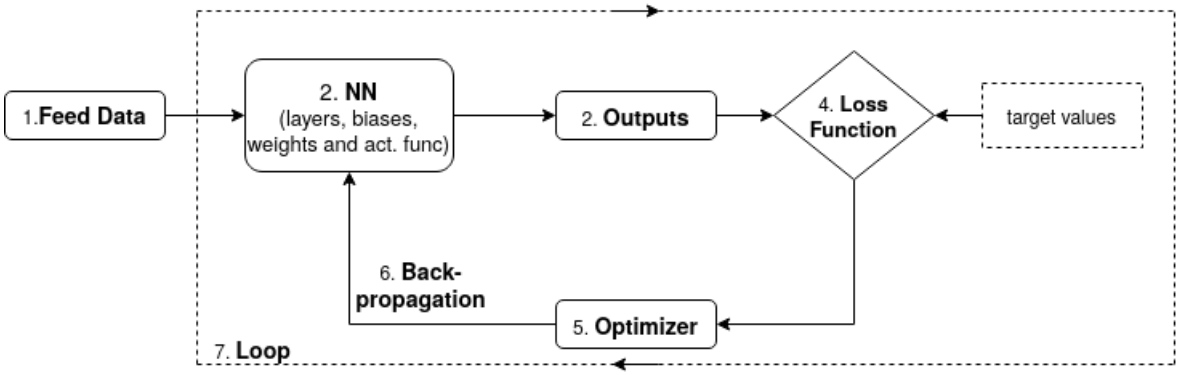


Figure 2.10: Back-propagation scheme

Figure 2.10 shows the back-propagation procedure and how the ingredients previously mentioned are used.
Fundamental steps are here introduce with reference to figure 2.10:

1. Feeding training data-set batch-wise to the model through the input layer.
2. Data samples go through the network interacting with weights, biases and activation function.
3. Output layer shows the prediction generated.
4. Loss is measured comparing outputs and targets value of the data-set. This comparison is evaluated after a number of data equal to the batch size.
5. Optimizer algorithm uses loss and learning rate to adjust model parameters
6. Back-propagation: The correction on the weights are feed-back at the beginning of loop.
7. The loop lasts until when the number of epochs fixed is reached. Remember that the number of epochs is an hyper-parameter, it's tuned by the developer before the training.

When step 7 is done the training phase is completed. Validation data-set and testing data-set are not mentioned. The former is taken into account in correspondence of step 4. It's used after each epoch to have unbiased prediction in order to give a measure on how well the model is learning from the training data-set.
After the training phase the latter one is used (testing data-set). This gives the final evaluation of the model. The goal is to understand the prediction capability on unseen value of the model. In figure 2.11 it is possible to clearly understand when they occur in NN life cycle.

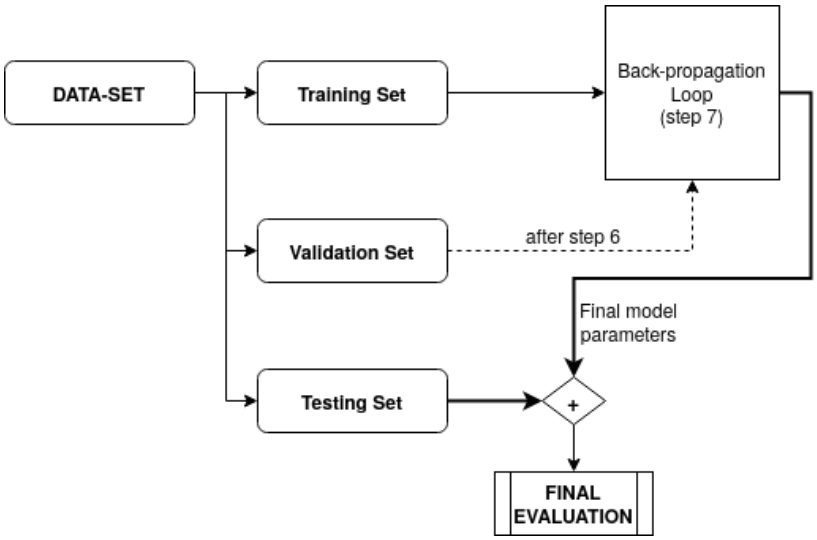


Figure 2.11: Subdivision and deployment of the data-set (step 7 refers to figure 2.10)

2.3 Activation Function

Activation function is a crucial component of a neural network. It's attached to each neuron and determines the output of the model. It has to be pointed out that it must be computationally efficient, because they are called thousands times in the training phase.

Different activation functions mean different output behavior of each neuron and this has an impact on the final result. Due to the problem involved the best choice for the activation function could differ.
An activation function introduction is required to understand how a NN works but also because they are important for the final result, it doesn't matter which one is the aim. Five of the most common activation function will be discussed:

- Binary step function
- Linear activation function
- Logistic sigmoid
- tanh - Hyperbolic Tangent
- ReLU - Rectified Linear Unit
- Leaky ReLU.

Qualities and limitations are introduced to have a background before to use them. Not all of these specifics have been faced in the related work. Binary step function is presented only because it is the activation function used in the first perceptron. It's used in classifier algorithm, which is not the case here.

2.3.1 Binary step function

A binary step function is the easiest activation function. The idea is to have a threshold-based activation function i.e. whether or not a neuron should be activated. Once the threshold is defined if the input of the neuron is greater the neuron is activated and its output is equal to the input. Figure 2.12 shows the binary step function with threshold equal to 0.

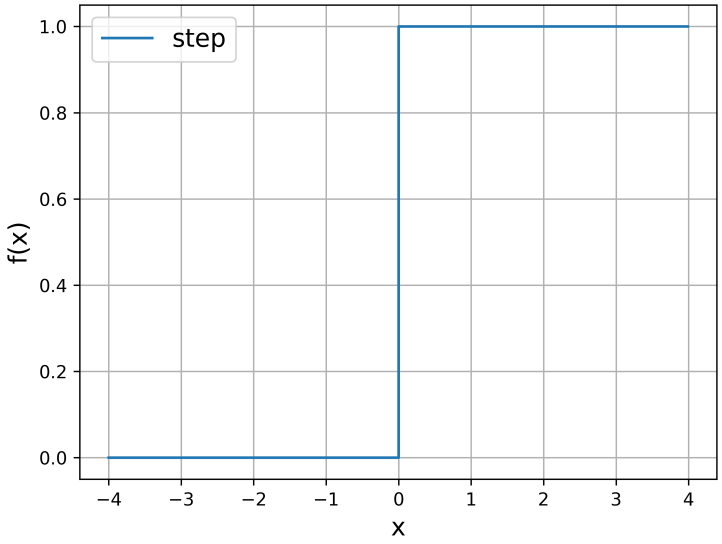


Figure 2.12: Step function as activation function

The most significant limitation of the binary step function as activation function is the impossibility to manage multi-value output.

2.3.2 Linear activation function

A linear activation function generates an output proportional to the input. Therefore the output is directly proportional to the multiplication of weight and input (output of the previous layer). Equation 2.1 represent the linear activation function and figure 2.13 is its own graph.

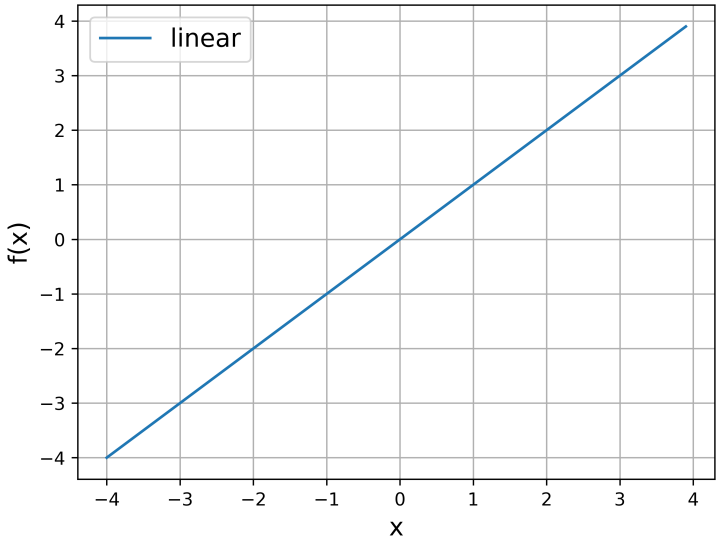


Figure 2.13: linear function as activation function

$$g(x) = x \ . \tag{2.1}$$

As binary step function, also linear activation function has a serious limitations:

- Neural Network with only one layer
  If linear activation function is used in each layer the resultant Neural Network will have only one layer. This comes from a property of Linear algebra: combination of linear function is a linear function.

2.3.3 Logistic sigmoid

Sigmoid, or Logistic,is the first non-linear activation function here introduced. It maps the interval (-∞, +∞) onto (0,1) and for this reason it is mostly used for probability prediction. It is defined as:

$$\sigma(x) = \frac{1}{1 + \exp(-x)}. \tag{2.2}$$

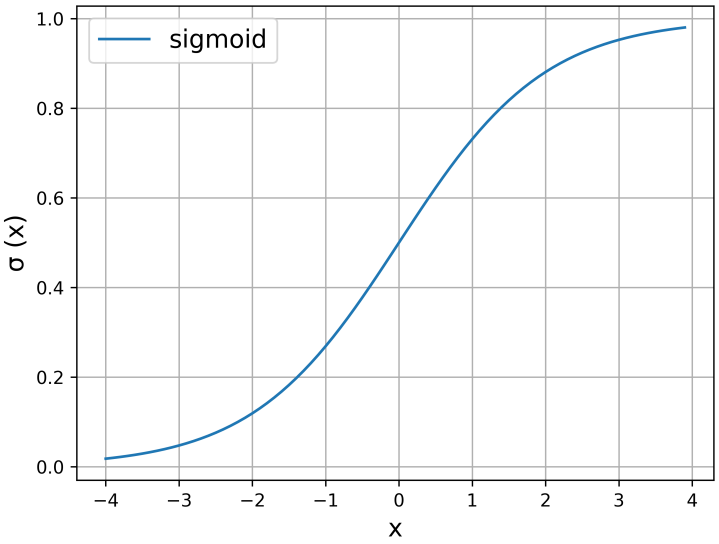


Figure 2.14: Sigmoid activation function

From its graph, figure 2.14, one can deduce that Sigmoid function saturates when its argument is very positive or very negative, this means that the function becomes very flat and insensitive to small changes in its input [21].

This activation function has some qualities as:

- Sigmoid is a non-linear activation function.
- Its output lay in the range (0,1) that with respect to step function is a great improvement and it's smooth, continuous and differentiable. The output could be interpreted as a probability value.
- No possibility for the activation to blow up since the output has a fix range.

Below are listed disadvantages :

- Vanishing and exploding gradients.
- Output not zero centered.
- Computationally expensive since it involve exponentials.

The exploding gradients problem require large weights and large gradient over multiple layers to cause exploding gradients. Therefore, it is worth to pay more attention to it when deep and/or recurrent networks are used. The vanishing gradient problem deserve more attention in comparison with the others.
As mentioned above, derivatives and gradients of the loss function are used from the optimization algorithm in the back-propagation process. Weights and biases change in proportion with these gradients.
This problem arise when gradients are very small. It's clear that is particular problematic with sigmoid activation function. Figure 2.15 shows sigmoid derivative function.It's very small for values greater than 3 and lower the -3.
When the gradients are too small this results in no-learning. The network is not able to improve its accuracy.

This can occur when weights are initialized poorly with values too large or too small. This does not mean that with a proper initialization of the weights this problem disappear.

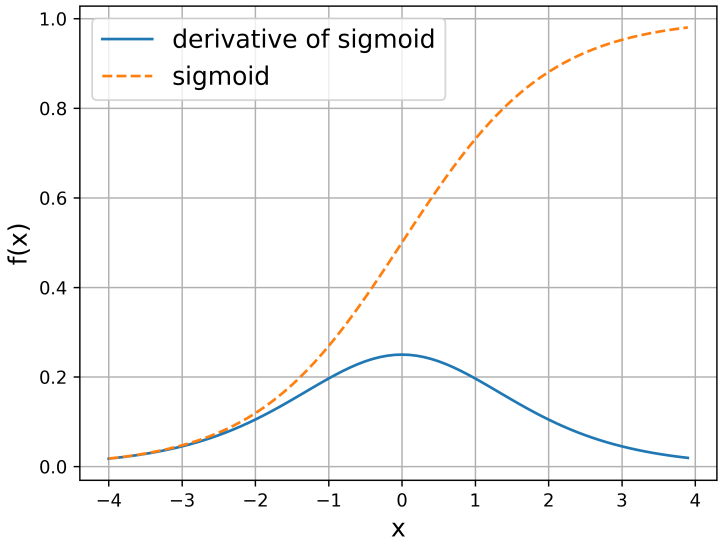


Figure 2.15: Sigmoid function and sigmoid derivative

2.3.4 Tanh - Hyperbolic tangent

The hyperbolic tangent is defined as the ratio between the hyperbolic sine and hyperbolic cosine as shown in equation :

$$g(x) = tanh(x) = \frac{sinh(x)}{cosh(x)} , \tag{2.3}$$

From figure 2.16 it possibly to understand that tanh is similar to sigmoid function: it's actually a mathematically transformed and shifted version of the sigmoid function.
It saturates after certain values of x and it's also characterized by a "S" shape. The main difference is that tanh is sym Loss function respect the axes origin. It's possible to point out the relation between tanh and sigmoid using the mathematical equation 2.2 and 2.3.

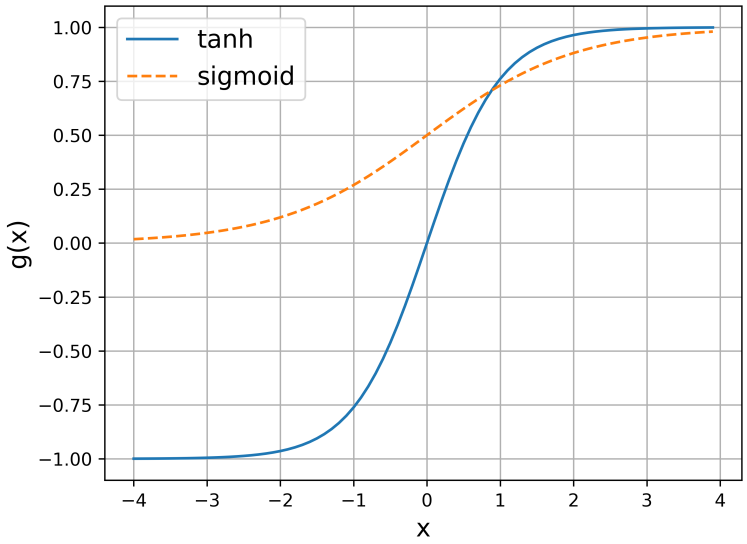


Figure 2.16: Hyperbolic tangent activation function

The 2.3 can be written as:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \ , \tag{2.4}$$

From equation 2.4 and 2.2 can be demonstrate the relation between $tanh(x)$ and the Sigmoid function:

$$tanh(x) = 2\sigma(2x) - 1 \ , \tag{2.5}$$

where σ is the sigmoid function. The hyperbolic tangent activation function usually performs better than sigmoid. Tanh goes through the origin, this means tanh(0) = 0 while σ (0) = $\frac{1}{2}$. In proximity of the origin tanh(x) is similar to x and this makes the training simpler([23]).
Disadvantages of this activation function are the ones from the sigmoid activation function.

2.3.5 ReLU - Rectified Linear Unit

ReLu is another non-linear activation function. It is defined by :

$$g(x) = max\{0, x\}. \tag{2.6}$$

The associated graph is shown in figure 2.17 . It's the most commonly used activation function in Neural networks. ReLU is a linear function for all positive input values and zero for all negative values .
Despite being a non-linear function it is very similar to a linear function: ReLU is a piece-wise linear activation function. An advantage related to employ ReLU as activation function is for example the computational cost: it's very efficient since the math involved is easier than for sigmoid and tanh; there is no need for computing the exponential functions in activations. This result in less training time needed in most of the case.

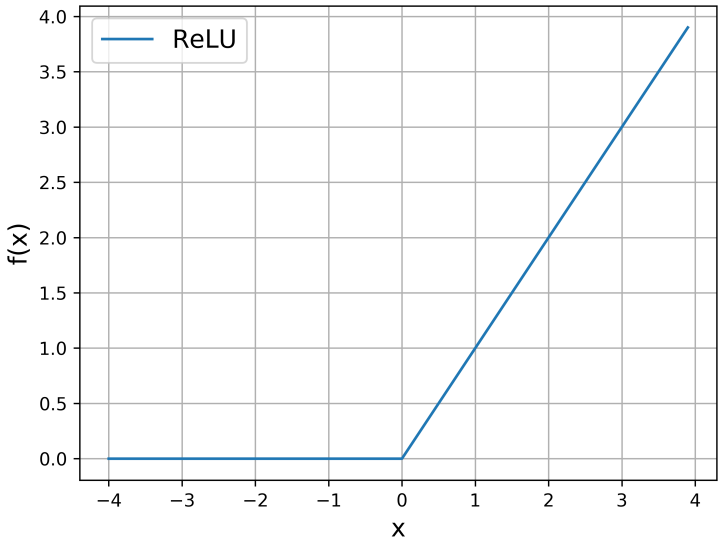


Figure 2.17: ReLU activation function

Linear behaviour for positive x means that it doesn't saturate. Thanks to that it avoids vanishing gradient problem suffered by sigmoid and tanh activation function. Another selling point is the capability to produce a true zero value.
This leads to a sparsely activated network, no all neurons are learn from each data sample.
This is an advantage if the neuron is not important. It can accelerate learning and simplify the model.

Dying ReLU happens when the biases is updated via back-propagation so that the input for the activation function becomes negative for nearly all data samples. The neuron will always output zero and unlikely for it to recover. This problem is called "Dying ReLU. It doesn't play any role and it's basically useless. The derivative is zero and the network cannot learn.
At the end it's possible to have a section of the network useless for the final result.

The "Dying ReLU" problem occurs with :

- high learning rate values
- large negative bias.

A solution to overcome this problem is Leaky ReLU activation function.

2.3.6 Parametric/Leaky ReLU

Parametric ReLU, or Leaky ReLU, activation function is a variant of ReLU attempt to fix the "dying ReLU" problem.

$$g(x) = \begin{cases} x & if\ x > 0, \\ \alpha x & otherwise. \end{cases} \tag{2.7}$$

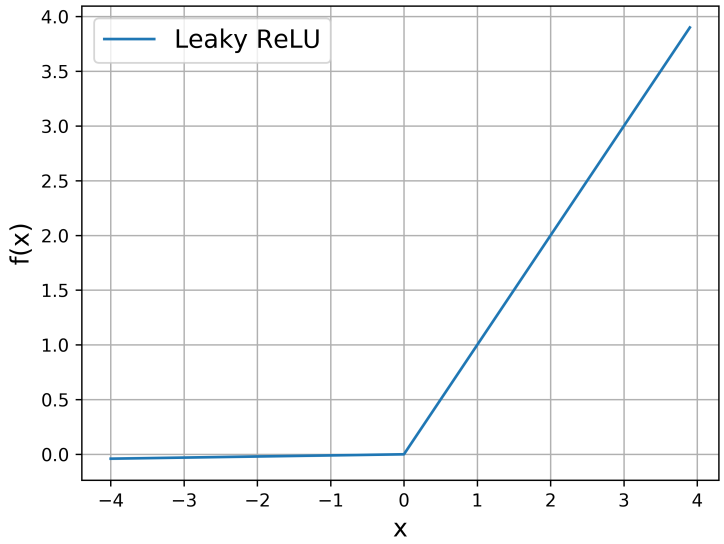


Figure 2.18: Parametric/ function ReLU activation function

Instead of the function being zero for the negative domain, para Loss function ReLu is characterized by a small negative slope. Commonly this slope is set to 0,01. However, the consistency of the benefit across tasks is presently unclear. In [24] the conclusion achieved is that Leaky ReLu performs nearly identical to standard ReLU.

2.4 Loss function

In the learning process is necessary to evaluate the ML algorithm implemented . This is done by Loss function. The Loss function describes how to calculate the loss. Loss functions are functions of true value versus predicted values. In the optimization process the loss function is the function to be maximized or minimized. The choice of the loss function is important since the function must capture the propertied of the problem.Therefore it is important that the function faithfully represent the design goals [25]. The aim of the related work lays in the regression sub-set of supervised learning. The most common cost functions used in this section are Mean Square Error (MSE), Root Mean Square Error (RMSE) and Mean Absolute Error (MAE).

2.4.1 Mean Square Error - MSE

The MSE asses the quality of a predictor. It provides information about the difference between target and predicted values in the following way:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{2.8}$$

where n represents the number of samples, y_i is the i^{th} predicted value and $\hat{y}_i$ is the i^{th} target value.

2.4.2 Root Mean Square Error- RMSE

As MSE, RMSE estimate the error between a target and a predicted values. It's defined as the square root of the MSE:

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}. \tag{2.9}$$

The terms inside equation 2.9 have the same meaning as in equation 2.8.

2.4.3 Mean Absolute Error - MAE

MAE measures the absolute difference between true and predicted values. It's defined as :

$$MAE = \frac{1}{n}\sum_{i=1}^{n} \mid (y_i - \hat{y}_i) \mid . \tag{2.10}$$

The main difference in the application of RMSE and MSE with respect to MAE is that the formers are more sensitive to outliers. They involve the square the error giving more importance to large errors.

2.5 Optimizer Algorithms

As illustrated in figure 2.10 optimizer is used after the comparison between target values and outputs of the network and the goal is to minimize (or maximize) the loss function. Always remember that cost function, loss function, error function or objective function can be used interchangeably.
The optimization is crucial for the learning process. The choice of a optimizer over another one can lead to a better optimization. This can result in a faster learning and/or in a better final prediction. In chapter 4 results for different optimizer will be present some difference between them will be highlighted. This section introduces some optimizers and their theory.

2.5.1 SGD Stochastic Gradient Descent

Stochastic Gradient Descent SDG derives from the gradient descent technique.
In order to understand SDG the former gradient descent has to be discussed.

Gradient Descent

Gradient descent, also called full-batch gradient descent, is the most important technique used to optimize an intelligent system. It's the "starting point" for each optimization algorithm developed afterwards.
This technique was originally proposed by Augustin-Louis Cauchy [26], a French mathematician .
His method is based on the derivative of a function. Such derivative has information that can help to minimize or maximize that function since it's the slope of function *f(x)* at the point *x*.

For example considering a function f it's possible to say $f(x - \epsilon sign(f'(x)))$ is less than $f(x)$ for small enough ϵ. This is the Cauchy gradient descent.

This method look for minimun or maximum. It's worth to recall differences between local and global minimun or maximum.
Local minimum is a point where *f(x)* is lower than all neighboring points.
The mathematical definition is : x^* is a local minimum if $f(x^*) \leq f(x)$ for all x in X within distance $\epsilon > 0$ of x^*, where X is a Loss function space.
A local maximum is a point where $f(x)$ is higher then f in all neighbor points.
Points where the derivative is zero are called stationary points or critical points.
Global minimum and maximum are the lowest and the highest points in the whole domain. No restrictions related to the neighbor points are made.

A problem related to the optimization in machine learning is that objective functions may have more the one local minimum that deceive the optimizer algorithm used. When functions with multiple inputs are used partial derivatives, $\frac{\delta}{\delta_{x_i}} f(\mathbf{x})$ represent the derivative of f with respect to x_i. The gradient $\nabla_x f(\mathbf{x})$ is the vector contains all the partial derivative of function f with respect to each variable x_i.

As before, now with multiple variables it is possible to decrease f following the direction given by negative gradient:

$$\boldsymbol{x'} = \boldsymbol{x} - \epsilon \boldsymbol{\nabla}_{\boldsymbol{x}} f(\boldsymbol{x}), \tag{2.11}$$

where ϵ is the learning rate.
In the framework of NN : $\mathbf{x}$ are the model parameters (weights and biases) or NN outputs, f is the objective function, $\boldsymbol{\nabla}_{\boldsymbol{x}} f(\boldsymbol{x})$ is the gradient of the objective function and $\mathbf{x}'$ are the adjusted model parameters. This is the fundamental formula to understand how an optimizer work.

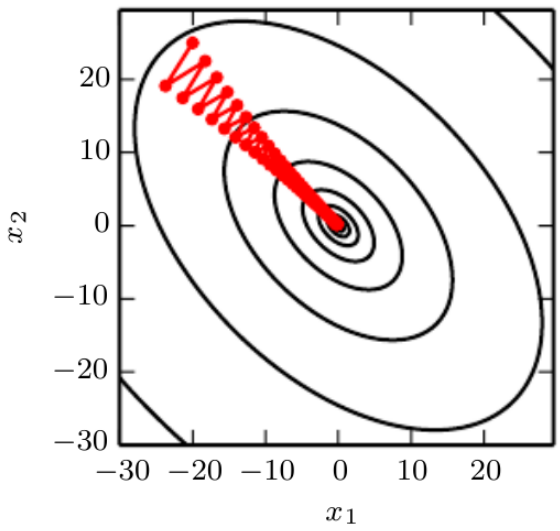


Figure 2.19: Gradient descent method used to minimize a quadratic function [21]

SGD

Stochastic gradient descent is the most common optimizer algorithm used in deep learning. It's an extension of the gradient descent method.

SGD use an expectation as defined in [21]. Machine learning algorithms manage big data-set. Evaluating a single gradient step could need long computational time. SGD refers to gradient descent where you update after each sample (batch size=1). What is used normally for NN is mini-batch gradient descent, which however is often referred to as SGD in literature. The expectation used by SGD (mini-batch SGD) is made on a small portion of data called mini-batch, $\mathbb{B} = \{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(m')}\}$ and not on the whole data-set involved in the training phase. This allow SGD to be faster than standard gradient descent(batch-size=1).
The mini-batch is usually a small portion of the whole data-set which should be small enough to be computationally efficient and large enough for the mini-batch's cost function to represent the actual cost function of the whole training data set well enough. Defining $L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$ the loss function, where $\boldsymbol{\theta}$ represents the model parameters the gradient could be introduced as:

$$\boldsymbol{g} = \frac{1}{m'} \boldsymbol{\nabla}_{\boldsymbol{\theta}} \sum_{i=1}^{m'} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}) \ , \tag{2.12}$$

and the step of SDG is:

$$\boldsymbol{\theta}' = \boldsymbol{\theta} - \epsilon \boldsymbol{g}. \tag{2.13}$$

The learning rate is a high priority parameter in SGD. It's common practice to decay the learning rate in the training phase. The operation of choosing the learning rate is more an art than a science.
If learning rate is too large the learning phase could show violent oscillations and it can drive to a local minimum instead if it's too small the learning phase could stuck with an high cost function value. The most important property of SGD is that the computational time required per update does not grow with the size of the data-set. Increasing the size of the data-set involved SGD may converge before processing it entirely [21].

2.5.2 Momentum Method

Here the momentum method is introduced (Polyak,1964). This method is designed to accelerate the learning process. The name derives from a physical analogy related to Newton's law of motion. It accumulates an exponential decaying moving average of the past gradients and continue to move in that directions. An exponential decaying average is a particular Exponential Moving Avarage (EMA).
EMA is a particular type of moving average (MA) where more recent data is given greater importance.
New update law is the following:

$$\boldsymbol{v} = \alpha \boldsymbol{v}' - \epsilon \frac{1}{m} \boldsymbol{\nabla}_{\boldsymbol{\theta}} \sum_{i=1}^{m} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}) \tag{2.14}$$

$$\boldsymbol{\theta} = \boldsymbol{\theta}' + \boldsymbol{v}. \tag{2.15}$$

Another parameter, $\alpha \in [0,1)$, is introduced. The exponential decays is controlled by this parameter. The parameters $\boldsymbol{v}$ is called velocity and it accumulates the previous gradients.

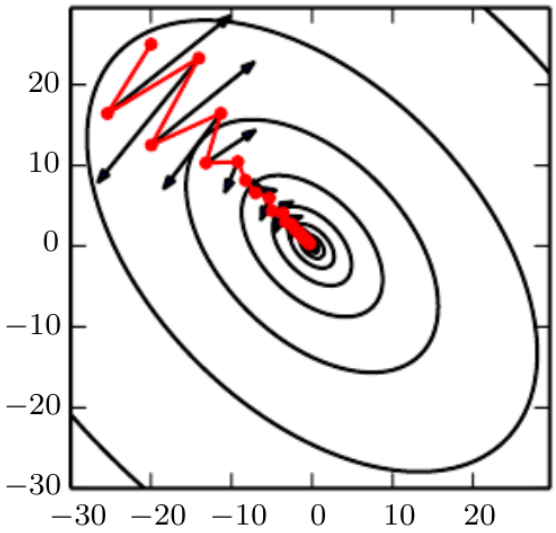


Figure 2.20: Momentum method vs Gradient [21]

Figure 2.20 shows the main differences between momentum method and gradient descent. Red line correspond to momentum instead black arrows represent the step that gradient descent would take at that point. This is the case of a poorly conditioned Hessian matrix in order to exasperate the difference. The condition number of Hessian measure how much the second derivative in one direction differs from the others with respect to the direction. Gradient descent doesn't use information from Hessian matrix so it doesn't know which one is the best direction to explore. Gradient steps waste computational time moving back and forth, instead momentum correctly traverses the canyon lengthwise.
In gradient descent the step size was aligned only with the gradient of the last learning, with momentum the step depend on the size and the direction of the previous update. For example if the learning process is characterized by a series of gradients pointing in the same directions the momentum accelerate the learning in the direction of $-\boldsymbol{g}$. This results, roughly speaking, in updates only for relevant samples.

2.5.3 RMSProp

RMSProp is the first algorithm here considered that is characterized by an adaptive learning rate.
Momentum method use a new hyper-parameter trying to take into account sensitivity to some directions. Methods with adaptive learning rate are an alternative to the latter, they don't introduce another input parameter. The basic idea is : if derivatives sign remain the same the learning rate should increase; if they change sign the learning rate should decrease.
In particular RMSProp [27] is a revision of AdaGrad [28], which *individually adapts the learning rates of all model parameters by scaling them inversely to the square root of the sum of all of their historical squared values.* This results is a rapid decrease of the learning rate for those parameters with large values and slow decrease for parameters with small values. RMSProp, with respectrespect to AdaGrad, introduces an exponential weighted moving average. Using the square root of the entire parameter history could result in a too small learning rate before the possibility to reach the convergence.

RMSProp use the previous mentioned decaying to give more importance to the most recent values.
In order to introduce the RMSProp algorithm it's necessary to introduce some parameters: ϵ is the learning rate, ρ is the decay rate of the exponential weighted moving average and δ is a small constant introduce to avoid division by zero (usually set to 10^{-6}).
Once the gradient is defined with the formula 2.12 seen before the algorithm goes on with the square gradients as follows:

$$\boldsymbol{r} = \rho \boldsymbol{r'} + (1-\rho)\boldsymbol{g} \odot \boldsymbol{g} \ , \tag{2.16}$$

$$\boldsymbol{\Delta\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g} \ , \tag{2.17}$$

$$\boldsymbol{\theta} = \boldsymbol{\theta'} + \boldsymbol{\Delta\theta} \ . \tag{2.18}$$

In 2.17 $\frac{\epsilon}{\sqrt{\delta+\boldsymbol{r}}}$ is applied element-wise.
Momentum method and RMSProp have been introduced in order to understand the following optimizer, Adam.

2.5.4 Adam

Adam [29] stands for adaptive moment estimation. Adam has the characteristic to be a combination of RMSProp and Momentum method. The algorithm introduce exponential moving average of gradients and the squared gradients. They are estimates of the first moment (mean) and the second moment (uncertered variance) of the gradients. Therefore it introduces corrections based on the past gradients (momentum method) and corrections base on the square root of square past gradients (RMSProp method). It also incorporate bias correction for both momentum terms that is important in the first step where the moving decaying average are lead towards zero since ρ_1 and ρ_2 are close to zero. This becomes clearer looking at the algorithm.
As for RMSProp new definition are introduced: ρ_1 and ρ_2 in [0,1) are exponential decays rates for first and second moment(suggested number are 0,9 and 0,999), δ has the same meaning as before, $\boldsymbol{s}$ is the first moment (gradients), $\boldsymbol{r}$ second moment variable (squared gradients).
The first step is always 2.12, then:

$$\boldsymbol{s} = \rho_1 \boldsymbol{s'} + (1-\rho_1)\boldsymbol{g}, \tag{2.19}$$

$$\boldsymbol{r} = \rho_1 \boldsymbol{r'} + (1-\rho_2)\boldsymbol{g} \odot \boldsymbol{g}, \tag{2.20}$$

$$\hat{\boldsymbol{s}} = \frac{\boldsymbol{s}}{1-\rho_1}, \tag{2.21}$$

$$\hat{\boldsymbol{r}} = \frac{\boldsymbol{r}}{1-\rho_2}, \tag{2.22}$$

$$\boldsymbol{\Delta\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}, \tag{2.23}$$

$$\boldsymbol{\theta} = \boldsymbol{\theta'} + \boldsymbol{\Delta\theta} \, . \tag{2.24}$$

Operation 2.23 is also applied element-wise. From [29], Adam is a robust and well suited algorithm for machine learning optimization.

2.6 Considerations

In this chapter all the ingredients necessary to understand the NN have been presented. In particular for activation functions, loss functions and optimizers a deeper mathematical background have been explained. Before staring with the related work a clarification may be helpful. In figure 2.10 step 7 represent the back-propagation: It has been introduced in that way to simplify the whole comprehension of NN. More precisely, once the correction of model parameters are evaluate with the optimizer, the corrections travel backwards in the following way:

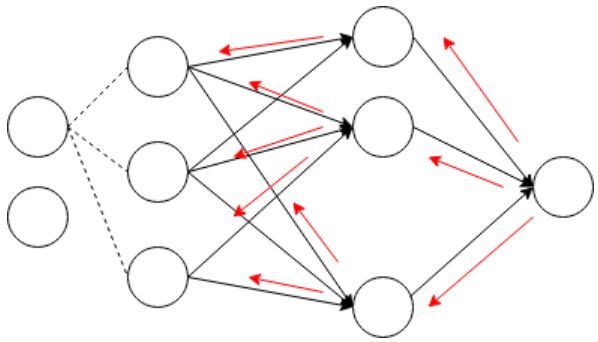


Figure 2.21: Neural Network architecture with back-propagation. The black arrows represent normal data propagation towards output layer and red arrows represent back-propagation afterwards corrections evaluations.

Chapter 3

Numerical simulations

In this chapter the work done in order to generate the data-set will be briefly introduced.
As mentioned in the introduction, NN are used to predict lift and drag coefficient for 4-digit series of NACA airfoil profiles. The dataset was developed from CFD simulations with DLR-TAU code.

3.1 Airfoil profile

The object of CFD simulations are NACA profiles from the 4-digit series.
NACA airfoils were developed by National Advisory Committee for Aeronautics (NACA). A series of digits is used to describe the shape: 4-digit series, 5-digit series, modified 4-/5- digit series and 6-digit series with more complicated shapes. Here only NACA 4-digit profiles are used.
NACA 4-digit series has 4 digits that are used to generated the airfoil shape. If four digits as MPXX are taken into account, the resulting NACA profiles has:

- maximum camber M as percentage of the chord. For example M=2 means maximum camber at 2%.
- P as maximum camber position with respect to the leading edge in tenths of the chord.
- thickness described by XX as percentage of the chord.

Figure 3.1 show the 4-digit system where m is the maximum camber (first digit), p denotes the chordwise position (second digit) and t is the maximum thickness (third and fourth digit). Again in figure 3.1 c denotes the chord and y is the half of the thickness. Figure 3.2 shows NACA 2412 profile that it is characterized by 2% of the chord as camber located at the 40% of the chord and with a thickness of 12%. The profile section is generated using a camber line and a thickness distribution perpendicular to the camber line. Equations for this particular series are:

Table 3.1: Camber and gradient for a cambered 4-digit NACA airfoil

	Camber	Gradient
0≤x<pc	$y_c = \frac{m}{p^2}(2p(\frac{x}{c}) - (\frac{x}{c})^2)$	$\frac{dy_c}{dx} = \frac{2m}{p^2}(p - \frac{x}{c})$
pc≤x<1	$y_c = \frac{m}{(1-p)^2}((1-2p) + 2p(\frac{x}{c}) - (\frac{x}{c})^2)$	$\frac{dy_c}{dx} = \frac{2m}{(1-p)^2}(p - \frac{x}{c})$

$$y = \frac{t}{0.2}(0.2969\sqrt{x} - 0.1260x - 0.3516x^2 + 0.2843x^3 - 0.1015x^4) \tag{3.1}$$

where the variables have the same meaning as in figure 3.1: x is the position along the chord, y is half of the thickness at position x that needs to be applied both sides of the camber line, p is the second digit P divided by 10. m is the maximum camber (100 m is the first digit M). When a closed trailing edge is considered, which is the case in this work, the last coefficient of equation 3.1 is modified from 0.1015 to −0.1036.

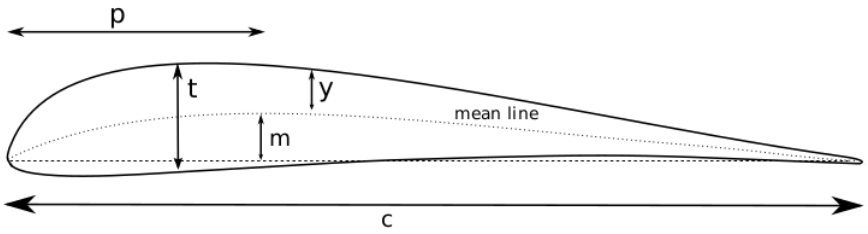


Figure 3.1: Description of NACA 4-digit system [8]

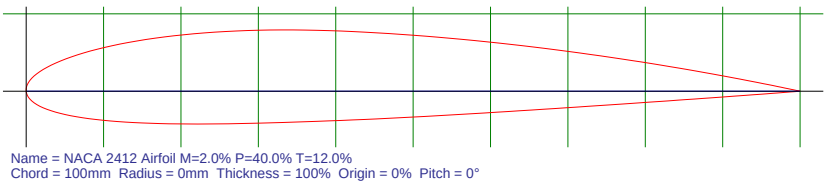


Figure 3.2: NACA 2412 profile from airfoiltools

3.2 Airfoil grid

The mesh generation is a fundamental preprocessing step for the applied CFD simulation in this work and it's also one of the most critical factor that must be considered to ensure simulation accuracy.

3.2.1 CAD

In this section a brief introduction of the process followed to generate airfoils CAD is given .
Steps in the chain process that leads to the final mesh:

- Profile coordinates for NACA 4-digit series from http://airfoiltools.com/
- Geocreate is used to create the CAD of each profile

- Centaur is the software used to generate mesh
- DLR-TAU code y+ adaptation.

Airfoiltools allows to download x and y coordinates for each profile . Maximum number of points for each profile, 200. is used with a closed trailing edge. Figure 3.2, that comes directly from airfoilstools.com ,shows NACA 2412 profile. The transition from the x,y coordinates to a 3D CAD was made with Geocreate.
From a "file.dat", where x and y coordinates are stored, only adding some rows specifying splines and related points allows to create the profile shape. In figure 3.3 is possible to see the 3D CAD of NACA 2412 generated with a random wingspan of 10mm. Since 2D CFD simulation are used the size of the wingspan is not a predefined value. Since 2D simulations were the goal to reach it was than necessary to switch from 3D to 2D. This step was performed in centaur before the mesh generation process started.

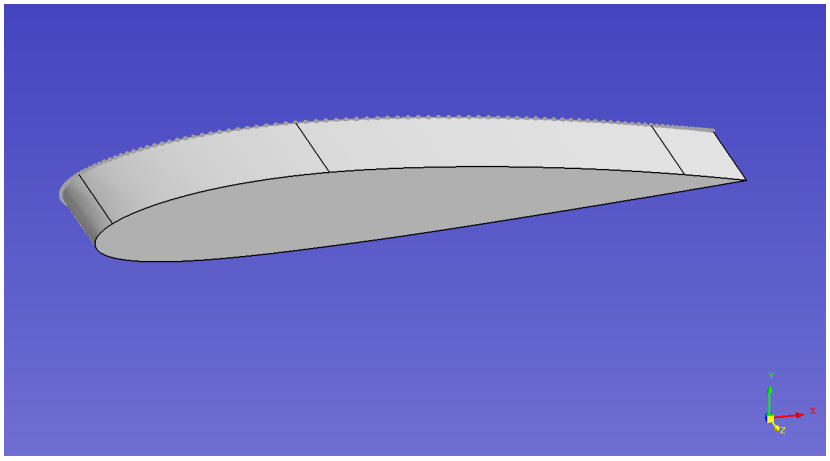


Figure 3.3: 3D cad of NACA 2412 profile from geocreate

3.2.2 Mesh generation

In order to generate the grid generator Centaur was used. As aforementioned it was necessary to switch from 3D profile to 2D.Once this step was done, two parameters were taken into account to generate the desired mesh:

- first prism layer thickness h,
- boundary layer thickness δ.

The mesh used is structured close to the airfoil surface in order to discretize the boundary layer and unstructured for the remaining part of the domain.
Structured mesh in 2D simulation means quads (or prisms). The previous considerations are both related to the structured part of the mesh.
In chapter 1 the law of the wall was introduced. The division of the inner layer, the one where the effect of the viscosity in not negligible, have been mentioned.
The full resolution of the boundary layer is required when the boundary layer effects are necessary for the goal of the simulation as adverse pressure gradients, aerodynamic drag,force over a body and heat transfer. In order to solve the near-wall region the first cell must be placed in the viscous sublayer ($y^+ \leq 5$), usually it's place at $y^+ \simeq 1$. This strategy is considered when the viscous sublayer needs to be solved, instead higher

value of y^+ could be considered when wall functions are used. The latter is used when boundary leyer effects are secondary.This is not the case.
The strategy of first cell at $y^+ \simeq 1$ needs to be followed with a turbulence model. In this work negative Spalart-allmaras model due to the required computational effort with respect to the others, for example the aforementioned $k - \omega$ model that involves two equations.
In the mesh process for a 2D case this results in the first prism layer thickness. This thickness is defined using a $y^+ = 1$.It's not necessary to have exactly this thickness at $y^+ = 1$, it's only required to have an idea od that size. A more detailed procedure also considers a y^+ checking after the flow development in the numerical simulations. In this work the main goal wasn't to achieve a CFD solution that best fit with experimental data, so this check point after the simulations is not considered. As mentioned before the data-set created was used for a neural network approach. Although time and resources were used to generate reliable result.
The math used for thickness approximation are based on the following equations:

$$\begin{aligned} Re_x &= \frac{\rho U_\infty L}{\mu}, \quad C_f = \frac{0.026}{Re_x^{1/7}}, \\ \tau_w &= \frac{Cf \rho U_\infty^2}{2}, \quad u_\tau = \sqrt{\frac{\tau_w}{\rho}} \\ \Delta s &= \frac{y^+ \mu}{u_\tau \rho}, \end{aligned} \tag{3.2}$$

where C_f is the skin friction factor and the related equation is for a flat plate.
Both considerations are steps of the preprocessing, no exact values are available. Also for the boundary layer thickness an approximation is used. In particular the equation 1.3 is used since with chord of 1 meter, $\rho = 1.204\ [kg/m^3]$, $\mu = 1,8375 \cdot 10^{-5}\ [Kg/ms]$ and a free-stream velocity $U_\infty \simeq 69,8 m/s$ the Reynolds number is $Re \simeq 4,5 \cdot 10^6$. The lower limit for a turbulent boundary layer over a flat plate is around $Re \simeq 10^5$.
Afterwards the process of mesh generation in centaur was complete by adding a bounding box for the far-field with a size of 100 chords. Also a source (see fig. 3.4), that take place for the majority behind the profile, has been introduced in order to better solve the wake.

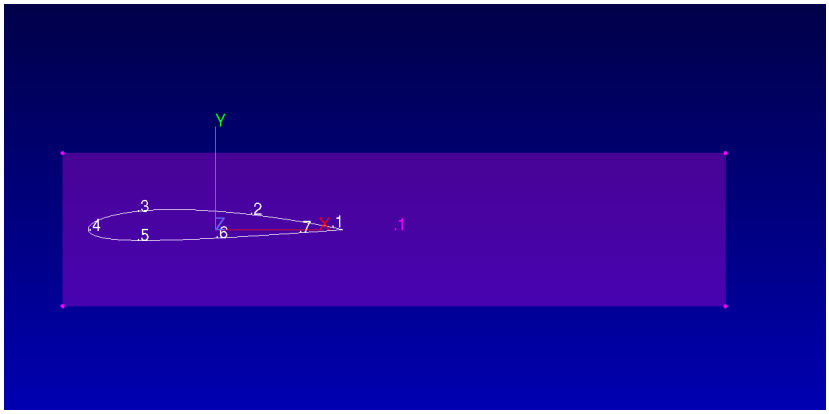


Figure 3.4: CAD-draw of a 2D NACA 2412 airfoil with wake source

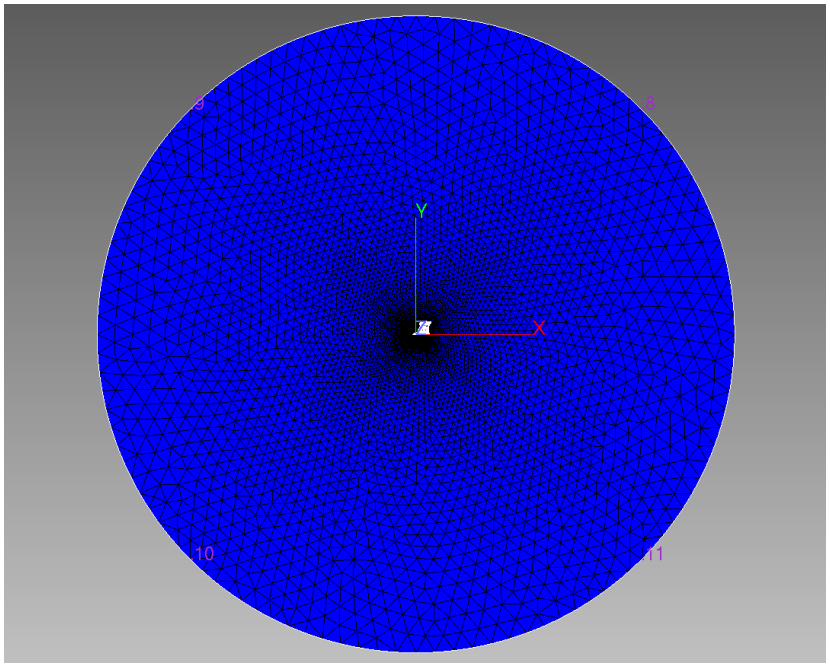


Figure 3.5: NACA 2412 complete mesh

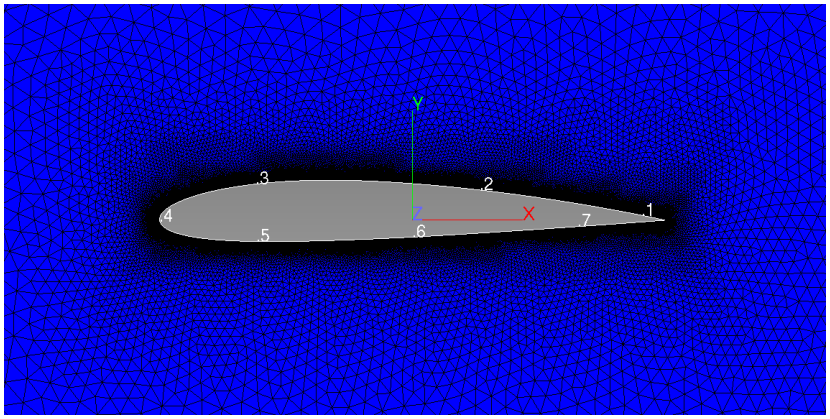


Figure 3.6: zoom of NACA 2412 around the airfoil

The meshing of the domain is showed in figure 3.5 and figure 3.6. The majority of the mesh in unstructured with triangular cells. Structured mesh is used in proximity of the airfoil with rectangular cells in order to capture the boundary layer.
One necessary clarification is that for a 2D simulation DLR-TAU use a 3D grid with only one cell layer in spanwise direction. The Mesh after centaur step is 2D, the final mesh ready for the simulation is generated with "centaur2tau" program.
The grid-adaptation algorithm available for DLR-TAU code was used. In the related work since the meshes at the beginning were created for profiles with chord = 50 mm it was necessary in order to not waste too much time and to not start again the whole process. The aim of this algorithm is to build, given a mesh and a solution on it, a new conforming mesh as a result of certain element refinements and in case of a viscous calculation on a hybrid mesh as a result of redistribution of points on wall normal rays depending on the given solution and the corresponding refinement indicator. A clear explanation of how it works is given in [30].

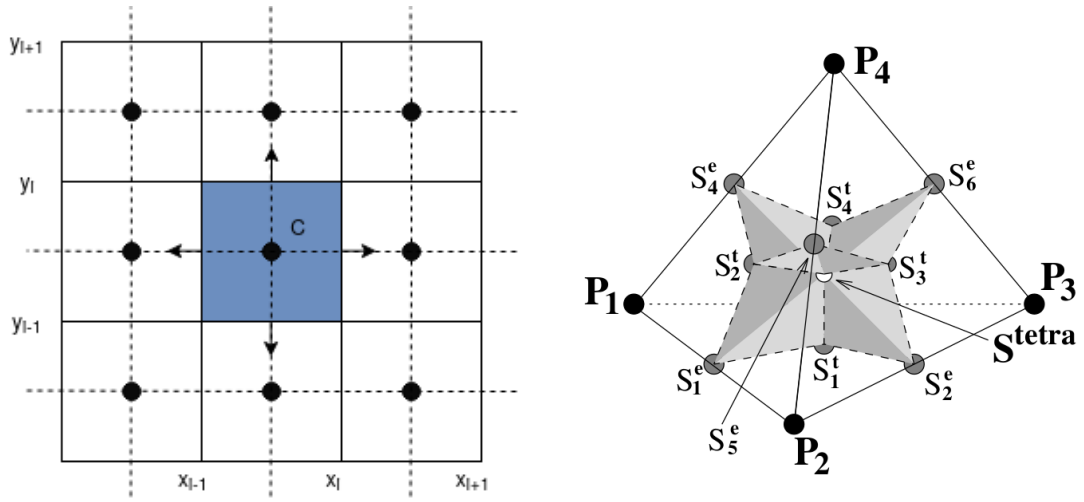
3.3 DLR-TAU solver

It has been already introduced that TAU code is a finite volume scheme for solving RANS equations. Three aspects of this code are here presented :

- dual grid
- spatial discretization schemes
- time discretization.

3.3.1 Dual grid

The flow solver used stores the flow variables on the vertices of the initial grid. This is cell vertex spatial discretization with dual metric. In DLR-TAU code this is done in the preprocessing step. In dual metric approach there are a primary grid and a secondary grid.
The primary grid is the developed through a third-party package, in this case Centaur. This primary grid consist of polyhedral elements with triangular and quadrilateral surfaces. It's the one that spatially discretize the physical space in cells. TAU allows hybrid mesh for the primary grid. The secondary grid is generated from the primary. It contains the data necessary for the flow solver. Two different approach could be realized: cell vertex and cell centered. The cell vertex grid metric is performed in TAU. It stores the variables with the vertices of the primary mesh. In this way primary and secondary grid share same point in the physical space. The difference is in the control volumes, They don't coincide. The control volume in the secondary grid surrounds the vertex, see as reference figure 3.7.



(a) 2D case

(b) 3D case [30]

Figure 3.7: Cell vertex grid metrics

In figure 3.7-a the 2D case is presented. The dashed lines are edges of the primary grid, solid lines are edges of the secondary grid, black points are vertices of the primary grid where flow variables are stored. Arrows indicate direction of flux vector and outer normal vectors of the central control volume. Instead figure 3.7-b represents the 3D case, in particular a tetrahedron. A clearer explanation is given in [30].

3.3.2 Spatial discretization schemes

The viscous fluxes for the one equation turbulence models with central schemes are discretized using central differences. For the inviscid part of flux vector central and upwind schemes are available.

In upwind scheme the flux discretization can be chosen from several different functions: Van Leer, AUSMDV, AUSMP, Roe, AUSM Van Leer, EFM, MAPS+. In the presented work AUSMDV [31] is considered when upwind scheme were used. When an upwind scheme is used , the same upwind scheme will be used for the turbulence equations. The central methods is available with two different artificial dissipation models: scalar dissipation and matrix dissipation.The artificial dissipation is required in order to ensure the stability of the computation [30]. It has been use the matrix dissipation model [32] .

After the introduction of the spatial discretization scheme equation 1.14 can be wrtitten as:

$$\frac{d}{dt}\vec{W} = -\frac{1}{V} \cdot \vec{Q}^F \tag{3.3}$$

where $\vec{Q}^F$ represents vector of fluxes over control volume boundaries and V is the site of the control volume considered.

3.3.3 Time discretization

Steady-state computations

The variation in time for flow quantities can be written as:

$$\frac{d}{dt}\vec{W}(j) + \vec{R}(j) = 0. \tag{3.4}$$

where j indicates the point considered. For steady state case $\frac{d}{dt}\vec{W} = 0$ and equation 3.4 becomes:

$$\vec{R}(j) = 0\ . \tag{3.5}$$

Looking at equation 3.3, equation 3.4 becomes:

$$\vec{R}(j) = \frac{1}{V(j)}\vec{Q}^F(j), \tag{3.6}$$

where $\vec{Q}^F(j)$ represent the discretised fluxes at control volumes boundaries. For steady state problems a fictitious pseudo-time t^* is introduced:

$$\frac{d}{dt^*}\vec{W}(j) + \vec{R}(j) = 0. \tag{3.7}$$

The integration over this fictitious time is performed by an explicit Runge-Kutta scheme [33]. In order to overcome the issues to use an explicit method instead of an implicit one, where for example the time step could be chosen much larger with respect to the other one, the LUSGS-method [34] is implemented in the DLR-TAU code.

Time-accurate computations

In this case DLR-TAU code provides two different options[30]:

- global time stepping
- dual time stepping.

Global time stepping follows K-steps Runge Kutta scheme [33].
Dual time stepping discetization, that it's used for unsteady simulations in this work, involves two discretization: with real time and fictitious time. The first step consider Backward difference formula for the time derivative. It's also possible to employ different order of accuracy: first , second or third. The following equation is for the second order of accuracy:

$$\frac{3}{2\Delta t}\vec{W}_j^{n+1} - \frac{4}{2\Delta t}\vec{W}_j^{n} + \frac{1}{2\Delta t}\vec{W}_j^{n-1} = -\vec{R}(\vec{W}_j)^{n+1} \ . \tag{3.8}$$

Here a sequence of steady state problems arises with unknown $\vec{W}_{(j)}^{n+1}$. The fictitious time in introduced as follows:

$$\frac{d}{dt^*}\vec{W}(j) = -\vec{R}^{DTS}(\vec{W}_j^{n+1}) \tag{3.9}$$

with the modified residual $\vec{R}^{DTS}$ (DTS - Dual Time Stepping):

$$\vec{R}^{DTS}(\vec{W}_j^{n+1}) = \vec{R}(\vec{W}_j^{n+1}) + \frac{3}{2\Delta t}\vec{W}_j^{v} - \frac{4}{2\Delta t}\vec{W}_j^{n} + \frac{1}{2\Delta t}\vec{W}_j^{n-1} \ . \tag{3.10}$$

The last equation can be integrated using Runge-Kutta or LUSGS scheme.

Acceleration Techniques

In order to accelerate the convergence DLR-TAU code provides three different techniques [30]: local time stepping, Residual smoothing and Multigrid. In local time stepping improves convergence by using the maximum allowed time step individually for each control volume. Stability limited is reached everywhere in the flowfield. Residual smoothing can be use after having calculate the residual and it can be done explicitly or implicitly. In Multigrid approach the solution is transferred between of different size of meshes to employ the advantages of both. Coarse grids offer not precise results but fast convergence. inversely, fine grids have precise results but they show slow convergence.

3.4 CFD results

The aim of CFD simulations involved in this work were lift and drag coefficients for NACA 4-digits airfoil. Not all the possible configurations were tested since it was not helpful for the main goal of the project.
For the numerical simulation the airfoil is treated as a viscous wall the farfield in DLR-TAU code implements inflow and outflow conditions.
The next table includes the profiles used in the related work:

Table 3.2: series of 4-digit NACA profile tested

0006	0025	2218	2221	2225	2306	2309
2312	2315	2406	2410	2411	2412	2421
2425	2506	2509	2521	2525	2606	2609
2612	2625	2706	4206	4306	4325	4506
4525	4606	4625	4706	4725	6206	6225
6306	6325	6406	6425	6506	6625	6706

Those profiles were tested for AoA from $-5°$ up to $25°$ and for AoA= $30°$ and for Mach = 0.2 and Mach = 0.5. Steady and Unsteady simulations were performed. Up to the stall steady state simulations were used, for angles of attack greater of α_{stall} unsteady simulations were performed. To verify the validity of the of the simulations a residual control was considered. Furthermore, a Cauchy convergence control was implemented. Some parameters has to be set when Cauchy convergence control is implemented in DLR-TAU cod. Firstly the Cauchy control variable has to be chosen, in the related work they are lift and drag coefficients. Additionally DLR-TAU code provides four different ways to perform the control: relative or relative dynamic, absolute or absolute dynamic. Here the relative is considered: $\frac{|c^n - c^{n-k}|}{|c^n|} \leq \epsilon_{conv}$ for all k=1,..,$(N_{sample} - 1)$, where c^n is the control variable at the inner iteration n,ϵ_{conv} is the threshold value and N_{sample} are the number of samples considered for convergence estimation. Table 3.3 includes the parameters considered for relative Cauchy convergence control.

	Steady	**Unsteady**
Control variables	C-lift C-drag	C-lift C-drg
$\boldsymbol{N_{samples}}$	3000	50
$\boldsymbol{\epsilon}$	$1e-6$	$5e-5$

Table 3.3: Cauchy convergence control setting

The data-set created has around 2700 samples. One can see that in Table 3.2 a great part of profile test have thickness -06 and -25. Since the data-set was then analyzed with NN and since NN learns from the data, they produce better results for interpolation than extrapolation. Following this a boundary in term of thickness for the profiles was considered. Some of those profile are used to test the final model.
The grid-adaptation aforementioned was applied twice for the same airfoil, one for each mach at AoA = $0°$. It was done for only one AoA since CFD results arrived from an automatic switching from an AoA to the next one. In this procedure it was not possible to automatize the grid-adaptation. Additionally, the procure repeated for each profile and for each AoA would have required a massive amount of time and then the automatic simulation for different AoA would not have been possible.

3.4.1 CFD analysis

For one the the profile tested, NACA 2606, a brief analysis and a presentation of the CFD results is here given. The following table (table 3.4) presents the most important values for the cfd simulations.

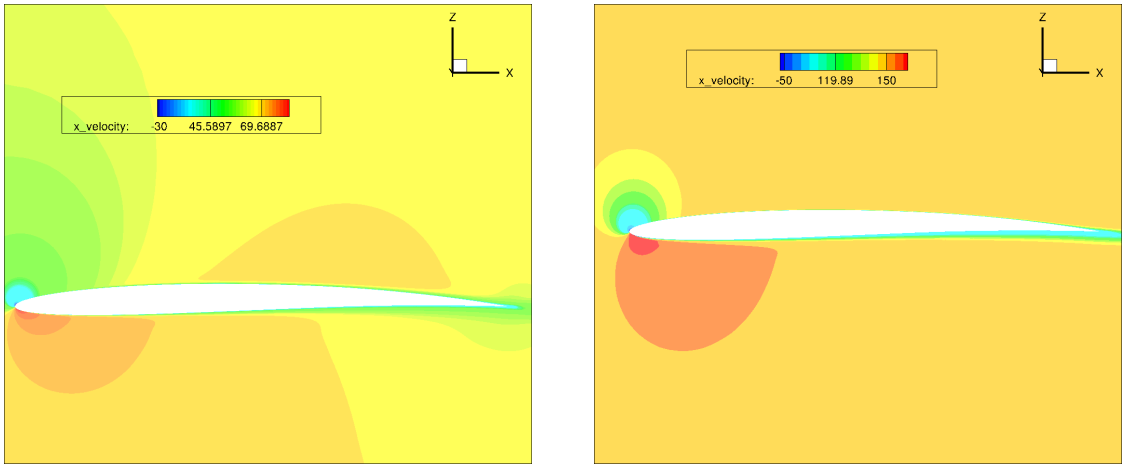
Variable	Value	Unit
Mach	0.2	-
Reynolds	$4.6 \cdot 10^6$	-
Temperature	293	K
Density	1.204	$\frac{kg}{m^3}$
Velocity	68,9	$\frac{m}{s}$

(a) simulations at Mach=0.2

Variable	Value	Unit
Mach	0.5	-
Reynolds	$11 \cdot 10^6$	-
Temperature	293	K
Density	1.204	$\frac{kg}{m^3}$
Velocity	174.5	$\frac{m}{s}$

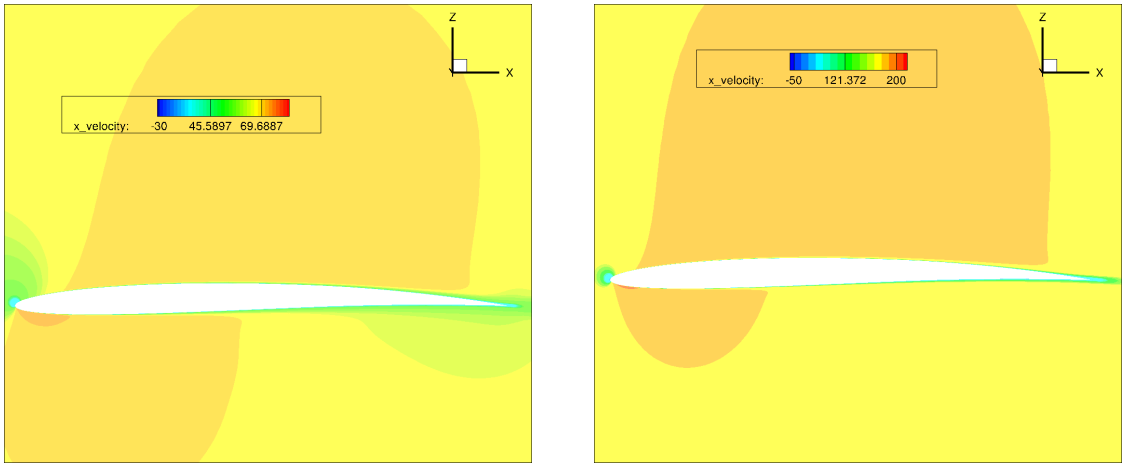
(b) simulations at Mach=0.5

Table 3.4: Reference values for CFD simulation



(a) Mach = 0.2 (b) Mach = 0.5

Figure 3.8: x-velocity profile for NACA 2606 at AoA=$-5°$

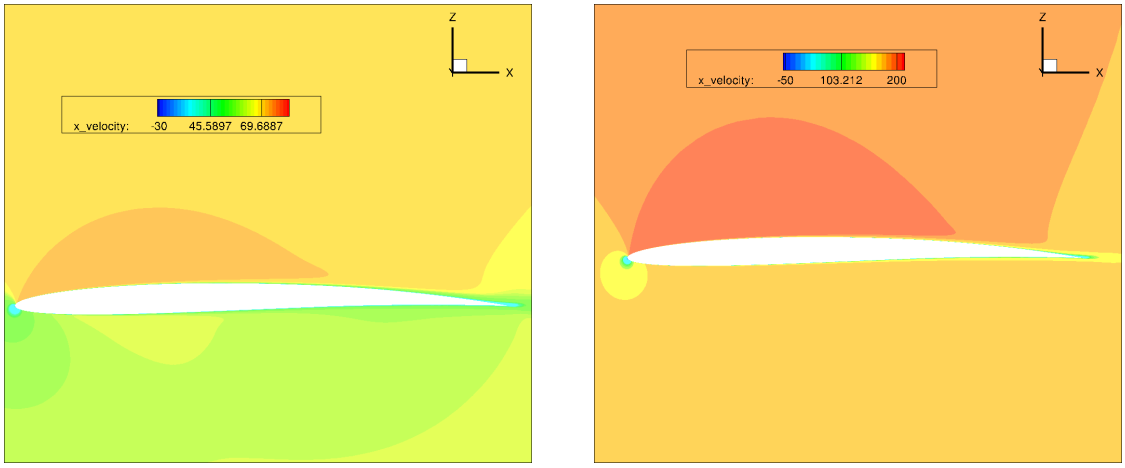


(a) Mach = 0.2 (b) Mach = 0.5

Figure 3.9: x-velocity profile for NACA 2606 at AoA=$-1°$

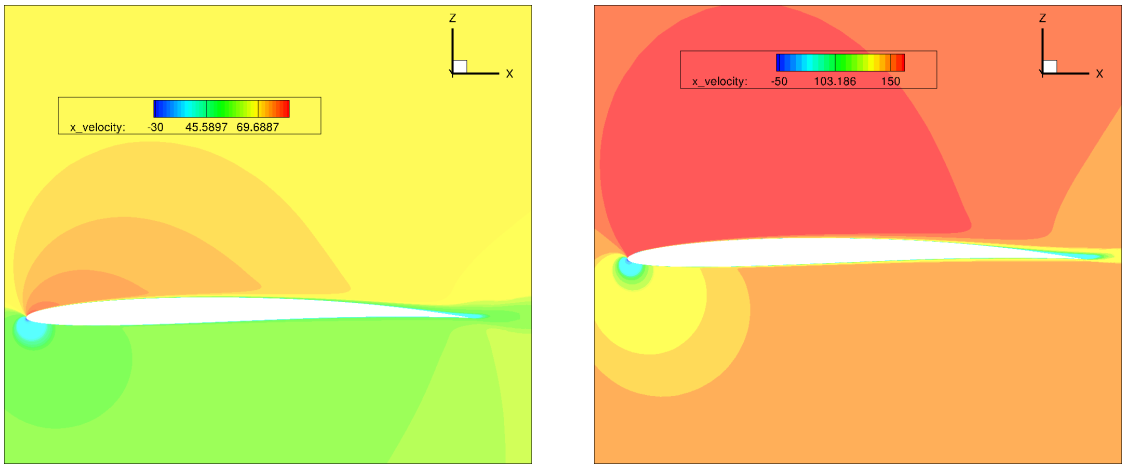
From the figure 3.8 to 3.15 it can be seen the x-velocity profile around the NACA 2606 airfoil at $Mach = 0.2$ and $Mach = 0.5$. In particular several AoA are shown

$(-5°, -1°, 2°, 5°, 8°, 10°, 12°$ and $15°)$. In figure 3.8 the velocity profiles look similar. They present in the lower surface a higher velocity due to to negative negative AoA. The stagnation point in slightly moved backward in $Mach = 0.5$. The lift coefficient for mach= 0.5, $Cl = -0.32047$ is more negative with respect to the one at $Mach = 0.2$, $Cl = -0.28683$. Respectively the drag coefficient is lower for $Mach = 0.2$ (at $Mach = 0.2$ $Cd = 0.00907$ instead at $Mach = S0.5$ $Cd = 0.01273$). Decreasing the AoA up to -1 degrees, figure 3.9 leads to an increase of velocity in both upper surfaces. Lift and drag are reduced for both cases.



(a) Mach = 0.2 (b) Mach = 0.5

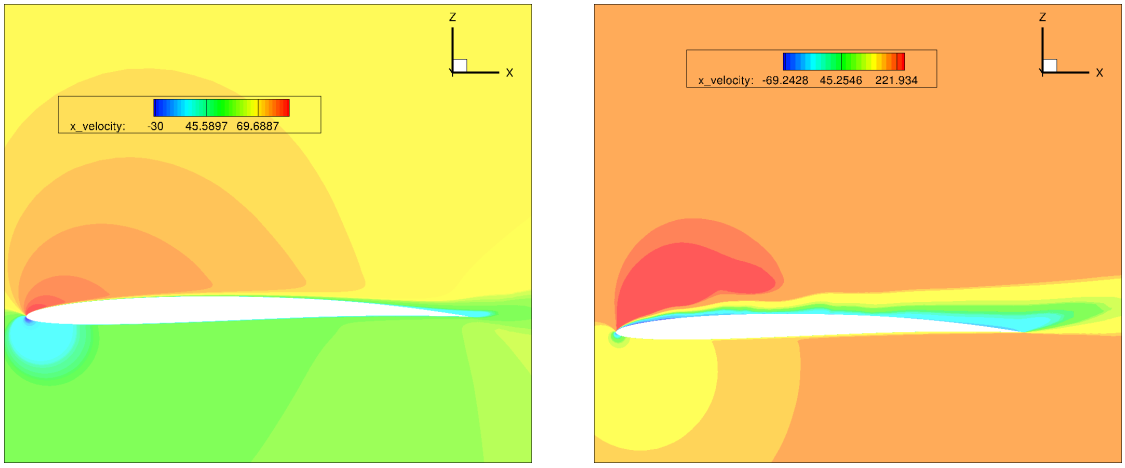
Figure 3.10: x-velocity profile for NACA 2606 at AoA=2°



(a) Mach = 0.2 (b) Mach = 0.5

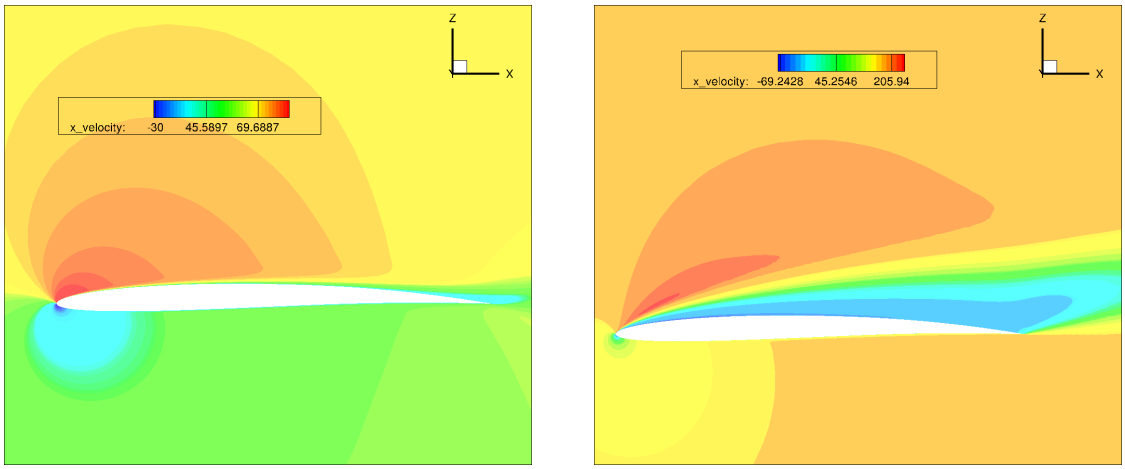
Figure 3.11: x-velocity profile for NACA 2606 at AoA=5°

In figure 3.10 and figure 3.11 one can notice that the upper surfaces are completely characterized by a higher velocity value that correspond to a lower pressure. Lift is now positive and drag start to increase after the reduction due to the previous decrease of the AoA. With $Mach = 0.5$ case at 8 degrees NACA 2606 is close to flow separation. The velocity at trailing edge is reducing but the flow is still attached.



(a) Mach = 0.2 (b) Mach = 0.5

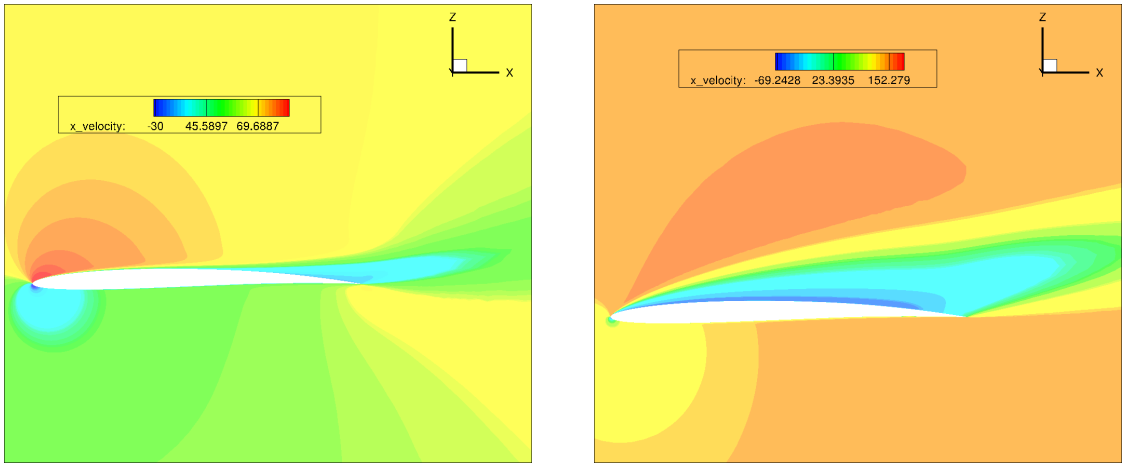
Figure 3.12: x-velocity profile for NACA 2606 at AoA=8°



(a) Mach = 0.2 (b) Mach = 0.5

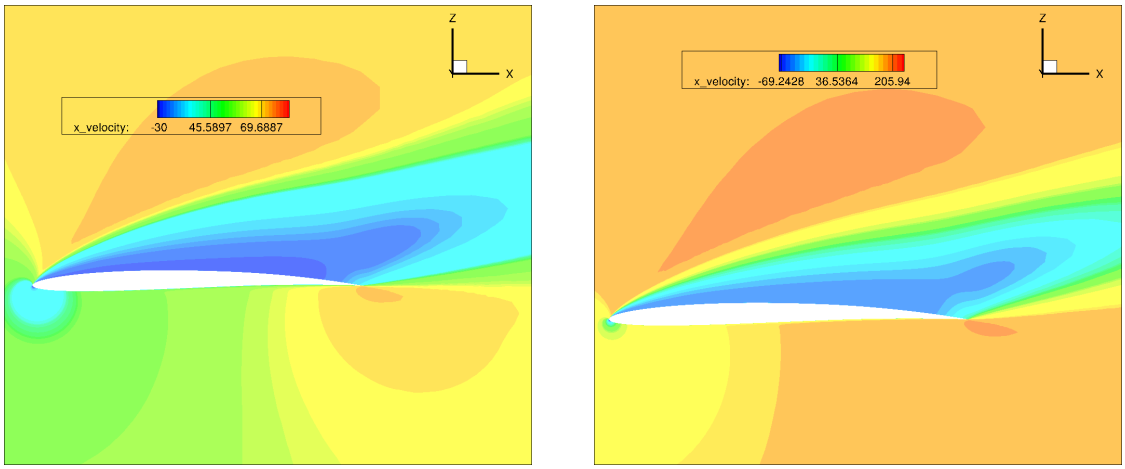
Figure 3.13: x-velocity profile for NACA 2606 at AoA=10°

Increasing again the AoA the stall is becoming closer, it is possible to notice it at $Mach = 0.5$ case in figure 3.12. The boundary layer thickness is increasing, see figure 3.13-b (light blue part around the airfoil). At both Mach at AoA = 8° the stagnation point is on the lower surface, increasing the AoA it is moving on the lower surface towards the trailing edge. The flow at mach=0.5 the flow start to separate. Figure 3.13-a exhibits the region of high suction on the upper surface (white zone) with the maximum velocity. Instead, in figure 3.13-b the flow is completely separated from the upper surface.



(a) Mach = 0.2 (b) Mach = 0.5

Figure 3.14: x-velocity profile for NACA 2606 at AoA=12°



(a) Mach = 0.2 (b) Mach = 0.5

Figure 3.15: x-velocity profile for NACA 2606 at AoA=15°

From figures 3.14 to figure 3.15 arise the same considerations presented for figure 3.12 and figure 3.13. One can also notice the increase of velocity close to the leading edge.
Including the aerodynamics of the stall will generate a more difficult relation to map with respect a model that try to learn only drag and lift coefficient in the so called linear part of lift coefficient. This may drive the Neural Network to not learn or to overfit. In the next chapter will be explained what overfit is.

Chapter 4

Neural Network approach

In this chapter machine learning approaches in fluid mechanics field are initially introduced followed by the exposition of related work with Neural Network approach. The main goal was not the exact prediction of the lift and drag coefficient but how a good result can be reached with this approach, trying to extract particular behavior for future application in this field.
In the following chapter, when the subdivision in the three different sets (training, validation and testing) is not mentioned these measure are used: 80% of the whole data-set as training-set (20% of which is considered as validation error) and the remaining 20% as testing-set. Moreover, the samples in each different set are chosen randomly.

4.1 Machine Learning for fluid mechanics

The related work falls into machine learning approach for fluid mechanics. In The last years this approach is rapidly increasing. Fluid mechanics has traditionally dealt with massive amounts of data from experiments, field measurements and large-scale numerical simulations. Machine Learning approaches allow to extract information and knowledge from data-set already available or data-set specifically generated.
The main advantage of using ML is the possibility to address this approach to wide ensembles of fluid mechanics application such as experimental data processing, shape optimization, turbulence closure modeling and control [35]. A general overview of fundamentals machine learning methodologies is presented in [35].

4.1.1 Common approach

The work done in this thesis is placed in supervised learning as mentioned in chapter 2. Neural Networks, that are the most well known method in supervised branch, are involved. Neural network has already been used to in order to predict aerodynamic properties od airfoils. In [36] convolutional neural network (CNN) has been used to predict aerodynamic flow-field around airfoils. In particular CNN is used to predict velocity and pressure. In this work CNN are used because as inputs are used visualizations of the flowfield. Specifically, an encoder-decoder CNN is used. It's a different approach but it illustrates the capability of prediction of NN.
A similar approach is involved in [37] where the object of the study was to illustrate

how the training data size influences the accuracy of the solutions. It has been point out that the ability of a CNN to generalize the knowledge extracted not only depends on type and amount of training data but also on the representative capacities of the chosen architecture. It's also mentioned that large errors characterize the most difficult cases of the test data-set. Furthermore the Adam optimizer was used to train the network. Later It will be shown that it's the best choice in the approach here used. In [38] again a CNN network approach but it contains also a comparison between MLP and CNN. It shows that for a given number of epochs CNN has better results than MLP but requires more time. Here the ReLU activation function was used. As for the optimizer, results in the related work leads to this activation function and to the parametric ReLU.
A very close approach to the one that has been followed here it used in [39]. Aerodynamic coefficients are predicted using as inputs the angle of attack,the Mach number and the Reynolds number. In part of [39] is also considered the prediction of drag coefficient for wing-body configuration with variable wing geometry with good results.

4.2 Tools

The data-based approach here involved was developed using TensorFLow [40] on 64-bit Linux platform with Python3 interface. It is a open-source software library developed by google. TensorFlow is Google Brain's second-generation system and the first version was released on February 2017. The version here used is the 2.0 available since January 2019. Additionally, Keras [41] was used. Keras is an application program interface (API) designed for deep learning. It is open-source and it is written in Python Programming Language. Basically Keras is able to run on the top of TensorFlow . It has a simple and consistent interface.

Figure 4.1: Integration of Keras, TensorFlow and Python for deep learning.

4.3 Neural Network analysis

Here the steps followed in this study are introduced. The codes were developed specifically for this case, no codes already defined have been used.
The work here presented, as aforementioned involves NACA 4-digit profiles tested for different AoA and Mach=0.2 and Mach 0.5. Lift and drag coefficients were the object of the predictions.

Firstly, It is necessary to point out two things: the subdivision between input and output and when NN are involved for prediction the most suited activation function between last hidden layer and output is the linear activation function, the other choices have limited range or not uniform behavior. In the related case inputs are flowfield information, profile shape (with the 4 NACA digit series) and the angle of attack. Instead the outputs are drag coefficients an lift coefficients, respectively. Facing an analysis with Machine Learning algorithms doesn't require only to know how the algorithm involved works but it also requires to use in a clever way the data-set. One can say that in machine learning is trivial the way used to handle the data-set. Related to the present work is clear that the neural network involved needs two input neurons, one for the mach number and one for the AoA. An additional input is required for the 4-digit series. From this last input the network has to learn the shape of the profile and it has to recognize the coherency between the considered profile and the lift and drag coefficient. Initially the profile shape was considered as a single input. Using a single input for the profile shape arises two conclusion: the first was that to learn and to recognize the profile shape with only one input neurons the data-set involved is not enough, this could be an interesting question but here it has not been investigated; the second is that in this case the normalization of input data was mandatory. Lets consider a Neural Network with 3 inputs: profile shape, AoA and mach number. The range of the 4-digit series goes from 0006 to 6709, instead for AoA goes from -5 to 25 with also AoA = 30^o and for mach number is only defined by 0.2 and 0.5. With reference to chapter 2, where how a Neural Network works is introduced, it is easy to conclude that having a great difference between the range of the inputs (different scale, look at profile shape range and mach number) is not good for the learning process. The big scale becomes dominating and the performed predictions will not be accurate.

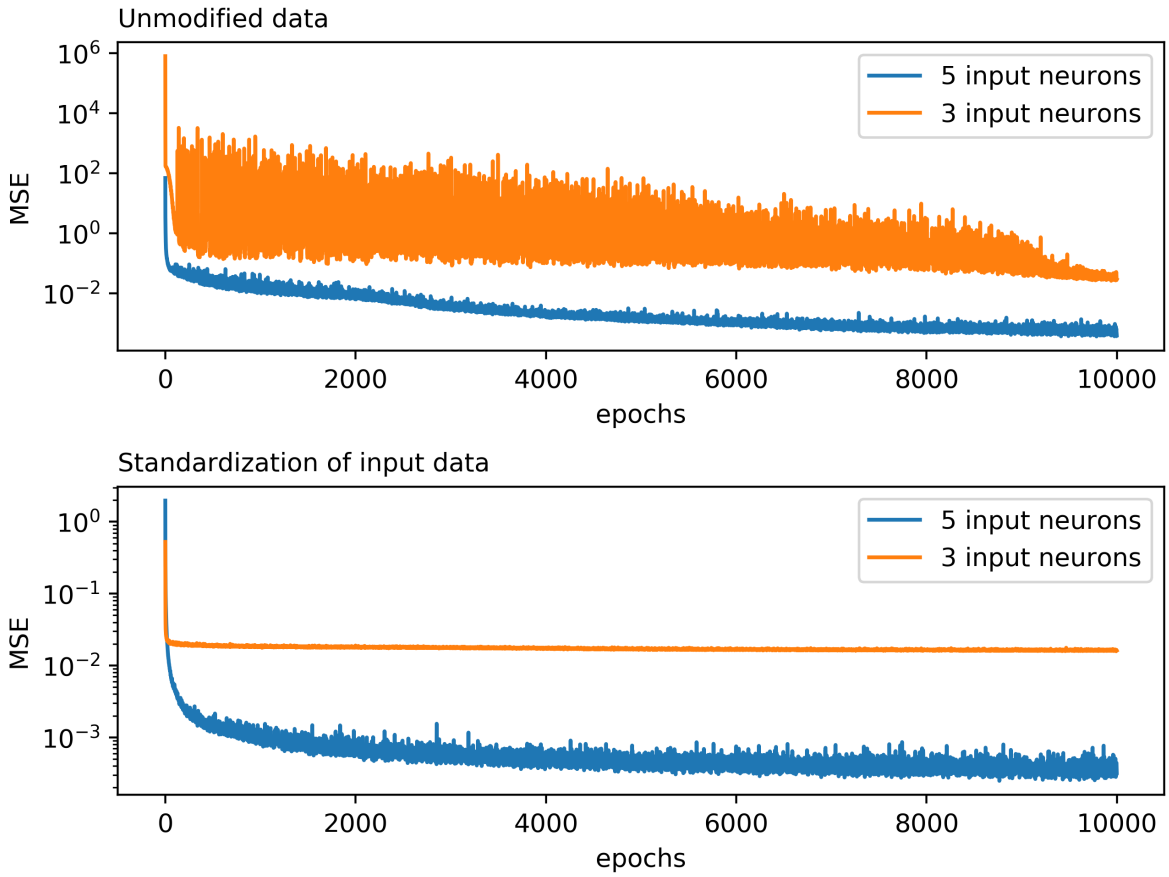


Figure 4.2: Training error vs epochs for NN with respectively 3 and 5 input neurons

Figure 4.2 shows the training error for the same NN architecture but with two different input layers. The relation that the network is trying to learn is the map with the lift coefficient, here only lift coefficient is shown. In particular for this analysis was performed with two hidden layers, each one made of 30 neurons. The other hyper-parameters are: learning rate = 1, batch-size = 30, optimizer = Adam (when no specification is given, TensorFLow default values are used: $\beta_1 = 0.9$ and $\beta_2 = 0.999$. See as reference the paragraph 3.5.4.), loss function = MSE, activation function = Leaky ReLu (with $\alpha = 0.1$) and 10000 epochs were used. Here it is not under investigation how the combination of the hyper-parameters work. Figure 4.2 points out the difference between training with and without normalization and with two different approach for the input layer. Only lift data are used because the purpose here is to point out that more input neurons are used for the NACA 4-digit series better is the learning process. An input layer with three input means that only one is used for the 4-digit series, instead when five inputs are used means that three input neurons are used for the 4-digit series (one for each different digit's meaning). Figure 4.2 shows the training error over the number of epochs. Each plot shows the difference between the two different approaches. Furthermore, the upper one is characterized by the absence of data manipulation. Each input has is normal scale. Instead in the lower one standardization is used. It's possible to see the improvement gave it to the learning when standardization is called. Sometimes standardization and normalization are used as interchangeable. Normalization usually can have multiple meanings. In machine Learning field it's necessary to underline the difference. When Normalization is called a data rescaling is considered. Mostly, it involves a data rescaling between 0 and 1.

One possible formula to achieve this is :

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}. \tag{4.1}$$

Standardization transforms data to have mean of zero and standard deviation of 1. It can be performed with the following formula:

$$x_{new} = \frac{x - \bar{x}}{s}, \tag{4.2}$$

where x is a data point, $\bar{x}$ is the mean and s is the standard deviation. In general standardization is preferred since it helps the convergence when NN are involved. It has to be remarked that it's always worth to test different data transformation and to not assume that one approach is better without testing it. In the related work only input data are modified. The prediction's subject, lift and drag coefficient, are not standardized. Standardization for target values is required when there are two or more target variables [**2002**] for the same reason aforementioned for the inputs, the variation of each target values affect the learning process and target values with different scales can results in one dominating target.
The approach with three input neurons was abandoned in favor of five input neurons approach.
For the following analysis these considerations are taken for granted. Sigmoid and tanh activation functions suffer of the vanishing gradient, if inputs are too large the gradient is zero. Xavier initialization, a specific weights initialization, achieve good variance that let both work better [42]. Instead for ReLu, where dead units problem can occur , He initialization is better [43]. He initialization is also considered for Leaky ReLU activation function. It has to be remarked that weights initialization is not mandatory but it helps the learning process. Weights initialization analysis has not been performed, the advises found in the literature mentioned before have been followed. After these considerations the hyper-parameters setting has been investigated.

4.3.1 Hyper-parameters setting

The design of a neural network involves the selection of some parameters called hyper-parameters as aforementioned. This is required to achieve fast convergence speed during the training and good accuracy when they are used to generate predictions. The main characteristic of this setting is that it doesn't exist a general rule to follow. In the last years many algorithms were developed in order to reach the optimal structure, one of those is the Taguchi-method [22]. Mostly the trial and error approach is used. In the related work in order to understand how to build a predictor with NN the trial and error approach was followed.

4.3.2 Activation function and Optimizer

A cross analysis on optimizer and activation function is here presented. The NN architecture here used is a two layers network with one hidden layers of 60 neurons. The epoch number was set to 5000, the learning rate was set as 10 and the batch size as 50. As most of the time in this work the loss function involved was MSE. The subdivision of the data-set is 80 % of the whole data-set as training set and 20% as testing set. Additionally 15 % of the training set was used as validation set. Input data

were standardized before the training. Scikit-learn [44], an open source library written in python language for machine learning, provided the tool used to standardize. The He initialization [43] is considered.

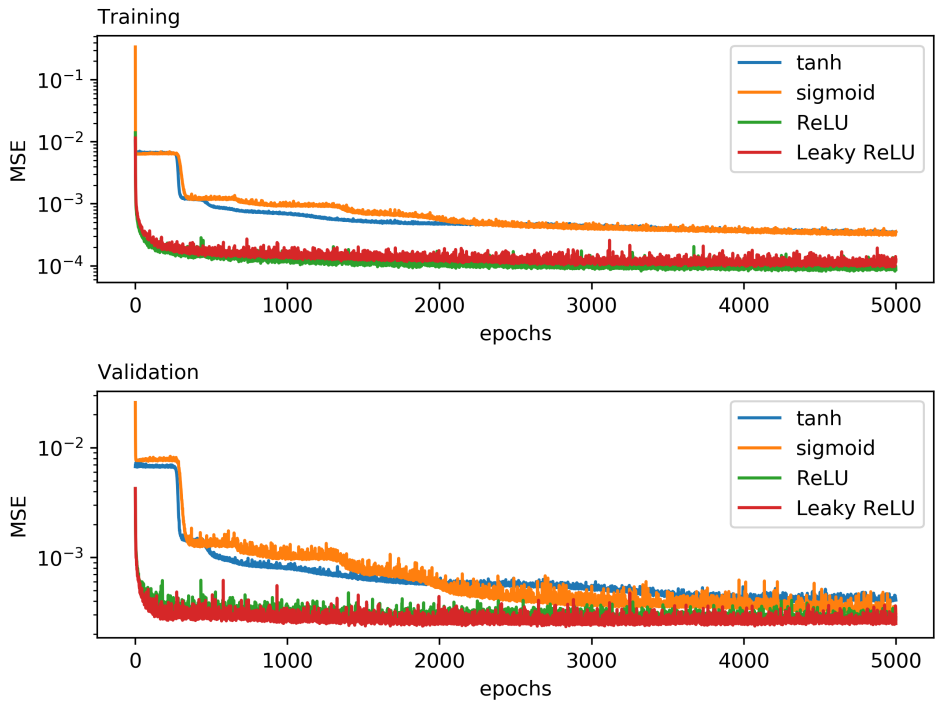


Figure 4.3: Training error and validation error with Adam optimizer.

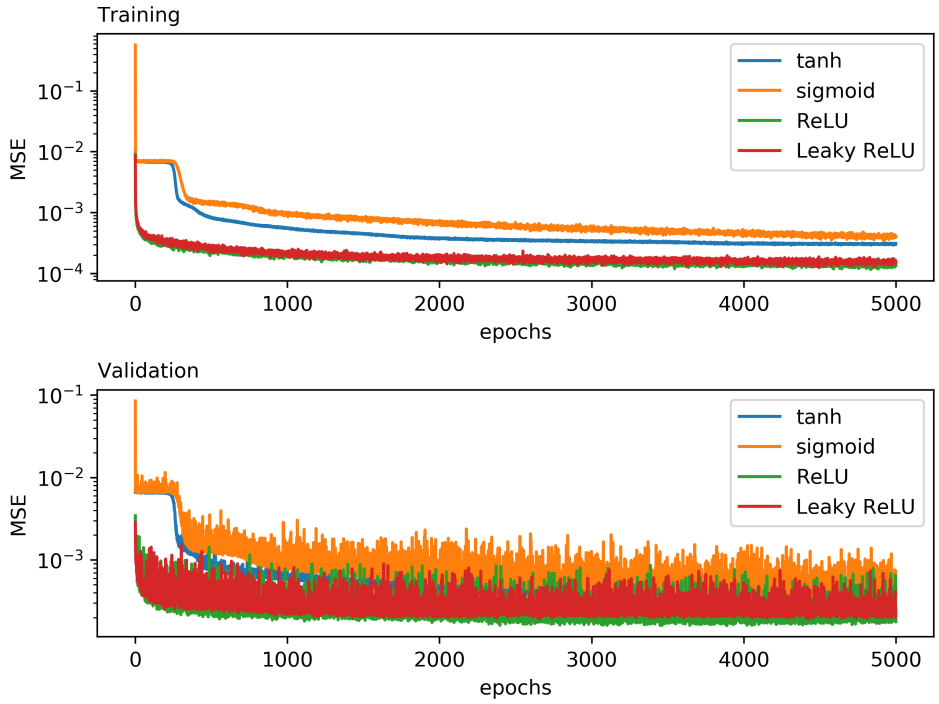


Figure 4.4: Training error and validation error with RMSprop optimizer.

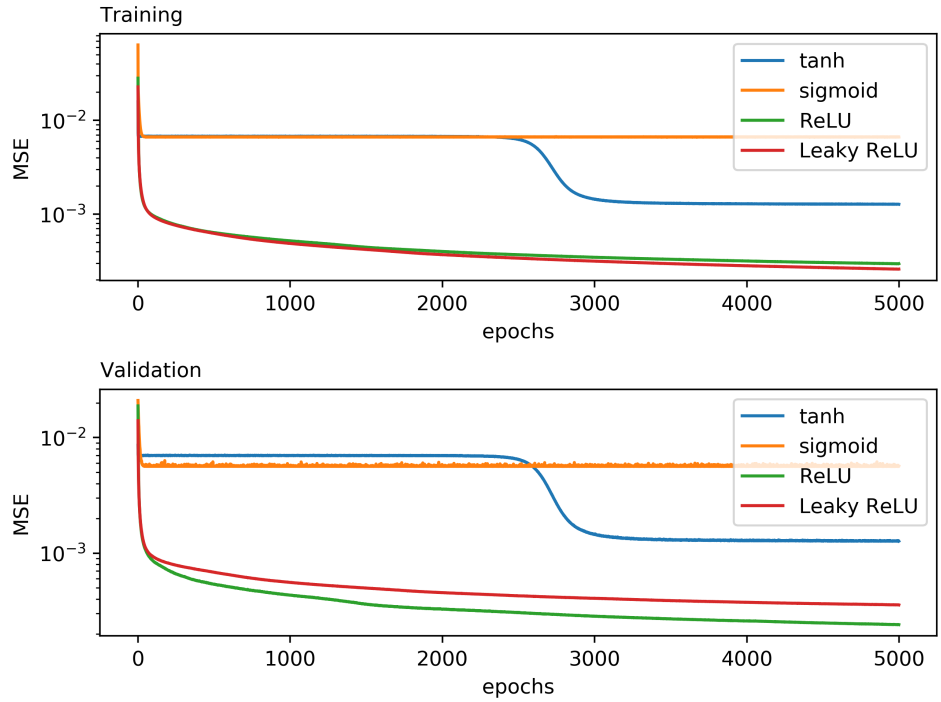


Figure 4.5: Training error and validation error with SGD optimizer.

The network in figure 4.3, 4.5 and 4.4 is using drag data. The learning process of the two coefficients is separately analyzed. When results are similar, as for this case, only one of the two learning process in showed. The aim of this cross analysis was understand how the combination of optimizer and different activation functions work for the related problem. From the the theory the best combination seemed to be the ReLU/leaky ReLU activation function and Adam optimizer. The latter one makes also the use of the second moments of the gradient with respect to RMSprop that involves only the first moment (the mean). The combination of this and the high initial learning rate lead to visible oscillations in RMSprop validation error (figure 4.4). The Adam optimizer, as improvement of RMSprop, has lower oscillations. Furthermore, it doesn't matter which optimizer is used, ReLU and Leaky ReLu perform better with respect to sigmoid and tanh activation functions. Relu and Leaky ReLU have a faster convergence in terms of epochs. Each different optimizer shows a particular behavior for tanh and sigmoid activation function: they got stuck in a local minima. In the presented case also with a learning rate of 10 using the sigmoid activation function,in figure 4.5, with 5000 epochs is not able to leave the local minimum (vanishing gradient problem). It was explained that usually tanh performs better the sigmoid, this is the case since after 3000 epochs using tanh activation function the network start to learn again. This analysis involved He initialization that is devoted for ReLU and Leaky ReLu activation functions. Instead, as mentioned before, for the sigmoid and tanh activation function that are affected by the vanishing gradient problem the recommend weights initialization is the Xavier initialization [42]. The following figure shows for the Adam optimizer how the initialization can help the sigmoid and tanh activation function at the beginning of the learning process. The same hyper-parameters setting is considered. The xavier initialization is applied only to tanh and sigmoid activation functions, for the ReLu and Leaky ReLu again the He is considered.

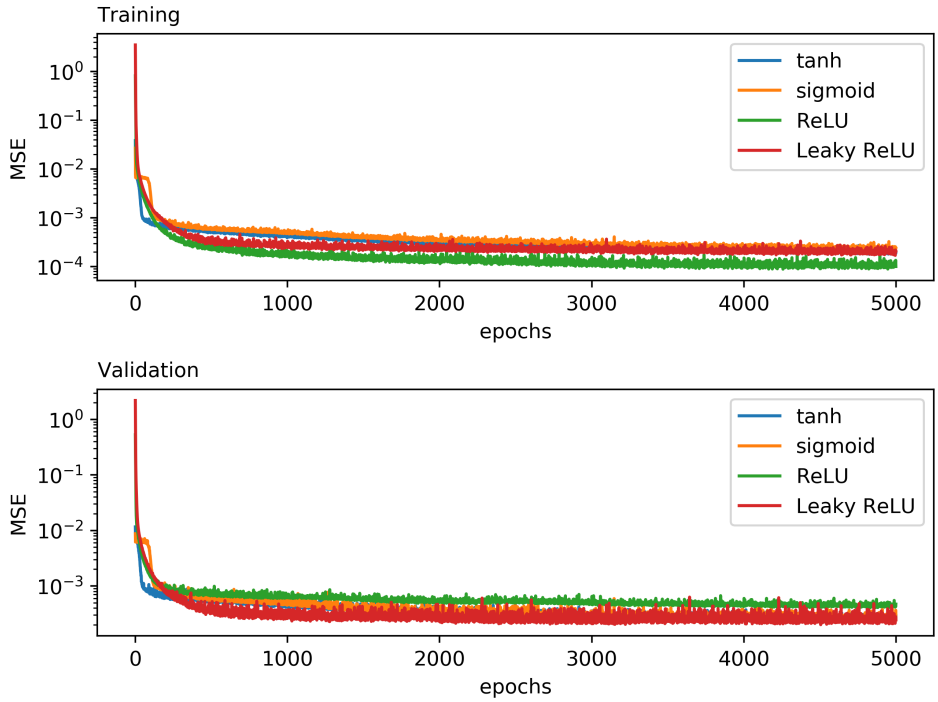


Figure 4.6: Training and validation error with ADAM optimizer. Furthermore, He initialization is used for ReLU and Leaky ReLU instead the xavier initialization is applied to Sigmoid and Tanh.

It is shown only the analysis with Adam optimizer because it has the most evident changes. The Adam optimizer has benefits respect to SGD and RMSprop , they have been aforementioned. These benefits with the right choice of weights initialization helps to overcome the vanishing gradient problem. The step present in figure 4.3 is almost vanished in figure 4.6. When Tanh is involved one can say that the weights initialization is enough to delete the step instead for sigmoid activation function some epochs are required. One can also notice the higher initial values for training and validation with ReLu and Leaky ReLu activation function. This behavior can be explained with the random characteristic of weights initialization. When a specific initialization is applied for two different learning process it doesn't imply that the weights are equals in both process. There is still a source of randomness. Also if the learning process starts with a higher loss values but the right choice have been done (activation function, optimizer and weights initialization), this initial higher value doesn't affect the remaining part. It is worth to mention that when more hidden layers are considered, this is not the case, it is common practice to applied weights initialization at the beginning of the training for each hidden layers introduced in the NN.

4.3.3 Learning rate and Batch size

The following analysis is related to batch size and and learning rate. Here lift data are used. The architecture includes 3 hidden layers, each with 30 neurons. The optimizer is Adam and the activation function is Leaky ReLU ($\alpha = 0.1$). Three different batch sizes (BS) are considered (BS = 1, BS = 30, BS = 200) and four different learning rate (LR) values are considered (LR = 0.001, LR = 1, LR = 50, LR = 100).The aim of this investigation was trying to extract the relation between this couple of hyper-parameters, specifically in the extreme case : low learning rate value-high batch size

and high learning rate-low batch size.

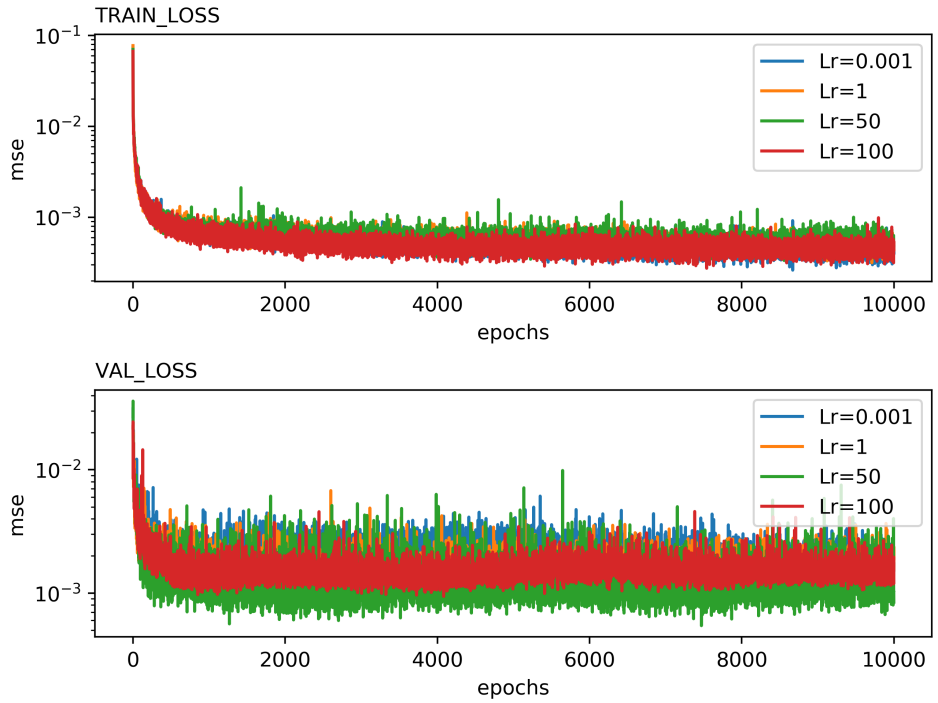


Figure 4.7: Training and validation error with BS = 1 and three differen LR values

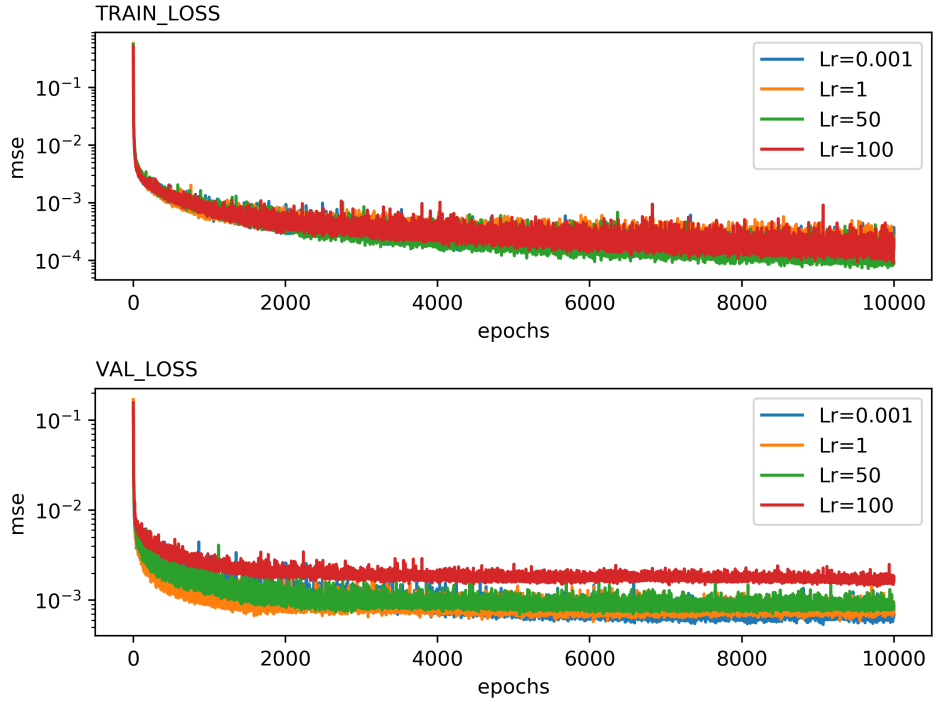


Figure 4.8: Training and validation error with BS = 30 and three differen LR values

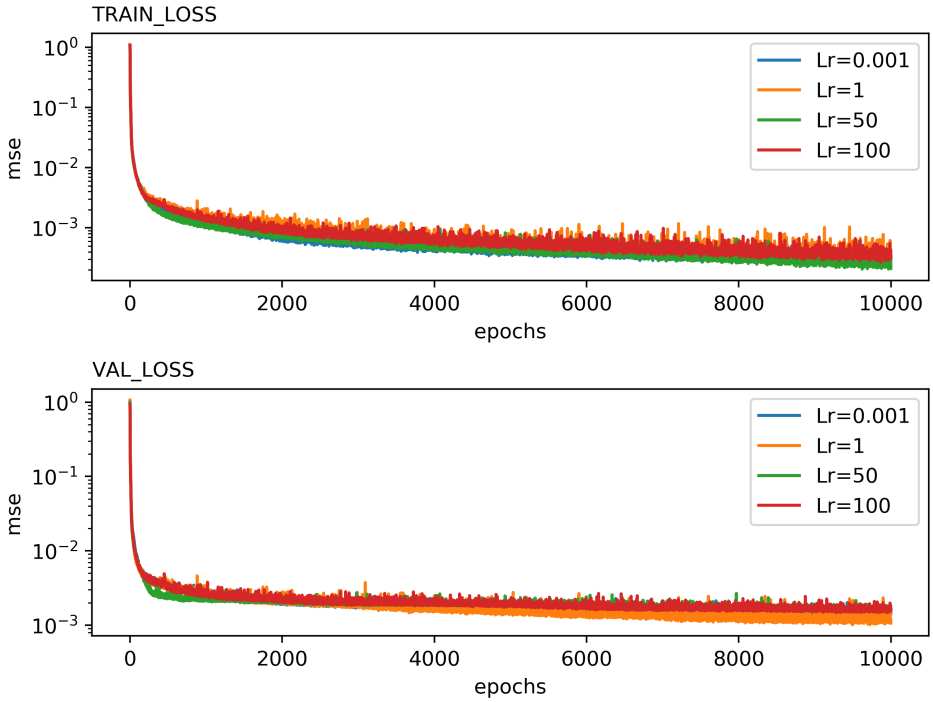


Figure 4.9: Training and validation error with BS = 200 and three differen LR values

Generally small batch size offers a lower generalization error, just think that with batch size equal to one the model hyper-parameters are updated after each data sample. The network try to find the coherency between each sample and this leads to a lower generalization since only one sample is involved in the learning process per time. It can been seen that with batch size equal to 1 the training and validation curves in figure 4.7 start with a lower value with respect to BS = 30 (figure 4.8) and BS = 200 (figure 4.9). This is due to the number of parameter updates. The number of parameter updates per epoch is the number of sample, used in the training, divided ny the batch size. Considering this is clear why with batch size equal to one correspond to lower training and lower validation error at the beginning of the learning process. The lower generalization ability is clearly visible from the asymptote reached. For BS = 1, see as reference figure 4.7, the learning process start with a lower value with respect to BS = , is around $2 \cdot 10^{-3}$, considering the validation loss, with minimum values around $6 \cdot 10^{-4}$. When batch size is set to 30 (figure 4.8) the asymptote is $1 \cdot 10^{-3}$. Looking at the training error is more evident, for BS = 30 training error leans 10^{-4}. Additionally , as expected, the loss function with BS = 1 shows important oscillations of the order of $9 \cdot 10^{-3}$. Usually the term "noisy" is used to indicate the BS = 1 case (or more in general for small BS values), since random samples are used sequentially and due to this the direction of the gradient may vary more than when a bigger batch size is used. See as reference the figure (figure 4.9) when oscillations have magnitude close to 10^{-3}. In conclusion high learning rate, when batch size 1 is used, increase oscillations for the validation error. It is possible to notice that when learning rate is set to 100 the amplitude od oscillations is lower than for LR = 50 but the minimun validation error is higher for LR = 100. With LR = 50 the learning process goes close to the global minimum, when LR increases until 100 it is not able to get closer. The oscillation problems can be considered solved when high batch size are used. As found in literature [45], using large batch size leads to a degradation in the quality model (ability to generalize). There is no exacts relation between the

batch size and the number of sample involved in the learning process. An intermediate value , as BS = 30 (smaller values are preferred to larger for th reasons just explained), is usually set. In the related work from figure (4.8), one can conclude that BS = 30 with LR = 1 or lower are a proper choice in order to achieve a good level of accuracy.

4.3.4 Epochs and Overfitting

Regarding the number of epochs a very common problem has to be mentioned: over fitting. In the chapter 2 the definition of epoch number has been introduced: it defines the number of times that the training error is feeded to the network. Its counter part is underfitting. In statistics, a fit refers to how well you approximate a target function. A particular model overfits when it models the training data too well. A drastic example of overfitting is shown in figure 4.10.

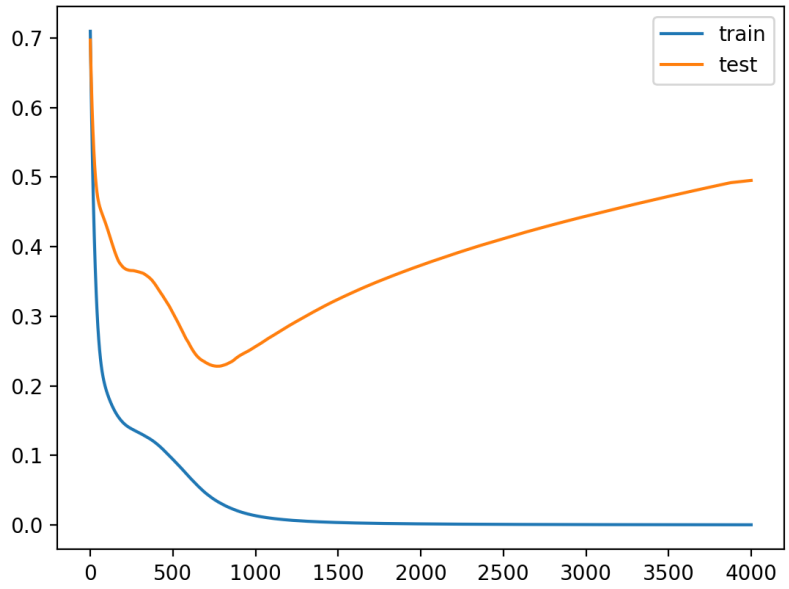
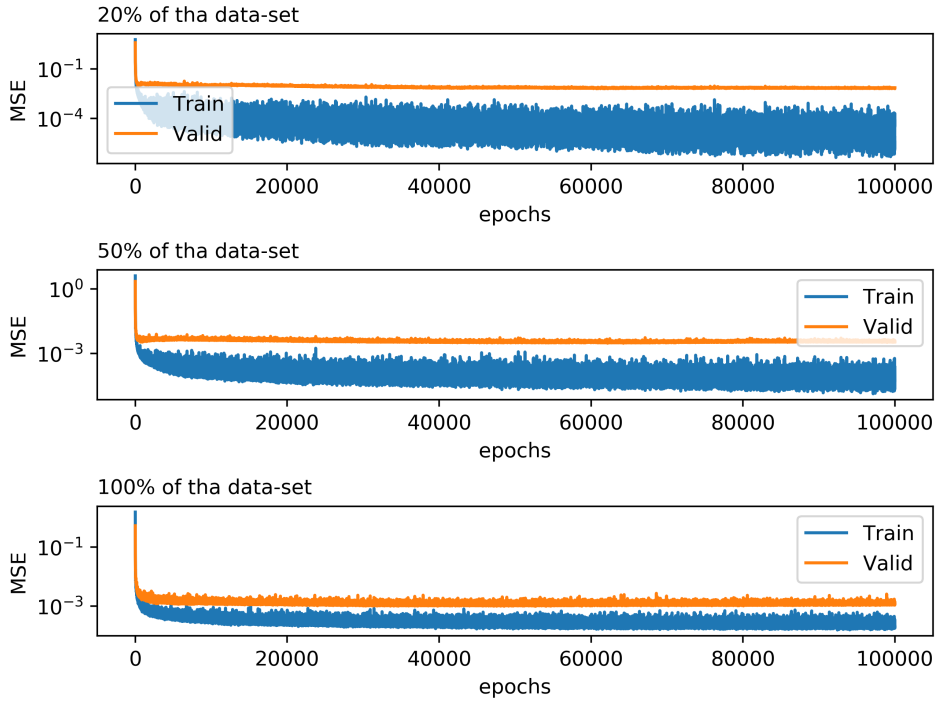
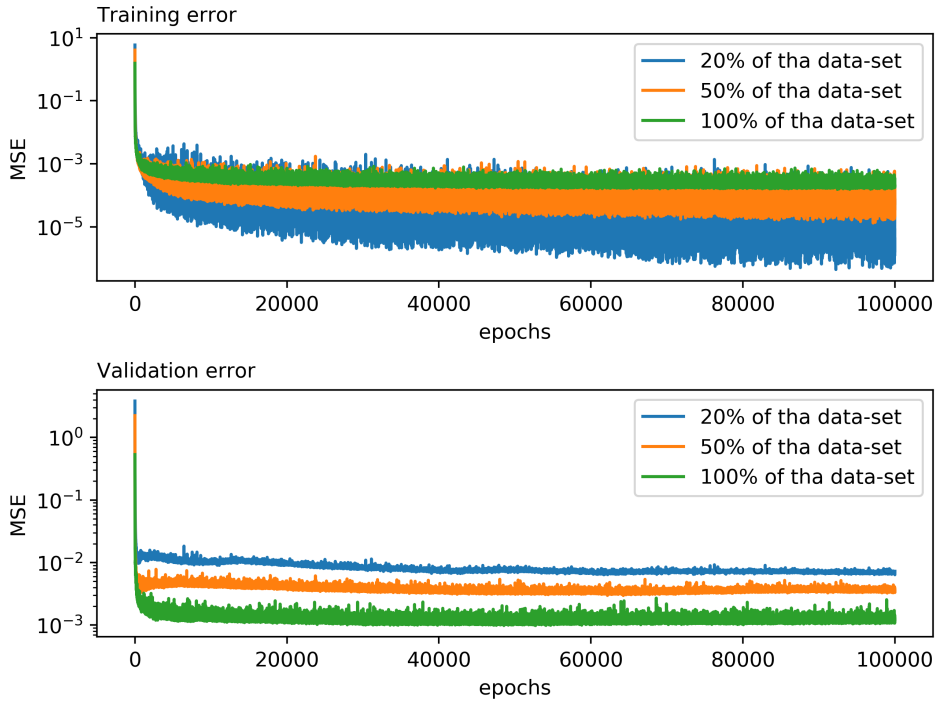


Figure 4.10: Overfitted model: error vs epochs.

Overfitting is a particular situation that arise when the model continues to learn from the training data but it's not able to generalize well using unseen data. In figure 4.10 the model start to overfit around 750 epochs. Before that 750th epoch is reached the testing error is observably higher than the training error, but until when it continues to decrease the model is not overfitted. The undefitting occours when both, training and testing error, have high value. The model is not able to learn from the data. An example of undefitting can be considered the 3 input neurons approach showed in figure 4.2 with unmodified data. The training error oscillates and it shows that the model is not able to learn from the data.



(a) Training error vs validation error



(b) Training and validation error for three different sizes of the data-set

Figure 4.11: Overfitting analysis

The results of figure 4.11 display training and validation error for lift data for the same structure in two different ways. Structured used concern: two layers of 30 neurons, Adam optimizer, Leaky ReLU activation function ($\alpha = 0.1$), learning rate set to 1 and batch size equal to 30. As epoch number a large number as 100 000 was considered in order to point out the learning behavior until the trend was clear. Figure 4.11 involves three different size of the whole data-set (20%, 50% and 100%). Some considerations can be done. From figure4.11-b, looking at the validation error, the validation error

for the whole data-set is lower respect to the other sizes. Respectively the validation error from the whole data-set goes for 10^{-3}, for half data-set it is between $3 \cdot 10^{-3}$ and $4 \cdot 10^{-3}$ instead for the smallest size considered (20%) is around $7 \cdot 10^{-3}$. Increasing the data-set size most of the time helps to achieve a better results. This consideration can be extended to the drag data, similar trend has been found. In figure 4.11-a training and validation error are analyzed in the same plot. With respect to figure 4.11-b, 4.11-a helps to underline if overfitting is present. Here, one can considered that using a smaller data-set allows to the model to learn better since the training error 10^{-4} but two additional considerations have to be done. The first one is that the samples used for training and validation are random, this means that it is possible that the training-set contains an easier "function" to map (similar data are considered) and the validation-set has sparse data that results in a lower model's ability to generalize. The second one is the focus has to be on the validation error, it provides a measure of how well the model performs over unseen data. Increasing the da-set size reduce the gap between training and validation error. The model, also with the whole data-set, can be considered overfitted. The validation error stop to lay on the training error. The good point is that overfitting, as in figure4.10, can results in a further increase of the validation error. In the related work this doesn't happen. The validation error curve, once it has reached his minimum, remains flat. These considerations cannot be considered general. Random choice of the data-set is helpful to have a more general measure on the final accuracy but it's not enough. It is not related to the problem involved, or to the field in which machine learning and neural networks are applied. A common approach to have a general trend is called cross-validation. In the following paragraph cross validation is considered

4.3.5 Cross validation

Cross validation is a common practice performed in supervised learning to avoid overfitting. When an algorithm is set to learn in supervised learning a part of the data-set is hold out to test the final performance of the model, it is called test set. Before the learning process starts the subdivision of the whole data-set has to be performed. This drastically reduce the number of the samples and it can lead to an high dependency on the random data considered. The basic approach of cross validation is k-fold cross validation (k-fold CV, see as reference figure 4.12). The data-set is divided in k sets, one is held out for the final evaluation (test-set) and the remaining $k-1$ sets are used as training set. This procedure is repeated for each k set. Divisions are also called "folds". Therefore, each fold is used $k-1$ times for the training and one as testing set. The error estimation is the averaged over all the training performed (k-times). This helps to reduce bias in the training since each fold is used k-1 times and it helps to reduce variance in validation since each fold is used to perform the testing error.

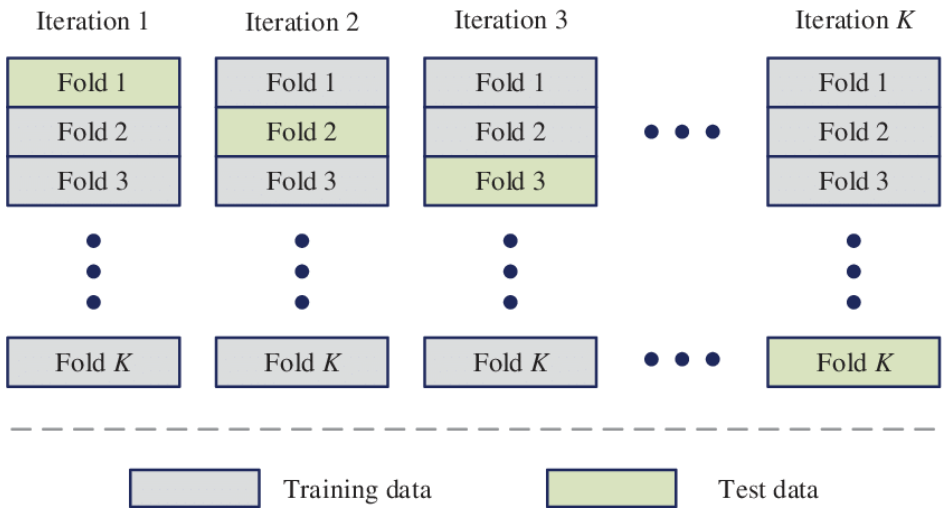


Figure 4.12: K-fold cross validation [46]

The general rule consider $k = 5$ or $k = 10$. In the related work $k = 5$ is considered. Moreover, the procedure is repeated ten times in order to increase the ability to generalize over the data-set.

Repeated K-fold cross validation

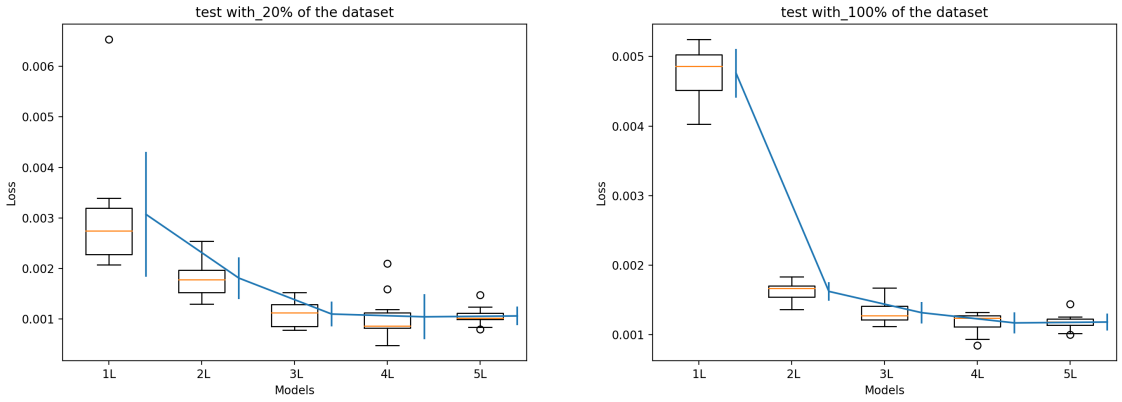
Cross validation is also useful to compare different models. For this reason is here consider not only to estimate the general performance of a particular model but to understand also, for the related case which model perform better. Here model is used to define the number of layer and neurons since the others hyper-parameters are set at the beginning of the analysis and never changed. When the overfitting was analyzed the structure mentioned in the previous section it was also done a prior analysis to understand if deeper models, in term of number of layer, exhibits overfitting around the same name of epochs. It was consider a values that could make sure the achievement of the lower limit for the validation error.

For this analysis the following hyper-parameters are considered:

- epoch number = 5000, this number was take into account after the aforementioned pre-analysis,
- the Adam optimizer,
- the Leaky ReLU activation function ($\alpha = 0.1$),
- 30 neurons (per each layer),
- learning rate = 1,
- batch size = 30,
- the mean squared error as loss function.

Structures with at the most with 5 layer have been under investigation. It has to be point, as previously mentioned, there in no a thumb rule for defining the hyper-parameter, and for each specific model a best solution can be find playing with the hyper-parameters. The listed hyper-parameters were took as granted after the result

shown above.
The results displayed from now on involve a k-fold cross validation with $k = 5$ repeated for 10 times. After the division in five folds, the 20% of the $k - 1$ folds used for the training is taken as validation set. This allows to have unvaried sizes with respect to the normal involved in a learning process (as mentioned at the beginning of the chapter). The k folds are different for each repeats. The values plotted are the mean values per each k-fold cross validation performed.



(a) Testing error over model depth with 20% of the data-set

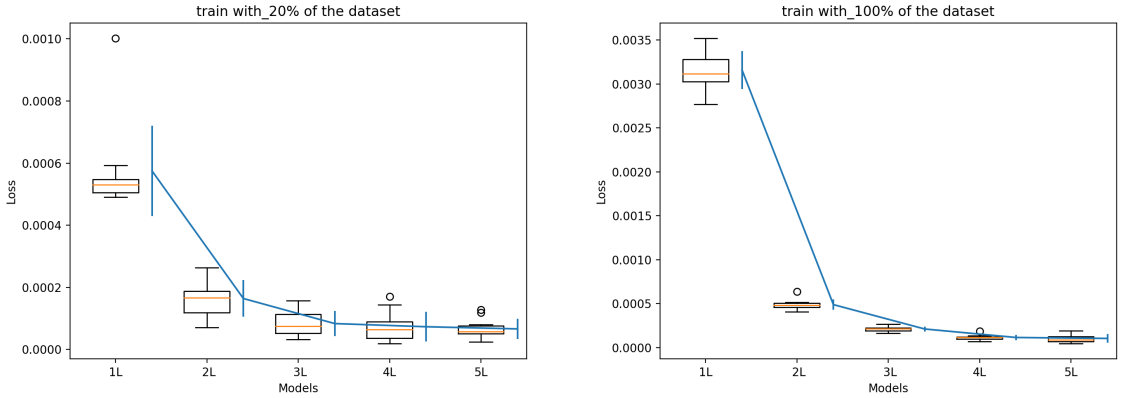
(b) Testing error over model depth with 100% of the data-set

Figure 4.13: Testing error box-plot and mean with standard deviation (blue lines) for lift prediction. On the x-axis depth of the model is labelled: 1L = one hidden layer, 2L = two hidden layers, 3L = three hidden layers, 4L = four hidden layers and 5L = five hidden layers. The box-plot shows the median with the orange line. The limit of the box are first and second quartile. The others values are displayed with whiskers with maximun 1.5 IQR, where IQR = third quartile (Q3) - first quartile (Q1). It is possible to have points that are drop out of this range, they are plotted as outliers.

With reference at figure (4.13) it is possible to deduce that for the prediction of the lift coefficient, using the hyper-parameter set and the input data handling, deeper data are better in generalization. Using the whole data-set shows a "massive" improvement from 1 hidden layer to two hidden layers. Not only in this case but in general models with 1 hidden layer are avoided, those structures are considered only for very simple relations to learn. Table 4.1 includes mean values related to figure 4.13. From table 4.1 one can see that a model with 5 layers is characterized by an higher mean test error, model with four layers generalize better.

	1 layer	**2 layer**	**3 layer**	**4 layer**	**5 layer**
20%-set	3.0698e-03	1.8100e-03	1.0983e-03	1.0439se-03	1.0612e-03
100%-set	4.7598e-03	1.6222e-03	1.31613e-03	1.1694e-03	1.1817e-03

Table 4.1: Mean lift testing error for different sizes of the data-set and for the five models considered.



(a) Training error over model depth with 20% of the data-set

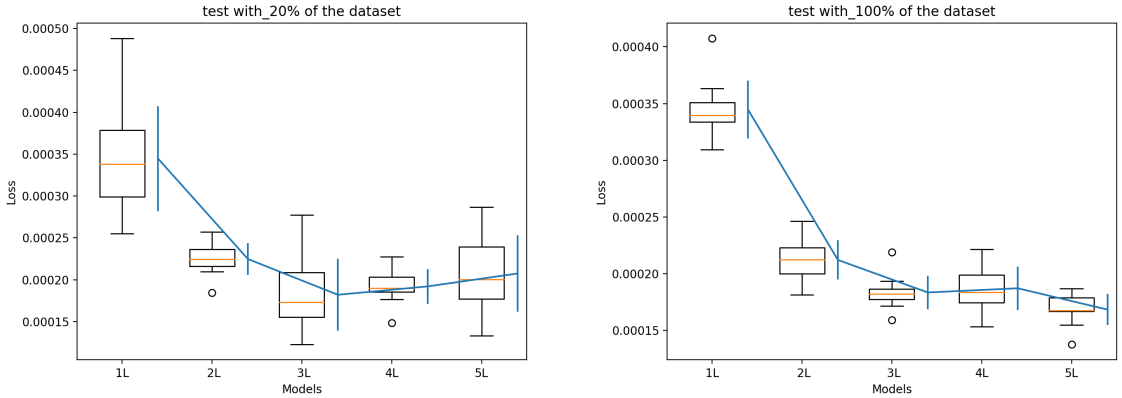
(b) Training error over model depth with 100% of the data-set

Figure 4.14: Lift training error box-plot and mean with standard deviation (blue lines)

	1 layer	**2 layer**	**3 layer**	**4 layer**	**5 layer**
20%-set	5.7476e-04	1.6407e-04	8.3439e-05	7.3270e-05	6.6370e-05
100%-set	3.1593e-03	4.8950e-04	2.1203e-04	1.1549e-04	1.0480e-04

Table 4.2: Mean lift training error for different sizes of the data-set and for the five models considered

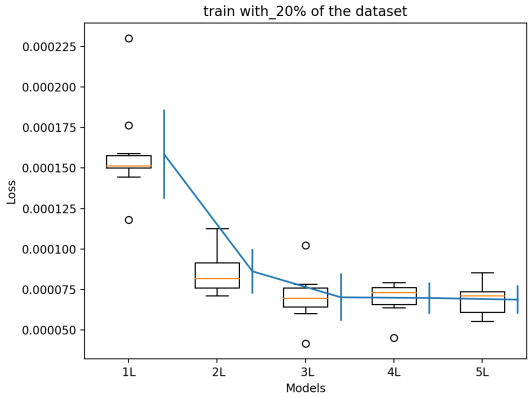
Comparing table 4.2 and figure 4.14 with table 4.1 and 4.13 one can notice that there is a difference of one order of magnitude between testing and training error. When a smaller data-set is used the difference increase, the model overfits easier. A good features of these models is that the validation error is low also if it overfits. When overfitting includes a high testing error it leads to a useless model. In this case, also if overfitting is present the testing error is low.



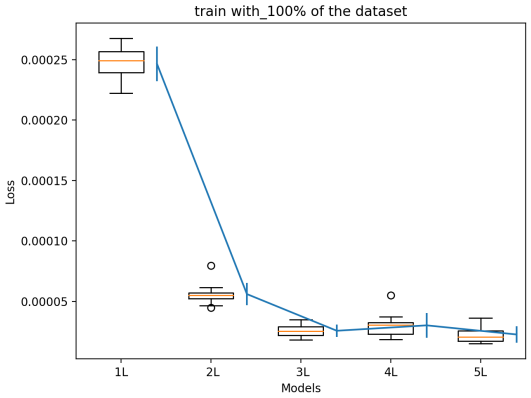
(a) Training error over model depth with 20% of the data-set

(b) Training error over model depth with 100% of the data-set

Figure 4.15: Testing error box plots and means/standard deviations for drag prediction



(a) Training error over model depth with 20% of the data-set



(b) Training error over model depth with 100% of the data-set

Figure 4.16: Training error box plots and means/standard deviations for drag prediction

	1 layer	**2 layer**	**3 layer**	**4 layer**	**5 layer**
20%-set	3.4455e-04	2.2465e-04	1.8198e-04	1.9200e-04	2.0740e-04
100%-set	3.4457e-04	2.1205e-04	1.8339e-04	1.8710e-04	1.6829e-04

Table 4.3: Mean drag testing error for different sizes of the data-set and for the five models considered

	1 layer	**2 layer**	**3 layer**	**4 layer**	**5 layer**
20%-set	1.5854e-04	8.6234e-05	7.0207e-05	6.9800e-05	6.8772e-05
100%-set	2.4642e-04	5.6072e-05	2.5733e-05	3.0208e-05	2.2658e-05

Table 4.4: Mean drag training error for different sizes of the data-set and for the five models considered

Figures 4.15 - 4.16 and Tables 4.3 4.4 show the drag case. The analysis performed is the one followed above. When drag prediction is involved the same models considered for lift prediction perform better. Tables 4.3 4.4 display the testing and training error respectively. Generally there is one order of magnitude of difference between the drag error and the lift error. These models are more suited for drag prediction. Using as references the figures 4.15 - 4.16 one can deduce a different behavior for drag prediction with respect to the depth of the models when the 20% of the data-set is used. Here increasing the model complexity, adding hidden layers to the model, leads to an example of overfitting that reduce the testing error and therefore the final performance of the model. Respectively, models with four and five hidden layers are excessively complicated for the data-involved. These models learn very well from the training data, it is possible to say too well since when unseen data are used the ability to generalize decrease (look at the table 4.3). One solution for overfitting is train the model with

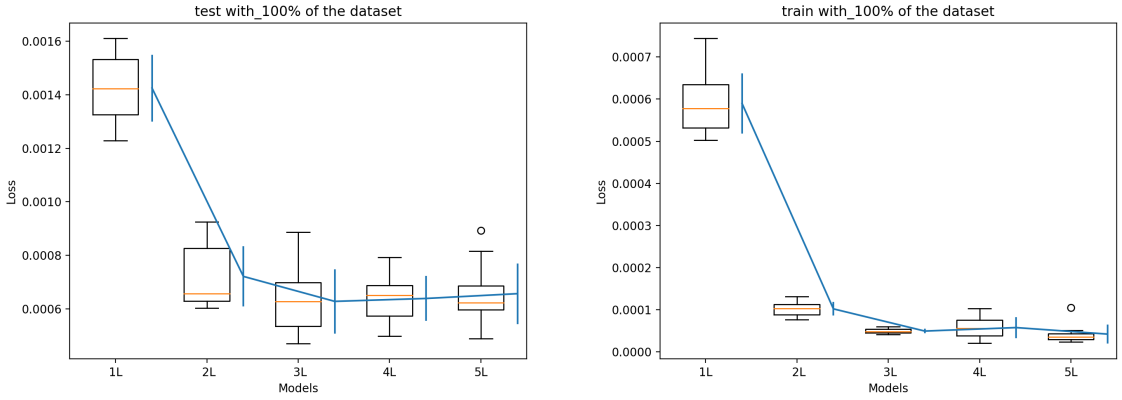
more data. From figure 4.15-b increasing the model complexity can reduce the mean testing error, also if the model with 4 layers shows a worst performance.

	1 layer	**2 layer**	**3 layer**	**4 layer**	**5 layer**
20%-set	6.2847e-05	1.8881e-05	4.2993e-05	2.0725e-05	4.5713e-05
100%-set	2.5495e-05	1.7299e-05	1.4700e-05	1.9213e-05	1.3631e-05

Table 4.5: Testing standard deviation for the five models considered

Not only model with three and five layers show lower mean values but they also show a lower standard deviation (see as reference table 4.5). The dispersion of testing error for these two models is lower, this means that the final testing errors tends to be close to the mean and since they have the lower mean it is possible to conclude that they perform better.

Until here, the results presented includes the aerodynamic stall. This introduce more complexity in the relation that the network tries to map in the learning process. In the following part data obtained with AoA in the range of -5^o and 10^o (with step= 1^o). Using the range of angles of attack allows to not take into account the aerodynamic of the stall for the most of the profile considered.



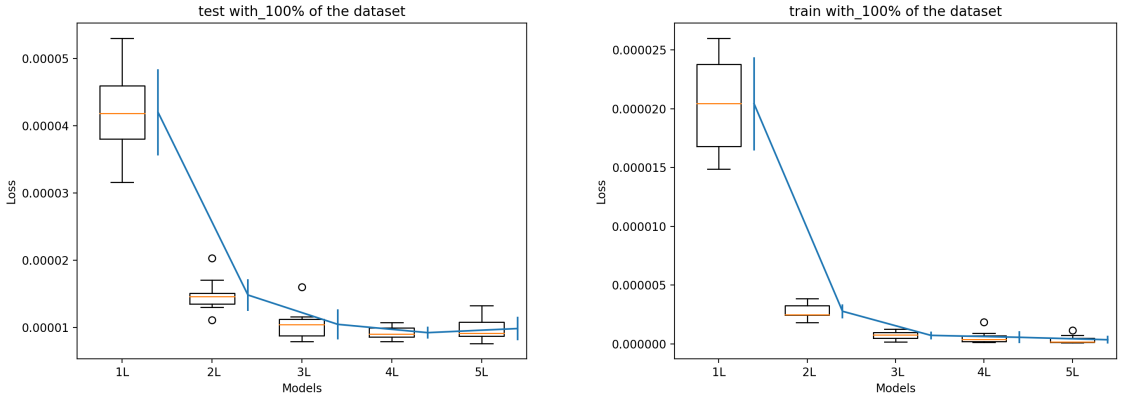
(a) Testing error over model depth

(b) Training error over model depth

Figure 4.17: Testing and training error box plots and means/standard deviations for lift prediction with 100% of the data-set and without the stall

	1 layer	**2 layer**	**3 layer**	**4 layer**	**5 layer**
Test error	1.4248e-03	7.2169e-04	6.2845e-04	6.3933e-04	6.5725e-04
Train error	5.8970e-04	1.0204e-04	4.9372e-05	5.7620e-05	4.2214e-05

Table 4.6: Mean testing and training errors for figure 4.17



(a) Testing error over model depth

(b) Training error over model depth

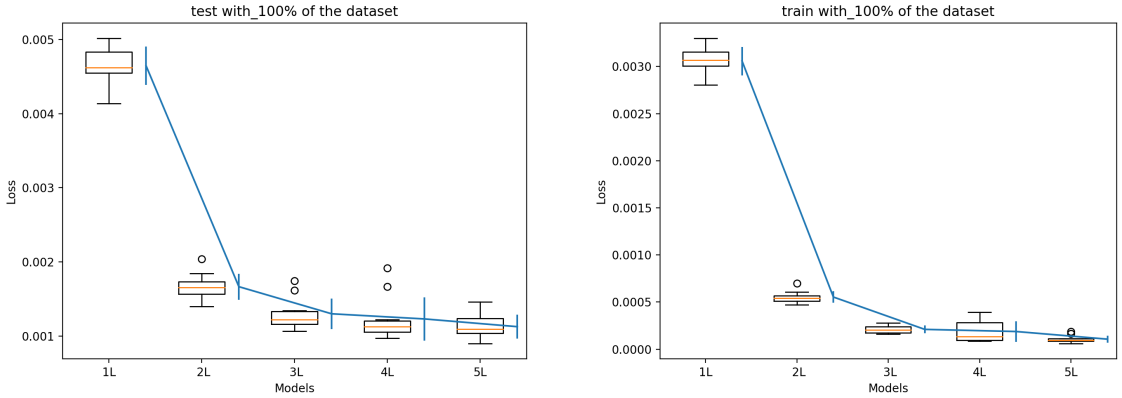
Figure 4.18: Testing and training error box plots and means/standard deviations for drag prediction with 100% of the data-set and without the stall

	1 layer	**2 layer**	**3 layer**	**4 layer**	**5 layer**
Test error	4.2005e-05	1.4851e-05	1.0489e-05	9.2552e-06	9.8667e-06
Train error	2.0414e-05	2.7836e-06	7.3525e-07	5.7009e-07	3.6375e-07

Table 4.7: Mean testing and training errors for figure 4.18

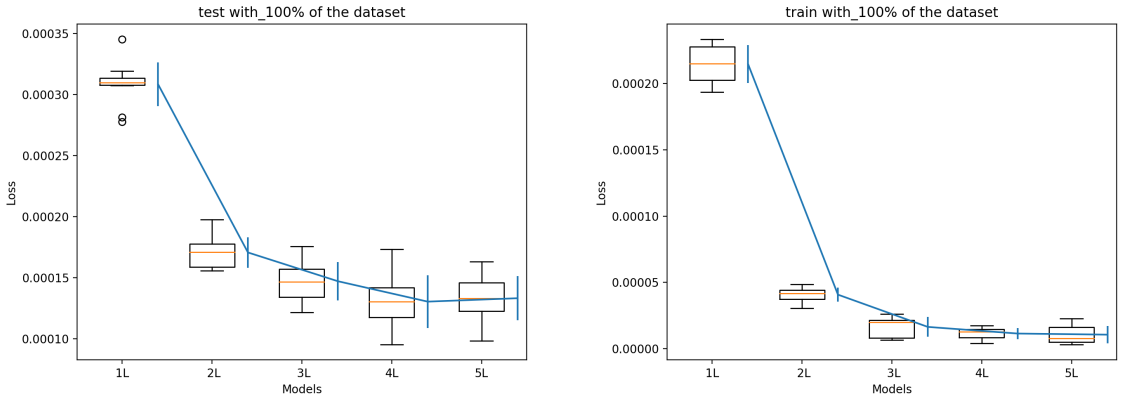
The first consideration that has to be made is the less number of samples involved respect to the previous analysis: here the samples are around 1300. Comparing figure 4.17-a with 4.13-b one can deduce that the lowest mean testing error is in the former one (see as reference 4.6 and 4.1). In addition, in figure 4.17 and in table 4.6 can be observed that a model with 3 layers, with the other choices already mentioned, has the best generalization capability. When aerodynamic stall was considered best results were achieved with the deepest network. The simplified problem and the lower number of samples involved considered can be considered the cause of the requirement of a simpler structure. Same considerations can be made for the drag model. Here without the aerodynamic stall the model with 4 layers performs better.

Considering time and the effort required to generate the data-set and without any further chances to increase it a different approach was considered. One input neurons was added and the input data introduced was the Reynolds number. Purpose of this analysis was to understand the behavior of the models already analyzed but adding an input. As aforementioned, when a different approach for the NACA 4-digit series was introduced an improvement in terms of training and validation was achieved. These results underlines the importance of how the data are handled and how they are feeded into the network.
The only modification is the number of input parameters, the other hyper-parameters respect the previous assumptions.



(a) Testing error over model depth (b) Training error over model depth

Figure 4.19: Testing and training error box plots and means/standard deviations for lift prediction with 100% including Reynolds number as input variable.



(a) Testing error over model depth (b) Training error over model depth

Figure 4.20: Testing and training error box plots and means/standard deviations for drag prediction with 100% including Reynolds number as input variable.

For lift case, figure 4.19-a and table 4.8 show that adding an input variables results in an improvement of the mean test error also with five hidden layers model. In figure 4.13-b and table 4.1 the best generalization capability was with four layers. Including the Reynolds number reduces the mean testing error and it allows to a deeper network to better generalization.

	1 layer	**2 layer**	**3 layer**	**4 layer**	**5 layer**
Test error	4.6465e-03	1.6658e-03	1.2999e-03	1.2307e-03	1.1263e-03
Train error	3.0580e-03	5.5309e-04	2.0977e-04	1.8708e-04	1.0608e-04

Table 4.8: Mean testing and training errors for figure 4.19

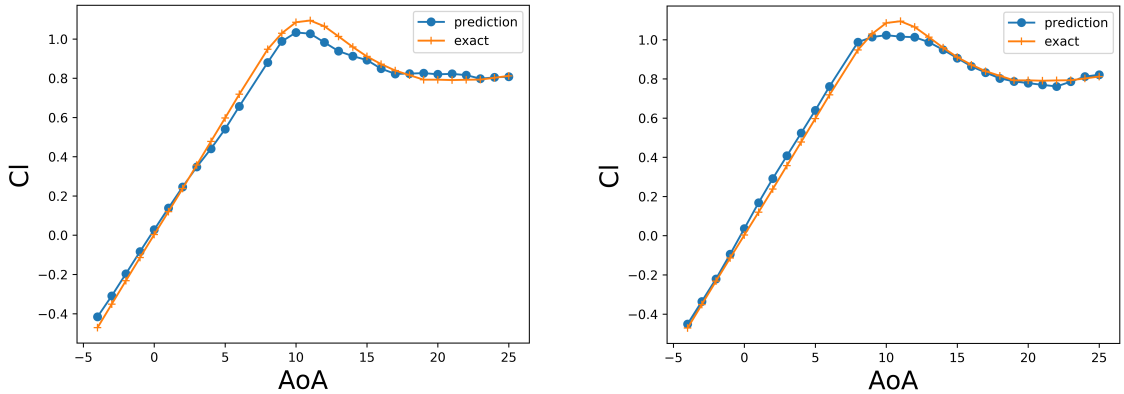
	1 layer	2 layer	3 layer	4 layer	5 layer
Test error	3.0836e-04	1.7074e-04	1.4722e-04	1.3052e-04	1.3330e-04
Train error	2.1466e-04	4.073e-05	1.6395e-05	1.1394e-05	1.0560e-05

Table 4.9: Mean testing and training errors for figure 4.20

When drag data are feeded into the network including the Reynolds number a different behavior can be noticed. Network with four layers (see as reference 4.20 and table 4.9) shows a smallest mean test value. Therefore, adding the Reynolds number for the drag case reduce the complexity of the problem. It can also be noticed that the improvement for the testing error is not related only to the "best architecture" but it is a general characteristic. The mean testing values in table 4.9 are lower than test errors displayed in table 4.1 with the whole data-set. In lift and drag prediction introducing the Reynolds number results in a lower training error. Looking at the difference between training e testing error it can be deduce that the last approach helps to reduce the overfitting.

4.4 Prediction examples

Using the knowledge extract from the previous analysis the lift and drag prediction of NACA 0021 are here briefly presented. The exact prediction was not the purpose of this study, due to this only one profile is presented (time and resources were limited), but it is interesting if the prediction on unseen profiles confirms the previous considerations. The coefficients involved are generated under on-flow condition at $Mach = 0.5$. This profile lays in the profile's domain used for training (see as reference 3.2) in terms the last two digit. The thickness digits, "21" is inside the range of table 3.2. This exhibition of results is presented only to underline once more the knowledge extract in the previous paragraph.



(a) 5 inputs approach. Reynolds number not considered

(b) 6 inputs approach. Reynolds number considered

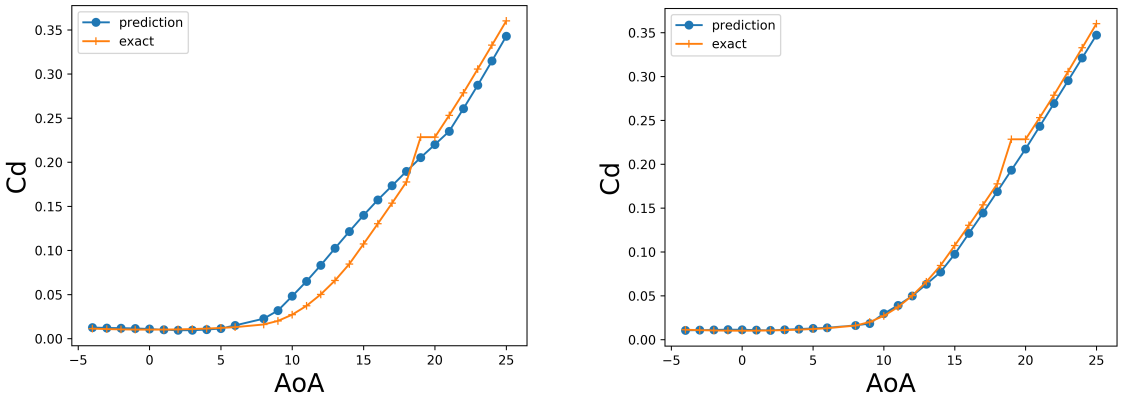
Figure 4.21: Lift coefficient for NACA 0021 at $Mach = 0.5$

The NN architecture and the hyper-parameters setting reflect section 4.3.5. When Reynolds number is not considered the structure with four hidden layers is employed

instead when Reynolds number is added as input variable the five hidden layers structure is applied (see as reference table 4.1 and table 4.8). In table 4.10 the mean square error of the prediction are presented. From figure 4.21, the Reynolds number helps to follow the aerodynamic stall zone: the error is lower (see 4.10) but also from the figure can be seen that the predicted curve lies better on the exact one.

5 inputs (without RE)	**6 inputs (with RE)**
1.6965e-03	1.1443e-03

Table 4.10: MSE for lift prediction of NACA 0021 airfoil



(a) 5 inputs approach. Reynolds number not considered

(b) 6 inputs approach. Reynolds number considered

Figure 4.22: Drag coefficient for NACA 0021 at $Mach = 0.5$

In figure 4.22 the drag coefficient at $AoA = 19^o$ is not well predicted with the CFD simulations. It could have been corrected but it was interesting to see the reaction of the NN on this outliers". The same procedure has been followed, the considerations made in section 4.3.5 have been applied. It means a structure of five hidden layers for drag prediction without Reynolds number and NN with 4 hidden layers for drag prediction when Reynolds is introduced as input variable (see as reference 4.3 and tab 4.9).

5 inputs (without RE)	**6 inputs (with RE)**
1.5282e-03	8.1878e-05

Table 4.11: MSE for drag prediction of NACA 0021 airfoil

In both cases the NN tries to interpolate the value at $AoA = 19^o$. As for lift case, including the Reynolds number allow a better prediction and for drag also high ability to overcome at outliers. It can be also mentioned that the outlier is present only in the prediction process and it is not involved in the training one. Using outliers in the training process can lead more easily to overfitting, the relation that the NN try to learn lose meaning and it can result in the learning of the relation input-output for each sample without the ability to generalize.

Conclusions

The purpose of the present study was to analyze and to investigate neural networks, a specific machine learning algorithm, in terms of applicability and constrains when the application field is an aerodynamic prediction, here lift and drag coefficients prediction under two different on-flow conditions and at several angles of attack. This work has two theoretical strengths: it involves theoretical background of CFD simulations and it provides a theoretical background of Neural Networks. The former one is not fundamental but it allows to have the basis to understand CFD simulations and in particular how the DLR-TAU code, the one used for the data-set generation, works. Artificial Neural Network is probably the most famous machine learning technique. Despite this, the theoretical understanding of Neural Network requires time and effort. Furthermore, in literature sometimes it is possible to find unclear explanations: overlapping of meanings, interchangeable nomenclature and also wrong examples. Chapter 2 tries to overcome these drawbacks. Thus, Chapter 2 and in general this work can be a "reference" when using the Neural Networks algorithm for the first time. Nonetheless, in order to have a deeper knowledge of NN and machine learning the bibliography related to Chapter 2 is highly recommended, in particular [21].
The application of Chapter 2 is carried on in Chapter 4 where NN algorithm is finally applied to the aerodynamic coefficients prediction. In the first part is underlined the relevance of handling in a smart way the input data. It's not only important in this specific study, testing different input data setting can help to overcome learning problems of the networks. In the related work the initial approach of only one input neuron for the NACA 4-digit series was abandoned for the three input neurons approach. The paragraph 4.3.2 points out a hyper-parameters analysis. The cross analysis of activation functions and optimizers shows that ReLu/leaky ReLu with Adam optimizer can be considered as a the best combination of activation function and optimizer. Initially, it can be observed that tanh and sigmoid activation function are not convenient in this particular case, but involving the right weights initialization the vanishing gradient problem can be solved. Adam optimizer allows a fast convergence and low oscillations. What concerns learning rate and batch size is that the extreme cases have drawbacks. Low values of batch size are defined noisy, these cases are characterized by oscillations. Additionally, when high values of learning rate are considered the oscillation are more important or the validation error has lower oscillations but it gets stuck "far" from the global minima. One can say that the low batch size values is dominant with respect the learning rate. A low value for the batch size doesn't allows to generalize well, it must be avoided. Instead using high value of batch size leads to a lower generalization and increasing the learning rate in the related work doesn't help to overcome this drawback. The number of epochs is set right when it allow to reach the minimum validation error and it doesn't drive the model to overfit. It has been explained that not always overfitting is harmful. In this study the model overfits but this doesn't drive to

a lower generalization ability. In this study there wasn't the possibility to increase the data-set, it is the first solution for overfitting. Over options are regularization, wich add a cost to the loss function for large weights, and dropout layers, which randomly remove certain connections. An intuitive solution is early stopping criteria, it stops the learning process when the validation error starts to increase. These approaches are not analyzed in these work, despite this it is worth to mention them.
Cross validation is a useful tool to analyze the generalization capability of a model, especially when a comparison with others model has to be performed. Using k-fold cross validation (in this case 5 folds) provides a general behavior of different model for the related study case. Involving the aerodynamic stall in the learning process requires deep model, the model with the best generalization capability are the ones with four or five hidden layers. When the stall is not taken into account the most remarkable alteration can be seen in lift prediction. Lift coefficient up to $AoA = 10^o$ is nearly linear and it results in a lower model complexity. For drag prediction switch from five to four hidden layers. Adding Reynolds number as input variable helps to reduce the mean testing error for both cases. For lift coefficient, five hidden layers lead to the best generalization. The "solution" improved with respect to the analysis without Reynolds. On the other hand, four hidden layers are enough to obtain a good solution for drag coefficient. There is not a theoretical explanation of these two different behaviors, the expectation was that the deepest model would have been the best choice when more inputs are provided. In literature, the hyper-parameters setting id often defined as "more an art than a science". This is the case. The over-all knowledge is that more complex structures performs better. One thing can be added about the model complexity, the more complicated is the relation to learn the more complex is the network required but there is no possibility to be sure in advance which one performs better. Usually the "one hidden layer networks" are avoided and also very deep networks are taken into consideration only in particular cases. For this reason the limit of five hidden layers has been chosen. The advantages that Reynolds number introduce in prediction are also clear for the final prediction of NACA 0021 .
An alternative approach could be use the coordinates of the profile as input samples and/or using one of the two coefficients as input data. It can be consider for further study additional samples with new on-flow conditions and also include the others NACA digit series.

Bibliography

[1] Deutsche Zentrum für Luft- und Raumfahrt (DLR)-Institute of Aerodynamics and FlowTechnology. "Technical documentation of the DLR [Tau]-code". In: (2018).

[2] Wilcox, David C, et al. *Turbulence modeling for CFD*. Vol. 2. DCW industries La Canada, CA, 1998.

[3] Tripathi and Siddhant. "Tollmien-Schlichting waves in planar and streaky airfoil boundary layers". PhD thesis. Aug. 2018.

[4] Ganesh Nampelly, S. Arunvinthan, and Nadaraja Pillai. "Effect of surface blowing on aerodynamic characteristics of tubercled straight wing". In: *Chinese Journal of Aeronautics* 32 (Feb. 2019). DOI: 10.1016/j.cja.2019.02.006.

[5] Peter Bradshaw and George P Huang. "The law of the wall in turbulent flow". In: *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* 451.1941 (1995), pp. 165–188.

[6] LEAP CFD TEAM. "Turbulence Part 3 - Selection of wall functions and Y+ to best capture the Turbulent Boundary Layer". In: *www.computationalfluiddynamics.com* (2013).

[7] Giancarlo Alfonsi. "Reynolds-averaged Navier-Stokes equations for turbulence modeling". In: *Applied Mechanics Reviews* 62.4 (2009).

[8] Philippe Spalart and Steven Allmaras. "A One-Equation Turbulence Model for Aerodynamic Flows". In: *AIAA* 439 (Jan. 1992). DOI: 10.2514/6.1992-439.

[9] Steven R Allmaras and Forrester T Johnson. "Modifications and clarifications for the implementation of the Spalart-Allmaras turbulence model". In: 2012.

[10] NASA Langely 2020. "Turbulence Modeling Resource". In: *Langley Research-Center (NASA) https://turbmodels.larc.nasa.gov/* (2020).

[11] Moukalled et al. *The finite volume method in computational fluid dynamics*. Vol. 113. Springer, 2016.

[12] H.R. Hiester et al. "Assessment of spurious mixing in adaptive mesh simulations of the two-dimensional lock-exchange". In: *Ocean Modelling* 73 (Jan. 2014), pp. 30–44. DOI: 10.1016/j.ocemod.2013.10.003.

[13] Alan M Turing. "Computing machinery and intelligence". In: *Parsing the Turing Test*. Springer, 2009, pp. 23–65.

[14] Michael Haenlein and Andreas Kaplan. "A brief history of artificial intelligence: On the past, present, and future of artificial intelligence". In: *California Management Review* 61.4 (2019), pp. 5–14.

[15] Mayank Mishra. "Computer Vision in Artificial Intelligence". In: *oracle.com/datascience/computer vision-in-artificial-intelligence* (2019).

[16] Thomas M Mitchell et al. *Machine learning*. 1997.

[17] OXFORDuniversityPress. In: *www.oxfordreference.com* (2020).

[18] Daniel D Gutierrez. *Machine learning and data science: an introduction to statistical learning methods with R*. Technics Publications, 2015.

[19] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[20] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[22] John FC Khaw, BS Lim, and Lennie EN Lim. "Optimal design of neural networks using the Taguchi method". In: *Neurocomputing* 7.3 (1995), pp. 225–245.

[23] Bekir Karlik and A Vehbi Olgac. "Performance analysis of various activation functions in generalized MLP architectures of neural networks". In: *International Journal of Artificial Intelligence and Expert Systems* 1.4 (2011), pp. 111–122.

[24] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. "Rectifier nonlinearities improve neural network acoustic models". In: *Proc. icml*. Vol. 30. 1. 2013, p. 3.

[25] Russell Reed and Robert J MarksII. *Neural smithing: supervised learning in feedforward artificial neural networks*. Mit Press, 1999.

[26] Augustin Cauchy. "Méthode générale pour la résolution des systemes d'équations simultanées". In: *Comp. Rend. Sci. Paris* 25.1847 (1847), pp. 536–538.

[27] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent". In: *Cited on* 14.8 (2012).

[28] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12 (July 2011), pp. 2121–2159.

[29] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[30] Deutsche Zentrum für Luft- und Raumfahrt (DLR)-Institute of Aerodynamics and FlowTechnology. "Technical documentation of the DLR [Tau]-code". In: (2018).

[31] Yasuhiro Wada and Meng-Sing Liou. "A flux splitting scheme with high-resolution and robustness for discontinuities". In: *32nd Aerospace Sciences Meeting and Exhibit*. 1994, p. 83.

[32] Eli Turkel. "Improving the accuracy of central difference schemes". In: *11th International Conference on Numerical Methods in Fluid Dynamics*. Springer. 1989, pp. 586–591.

[33] A Jameson. "Transonic Flow Calculations, MAE Report 1651". In: *Princeton University, Princeton* (1983).

[34] Richard P Dwight. "Efficiency improvements of RANS-based analysis and optimization using implicit and adjoint methods on unstructured grids". PhD thesis. University of Manchester, 2006.

[35] Steven L Brunton, Bernd R Noack, and Petros Koumoutsakos. "Machine learning for fluid mechanics". In: *Annual Review of Fluid Mechanics* 52 (2020), pp. 477–508.

[36] Saakaar Bhatnagar et al. "Prediction of aerodynamic flow fields using convolutional neural networks". In: *Computational Mechanics* 64.2 (2019), pp. 525–545.

[37] Nils Thuerey et al. "Deep Learning Methods for Reynolds-Averaged Navier–Stokes Simulations of Airfoil Flows". In: *AIAA Journal* (2019), pp. 1–12.

[38] Yao Zhang, Woong Je Sung, and Dimitri N Mavris. "Application of convolutional neural network to predict airfoil lift coefficient". In: *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*. 2018, p. 1903.

[39] Ricardo Wallach, Bento S de Mattos, and Roberto da Mota Girardi. "Aerodynamic Coefficient Prediction of a General Transport Aircraft Using Neural Network". In: ().

[40] Google. "TensorFlow". In: *https://www.tensorflow.org* (2020).

[41] François Chollet et al. "keras documentation". In: *https://keras.io/* (2020).

[42] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.

[43] K. He et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1026–1034.

[44] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[45] Nitish Shirish Keskar et al. "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima". In: *CoRR* abs/1609.04836 (2016). arXiv: 1609.04836. URL: http://arxiv.org/abs/1609.04836.

[46] Qiubing Ren, Mingchao Li, and Shuai Han. "Tectonic discrimination of olivine in basalt using data mining techniques based on major elements: a comparative study from multiple perspectives". In: *Big Earth Data* (Feb. 2019), pp. 1–18. DOI: 10.1080/20964471.2019.1572452.