UNIVERSIDADE
BEIRA INTERIOR

Predicting Transient Aerodynamics Using a Neural Network

(Versão final após defesa)

João Pedro Alves Fonseca Pereira

Dissertação para obtenção do Grau de Mestre em
Engenharia Aeronáutica
(Ciclo de estudos integrado)

Orientador: Prof. Doutor André Resende Rodrigues da Silva
Co-orientador: Doutor Emanuel António Rodrigues Camacho

Agosto, 2025.

This page was intentionally left blank.

Declaração Integridade

Eu, João Pedro Alves Fonseca Pereira, que abaixo assino, estudante com o número de inscrição a41476 de/o Mestrado Integrado em Engenharia Aeronáutica da Faculdade de Engenharia, declaro ter desenvolvido o presente trabalho e elaborado o presente texto em total consonância com o **Código de Integridades da Universidade da Beira Interior.**

Mais concretamente afirmo não ter incorrido em qualquer das variedades de Fraude Académica, e que aqui declaro conhecer, que em particular atendi à exigida referenciação de frases, extratos, imagens e outras formas de trabalho intelectual, e assumindo assim na íntegra as responsabilidades da autoria.

Universidade da Beira Interior, Covilhã 04/08/2025

Assinado por: João Pedro Alves Fonseca Pereira
Identificação: BI30399185
Data: 2025-08-06 às 09:54:06

This page was intentionally left blank.

Dedicatória

Aos meus avós e ”segundos-pais”, Joaquim da Fonseca e Filomena Carvalho Alves.

This page was intentionally left blank.

Agradecimentos

Diz-se que uma pessoa sozinha, com esforço e determinação, pode ir longe, mas a verdade é que com ajuda pode ir bastante mais. E se rodeado das pessoas certas, apenas o céu é o limite. Esta dissertação é um testemunho, não só da minha própria força de vontade e esforço, mas sobretudo das pessoas que me acompanharam nesta jornada. Agora, é o momento de vos agradecer.

Começando com uma pessoa muito especial, a minha Catarina, que melhor que ninguém sabe as noites de choro e tardes de frustração que foram precisas. Tu que partilhaste do meu suor e lágrimas. Tu que sempre me incentivaste e me disseste para ter calma, para não desistir. Se estou hoje a entregar esta tese e acabar este curso, em grande parte to devo a ti. Por tudo isso e muito mais, um grande obrigado meu amor. Outra pessoa muito importante, o meu irmão de outra mãe. Meu caro Miguel obrigado, por todo o apoio e incentivo. Obrigado pelos cafés e passeios à beira-rio. Obrigado pelos desabafos e pelas palavras de calma. Obrigado por saber que, mesmo longe, posso contar sempre contigo.

Um grande obrigado às pessoas que a Covilhã acrescentou à minha vida. Agradeço à minha afilhada, Leonor por todas as conversas e noites de jogos. Pela companhia, pelo alento e pela partilha, enquanto longe de casa estava. Agradeço também à Joana e ao Diogo pela vossa amizade sincera e companhia. Um obrigado ainda aos meus amigos Greg e Tjorven pelas parvoíces e jogadas, que me ajudaram a distrair. Aos mencionados até agora, o maior dos obrigados, pois se estes não foram dos piores e mais difíceis anos da minha vida, a vocês devo os sorrisos e as boas memórias que vou levar.

Queria ainda agradecer aos meus pais e avós pelo esforço descomunal que fizeram para que eu pudesse "ser alguém" um dia. Nada que eu possa fazer neste mundo pagará o vosso esforço. Da renda e propinas, à comida feita e roupa passada. Às visitas e aos passeios, aos almoços de domingo e à "notinha" para o café. Obrigado por tudo. Agradeço ainda ao meu padrinho e madrinha e aos meus primos pelos gestos e palavras.

Outro grande obrigado devo ao Emanuel que, incansavelmente, me ajudou a tornar este trabalho realidade. Tu que partilhaste da dificuldade em primeira mão e em tanto me auxiliaste, obrigado! Agradeço ainda ao Professor André Silva, por ter aceite e embarcado comigo neste desafio. Obrigado pela paciência e pela aprendizagem.

Finalmente, queria agradecer ainda às pessoas do CEiiA por esta oportunidade. Nomeadamente à Sara pelo voto de confiança que me deu. Agradeço também ao Paulo e aos meus colegas de Sistemas pelo ambiente fantástico que me faz querer ir trabalhar todos os dias. Agradeço ainda às pessoas que Évora tem colocado no meu caminho, especialmente ao Miguel e ao Luís, pelo vosso companheirismo com que me fizeram esquecer tanto a distância a casa, como a monumentalidade desta tarefa.

A todos vocês que me ajudaram a ir mais longe, um enorme obrigado!

This page was intentionally left blank.

Resumo

O movimento de batimento das asas de pássaros e insetos sempre cativou a mente humana. Desenvolvimentos recentes em veículos aéreos micro e extração de energia renovaram o interesse no estudo da aerodinâmica do batimento. Embora a modelação de aerodinâmica transiente tenha sempre trazido desafios, grandes avanços na área da inteligência artificial, nomeadamente em redes neuronais, prometem a capacidade para realizar computações mais rápidas e complexas. Em particular, redes neuronais recorrentes têm sido aplicadas para modelar aerodinâmica transiente e solucionar problemas dependentes do tempo. Esta dissertação procura a implementação, treino e validação de uma rede neuronal recorrente para a previsão de coeficientes aerodinâmicos de um perfil alar sujeito a batimento no tempo. A rede é composta por uma camada de atraso, seguida de camadas completamente ligadas e com uma camada de saída linear. A rede estima os valores de C_l e de C_m. Como entradas, a rede recebe os valores de Re, k, h, kh e A_α, assim como o histórico de α_{eff}, aos quais ainda se juntam as estimativas anteriores de C_l e de C_m. A rede foi treinada para um perfil NACA0012 sujeito a deslocamento vertical, cujos dados foram gerados através de um código de painéis, o HSPM. Dois treinos foram feitos para verificar o impacto a inicialização da camada de atraso. Os resultados mostram que a rede é capaz de prever a evolução dos coeficientes com boa precisão. Também foi visto que a inicialização dos atrasos tem um efeito muito reduzido no desempenho do modelo. Trabalho futuro focar-se-á em expandir a base de dados de treino para incluir variação de ângulo de ataque, assim como analisar a variação de mais parâmetros de entrada.

Palavras-chave

Redes neuronais, Perfis oscilantes, Inteligência artificial, Aerodinâmica transiente, Coeficientes Aerodinâmicos

This page was intentionally left blank.

Abstract

The flapping motion of birds and insects always captivated the human mind. Recent developments in micro air vehicles and energy extraction have brought up interest in the study of flapping aerodynamics. While modelling transient aerodynamics always poses some challenges, large advancements in machine learning, namely in artificial neural networks seam to promise faster and more complex computations. Recurrent neural networks have found plenty of applications in transient aerodynamics and time-dependent problems. This dissertation aims to implement a recurrent neural network for the time-wise prediction of aerodynamic coefficients on flapping airfoils. The network features a delay layer, followed by a series of fully connected layers and a linear output layer. The network estimates the instantaneous C_l and C_m. It takes as inputs the Re, k, h, kh and A_α, plus the time history of α_{eff}, together with the previous C_l and C_m. Training data for a plunging NACA0012 airfoil was generated using a panel code (HSPM). Two trainings were conducted to assess the impact of the delay initialization of the network. Results show that the network is capable of prediction most of the flapping cycles with good accuracy. It was also found that the delay initialization has little effect on the model's performance. Future work intends to expand the training data to include pitching and flapping motions and expand the parameter test set.

Keywords

Neural networks, Flapping airfoils, Artificial intelligence, Transient aerodynamics, Aerodynamic coefficients

This page was intentionally left blank.

Contents

List of Figures

This page was intentionally left blank.

List of Tables

This page was intentionally left blank.

List of Algorithms

This page was intentionally left blank.

Nomenclature

α_{eff}	Effective angle of attack	rad
α_{mean}	Mean angle of attack	rad
β	Momentum coefficient	$-$
η	Learning rate	$-$
λ	Leady ReLU slope	$-$
ϕ	Phase angle between plunging and pitching	rad
μ	Kinematic viscosity	$Pa \cdot s$
ρ	Fluid density	$kg \cdot m^{-3}$
σ	Activation function	$-$
a	Neuron activation	$-$
A'	Plunging amplitude	m
A_α	Pitching amplitude	rad
AR	Aspect ratio	$-$
b	Neuron bias	$-$
b_W	Wing span	m
B	Bias vector	$-$
c	Aerodynamic cord	m
C_l	Lift coefficient	$-$
C_d	Drag coefficient	$-$
C_m	Pitching moment coefficient	$-$
D	Drag force	N
e	Output deviation	$-$
f	Plunging frequency	Hz
h	Nondimensional amplitude	$-$
J	Cost fucntion	$-$
k	Reduced frequency	$-$
kh	Nondimensional velocity	$-$
L	Lift force	N
m	Batch size	$-$
M	Pitching moment	$N.m$
N	Time steps in a flapping cycle velocity	$-$
r	Target output	$-$
Re	Reynolds number	$-$
S	Wing area	m^2
St	Strouhal number	$-$
t	Time step	$-$
U_∞	Free-stream velocity	$m \cdot s^{-1}$
w	Connection weight	$-$

W	Weight matrix	$-$
W_L	Wake length	m
x	Neuron input	$-$
y	Neuron output	$-$
y_{pos}	Airfoil's vertical coordinate	m
$\dot{y_{\text{pos}}}$	Airfoil's vertical speed	$m \cdot s^{-1}$
z	Neuron state	$-$
Z^{-1}	Network delay	$-$

List of Acronyms

2D	Two-dimensional
3D	Three-dimensional
ADAM	Adptive Moment Estimation
AI	Artificial Intelligence
ANN	Artificial Neural Network
AoA	Angle of Attack
BPTT	Backpropagation Through Time
CEC	Constant-Error Carrousel
CFD	Computational Fluids Dynamics
CGRL	Conjugate Gradient Recurrent Learning
CNN	Convolutional Neural Network
DCNN	Deconvolutional Neural Network
DNS	Direct Numerical Simulation
ELU	Exponential Linear Unit
ESCNN	Element Spatial Convolution Neural Network
FNN	Feed-forward Neural Network
GRU	Gated Recurrent Unit
HSPM	Hess Smith Panel Method
LES	Large Eddy Simulation
LEV	Leading Edge Vortex
LOC	Loss Of Control
LSTM	Long Short-Term Memory
MAV	Micro Air Vehicle
MNN	Memory Neuron Network
MSE	Mean Squared Error
NACA	National Advisory Committee for Aeronautics
NAG	Nesterov Accelerated Gradient
NS	Navier-Stokes
PINN	Physics Informed Neural Network
PReLU	Parametric Rectified Linear Unit
QRNN	Quasi-Recurrent Neural Network
RANS	Reynolds-Averaged Navier-Stokes
ReLU	Rectified Linear Unit
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
ROM	Reduced Order Model
RTRL	Real Time Recurrent Learning
SGD	Stochastic Gradient Descent

SHF	Stochastic Hessian-free Optimization
TDNN	Time-Delay Neural Network
UAV	Unmanned Aerial Vehicle
UPM	Unsteady Panel Method
URANS	Unsteady Reynolds-Averaged Navier-Stokes

Chapter 1

Introduction

In this first chapter, the motivation behind the present work is given, followed by the proposed objectives. Lastly, an outline for the structure of the remaining document is presented.

1.1 Motivation

Unlike the scientific fields of human knowledge, nature has had millions of years to evolve and perfect it self to the world we live in. From the smallest inset, to birds, fish and the largest whales, flapping has become a predominant mechanism for airborne and aquatic locomotion. The biological flapping system is so well adapted that birds can control the direction of thrust and keep peak propulsive efficiency by just morphing their wings. Of course, as often happens, scientists and engineers would look into nature for inspiration. The idea of flying like a bird captivated the human mind for ages [1].

When developing the helicopter, flapping aerodynamics where found to play an important role in its forward flight. During this phase, helicopter blades are subject to dynamic delay of stall, a non-linear aerodynamic process that results in large variations in lift and pitching moment [2]. Nowadays, there has been a recent surge in interest for flapping systems, with several new applications. The development of micro air vehicles (MAVs) is one of the main contributors to this increase in flapping investigation. These air-borne vehicles can be as small as insects and deploy bio-inspired flapping wing systems for propulsion [3]. Other emerging applications of flapping dynamics include wind energy extraction [4] and fish-like propulsion [5].

However, unveiling the secrets behind the flapping mechanism would not be an easy task. By the hands of Knoller [6] and Betz [7], Katzmayr [8] and von Kárman and Burges[9], the first theories and models about flapping dynamics were developed in the first half of the XIX Century. These initial flapping models were very limited and often simplified the aerodynamic behaviour. For a long time, experimental methods based on wind and water tunnel were the common way to study flapping aerodynamics. With time, this extensive and time consuming experiments allowed for data bases of flapping aerodynamics to be built. This data would be used to produce semi-empirical models that could provided good, but very limited approximations of a flapping system [3].

With the introduction of the computer, a new prospect of fast fluid simulations was open.

Unsteady panel methods, like the one implemented by Teng [10] allowed for faster and more detailed computations of transient flows. Nonetheless, these methods still required some assumptions to be made regarding the flow, which hindered their capabilities [11]. Advancements in computer power eventually led to the appearance of Computational Fluid Dynamics (CFD). These numerical methods could solve increasingly complex approximations of the Navier-Stokes equations, like the RANS. However, CFD simulations can become computational expensive, specially as the complexity of the model increases. Even so, these simulations can have difficulties capturing very complex non-linear behaviours, like the ones experienced by a flapping airfoil [12].

From this, it is clear that there is still interest in studying and modelling flapping aerodynamics. Not only that, but even with advancements in CFD and reduced-order models, the need for fast and reliable aerodynamic prediction tools as not been fulfilled. This is specially true for the early design stages, when a large number of variables and configurations need to be analysed [13]. Reduced order models (ROM) have been developed to accelerate the aerodynamic study, but these lack generalization capabilities and show difficulties when dealing with non-linear aerodynamic processes [14]. Furthermore, with recent advancements in active morphing wing and real-time airfoil optimization applications, being able to quickly compute aerodynamic conditions is a must.

Luckily, these past decades have seen an incredible quick development of artificial intelligence (AI) models, which include neural networks. Once again, natural inspiration came as a aid, this time from the human nervous system itself, which inspired the creation of the artificial neuron by Frank Rosenblatt [15] in 1958. Despite being originally created for imagine recognition and language modelling, these tools have found to be useful in many other fields, and unsteady aerodynamics is no exception. The early 2000s saw some of the first attempts to use neural networks for aerodynamic prediction. Such were the works of Rajkumar et al. [16], Suresh et al. [17] which attempted to estimate aerodynamic coefficients.

With further improvements to computation power and machine learning models, the 2020s saw a surge in the application of neural networks to study aerodynamics. Moreover, these have seen increasingly use for aerodynamic coefficient prediction. Such studies include the works of Peng et al. [18], Zhang et al. [19], Balla et al. [13] and Moin et al. [14]. Recurrent neural networks (RNNs) have also been used to model time-dependent phenomena. These include the recent works of Mersha et al. [20], who tried to estimate the angle of attack of a fighter jet, and Altena et al. [21] who attempted to predict the loss of control of quadcopters.

The present dissertation follows their footsteps by attempting to implement a recurrent neural network for the time-wise prediction of aerodynamic coefficients of a flapping airfoil. In particular, this work focus on the estimation of the lift, C_l, and pitching moment, C_m, coefficients for an airfoil undergoing pure plunging motion. The RNN takes two kinds of inputs: direct and time-varying. The direct inputs include Re, k, h, kh and A_α. The

time-varying inputs enter the network via a delay layer and consist of a window of points of the effective angle of attack, α_{eff}, together with some of the aerodynamic coefficients estimated previously, fed by the feedback loop. The proposed network consists of a delay layer, followed by a series of fully connected layers with Leaky ReLU activation and a final linear output layer.

In order to train the RNN, a data set was generated using an implementation of the Hess Smith Panel Method (HSPM). The neural network was subject to an initial study where the best learning rate for the training was found. After this, two RNNs were trained to test the impacts of the delay initialization on the model's performance. One network was trained with the delay layer initialized as zero, while the other performed four extra time steps to initialize the delays. The trained models were then tested with conditions distinct from those used during training. Finally, the results from both trainings were analysed and the model's performance evaluated.

1.2 Objectives

The goals this work proposes to achieve are as follows:

1. Raise awareness to the importance of flapping and the challenges faced when modelling unsteady aerodynamics.
2. To develop a recurrent neural network framework that can be used for the prediction of aerodynamic coefficients of an airfoil subject to flapping motion.
3. Train and validate the recurrent neural network for an airfoil undergoing pure plunging.
4. Test a delay initialization scheme and evaluate its impact on the model's performance.

1.3 Outline

As a reader, one will find this dissertation organized into five main chapters, these being: Introduction, Literature Review, Methodology, Results and Discussion and lastly, Conclusions and Future Work.

The first chapter focused mostly on providing the motivation behind the work done, a summary of what was done and its objectives.

In the second chapter, an extensive review on published literature is provided. This literature serves both to expose the captivating and challenging history of modelling flapping aerodynamics, while also providing the groundwork on the application of neural networks in aerodynamics that inspired this work. As such, the chapter is split into two major sections, the first of which focus on the history of flapping aerodynamics, and the second on the development and use of neural networks.

Then, the third chapter explains the methodology behind this work. This chapter is divided into multiple sections that describe the recurrent neural network itself, the training and validation procedures and the data generation and test conditions. In here, the network's mathematical model is described, together with the RNN's typology. The training algorithm and validation procedures are also shown, as well as the chosen test conditions.

This is followed by the presentation of the results obtained in chapter four. The chapter starts with the results from the learning rate study. Then the findings from both trainings on the plunging airfoil are provided. The effects of the delay initialization are compared and the overall model's performance is evaluated. Plots of the objective functions, network outputs can be found, together with scores of MSE and R^2.

Finally, chapter five summarises the work performed. A brief overview on the history of flapping aerodynamics is made. This is followed by a short description of the work done and conclusions are drawn. To finalize, suggestions for improvements and further investigation are proposed, closing with future prospects on the use of neural networks as aerodynamic modelling tools.

Chapter 2

Literature Review

In the present chapter previous works regarding flapping aerodynamics and aerodynamic prediction using neural networks. The methodologies, assumptions and findings are presented and discussed. The chapter is split into two major sections, the first of which is about flapping aerodynamics and aerodynamic modelling. The second section discusses the challenge of applying artificial intelligence into aerodynamics, namely the use the neural networks.

2.1 Overview of Flapping Aerodynamics

As mentioned, this section will discuss the evolution of modelling of flapping and unsteady aerodynamics. From early experimental and data-driven methods, to numerical and simplified models, to the use of computational fluid dynamics. First, an overview of unsteady aerodynamics, flapping systems and their governing parameters is provided. After that, a review of works regarding modelling flapping aerodynamics from the context of biological and bio-inspired propulsion is made. Finally, unsteady aerodynamics will be presented from the perspective of rotorcraft and power extraction applications.

2.1.1 Governing Parameters in Flapping Aerodynamics

Oscillating foils are commonly used by birds, insects and aquatic animals such as fish and cetaceans for their locomotion. Their dynamics have also been exploited by humans to better understand natural flight and marine propulsion, as well as a mean for power extraction, using both simplified or zoological approach based flapping devices.

The oscillating motion of wings are comprised of plunge, pitching or a combination of both, typically denominated as flapping. Describing the dynamics of oscillating foils is not an easy task, mainly due to the large number of governing parameters that have been being used in the literature. Wu et al. [12] groups these parameters into four main categories: environmental, geometric, kinematic and performance parameters.

One of the central governing parameters is the Reynolds number, Re, which is used to categorize the flow regime and the importance of inertial and viscous effects, in other words, the convective and dissipation effects of momentum. The Reynolds number Re is

given as

$$Re = \frac{\rho U_\infty c}{\mu}, \tag{2.1}$$

where U_∞ is the flow velocity, c is the airfoil chord and ρ and μ are the fluid's density and dynamic viscosity, respectively.

When it comes to the motion parameters, we can start by the reduced frequency, which provides the ratio between the flapping frequency and the velocity of the incoming flow. The reduced frequency k is a nondimensional parameter widely used when studying unsteady aerodynamics and can be expressed as

$$k = \frac{2\pi f c}{U_\infty}, \tag{2.2}$$

where f is the motion frequency. This parameter is often accompanied by the nondimensional amplitude, h. It relates the airfoil's plunging amplitude with its chord length as

$$h = \frac{A'}{c}, \tag{2.3}$$

where A' is the plunging amplitude.

With both k and h defined, one can define the maximum nondimensional plunge velocity, kh, which provides the ratio between the oscillating airfoil and flow velocities. It is important to note that, based on a wide variety of publications, this parameter does not represent a flow condition, but rather a series of combinations of k and h.

Another parameter used, which presents a similar concept as the nondimensional velocity is the Strouhal number, St, representing the ratio between the inertial forces from the oscillations of the airfoil and those due to the convective acceleration of the flow field. It is written as

$$St = \frac{f W_L}{U_\infty}, \tag{2.4}$$

where W_L is the wake length, properly calculated based on the imposed kinematics. It is related to the nondimensional velocity as $kh = \pi St$.

Regarding kinematics, the flapping motion typically results from the combination of vertical (plunging) and rotational (pitching) oscillatory motions. Traditionally, in their most simple form, these motions are described as sinusoidal functions, given by

$$y_{\text{pos}} = A' \sin(2\pi f t), \tag{2.5}$$

and

$$\alpha = \alpha_{\text{mean}} + A_\alpha \sin(2\pi f t + \phi), \tag{2.6}$$

where $y_{\text{pos}}(t)$ and $\alpha(t)$ are the instantaneous vertical position of the airfoil and instantaneous angle of attack. α_{mean} represents the mean angle of attack and A_α is the pitching amplitude. Lastly, ϕ is the phase difference between the pitching and plunging motions. These motions can be seen more clearly in Figure 2.1.

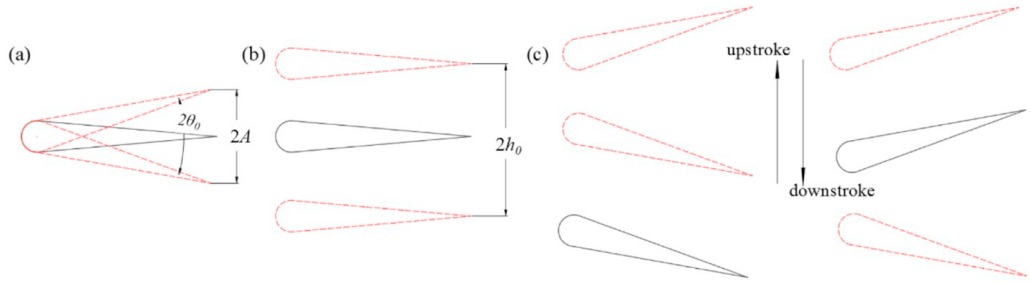

Figure 2.1: Different types of airfoil motions, from left to right: (a) pitching motion; (b) plunging (or heaving) motion and (c) flapping motion, from [22].

An additional parameter is the effective angle of attack identified by Knoller [6] and Betz [7]. They noticed how the relative motion between the approaching flow and the airfoil's vertical motion created an effective angle of attack, as illustrated in Figure 2.2.

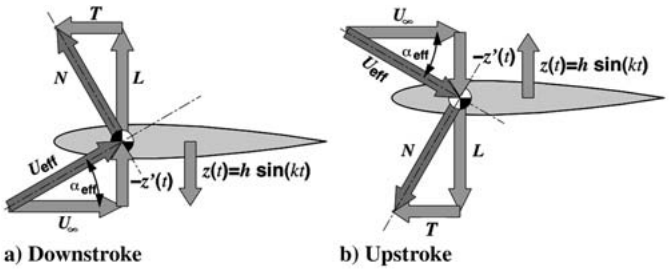

Figure 2.2: Creation of an effective angle of attack by a flapping airofil, from [3]

This instantaneous effective angle of attack α_{eff} can be calculated as

$$\alpha_{\text{eff}} = \arctan\left(\frac{-\dot{y_{\text{pos}}}}{U_\infty}\right) + \alpha. \tag{2.7}$$

In the case of a pure sinusoidal plunging motion, where $\dot{\alpha} = 0$, the effective angle of attack may be rewritten as

$$\alpha_{\text{eff}} = \arctan(-kh\sin(2\pi ft)) + \alpha_{\text{mean}}, \tag{2.8}$$

which has a maximum value of $\arctan(kh) + \alpha_{\text{mean}}$.

These are the central parameters used to control the kinematics, which work as inputs to the problem. Moreover, most of the aerodynamic properties of the airfoil can be predicted from its aerodynamic coefficients, making them of special interest to study. These include

the lift coefficient

$$C_L = \frac{L}{\frac{1}{2}\rho U_\infty^2 S}, \tag{2.9}$$

the drag coefficient

$$C_D = \frac{D}{\frac{1}{2}\rho U_\infty^2 S}, \tag{2.10}$$

and the pitching moment coefficient

$$C_M = \frac{M}{\frac{1}{2}\rho U_\infty^2 Sc}. \tag{2.11}$$

In the bi-dimensional case, the wing area, $S = b_W c$, is considered for a unitary wing span of $b_W = 1$, and the coefficients take the form of C_l, C_d and C_m.

Furthermore, it is important to mention that the formulas presented for propulsion efficiency stand for the condition of a flapping airfoil with combined pitching and plunging motions. Other cases, as is for the case of power extraction with forced plunging and/or pitching motions, present different formulas, as the ones presented by Kinsey and Dumas [22]. Other geometrical parameters include the maximum thickness of airfoils, the wingspan, b_W, and the aspect ratio, $AR = b_W^2/S$.

2.1.2 Flapping Aerodynamics in the Context of Propulsion

After presenting the parameters that govern the flapping aerodynamics, it is now time to see how natural locomotion has led to the investigation of flapping aerodynamics as a mean of propulsion and energy extraction. This section offers an overview from initial theories, to flat plate theory and potential flow models, ending in CFD simulations and experiments. Themes ranging from pure kinematic analysis, to the investigation of the complex wake dynamics, as well as the role of leading-edge vortices in performance enhancement, will be explored.

Knoller [6] and Betz [7], between 1909 and 1912, were the first to describe the thrust generation mechanism created by flapping airfoils. While investigating flapping wings, they saw an effective angle of attack being produced by the plunging motion due to the interaction between the airfoil's vertical and airflow velocities. This effective angle of attack changes with time that causing an oscillating aerodynamic force. This force, being normal to the relative velocity, can be decomposed into both lift and thrust components, as seen in Figure 2.2.

This effect, named the Knoller-Betz effect, was then experimentally verified by Katzmayr [8] in 1922, when he placed a stationary airfoil into a sinusoidal oscillating air flow and measured its average thrust. However, Knoller and Betz analysis focused on the kinematics of the airfoil, and did not take into account the vorticity mechanisms, which play a major role in the thrust and lift generation.

Such mechanisms were only explored in 1935, when von Kármán and Burgers [9] offer the first theoretical explanation of the lift and thrust production by flapping wings from the analysis of wake vortices.

They looked into the flow interactions around bluff bodies, flat plates and airfoils, at low Reynolds numbers. They noticed how moving bodies sometimes produce inverted von Kármán vortex streets, which create a surplus of momentum in the wake, compared to the upstream flow. Such wake is illustrated in Figure 2.3. This wake structure contrasts the typical von Kármán vortex street, which presents a momentum deficit in the wake, as seen in Figure 2.4. The presence of an inverted von Kármán street is associated with thrust and lift generation.

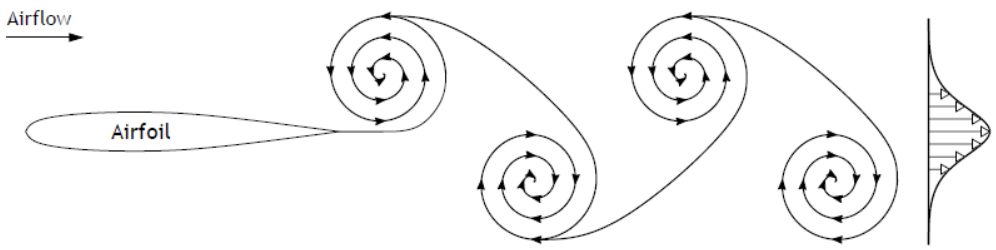

Figure 2.3: Inverted von Kármán wake vortex street with a momentum surplus, from [23].

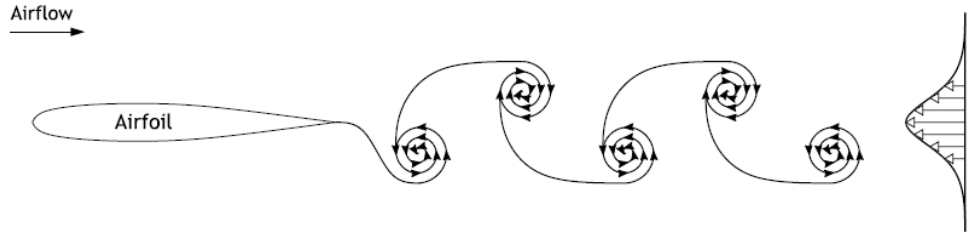

Figure 2.4: Typical von Kármán vortex street showing a momentum deficit, from [23].

Around this time, Garrick [24] applied Theodorsen's flat-plate theory to predict the thrust generation properties of pure plunging and pitching plates, as a function of the motion frequency. However, by using a potential flow analysis, the viscous effects were neglected and only small amplitude motions could be considered. Garrick noticed how, for those small amplitudes, the plates only started generating thrust at relatively high frequencies. His predictions would later be proven by Silverstein and Joyner [25] in 1939 and, again, by Bratt [26] in 1950.

Later, in 1987, Teng [10] published his thesis, where he develops a computer code for the numerical solution of inviscid, incompressible, unsteady flows over moving airfoils. His work is based on a extension of the already known Hess-Smith panel method, which was used for steady, potential flow problems. In order to achieve this, Teng includes the vortex shedding process into the wake. While ground-breaking, his code still requires the flow to be inviscid and thus, to remain to attached to the surface at all time. Still, his code exhibits good agreement with data obtained from other panel methods. Two important remarks about his results are the fact that the lift and moment coefficients have the same period as the one of the oscillation motion, while the drag coefficient shows the twice the frequency.

The following year, Freymuth [27] set to further investigate the vortex formations of pitching and plunging airfoils. Using flow visualization, Freymuth identified the vortical structures produced by the oscillating airfoils, such as the inverted von Kármán vortex street. He concluded that these wake structures were produced by pure pitching or plunging airfoils at low angles of attack, high reduced frequencies and small plunging amplitudes.

Then, Erickson [11] presented an extensive report on panel-method codes. The author defines panel methods as numerical schemes for solving the Prandtl-Glauert equation for linear, inviscid, irrotational flows, at subsonic or supersonic regimes. Three fundamental analytical solutions of the Prandtl-Gauert equation are know: the source, doublet and vorticity singularities. Panel methods work by superimposing surface distributions of these singularities over discrete elements of the aircraft surface, the panels. Finally, boundary conditions are imposed at the edges of the panels, called control points, in order to make the solution fit the desired geometry. These codes can be higher or lower-order, depending on the type of function used to describe the strength of the singularity distribution. Lower order were the first to be developed and use constant-strength distributions. While

these worked fine for incompressible subsonic flows, they tended to be unstable under supersonic conditions. Higher order codes were later developed, but these required greater analytic work to calculate their coefficients, so an interest remained on lower-order panel methods.

Additional limitations of panel methods are also described. The author refers to the Prandtl-Gauert equation as being the simplest fluid-flow equation that includes compressibility effects. It is derived from the Navier-Stokes (NS) equations by neglecting all the viscous and heat-transfer terms. It also assumes the flow to be irrotational and discards all nonlinear terms. As a consequence of these assumptions, these methods cannot compute aerodynamic phenomena such as flow separation, skin-friction drag and transonic shock. However, induced drag and wave drag are predicted. Another consequence of not considering explicit viscous effects is that the Kutta condition must be imposed, or the solution will not be unique. Panel codes achieve this by releasing wake panels from the trailing edge that propagate downstream, causing the flow to leave smoothly.

Some improvements have been made to panel codes, such as including the presence of the wing boundary layer into the code. Other improvements have been made into modeling flow separation for high swept wings. However, these often require iterative solving methods, which are more computational expensive. Furthermore, recent progresses have been performed into extending panel methods to transonic flows, by combining them with other numerical methods.

Some time later, in 1996, Tuncer and Platzer [28] investigated the thrust generation due to airfoil flapping. In their paper, the flow fields around both a single flapping airfoil and a configuration of a flapping and stationary airfoils in tandem were computed. For this, a combination of a NS solver with unsteady flow solutions and an iterative Navier-Stokes/potential flow solution were used. The nondimensional amplitude and reduced frequency were used to describe the flapping motions. With that, the authors were able to compute the unsteady flow fields over NACA 0012 airfoils with good accuracy.

And two years later, Jones et al. [29] performed both numerical and experimental investigations on the Knoller-Betz effect. They concluded that, for Strouhal numbers between 1 and 2, the experimental and numerical results are in agreement with each other. However, for St lower than 0.3, the viscous effects become prominent, causing a disagreement between both methods. Similarly, for St greater than 2, the experimental results showed higher asymmetry than the numerical ones.

Then, in the year 2000, Tuncer and Platzer [30] published another paper regarding the computational study of flapping airfoils. This time, the authors remark how advancements in computational fluid dynamics (CFD) have allowed for the numerical solution of the Reynolds-Averaged Navier-Stokes (RANS) equations. Their work focused on the computation of the unsteady, viscous flow fields over a NACA0012 airfoil, flapping at several

reduced frequencies and amplitudes. From the results, they concluded that the highest propulsive efficiency occurred when the flow remained fully attached to the airfoil's surface at all time. The authors also observed that, for the range of values tested, high frequency plunging motions lead to the formation of large leading edge vortices which would be released while the airfoil entered a dynamic stall loop. This resulted in higher thrust coefficients, at the expense of lower efficiency. They explained this fact by the creation of a low pressure, suction zone on the upper surface of the airfoil by the leading edge vortex.

Biomimetics was an emerging field at this time, where inspiration from the natural world was used to produce man-made devices capable of achieving the performance level of living organisms. Aquatic propulsion was one the areas under interest, where the study of the flapping mechanism of fish could be adapted for the propulsion of surface and underwater vehicles. Investigations in that regard are explored by Triantafyllou et al. [5] in their 2000 paper. Hydrodynamic, fish-like flapping is governed by the same parameters as airborne flapping. According to the authors, the interaction between oncoming vorticity and a fin is a basic problem in the development of vorticity control systems, which are important for the understanding of fish swim and manoeuvring. The authors mention that the reduced frequency reflects the flow unsteadiness better than the Strouhal number, while the latter is better reflects the wake flow dynamics. One of the challenges of aquatic flapping propulsion relies on the parametric window in which propulsive efficiency can be achieved.

Three years later, Lewin and Haj-Hariri [31] developed a numerical model for the thrust generation of flapping airfoils in viscous flows where a sinusoidal heaving motion was prescribed. From the results, it is noted how, for a given Strouhal number, that the maximum efficiency occurs at an intermediate reduced frequency, in contrast with ideal flow models. It is also noticed how that for low heaving frequencies, the separation of leading edge vortices actually results in lower thrust and efficiency. Finally, the authors observed that the power efficiency of the flapping airfoil is related to the similarity between the heaving frequency and the frequency of the most spatially unstable mode of the wake's velocity profile.

The following year, Young and Lai [32] further investigated the wake formation of sinusoidal plunging airfoils, together with the dependency of lift and thrust on the oscillation frequency and amplitude. From Garrick's linearised potential flow theory for a plunging flat plate, the authors show that, for large enough values of k, the thrust coefficient depends mostly on the heave velocity kh. Furthermore, it is noted how lines of constant average thrust coefficient, when plotted in the $k - h$ plane, tend to lines of constant kh. However, such lines of constant average thrust are only valid for small h. Additionally, the flat plate model would imply that, for values of k greater than 4, flows with the same kh, but distinct values of k and h would have the same vortex structure. Such fact is not true, as shown by the experimental boundaries between wake structure in the $k-h$ plane, as can be seen in Figure 2.5. Because of this, the authors imply that using just kh, is not

enough for the determination of the wake structure, contradicting previous findings.

Rather, Young and Lai [32] defend that both k and h must be regarded as distinct parameters. In fact, results from the unsteady panel method (UPM), based on the works of Basu and Hancock [33] and Jones et al. [29] corroborate such affirmations. The authors also conclude that the leading edge effects play a major role in the determination of the forces produced by the plunging airfoil. While trailing edge effects influence the wake's structure, these only carry a secondary contribution to lift and thrust. Finally, the computations also show that, for values of k larger than 4, both k and kh should be considered.

Also in 2004, Wu and Sun [34] studied the unsteady aerodynamic force generation from the flapping wings of a fruit fly. The flapping motion was described as a combination of both linear and rotational motions. The authors used CFD to numerically solve the NS equations, based on the procedures of Sun and Tang [35], from 2002. Moreover, the fruit fly model was the same as the one used by Dickinson et al. [36] in 1999. After investigating the influence of several parameters, the authors concluded that, for Reynolds numbers above 100, the mean lift and drag coefficients, only slightly depend on Re, agreeing with the results observed by Dickinson et al. However, for Reynolds numbers under 100, the mean lift coefficient drops, while the mean drag increases greatly. The authors explain this from the fact that, even though present, the leading edge vortices are weak and their vorticity diffused.

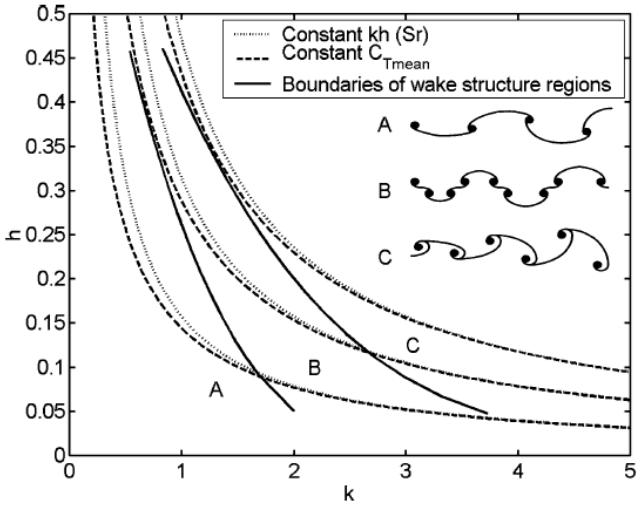

Figure 2.5: Comparison between constant average thrust coefficient with k and h, predicted by linearized potential theory, and experimental regions of differing wake structure, from [32].

Three years later, Shyy and Liu [37] studied the role of leading edge vortices in the lift generation of flapping wings. According to the authors, the leading edge vortices might be the result from the balance between the pressure gradient, the centrifugal force and the Coriolis force in the momentum equation. These vortices create a large low pressure region on the upper surface, increasing lift. In flapping wings, the stall is delayed if the

vortex stability is kept by the spanwise axial flow through the vortex core. Their paper mostly focus on the computation of flow fields for biological fliers, which are subject to lower Reynolds numbers and higher angle of attack than helicopter rotor blades. More specifically, the hawk-moth and the fruit fly were studied, with Reynolds number of 6000 and 134, respectively.

In the following year, Kurtulus et al. [38] further investigated the aerodynamics of flapping motion during hovering. Their work was mostly motivated by recent interests in MAVs and zoological wing configurations. They describe the flapping motion as a combination of a translation (the sidestroke), followed by a rotation and then a translation in the opposite direction. A Direct Numerical Simulation (DNS) approach was used to compute the unsteady, 2D, laminar flow fields around the NACA0012 airfoil. The authors remark how, even though plenty of research has been done with numerical simulations, it is still very difficult to solve the NS equations for the unsteady flows of insect wings. An experimental study was also carried out to compare and validate the numerical model. In both works, the authors were able to distinguish between three types of vortices: the leading edge vortices, which form around the leading edge of the airfoil, the translational vortices, which appear at the trailing edge of the airfoil during the translation phase, and the rotational stopping vortex, also formed at the trailing edge, but during the rotation phase.

Also in 2008, Platzer et al. [3] make a comprehensive compilation of the many advancements made in the field of flapping wing aerodynamics, as well as some of the challenges to be solved. In the introduction, the authors mention the importance of providing designers with prediction tools and aerodynamic knowledge about flapping mechanisms. They then proceed to elaborate about the developments in the understanding of the physics behind flapping airfoils, many of which have already been discussed. The authors also summarize the evolution from linear flat plate approaches to panel methods, based on potential flow, which became common with the development of better computers. Further advancements in computer technology have made it possible to include viscous analysis into the NS equations, in order to solve fluid dynamics, using the RANS coupled with turbulence models. All together, these form the three major methods for the aerodynamic study of flapping airfoils.

Platzer et al. [3] then proceed to discuss many findings and studies that have been performed around flapping aerodynamics. From such studies it can be noted how the thrust coefficient tends to increase with increasing reduced frequency, heave velocity and Strouhal number. The propulsive efficiency, on one hand, tends to decrease with greater reduced frequency. On the other hand, it achieves a maximum peak and then decreases with both heave velocity and Strouhal number. Regarding the phase angle between the plunging and pitching motions, it is possible to find a value where the propulsive efficiency, of a given motion, hits a maximum.

The authors also conclude that an inviscid analysis can be used to estimate the lift force of the airfoil, as long as the Reynolds number is large enough and the angle of attack is kept small. This is possible because, at high Reynolds numbers, the thin boundary layer has little effect on the force normal to the free stream direction. Even the wing flutter can be analysed with inviscid theory, since the normal forces are the main contribute to flutter. Moreover, inviscid flow analysis can be used to predict thrust and propulsive efficiency, as well. However, in steady-state aerodynamics, the stream force, which mainly comprises the drag, can only be calculated with viscous flow analysis [3].

Finally, the importance of being able to find the optimal conditions for lift, thrust and propulsive efficiency of flapping wigs is noted. It is mentioned how the aerodynamics of flapping wings is influenced by several parameters, which the authors hope advancements in optimization methods might aid to compute. Lastly, it is made clear that many problems remain to be tackled, specially regarding the computation and experimental study of complex, unsteady, viscous and separated flows on flapping wings [3].

Even though flapping mechanisms are usually associated with propulsion, they can also be used for power extraction. Since previous studies focused mostly on pure sinusoidal flapping, Lu et al. [39] explored the effects of non-sinusoidal modes of pitching and plunging on power extraction efficiency. A parameter was introduced into the motion equations that reflects how fast the airfoil reverses the direction of motion. This parameter allows for the modulation of the pitching and plunging motions between a sine and square wave. From studying several configurations of motions, the authors concluded that the power output of a flapping airfoil can be increased with an appropriate combination of non-sinusoidal plunging and pitching motions.

Also in 2014, Young et al. [4] presented an extensive review of many findings and challenges facing power extraction from flapping airfoils. The authors present flapping foils as an alternative to conventional rotary turbine designs. Flapping foil based-turbines would allow for lower foil velocities, and thus less noise and wildlife disturbance, as well as being able to operate small-scale devices on slower flows. Furthermore, some studies have suggested that flapping airfoils may not be bonded by the theoretical maximum Lanchester-Betz-Joukowsky limit. Finally, the authors remark the importance of understand the dependency on several parameters, including geometric and kinematic parameters. It is noted the field of flapping for energy extraction has benefited from recent developments in numerical code simulations and surrogate model optimization schemes.

The following year, Arora et al. [40] investigated the effects of flexibility in thrust generation of a plunging airfoil. In order to do so, the plunging surface was subdivided into discrete rigid components, linked by a torsion spring, to simulate flexibility. Two numerical solvers had to be run simultaneously, one to compute the wing forces, and another for the deformation. It was concluded that the introduction of flexibility allowed for a reduction of power consumption for forward flight. The propulsive efficiency was also improved when compared to a rigid wing. Lastly, the authors remark how there are still many gaps

in fully understanding the dynamics of a flapping wings, notably when regarding flexible flapping wings in propulsion, which could be beneficial for MAVs.

Some time later, in 2019, Zhu et al. [41] investigated the use of flapping wings for power extraction. According to the authors, the use of flapping wings for power extraction has been inspired by fish and birds and carries many benefits. Despite it being a relatively new field, several studies have suggested up to 7% increased in power extraction efficiency, when compared with traditional fixed-wing technology. The URANS model was used to numerically simulate the undergoing flapping motion of an airfoil with controlled deformation of its section. One of the main challenges relies in simulation the interaction between flow and wing, and in turn, the wing's motion impact on that same flow. Their work points to an improvement in power extraction obtained from the active airfoil deformation., which the authors justify from the increase in effective camber, which in return, increases the velocity circulation around the airfoil.

At the same time, Gursul and Cleaver [42] presented an analysis of previous studies on the flow structures and forces produced by plunging airfoils at low Reynolds numbers. The authors start by mentioning that unsteady, separated, vortex flows are commonly seen in natural swimmers and fliers, and even in MAVs. It is also pointed that mean thrust cannot be well predicted by inviscid methods. However, viscous predictions, even if based on laminar flow assumptions, provide good results. Still regarding thrust, this seams to mostly depend on Strouhal number, for a given amplitude. It was also seen that, for large-amplitude motions, LEVs also have a major contribute into the generation of thrust. Even though Strouhal number alone does not provide information about leading-edge separation, at large St, severe destruction of the LEV occurs from its interaction with the wing. Furthermore, the authors mention that this phenomenon appears to be linked with the change from a drag force to thrust. Mean lift can also be manipulated by controlling the tip vortex. Lastly, it is noted how the three-dimensionality of the LEV only becomes significant at both high angles of attack and Strouhal numbers.

In the following year, Bao et al. [43] investigated the effect of an alula in the aerodynamics of a flapping airfoil. An alula consists of a high-lift device that is common in birds and improves flight performance at height angles of attack. Understanding its effects is important for the development of bionic micro air vehicles. The flows around flapping NACA0012 airfoils with and without an alula where simulated with the URANS equations, coupled with the shear stress transport $k - \omega$ turbulence model, which is indicated for low Reynolds. The authors concluded that, in the pre-stall regime, the alula can provide a slight increase in lift. However, in certain configurations, the alula deflection can harm the aerodynamic performance. Once in the post-stall regime, the deflection of the alula does indeed delay flow separation during the full flapping cycle, whilst providing an increase in lift force.

Also in 2020, Wu et al. [12] provided their own comprehensive review on the fluid dynamics of flapping airfoils. The authors find there to be a massive spike in publishing regarding flapping aerodynamics since 2005. The vast majority of which are related to bi-dimensional flapping airfoils and focus on the flow characterization and performance studies. The major research gaps found include the investigation of three-dimensional effects in the flow and simulations based on real environmental conditions. The comparison between conventional rotary wind turbines and flapping foil systems was found to be also needing further research. Furthermore, the investigation of multiple flapping airfoil configurations instead of single-foil ones, as well as the study of non-sinusoidal motions and structure flexibility were also found to be lacking. Lastly, the authors conclude by alluding to the recent application of artificial intelligence into the field of aerodynamics and how deep machine learning could be used in the future to predict the forces and wake structures of flapping airfoils.

Then, in 2022, Wu et al. [44] set to investigate the influence of surging motion on the aerodynamic performance of flapping airfoils. In a pursuit for natural, bio-inspired flight and development of MAVs, further combinations of motion need to be considered besides the more traditional pitching and plunging. The author defines surging as a horizontal translation of the airfoil, which can be observed in turtles, pigeons and barn owls. In their work, the unsymmetrical E377 airfoil is studied under combined pitching, plunging and surging. The PMNS2D solver was used to compute the URANS equations with a finite volume method. The LU-SGS scheme was adopted in order to accelerate convergence and the SA turbulence model was used. Their results showed that the including of surging motion can increase lift up to $66\,\%$.

Also in 2023, Camacho et al. [45] investigated the effects of airfoil deformation on the propulsive efficiency of a flapping wing. According to the authors, flapping has been mostly studied under pure pitching and plunging motions. Moreover, a combination of pitching and plunging can often produce an improvement in propulsive efficiency. Another method to increase propulsive efficiency of flapping airfoils is to modify the shape of its leading edge. Some works mentioned by the authors refer to the use of dynamically deformed leading-edges or variable droop leading-edges.

The authors focus on the effects of a pitching leading edge on the propulsive efficiency of a flapping airfoil at low Reynolds number. For this, a modified NACA0012 with a pivoting leading edge was tested. The flow around the new airfoil was also investigated with CFD and the system dynamics were characterized by a set of dimensionless parameters, such as the Reynolds number, reduced frequency, nondimensional amplitude and nondimensional velocity. Their results show that the pitching leading-edge can increase the propulsive efficiency for flows with kh greater than 0.5, explained by the exploitation of the leading-edge vortices. However, for flows near the drag-producing regime, the pitching leading-edge showed to add little benefit.

In fact this is one of long-standing series of works that have been conducted at Aeronautics Research Center (UBI), with later collaboration of LabDin (EESC-USP), regarding the transient aerodynamics of plunging airfoils. For further reading includes the works of Lopes et al. [46] and Rodrigues et al. [47], Torres et al. [48] and Joana et al. [49] where they tackle both numerical and experimental studies of plunging airfoils. This is followed by the investigations of Camacho et al [50, 51], on the effects of various governing parameters, such as Re, AoA, frequency and oscillation amplitude, on the aerodynamics of plunging airfoils. Such studies led the path to the search of real-time flapping wing optimization, by active morphing of its leading-edge.

A comprehensive series of works were done into the effects of a deflectable leading edge on a plunging airfoil and its behaviour. These include further investigations by Camacho et al. [52, 53, 54, 55], which led to the study and creation of movable leading-edge prototype wing for wind tunnel testing. Trials and experiments with the prototype wing can be found in Camacho et al.[56, 57], the later of which attempts to use the control of the leading edge to mitigate the effects of dynamic stall. Additional works have been developed regarding the use of morphing trailing-edges by Sullivan et al. [58] and Pinho et al. [59] and even the recent analysis of a tail-like robotic fish propulsion system by Maria et al. [60].

2.1.3 Flapping Aerodynamics in the Context of Rotors

Previously, the way biological flapping inspired many applications of airborne and aquatic propulsion has been discussed. However, those are not the only applications of flapping systems, as it is a natural occurrence in helicopter rotors and wind turbine blades. In this section, the way unsteady aerodynamics form in these conditions and how they have been mode lled is presented. From early experimental studies to numerical and statistical approaches, modelling unsteady aerodynamics has presented a challenge. The complex and nonlinear nature of the flapping system can produce behaviour that even computational fluid dynamics struggles to recreate.

McCroskey et al. [61], in 1981, set to investigate the phenomenon of dynamic stall on pitching airfoils. The authors define dynamic stall as the occurrence of vortex shedding and convection on the upper surface of the airfoil, which causes a highly fluctuating pressure field. This in turn is responsible for hysteresis loops of the aerodynamic forces and negative aerodynamic damping. Dynamic stall can occur naturally on a rotor's retreating blade and is a common limitation for the high-speed performance of helicopters. However, it is noted how most blade designs fail to consider the effects from dynamic stall. Several airfoils were tested. As expected, their results show that a higher maximum lift is achieved during dynamic stall. Nevertheless, the associated vibration loads make the use of such a regime impractical. It was also shown that airfoils with good static stall performance also tend to have better dynamic stall behaviour, but such criteria is not the most reliably, since it is largely affected by airfoil shape.

Then, in 1987, Costes and Jones [62] approached the challenge of computing the transonic flow of the advancing blade, as well as the stall on the retreating blade of a rotor. Furthermore, it was their goal to perform such computations in a relatively short amount of time, as required for design tasks. On their work, two methods are tested, both relying on the solution of the potential equation, which is used under the assumption that the flow over the advancing blade has a Mach number low enough for it to be considered isentropic and irrotational. Although fairly good, the results appear to underestimate the experimental data, being of the possible causes attributed to insufficient refinement of the computational grid.

Some years later, in 1991, Baron and Boffadossi [63] aim to compute the instantaneous wake and aerodynamic rotor load using an unsteady vortex lattice scheme. On their work, the non-linearity problem that tip vortices introduce in rotorcraft design is addressed with the requirement for time-dependent methods. The authors remark the tremendous task that would be to fully compute the time dependent fluid dynamics directly from the Navier-Stokes equations. Moreover, they note the difficulties in choosing the discretization of the blade geometry, as well as in choosing the integration time step, as a compromise between model accuracy and required computing time.

Then in 1998, Ekaterinaris and Platzer [64] presented an comprehensive review on modelling and simulation of dynamic stall. The authors define dynamic stall as an unsteady flow separation that happens on aerodynamic bodies and can be caused by an unsteady motion or unsteady flow. Its prediction important for wing, propeller, turbomachinery, rotorcraft and wind turbine applications. The onset of dynamic stall results more severe aerodynamic loads than during static stall, which can lead to increased structural stress and control loads. Dynamic stall occurs with the development of a leading edge vortex as the airfoil exceeds the static stall angle of attack. As the angle of attack increases, the vortex traverses downstream along the airfoil's top surface, causing a suction that increases lift. However, once the vortex goes past the trailing edge, both lift and pitching moment decrease abruptly, while drag spikes, as the flow fully separates. The flow then remains separated until the airfoil returns to a low enough angle of attack, which allows it to reattach. This behaviour results in the characteristic hysteresis of aerodynamic loads, such as illustrated in Figures 2.6 and 2.7, for the light and deep stall cases, respectively.

As can be seen, numerical methods, while reasonable, prove to be quite unstable during the down-stroke, stalled phase. The authors remark how most of the existing prediction tools have to rely on empirical or semi-empirical methods. Other computational methods have been deployed, including boundary-layer methods and viscous-inviscid methods. Only recently has the computing of the RANS equations became feasible, despite it being very computational expensive and slow, as wells as dependent on the choice of adequate turbulence and transition models. Despite the many advances made into the prediction and study of dynamic stall, five major challenges remain to be addressed. These are the modelling of compressibility effects, flow reattachment, transitional flow, transonic effect

and three-dimensional flow effects [64].

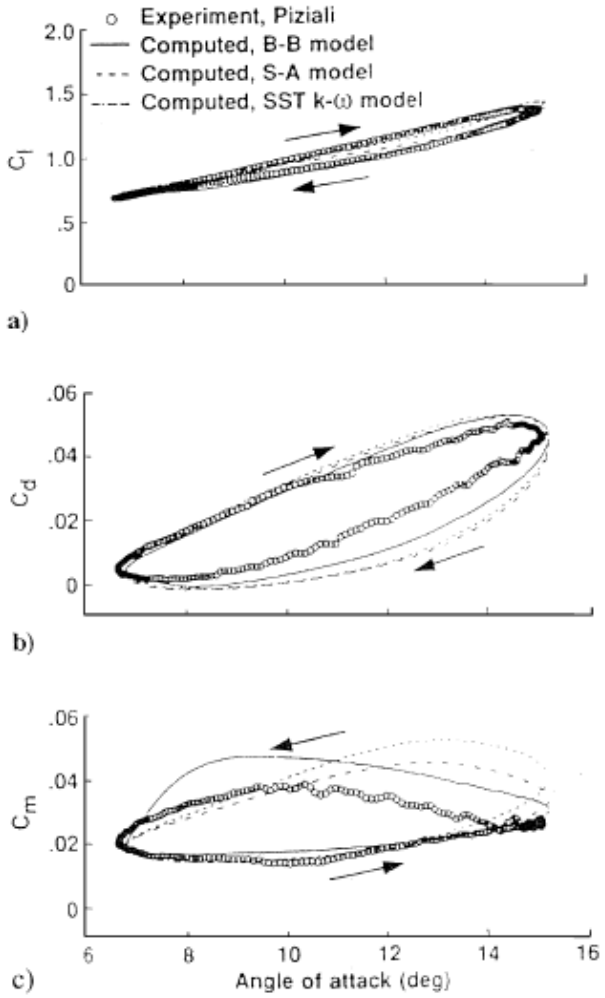

Figure 2.6: Comparison between computed and experimental aerodynamic loads for the NACA0015 undergoing light dynamic stall, from [64].

Some time later, in 2008, Majhi and Ganguli [65] investigated the impact of the small angle of attack assumption on the simulation of flapping rotor dynamics, as well as developing a mathematical model for the non-linear aerodynamics of the flapping rotor blade. During hover, the flap angle is constant, unlike forward flight, where the symmetry in aerodynamic load is lost, resulting in the azimuthal variation of flap angle. Furthermore, it is mentioned that, at low angles of attack, the flow remains fully attached to the blade and thus, can be modelled with quasi-steady approximations. However, at high angles of attack, the occurrence of separation and stall cause the lift and drag forces to become non-linear functions of angle of attack. Their results show that, for high enough values of blade pitch, the small angle assumption causes deviations in the magnitude and phase of the flapping response.

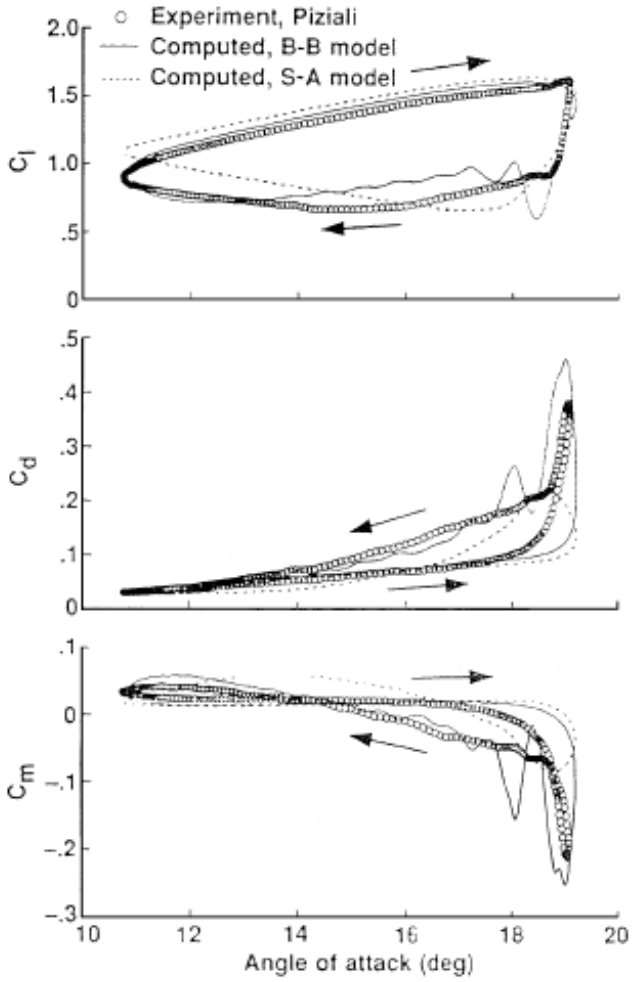

Figure 2.7: Comparison between computed and experimental aerodynamic loads for the NACA0015 undergoing deep dynamic stall, from [64].

While dynamic stall is often associated with helicopter rotors moving at high speed, dynamic stall vortices can also appear in MAVs. Furthermore, a vast majority of previous works focused on 2D flows. So, in 2010, Miguel Visbal [66] simulated plunging wings on 3D transitional flows at low Reynold. A high-order implicit Large-Eddy Simulation (LES) scheme was used to solve the flows. This model has been found to work well with laminar, transitional and turbulent flows at moderate Re. The results showed good agreement with experimental data for the formation and initial convection stages of the turbulent dynamic stall vortices. However, the later phases of the vortex shedding process proved to be substantially more sensitive, resulting in differences between results. The effects of grid resolution and spanwise extent on the models were both ruled out as the root cause of such discrepancies. Instead, the differences in results were pointed to be caused by the large-scale three dimensionality of the flow.

The following year, Heine et al. [67] investigated the impact of leading edge disturbance generators on the performance of an airfoil undergoing dynamic stall. According to the authors, dynamic stall is not only a limiting factor for helicopter forward flight and manoeuvring, but also for the aeroelastic stability of wind turbines. Several configurations of those devices were tested on a pitching OA209 airfoil, in a wing tunnel. The introduction of the disturbance generators on the leading edge allows for a lower negative pitching moment peak during dynamic stall, while producing higher lift during the downstroke. It also leads to a reduced hysteresis loop.

Two years later, Gharali et al. [68] studied the effects from oscillating horizontal freestream velocity on a pitching NACA0012 airfoil. CFD was used, with the SST $k-\omega$ model combined with a low-Reynolds correction factor. Their results show that an increase in lift coefficient during dynamic stall, relatively to the static stall lift, can be achieved for a pure pitching airfoil under constant freestream velocity. In-phase flow oscillations can provide a 5 times increase in lift, compared to the steady flow case, which can be further improved by increasing the reduced amplitude. In contrast, when the phase difference was high, the aerodynamics loads were reduced, and can be even lower than those from static stall.

Then in 2015, Vieira and Maughmer [69] discussed the role of dynamic stall in the airfoil design of rotorcraft. According to the authors, rotorcraft airfoil design has been mostly based on steady-flow aerodynamics. Because of this, traditional design requirements are reviewed in an attempt to include the effects of dynamic stall. As the consequence of the use of collective and cyclic pitch during manoeuvres and forward flight, parts of the blade can reach angles of attack superiors to the static stall angle. When combined with the unsteady effects from flapping and fluctuating pressure gradients, dynamic stall occurs. As already mentioned, this phenomenon allows the airfoil to momentarily exceed the maximum static lift. However, it is also companioned by abrupt increases in nose-down pitching moment and drag.

The high fidelity analysis required to consider these unsteady dynamics has only become possible with the improvements in computational power. The most popular method for unsteady analysis in the design of aerofoils relied on adjoint-based optimization, whose main advantage is the fact that is can process a large number of design variables. Nonetheless, since it is gradient based, it only explores a limited section of design space. This problem can be tackled by using genetic algorithms, combined with reduced-order models, to evaluate objective functions. The authors implemented an hybrid design approaches, using PROFIL, a potential-flow code, for the manipulation of the airfoil's pressure distributions. With this hybrid method, they were able to slightly improve the airfoil designs for unsteady flow conditions [69].

In the following year, Perdomo and Wei [70] further tackled the problem of numerically solving the non-linear, time-depend, differential equations that govern rotor dynamics. The authors describe the blade flapping motion as the result of the balance between the

moments from the blade's thrust, gravitational, centrifugal and inertia forces. While in hover the forces remain symmetrical among all blades, during forward flight, the advancing blades encounter a higher flow airspeed, while the retreating blade faces a lower airspeed. This difference results in the blades flapping motion. Many approaches to solve this mechanism have been carried out, such as the simplification of the flapping aerodynamics to linear equations. On their paper, the authors focus on the approximation of flapping dynamics with an inhomogeneous, linear, differential equation, coupled with periodic coefficients. Two methods to compute the Fourier coefficients of the motion were evaluated. Furthermore, the authors remark how it is possible to use a truncated solution of the model to estimate the periodic solution. However, this approach can lead to wrong results when higher harmonics of the motion play a substantial role.

The next year, Kai et al. [2] set to study the effects of plunging and pitching motions on the dynamic stall of the NACA0012 airfoil. Dynamic stall is referred as a collection of unsteady aerodynamic phenomena that result in the dynamic delay of stall to angles of attack greater than the static-stall angle of attack. This results in large oscillation of lift and pitching moment. Dynamic stall occurs during an helicopter's forward flight, mainly on the retreating blades. On their paper, finite volume CFD methods were used to calculate the flow fields based on the URANS equations. A NACA0012 airfoil undergoing a pitching-plunging motion was simulated at a Re of 1×10^5. Their results show good agreement with the experimental data during he up-stroke. The down-stroke is also close to the experimental data, apart from some oscillations of the computed values. However, it is during the flow separation phase that the numerical model exhibits the largest deviation from the data, showing large turbulence effects that make the aerodynamic forces' computation unreliable.

Another study into dynamic stall and unsteady aerodynamics modelling was performed by Truong [71] in the same year. According to the author, any predictive tool developed for the comprehensive analysis of rotorcraft is required to be both fast and accurate at estimating air loads. It is mentioned how the local aerodynamics of rotor blades are very complex, resulting in the two major problems of semi-empirical models: result accuracy and numerical convergence. Despite the fact that semi-empirical models of dynamic stall had been used since the 1970's, these often fail to reproduce the experimental results in some way. Most of these models also have trouble achieving numerical convergence. While the integration of CFD into comprehensive analysis is able to better compute the air loads, it lacks the speed required. A more recent approach separates the flow over the rotor blades into local and global flows. This separation allows for the use of distinct computation methods for each flow. Moment and wake models are used to compute the global flow, which determines the inflow conditions of the blades. Then, the forces produced by the blade cab be quickly evaluated from polars, which have been constructed from wind tunnel tests and expanded with CFD. However, these forces also have an effect on the global flow conditions, which needs to be taken into account, resulting in an iterative solving method. Overall, the results show a reasonable agreement with experimental data, but

demonstrated some difficulty into fully predicting the onset of dynamic stall. Still, the authors remarks that the computational time is substantially less than the required for coupled analysis with CFD.

Also in 2017, Geissler and van der Wall [72] attempted to improve the propulsive efficiency of plunging airfoils during dynamic stall. While helicopter blades operate in a distinct regime from natural flyers and MAVs, the authors enumerate some common features between the two: highly unsteady flows, appearance of concentrated leading edge vortices and flow separation being a limiting factor on the flight envelope. Motivated by this last fact and the recent push into using numerical codes to simulate dynamic stall, the authors apply a time-accurate 2D-Navier-Stokes code to solve a deforming plunging airfoil undergoing dynamic stall. Their results show that, while the dynamic stall vortex stays attached to the surface, the deformation can increase lift. However, it may be followed by a sudden loss of lift, increased drag and negative pitching moments, as the vortex detaches from the surface. Thus, it is concluded that for the best efficiency, concentrated vortices should be avoided altogether.

Some years later, in 2020, Wang and Zhao [73] attempted to improve the fidelity of rotor airfoil simulations by including the oscillating free-stream velocity. As already mentioned, the authors remark the importance of considering unsteady aerodynamic characteristics of the rotor for the study of dynamic stall. For this, the authors conduct simulations on the OA209 airfoil using CFD under oscillating freestream velocity. The URANS equations were used, together with the two-equation SST $k-\omega$ turbulence model. Their results show that, even though the computations of the 2D flowfields exhibit some deviation from the 3D aerodynamics of the rotor, they are much closer than the results obtained with 2D steady flow approaches.

Also in 2020, Samara and Johnson [74] investigated the potential of trailing edge flaps to control the aerodynamic loads on wing turbine blades. For that, a pitching cambered airfoil was investigated at a Re of 170000 over a range of reduced frequencies. The authors remark that dynamic stall often leads to severe fatigue on wind turbine blades. Their tests show that the trailing edge flap is able to produce an increase or decrease in C_l and C_m for the range of α evaluated. After passing the stall point, it was seen in the experimental results that there was a temporary loss of lift, while there is a reduction in C_m, accompanied by a large nose-down moment. With an increase in reduced frequency, the peak C_l increased, which suggested a stronger leading edge vortex. The formation of such vortices and the flow reattachment were both delayed with increasing k.

2.2 Neural Networks in Aerodynamics

Previously, the most common modelling tools for aerodynamics and, in particular, unsteady aerodynamics, have been presented. However, the majority of those methods require some degree of knowledge about the underlying physics of the system. Instead, neural networks can learn to reproduce a system's behaviour without any prior knowledge of its dynamics. Furthermore, one trained, they are able to produce results very fast. For these reasons, it is no surprise that there has been a surge in interest of applying neural networks and other forms of artificial intelligence to fluid mechanics. The advancements in neural networks, their structures and implementation approaches, specially in regard of aeronautical application will now be discussed.

2.2.1 Machine Learning and Neural Networks

The human desire to understand himself and the world around him knows no limits. The origins of human intelligence have always posed a major mystery. As the understanding of neurology evolved, so did the desired to mimic the processing power of the brain on artificial machines. Even though mathematical descriptions of artificial intelligence algorithms have existed for a long time, only with the advancements in computer science and computational power, it has become feasible to pursue its implementation. In this section, the basis of how neural networks function will be presented. Their mathematical model, most common architectures and design parameters are introduced.

Artificial Intelligence (AI) can be defined has the field of study which seeks the development of artificial entities with cognitive capabilities similar to the ones of human beings. Many different approaches have been proposed regarding the implementation of artificial intelligence, mainly the connectivity and biological approaches. In the connectivity approach, artificial intelligence is seen as an emergent property from the multiple interactions between a large number of basic processing units. This method is largely inspired by the structure of the brain, which is comprised of thousands of millions of cognitive units, the neurons, which one with multiple connections. The neuron takes in input signals, which are combined and excite the neuron if above a certain threshold. The signal is then sent to the other neurons and the process repeats [75].

Almost all artificial neurons follow basic mathematical model proposed by McCulloch and Pitts [76] from 1943. They described a neuron made of several inputs, which one with its associated weight, that describe its excitement of inhibition strength. The neuron takes the weighted sum from all its inputs, which, when compared against a threshold, can result in the neuron's activation. For this reason, this simple type of artificial neuron is also denominated as a linear threshold unit, which can be seen in Figure 2.8, where ,f represents the linear threshold, or step function.

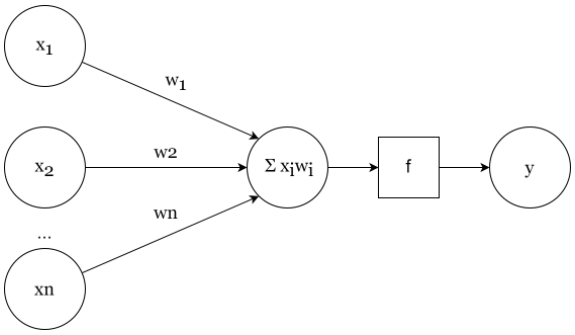

Figure 2.8: Model behind the linear threshold unit.

Most applications however include an additional parameter, the bias, which was derived as the symmetric of the threshold on the linear threshold unit. The bias is added to the weighted sum, as shown in Figure 2.9 and f now becomes a generic activation function. Mathematically, the bias is the symmetric of the threshold. If the weights reflect the strength of a particular input, the bias represents the overall neuron's activation sensitivity.

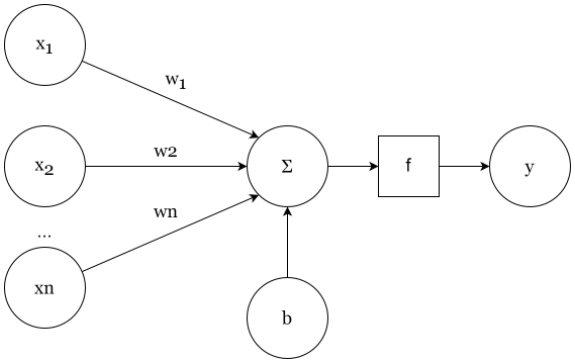

Figure 2.9: Model of a basic artificial neuron.

An artificial neural network (ANN) is then formed by a network of several connected artificial neurons. These networks are normally organised into layers, which one containing many artificial neurons. The layers are connected in such a way that the outputs of the neurons from the previous layer forms the input of the next layer. The first layer of a NN, which receives the external signals, is the input layer. An the final layer, which provides the external output, is the output layer. The layers in between form the hidden layers, as shown in Figure 2.10. Artificial neural networks can have just one single layer to multiple layers.

If the neurons of a layer have connections to all the neurons of the next layer, it is said that the layers are fully connected. Furthermore, artificial ANNs can be separated into two main categories. On feed-forward neural networks (FNNs), the output of any neuron can only affect the neurons of the following layers, which means no layer can directly, nor indirectly, feed into itself. In contrast, recurrent neural networks (RNN) feature feedback loops from the outputs of a layer to the inputs of a previous layer. This means that the

output of the network depends on its previous state.

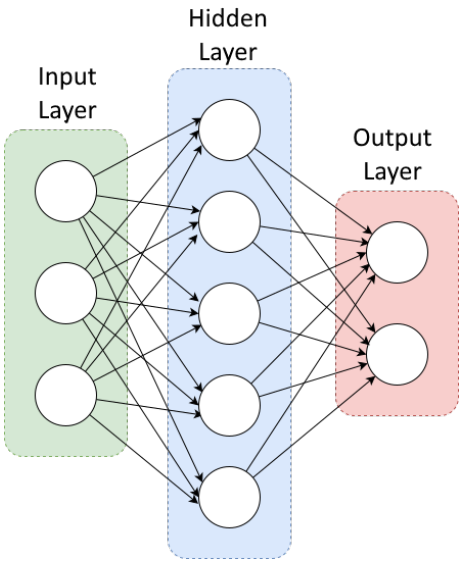

Figure 2.10: Example topology of a multi-layer artificial neural network.

Another common network typology is the auto-encoder, which comprises of an encoder followed by a decoder architecture. This type of ANN makes use of the convolution operation to compress and decompress information. The encoder is responsible for detecting features in the data, which can be used by the decoder to construct the output. This scheme is often found in imaging processing applications.

However, in order to function, the weights associated to the connections must be such that the network produces the correct outputs. For that, a training algorithm is required. At their core, neural networks are empirical methods, which rely on data to be trained. While gathering the amount of training examples necessary can be difficult, it is their ability to learn to model any set of data that makes ANNs very advantageous. There are two main approaches regarding training an artificial neural network. One way is to subject the ANN to a supervised learning task, the NN tries to adjust to fit the labelled training examples, which have previously been classified by an instructor. In this approach, the network has access to the input data and its respective desired output. The second away is for the ANN to undergo a unsupervised learning task, in which the network it self must explore the unlabelled data and figure out its hidden relations, as it learns to classify it.

Generally, artificial neural networks learn by adjusting the weights of the sections between neurons, based on Donald Hebb's [77] idea that if two neurons fire at the same time, their connection should be reinforced. This can be described mathematically as

$$\Delta_{w_{ij}} = \eta x_i x_j, \tag{2.12}$$

where $\Delta_{w_{ij}}$ is the change of weight w_{ij}, η is the learning rate, x_i represents the neuron's input and x_j its output. Since the network will be trained from examples, the weight change

can be given in terms of the error as

$$\Delta_{w_{ij}} = \eta(d_j - x_j)x_i, \tag{2.13}$$

where d_j is the correct value of the output and x_i the actual output. This last formula is known as the delta rule, or Widrow-Hoff rule [78]. However this is the most simple case for a single layer neural network. Still, its generalized derivation for multi-layer networks which relies on the minimization of the square error is the most commonly used.

Since neural networks learn from examples, the basis behind training is that one can take the output error and feed it through the network, in order to adjust the subsequent weights. Since this step takes propagates the error across the network, from the output to the first layer, it is called back-propagation, and it forms the core in the training of artificial neural networks. As usual, several different methods have been proposed throughout the years, seeking improved speed and network accuracy.

There are also many different ways the training examples can be presented to the neural network. In the most straight-forward approach, the weights and biases are updated after each training example. In this case, each example corresponds to one training epoch. However, when there is a large amount of data, it is usual to separate the data into smaller samples, called batches. The neural network runs through the batches and the weights are only updated once the full batch has been tested.

Returning to the learning rule's parameters, the learning rate, η, is one of the most important hyper-parameters for the training of any ANN. It defines the rate at which the value of the weights converges towards the suggested value. If left too low, then the network will take longer to converge to a suitable model. In contrast, if too hight, the training will suffer oscillations and may not converge.

Another hyper-parameter often included in the learning rule is the momentum coefficient β. This coefficient interpolates between the previous and the suggested weights in order to obtain the new network weights. Whilst the learning rate defines how much the back propagation influences the suggestion of new weights, the momentum coefficient reflects how much the ANN should value the current weights over the new suggested ones. It can be thought as the level of trust in the current suggestion and it allows for a smother training convergence at higher learning rates. However, a high momentum coefficient can also delay the convergence rate, as the weight suggestions will take longer to accumulate and produce a considerate change.

Other hyper-parameters include the batch size, m, the number of training iterations, the number of hidden units and regularization coefficients such as weight decay and sparsity of activation coefficients. There are many ways to set and optimize this parameters, from manual to automated search methods.

The activation function is also very important and must be carefully chosen, as it has a direct impact in both the network's model and training. This function is what separates a neural network from an average multiple equation linear model. Many different types of activation functions have been proposed through the years, which one with its advantages and disadvantages. This function is responsible for the neuron's activation. Some examples of activation functions are the step function, the sigmoid, the hyperbolic tangent and more recent functions such as the ReLU, PReLU and softmax, among others. Figure 2.11 shows some of the most common activation functions adopted nowadays.

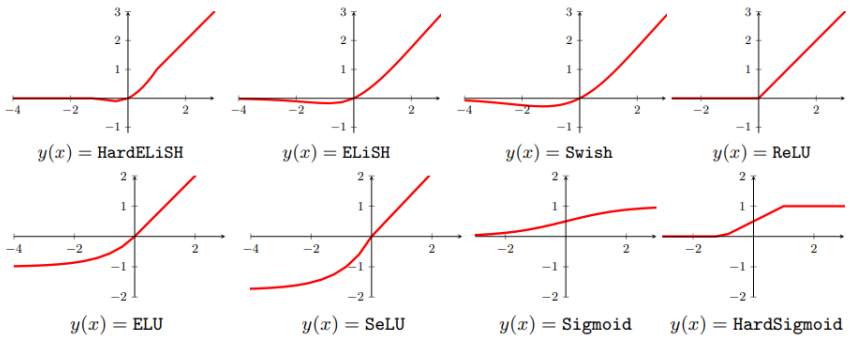

Figure 2.11: Commonly used contemporary activation functions, from [79].

2.2.2 Development and Application of Neural Networks

Now that the basis behind neural networks has been introduced, an overview of the evolution and development of neural networks is performed. As seen, neural networks can exist in many different formats, from the simple perception, to deep multi-layer networks. Neural networks can have simple feed-forward typologies or present feedback loops or memory cells. The advantages and disadvantages of the many architectures are discussed, in particular regarding recurrent neural networks. The most common parameters, activation functions, cost function and optimization algorithms are presented and reviewed.

Starting in 1996, Rojas [80] described one of the most basic building blocks of artificial neural networks, the perceptron. According to the author, the perceptron was first introduced by Frank Rosenblatt, in 1958, and was improved throughout the 1960's. The perceptron was proposed as a more general model than McCulloch-Pitts units, by pioneering the application of weighted threshold elements and interconnection patterns. Rosenblatt's classical perceptron was then simplified by Minskty and Paper, who studied its computational capabilities. The development of the perceptron was largely inspired by the understanding of the human visual pathway and the neuroanatomy of the retina, which came around the same time. This biological isnpiration led to further developments into the cognitron by Fukushima et al., whose proposed network architecture composes the base of imagine recognition algorithms.

The year prior, in 1995, Williams and Zipser [81] publishes the first edition of his book, in which an extensive insight into the learning algorithms used in recurrent neural networks is provided. The authors describe RNNs as networks which feature feedback loops, which gives them a very nonlinear dynamic nature. The networks are trained to perform temporal supervised learning tasks, from which the most common type features both time-varying inputs and target outputs. Two kinds of network operation are presented: continual or epochwise. In epochwise operation, the network is run for a prescribed amount of time, an epoch, after which its internal state is reset. This means that the start of an epoch is not dependent on the state of the previous epoch. This is not true for continuous operation, where there is no clear delimitation in the data. Training can also be done incrementally or in batches. In batch-training, the network is run over a series of training examples (a batch), and only after are the weights updated. On the other hand, in the incremental method, the weights are updated after each training example, ie. after each epoch.

The training methods prescribed on their work are gradient-based, in which the gradient of a measurement of performance is calculated with respect to the network's weight space. The algorithms presented are for time-discrete networks. The most basic objective of the learning process can be thought as minimizing the total error of the network. In order to obtain the gradient of this quantity in weight space, the error must be fed backwards through the network, in a process dominated backpropagation through time (BPTT). The authors distinguise real-time BPTT algorithms and epochwise BPTT. This backpropagation algorithms are the same as the ones used in feedforward networks, except that the recurrent neural network must be first unrolled in time into a larger feedforward network. The authors refer to one of the biggest disadvantages of the epochwise BPTT as being the need to save the history of the network inputs, state and errors over the complete epoch. However, this BPTT can be made more efficient by applying truncated BPTT, in which only a fixed number of previous time steps are taken into account. This approach is specially suitable for networks whose state tends to converge to fixed points [81].

Some years forward, in 1999, Chang and Mak [82] presented a modified version of conjugate gradient learning for recurrent neural networks. According to the authors, although backpropagation has been shown to be able to train FNNs, it requires three arbitrary coefficients to be chosen: the learning rate, momentum and number of hidden nodes. While recurrent neural networks contain feedback loops, these can be trained very similarly to feedforward networks. RNNs are used mainly for their capability of processing temporal patterns. However, in order to achieve convergence, a small learning rate is required, and thus, a larger number of training epochs. In order to improve training, Chang and Mak extend the conjugate gradient recurrent learning (CGRL) algorithm from Johansson et al. to recurrent neural networks. The major advantage of the conjugate gradient method is that the minimization performed in a step won't be undone by the following step. This is because the CGRL estimates the optimum learning rate for each gradient descent step. The authors tested a standard RNN with RTRL, batch-mode RTRL learning and the CGRL

algorithms. Their experiment showed the fluctuating behaviour typical of standard gradient descent, since this methods not lot always lead directly to a minimum. The CGRL network had a much smoother and quicker descent, achieving a lower minimum faster. It is also important to note that the RTRL and batch mode RTRL algorithms achieved about the same minimum in a similar time, although the batch mode training seamed to produce a smother descent than the RTRL.

Some years later, in 2012, Yoshua Bengio [83] compiled a vast assortment of recommendations for the training of deep neural networks. Gradient based optimization is the most generalized method for optimization of the parameters of a neural network and most of the recommendations provided apply to SGD. The learning rate is regarded as a crucial hyper-parameter of training. The optimal learning rate can normally be found to be close to twice the largest learning rate that does not lead to divergence of the training criterion. It has been found that a value of 0.01 can be good starting point for multi-layer networks. However, the authors remark how one must not rely on such default values, since the optimal results depend greatly on the model, making the learning rate the first parameter that should be tuned.

While the weights and biases of the neural network can be updated at the end of each training example, it is common to let the ANN run over a sample of training examples before updating those parameters. The sample of data is called a batch and its size, m, is another important hyper-parameter to consider. Increasing the size of the batch can be specially advantageous when using parallel computing. On the other hand, increasing the batch size also reduces the number of updates per computational cycle, which slows down convergence. However, because the local gradient direction does not fully align with the true direction of descent, performing parameter updates more frequently can be beneficial. Because of this, small values of m can provide better and faster exploration of the parameter space, specially for larger learning rates. For this reason, mini-batch gradient descent is often used and 32 for a batch size is regarded as a good starting value [83].

Continuing with the hyper-parameters, momentum, β, was introduced in order to smooth out the stochastic gradient descent. It can be easily implemented as the moving average of past gradients, where the momentum controls how fast the previous gradients get down-weighted. Its goal is to filter out some of the noise and oscillations from the gradient descent, particularly in regions of high curvature of the objective function. Even though no momentum can work well is most situations, its incorporation has been found to often allow for faster convergence [83].

Other hyper-parameters include regularization coefficients like weight decay and sparsity of activation. Weight decay is a regularization term that can aid in preventing over-fitting by pushing values to be near zero. This coefficient can be implemented for two types of regularization, L1 and L2 regularization. Similarly to weight decay, the sparsity of activation also works by including a penalty in the training criterion, encouraging the weights to

approach zero. This way, it helps to create sparsity in the hidden units and is very common on deep learning models [83].

Weight and bias initialization is also an important consideration. While biases can be initialized to zero, weights need to be initialized in such a away that breaks symmetry inside the hidden layers. A zero-mean Gaussian distribution with a standard deviation between 0.01 and 0.1 can also work well. Unsupervised pre-training can also be used to initialize the network [83].

Nevertheless, Yoshua Bengio [83] remarks how these recommendations are little more than a mere starting point and further evaluation is always required. When it comes to optimize a model, three main possibilities arise: manual search, automated (or semi-automated) grid-search and random sampling. The main criterion for comparing and evaluation a ANN performance is the validation error, since it serves as an estimation of the generalization error. However, the relationships between the validation error and the hyper-parameters can be very complex to understand for a manual search. Coordinated descent, in which only one parameter is changed at a time is a commonly used method for manual search.

The following year, Ryan Kiros [84] attempted to incorporate Hessian-free Optimization into training neural networks by introducing stochastic Hessian-free Optimization (SHF). According to the author, the first-order SGD has been the default optimization method used to train neural networks. This is because SGD is simple to implement, can operate in mini-batches which allow scaling to large datasets and its inherent noise often provides solutions that are well generalized. However, second-order methods are of their own interest since they could still operate in batch and produce more substantial weight updates. Previously, Martens [85] was able to adapt Hessian-free, a truncated Newton method, to train a ANN. Hessian-free uses a conjugate gradient algorithm to iteratively produce updating directions. This algorithm only requires curvature-vector products, making it as computational efficient as computing gradients. Building up on Martens' work, the author proposes stochastic Hessian-free optimization that works on both the gradient and curvature of mini-batches. It seeks to combine the generalization capability of SGD with the second-order information of Hessian-free optimization. After testing SHF with and without dropout, it is concluded that it can perform well on classification tasks, when compared against dropout SGD, NAG and momentum methods. However, the author finishes by remarking that this new methods still lacks further generalization to other ANN architectures, such as convolutional, recurrent and recursive neural networks.

Then in 2014, Kingma and Ba [86] presented a new method for first order stochastic gradient-based optimization, ADAM. ADAM, or adaptive moment estimation, calculates different adaptive learning rates for the update of each singular network parameter. It uses moving averages of the gradients and squared gradients to compute the first and second moments, which correspond to the mean and uncentered variance of the gradient, respectively. Because these moving averages are initialized to zero, a correction is

performed to remove their bias towards zero. These bias-corrected moments are then used to compute the parameter updates. The authors then compare ADAM to known and used algorithms, like RMSProp and AdaGrad, on multi-layer networks activated with ReLU and using mini-batches. From their results it was concluded that ADAM is able to converge faster and reaches lower minimums than other methods, while requiring less memory storage. A variant of ADAM, AdaMax is also presented, which solves the zero bias of ADAM.

A couple of years after, in 2016, Bradbury et al. [87] presented a new model for sequential time-dependent neural networks, the quasi-recurrent neural network (QRNN). The authors mention how recurrent neural networks (RNNs), including long short-term memory (LSTM) gated structures, became the most used neural networks for dealing with time-dependent and sequence modelling tasks. These networks have been widely used in natural language, machine translation and question answering problems. However, the fact the computation of a time step on a RNN requires the previous time step to be done, limits the parallelism of the algorithm. On the other hand, convolutional neural networks offer a high level of parallel computation, but cannot process sequential tasks that depend on a long time history. The quasi-recurrent neural network tries to take advantage of both models, enabling for the execution of long sequential tasks while maintaining a high degree of parallelism. When compared to LSTM performing language related tasks, the QRNN where able to achieve higher accuracy, while being faster.

In the same year, Keskar et al. [88] set to investigate why the use of large-batch in SGD optimization leads to poor model quality. This contrasts with training with small batches, raging anywhere from 32 to 512 data points, which are able to produce good generalization on trained neural networks. The authors mention that training a neural network can be described as performing a non-convex optimization of a loss function. Using SGD and its variants coupled with small batches is a well proven way to train a neural network. However, the sequential nature of iterations and small batch sizes, this method limits the degree of parallel processing that is possible to achieve. One way to increase parallelism would be to increase the batch size. The problem is that increased batch sizes has been shown to lead to worst performing models. This is thought to be caused by the reduced gradient noise from training large-batches, which is not enough to push the iterator out of the minimizer concavity. With small batches, the larger noise in the gradient helps to push the gradient towards flatter minimums, where it won't cause the function to exit the minimum vicinity. Finally, it was shown that a large-batch approach can be deployed after the small-batch has found a flat minimum of the function, wielding still good generalization of the model.

Also in 2016, Sebastian Ruder [89] presented a review of gradient descent algorithms used for training neural networks. Gradient descent is the most common method used to optimize neural networks by minimizing the objective function, with respect to the network's parameters. Batch gradient descent, or just standard gradient descent, computes

the gradient of the cost function for the complete training dataset before updating the parameters. This method can be very slow can become impracticable for very large datasets. In contrast, SGD executes a parameter update after each training example. This approach allows for faster convergence and for online training, however it introduces large fluctuations in the objective function. While this fluctuations can make it harder for the function to converge to a exact minimum, its noise enables it to jump to better local minimums. Mini-batch gradient descent tries to combine the best of batch and stochastic gradient descent. It works by updating the parameters after each mini-batch of training examples. This approach reduces the fluctuation of the objective function, which can produce better convergence. Mini batches often range from 50 to 256 training examples and this is the most used algorithm for training neural networks.

Following this, the author mentions some improvements have been proposed for gradient descent optimization. The first of these is the introduction of momentum, which reduces oscillations and helps to accelerate convergence by pushing the gradient in the relevant direction. It works by adding part of the previous suggestion to the current one. This way it increases the update vector when the update is in the same direction and reduces it when opposed. Nesterov's accelerated gradient (NAG) is more advanced form of momentum which can slow down updates when getting near the minimum. NAG works by using the previous momentum to estimate where the new set of parameters should be. Adagrad is another gradient optimization scheme which adapts the learning rate to the sparsity. This method allows for the use of different, optimized learning rate for each singular parameter, proving larger updates for less frequent parameters and smaller updates for frequent ones. This makes it a good option to tackle sparse data and has the large advantage of eliminating the need for manually tuning the learning rate. However, because it accumulates the squared gradients in the denominator, the learning rate becomes increasingly small, reaching a point at which the network is not able to learn any more. Adadelta is an improvement on Adagrad that seeks to solve the this issue. Instead of just storing all past squared gradients, the Adadelta uses a decaying average of a fixed window of previous squared gradients [89].

Other optimization strategies include the already-mentioned ADAM, which is another strategy for computing adaptive learning rates for the parameters. Not only does it store the previous history of square gradients, like Adadelta, as it also relies on the past gradients. Adamax and Nadam are two promising variants of ADAM. Nadam is like ADAM, but instead of using regular momentum, it uses NAG. Additional considerations include shuffling the training data after each epoch, unless the objective is to solve increasingly harder problems. Another strategy is perform batch normalization where the network parameters are normalized to zero mean and unit variance after each batch. This practise has been shown to allow for higher learning rates, while also acting as a form of regularization [89].

Then, in 2019, Basirat and Roth [79] provide a comprehensive review of the most commonly used activation functions used for training of deep networks. The authors start by stating that as neural networks get deeper and more complex, the grater the interest in making the training process more efficient and stable. Squashing functions, like the sigmoid and hyperbolic tangent where the first activation functions used in neural networks. However, these functions are subject to the vanishing gradient problem. Because of this, non-squashing functions were proposed, like the rectified linear unit (ReLU). While the ReLU solves the vanishing gradient, it does not allow for information to flow with negative values. This is called the dying ReLU problem. Despite this, the rectified linear units are one of the most used activation functions for their simplicity and reliability. Some variants of the ReLU have been introduced that mitigate the dying ReLU problem, such as the Leaky ReLU, the exponential linear unit (ELU) and the scaled exponentiation linear unit (SeLU). The graphs for these functions can be found in Figure 2.12.

The Swish was the next major development in activation functions. This function has been found to propagated information better than the ReLU. The authors then attempt to use genetic algorithms, a form of machine learning, to try and find the best activation function. From their findings, the ELiSH and HardELiSH were introduced, which are also shown in Figure 2.12. These presented very promising performance when compared to other functions.

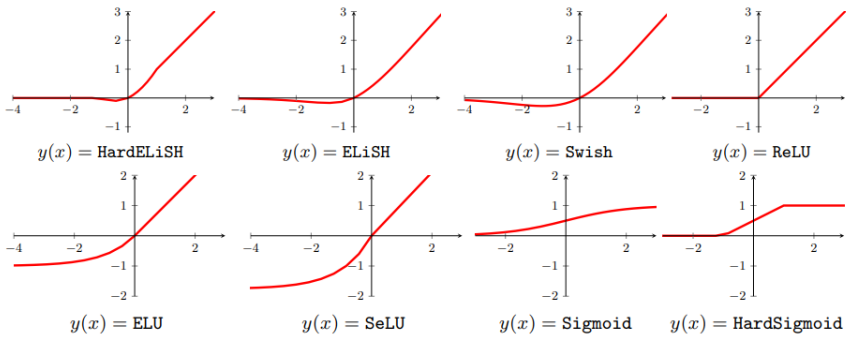

Figure 2.12: Commonly used contemporary activation functions, from [79].

Also in 2019, Staudemeyer and Morris [90] provide insights into Long Short-Term Memory (LSTM) Recurrent Neural Networks. The authors stated that while RNNs are good for processing time-depend tasks, they often cannot rely on information from more than the last 5 to 10 time steps. This is due to the fact that during backpropagation the error tends to either blow up or vanish, in what is called the vanishing error problem. The LSTM is then presented as a model that seeks to solve this problem, by using constant error carousels (CECs), which allow them to process context of the past 1000 time steps or more. The CEC is a unit with a single feedback loop connect to itself with a weight of 1.0. The access to the information on the CEC is controlled by input and output gates, which combine with the CEC to form a memory block, as shown in Figure 2.13. LSTM networks

have been successfully used for tasks that require both precise timing and counting, as well as noisy data sequences. LSTM-RNNs have been demonstrated to perform well with speech and handwriting recognition, machine translation and even text generation. More recently, the Gated Recurrent Unit (GRU) has been introduced as an RNN alternative to LSTMs. Instead of a memory cell, the GRU uses reset and update gates, as it shows promising results when compared to LSTM-RNNs.

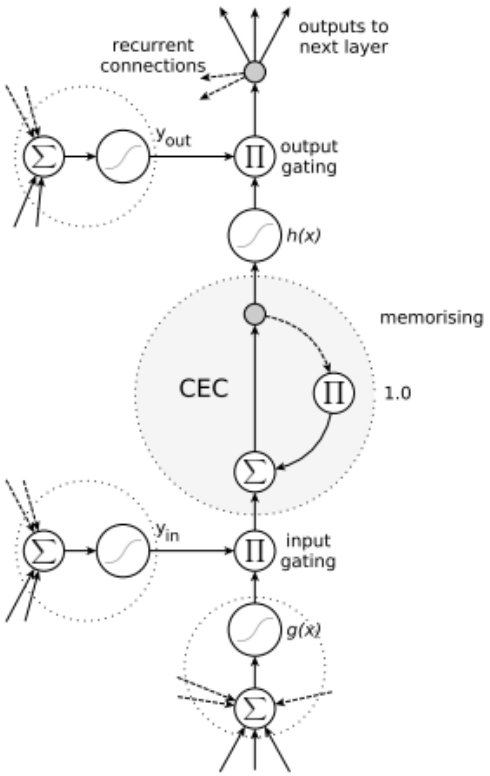

Figure 2.13: LSTM memory block model, from [90].

Then, in 2021, Johannes Lederer [91] provides his own extensive overview of activation functions used in neural networks for both general and deep learning applications. The activation function plays a major role in the model's performance. However, unlike the other parameters that are chosen to best fit the training data, the activation function is often chosen a priori and left fixed. Another important point is that networks with linear activations will always produce linear models. For this reason, the linear (or identity) activation function is rarely used, and when used is usually limited to the output layer. Among the reviewed functions are the sigmoid functions, like the logistic, arctan, tanh and Softsign. Regarding piecewise-linear functions, these include the identity, ReLU and Leaky ReLU. Other functions presented are the Softplus, ELU, SELU and Swish functions. It is also mentioned how similar activation functions should produce similar trained networks when the network parameters are well optimized. Because of this, when choosing between similar activation functions, the choice should focus on computational efficiency.

From their findings, the authors remark the Softsign and ReLU as good default functions.

2.2.3 Neural Networks in Aeronautics

Even though the early applications of artificial intelligence and machine learning was focused on imagine recognition and language models, they also found their way into the field of aerodynamics. Neural network's potential as aerodynamic modelling tools was quickly realised. Several works have been made and proposed in that regard, some of which are discussed here. However, many challenges remain to be tackled, namely regarding from model generalization and accuracy improvement. It is clear that the use of neural networks as a modelling tool is still in its infancy. In this section, many attempts to implement neural networks, namely recurrent neural networks, as prediction tools for unsteady aerodynamics are reviewed. Their approaches, methodologies, results and difficulties are discussed.

In 2002, Rajkumar et al. [16] attempted to estimate aerodynamic coefficients using neural networks. The network was presented as an alternative to interpolate between sparse experimental data. The network had a single hidden layer and took two inputs, the angle of attack and Mach number, to produce a single output, the lift coefficient was estimated. The neurons used a sigmoid activation, so all inputs and outputs were scaled to the $[0.1, 0.9]$ range. The network was trained with varying angles of attack and Mach numbers. An iterative approach was used to optimize the number of hidden neurons. The trained network was then used to estimate the evolution of the lift coefficient. The results exhibited some deviations, but still under 10% for the predicted values, however the network was only trained for 1000 iterations.

The following year, Suresh et al. [17] trained a recurrent neural network for the prediction of lift coefficient of an airfoil at high angles of attack. Two distinct network topologies were proposed and tested: a memory neuron network (MNN) and a recurrent neural network (RNN). The networks take the mean angle of attack and the cyclic angle of attack as inputs, and output the instantaneous lift coefficient. Both networks feature a single hidden layer. The weights are randomly initialized near zero and the delays started with zero. The learning rate was set as 0.03 and 0.01 for the MNN and RNN, respectively. The training error was limited to 0.002, after which training was stopped. The results show that both networks were able to estimate the lift coefficients with less than 1% deviations. It is worth noting that the largest error occurs at the start of the estimation period. Finally, the authors remark how the speed of the trained model could allow them to be incorporated in commercial rotor codes.

Two years after, Marques et al. [92] implemented a recurrent neural network to estimate the flapping and torsion motion of a hingeless flapping wing. In particular, both a RNN and a time-delay RNN was used. A time-delay neural network (TDNN) is a type of RNN that relies only on its previous inputs, and does not take a feedback of its previous out-

puts. Both networks used the sigmoid activation function on the hidden layer and a linear activation on the output neurons. A data set of blade motions was created using a finite element model for varying motion frequencies. While both networks converged at similar rates, the RNN achieved a minimum one order of magnitude lower than the TDNN. Comparing the results from the trained networks, they captured the overall behaviour of the system, but retained some evident noise in the response. An analysis of the power spectrum obtained from the network's results show that these also learned the frequency dynamic of the flapping system.

Some years later, in 2013, Dmitry Ignateyv [93] used a recurrent neural network to predict the unsteady flight characteristics of aircraft. His network features a delay layer for both a fixed-length window of the previous inputs and outputs. It is a shallow network with just one hidden layer and linear output layer. It outputs the lift and moment coefficients. Weight regularization was used to prevent overfitting. Data was generated for two cases: a delta wing and a swept wing with canard aircraft. The aircraft were subject to large amplitude oscillation of angle of attack. The trained networks produced overall good results, with some small deviations from the training data. The errors on the test set sat between 5.8% and 8.6%, compared to 4.5% to 7.1% on the training set.

And in 2019, Han et al. [94] implemented a hybrid neural network for the temporal prediction of unsteady wake flows. The network is comprised of a series of convolutional layers, followed by convolutional LSTM cells and deconvolutional layers. The networks outputs the pressure and velocity fields. The root mean squared error was used as the objective function, together with the PReLU as the nonlinear activation function. A dataset was built using CFD for a cylinder and NACA0012 airfoil wake flows, at different $Reynolds$ numbers and angle of attack of 20 degrees. Each dataset featured 10 000 time-setps of data, 500 of which were used as validation. The trained network showed good agreement with the CFD data, predicting 500 time-steps of flow in just 74s, about 11 times faster than CFD technology. The networks also exhibited good generalization with different Reynolds numbers. The largest deviation in results was found on the cylinder wall or wake, where the flowfields change the most.

Then in 2020, Peng et al. [18] proposed an convolutional-deconvolutional neural network (CNN-DCNN) as a reduced order model for the prediction of the velocity fields around airfoils. The network takes as inputs the pressure distribution, the Reynolds number, the geometry of the airfoil and the angle of attack as images. The first network, the CNN, works as an encoder that compresses that information. Then, the DCNN decompresses that information to construct the velocity field. A data set was build using CFD and the data parametrized. Ten percent of the produced data was used for validation, while the remaining formed the training data. The Adam optimizer was used. From the validation it was concluded that the network had achieve good generalization and was able to produce accurate estimates, while being much faster than traditional CFD computations.

One year later, Hui et al. [95] implemented a convolutional neural network to compute the pressure distribution of an airfoil. The network gets the airfoil geometry as an image input. The input is processed by a series of convolution kernels and neurons activated with ReLU. The network then outputs the pressure distribution of the airfoil. The data was generated with CFD for a variation of 1500 airfoil geometries and it is separated into training and validation sets. The geometry of the airfoils was converted into a grid of points by a signed distance function. Batch normalization was applied to the network parameters. Several configurations of number of layers and number of neurons were tested. The authors concluded that the CNNs were able to accurately estimate the pressure distributions for the airfoils, even with double and strong shocks present, while being much faster than CFD methods.

Still in 2021, Balla et al. [13] also tried to perform aerodynamic coefficient prediction with neural networks. Three different types of neural networks are compared. One that produces only one aerodynamic coefficient at a time, another that estimates the three aerodynamic coefficients simultaneously and finally, one that estimates the pressure distribution over the airfoil/wing section. All networks take the freestream Mach number and angle of attack as inputs. After testing all networks on both airfoils and wings, it was concluded that the network that predicted the pressure distribution performed better than the other ones. It was also found that predicting the different the aerodynamic coefficients simultaneously worked better than predicting only one at a time. It was found that the 3D trained network produces better results for swept wings than the 2D one on its individual airfoil sections.

In the same year, several works were published regarding applications of recurrent neural networks. One of such papers was the one presented by Zafar et al. [96], in which a recurrent neural network is used to predict the location of the laminar-turbulent transition of the boundary layer flow over an airfoil. The proposed RNN takes a sequence of mean boundary-layer flow properties as inputs and outputs the local growth rate of the most amplified disturbance. The flow properties considered are the Reynolds number, velocity profile and its first and second derivatives. Before the RNN, the data is feed to a CNN that compresses the spatial information into its latent features. A data set of flow conditions was produced for 53 airfoils and several training strategies were tested. The authors conclude that the trained network is able to predict the transition location within 0.7% of the mean aerodynamic chord. Error based data augmentation allowed for the best accuracy and lowest amount of training examples required. However, randomly sampling cases from the full training set for each airfoil also produced similar results. It was also found that the network is able to extrapolate well within the trained airfoil families, but its accuracy diminishes for airfoils of other families.

Zhou et al. [97] implements a recurrent neural network as a non-linear solver for a closed-loop UAV controller. A sliding-window-like LSTM structure is proposed, in which the current LSMT cell only depends on the last 5 cell outputs. The network uses the inertial

accelerations and velocity of the aircraft as inputs and outputs the pitch, roll, yaw and body acceleration. The LSTM layer feeds a connected network with a ReLU activated hidden layer of 36 neurons, followed by a regression hidden layer. The MSE was chosen as the loss function and RMSprop as the optimizer. The 1200 data samples are split into 1080 for training and 120 for testing. At each iteration, a random training example is sampled from the 1080. Updates are performed after a batch of 10 training cases. The trained network is able to produce accurate estimates much faster than a traditional solver, which allows it to be used for real-time control applications.

And Zhang et al. [19] proposed a recurrent neural network for the prediction of unsteady aerodynamic properties of airfoils undergoing stall. A four-layer networks is created, with a nonlinear and linear hidden layers. The previous outputs of the network are fed back to it from a delay layer. The inputs are also feed into the hidden layer. The L-M method is used for the training. The data was created using CFD for the NACA0012 airfoil. The variation of pitch angle is given by sinusoidal chirp signals. The networks are trained to predict the lift and moment coefficient of the oscillating airfoils. A set of composite signals was then used to validate the network's response. The results showed that the network was indeed able to predict the lift and moment coefficients evolution for the combined sinusoidal oscillating airfoils, even at large amplitude of motion.

The following year, Wu et al. [98] propose the use of a neural networks as a mean for the prediction of flow fields around airfoils. On their paper, a convolutional-deconvolutional (CNN-DCNN) neural network model is proposed as a data-driven model to construct the velocity and pressure flow fields of an airfoil. The network comprises of tow stages, the CNN that encodes the high dimensional properties of the physical field into a smaller space, followed by the DCNN that decodes that information into the output physical field. This type of network is used for image processing and because of this, the inputs are treated as 2D images. It's inputs are the local Re and a signed distance function that represents the airfoil's geometry. CFD was used to construct a labelled database for the NACA0012 airfoil for a range of angles of attack and Re. The ELU activation function was used. After training, the network presented a mean accuracy of 98.2% and 99.6% for the pressure and velocity fields, respectively, including the prediction of separated flows. Proper orthogonal distribution was then implemented to increase the networks accuracy, wielding a final trained accuracy of 99.2% and 99.8%.

In the same year, Peng et al. [99] implemented a element spatial convolution neural network (ESCNN) for the estimation of an airfoil's lift coefficient. Following developments with physics informed neural networks (PINNs), the proposed ESCNN model takes the airfoil coordinates and angle of attack as input and produces the the total lift coefficient as output. The work uses a series of convolution operations followed by non-linear activations, ending with a fully connected layer. This means that the lift coefficient is the weighted sum of the previous neuron's state. Airfoil data samples are taken from an airfoil database for a Mach pf 0.3 and Re of 13 million. The angle of attack is restrained to

the no separation regime, from -2 to 10 degrees. The lift coefficients where computed with CFD producing a total of 15678 data pairs. The network was trained with both the LeadkyReLU and ReLU activation functions. The trained network achieved a minimum error of 0.97%, performing the best when compared against other existing NNs. Furthermore, the authors investigate the model's weights and activations to try and get an insight into the trained "black-box" function of the network. From the final neuron activations they find that these correlate with the vortex strength distribution over the airfoil's surface, and that the first and final neurons learned to produce the same output. These facts are interpreted as the proof that the network somehow learned the underlying physical behaviour of the system.

Still in 2022, Yan et al. [100] proposed a masked gated recurrent neural network to predict the surface pressure distribution on a square cylinder. The proposed model is a combination of two distinct networks that operate in series. The first, a convolutional network, is used to mask out the relevant points of the velocity profile. Then, this masked velocity profile is fed to a GRU network to estimate the pressure distributions. The GRU network was trained using the MSE as a loss function under the Adam algorithm, with a learning rate of 0.001 and a batch size of 20. The data set was constructed using large eddy simulations. The network takes 300 wake velocity values as input and produces 60 pressure points. The lift and drag coefficients can then be derived from the pressure distribution. The trained network was able to predict the mean pressure coefficient around the square cylinder accurately. However, the accuracy of the instantaneous pressure coefficients depends on the location of the sample point, being the precision higher in the cross-flow direction, while the lowest correlation was found at the corners of the cylinder. It is also noted that the computed drag coefficient history shows very large fluctuations, probably caused by the randomness of the drag force.

At the same time, Lan et al. [101] implemented an artificial neural network to estimate the aerodynamic properties of flapping wings. Flapping wings offer a good alternative for MAV's flight. However, simulating a flapping wing with CFD takes a considerable amount of time. Because of this, the authors seek to implement a neural network model that is capable of predicting the aerodynamic characteristics of a flapping wing. Computational fluid dynamics were used to establish a data base for training. The flapping cycle was separated into 250 discrete time steps. In order to reduce the complexity of the ANN and the amount of data, instead of producing the instantaneous aerodynamic coefficients, the set of aerodynamic coefficients was represented by a Fourier series that fits the curve. The network thus only needs to output 11 Fourier coefficients, for a total of 44 outputs. Regarding inputs, the network takes the body oscillation and wing rotation as inputs. The training used Adam as the optimization method and two activation functions were tested, Sigmoid and ReLU. The mean square error (MSE)was adopted as a loss function and the learning rate was set to 0.001. It concluded that both ReLU and Sigmoid models were able to accurately predict the aerodynamic behaviour of the coefficient. The authors hope that ANN based models could be used to complement CFD calculations to accelerate the

development of flight mechanisms and MAVs.

Also in 2022, Ahmed et al. [102] propose the use of neural networks as an alternative to RANS for the aerodynamic analysis of airfoils. The authors focus is on predicting the lift and drag coefficients of airfoils, more specific of the NACA series. A fully connected, multi-layer, feed-forward neural network is presented. The network is fed the airfoil's name, Reynolds number, Mach number and angle of attack. The network featured a single hidden layer and the best performance was found by using 10 hidden neurons. A total of 440 cases were used for training, from Reynolds ranging from 0.5 to 5 million and angles of attack from 0 up to 20 degrees, at a Mach of 0.5. A generalized delta-rule with gradient descent is used for the network's training and its performance is evaluated with both RMSE and Pearson's correlation coefficient. It was concluded that the network was able to accurately estimate the lift and drag coefficients for both the training and validation datasets.

And Moin et al. [14] attempted to use artificial neural networks to predict the aerodynamic coefficients of different airfoils, at several angles of attack, Mach and Reynolds numbers. A data set for 560 NACA 4-digit and 1120 NACA 5-digit airfoil series was constructed, with each airfoil undergoing 315 flight conditions. These airfoil geometries are discretized into 101 points and the lift, drag and moment coefficients obtained. Several multi-layer networks with varying neuron and layer count were trained. The MSE was chosen as a loss function and the Adam optimizer used. The activation function adopted was the ReLU and the network was trained in batches of 128. Results were evaluated in terms of both RMSE and R^2. The authors observe the best accuracy for the NACA 4-digit series airfoils, while the NACA 5-digit series also wielded satisfactory results.

The following year, Mersha et al. [20] experimented with several types of recurrent neural networks to predict the angle of attack of a fighter jet. Simple RNN architecture, LSTM-RNN, GRU, and combinations of LSTM-GRU, RNN-GRU and RNN-LSTM were implemented and tested. The networks feature an input layer, three hidden layers and a linear output layer. A simulation model was used to produce 600 data matrices with a time-step of 0.1s. The data was split into training and validation data with a 60% to 40% ratio. A batch size of 400 was used with a dropout regularization coefficient of 20%. The networks were trained with both RMSprop and Adam as optimizers. It was found that Adam was able to achieve a lower MSE. From the validation, it was seen that all networks could indeed predict the evolution of angle of attack with good precision. From the pure implementations, the standard RNN showed the best results.

Also in, in 2023, Altena et al. [21] attempted to use a recurrent neural network to predict the occurrence of loss of control for quadcopters. Previous studies have relied on the flight envelop to anticipate the occurrence of loss of control. The authors propose a data-driven method, bases on a RNN architecture. More specifically, the authors test LSTM-RNN, a bidirectional LSTM-RNN (BILSTM) GRU and CNN-LSTM networks. The LSTM and GRU are often used when there is a need to process long-term temporal information. Regarding

data, 172 real-world examples are used and the network is feed rotor control, acceleration measures and attitude readings. To assess the trained network's accuracy, the RMSE is used. After training 120 different models, the authors conclude that, for the training set, all models are able to predict flight conditions that will lead to a loss of control event. However, when testing the model for generalization outside of the training set, the result accuracy is slightly lower.

Still in 2023, Pereira et al. [103] proposed the use of a recurrent neural network to predict the aerodynamic coefficients of flapping airfoils. A multi-layer recurrent neural network typology was proposed, with delays for the previous inputs and outputs. The network takes as inputs the Re, k, kh, h and a sliding window of angle of attack, while outputting the instantaneous lift, drag and pitching moment coefficients. The MSE was used as the objective function to be minimized with standard backpropagation through time. The neurons were activated with the hyperbolic tangent. A small data set was built using an implementation of the Hess-Smith Panel Method (HSPM), for a total of 30 training cases and 5 validation examples for a pure plunging airfoil. Training was conducted for 1000 epochs with a learning rate of 1.0. The network showed promising results for the prediction of the lift and pitching moment coefficients, being the latter the first to one to converge to the data. In contrast, the drag coefficient failed to fit the data within the training parameters. It also evident how the delay initialization could pose a problem for the model, being the largest deviation found at the start of flapping cycle.

Then, in 2024, Pereira et al. [104] continued their work on fast aerodynamic prediction using a recurrent neural network. Once again, a recurrent neural network was used, which took the Re, k, h and kh as direct inputs, plus the effective angle of attack as delayed time-varying input. This time the network was made to only predict the lift coefficient, which also feature a feedback loop to the delay layer. Despite reducing the output space, the training domain was expanded in the $k - h$ and α_{eff}, for a total of 240 training and 48 validation examples for a pure plunging airfoil. The hidden layers used the PReLU activation function, while the output layer was linear. Training was conducted for a learning rate of 0.1, with a moment coefficient of 0.75 and a mini-batch size of 25. The trained network was able to predict the evolution of lift coefficient with reasonable accuracy, showing good generalization. The best results were found on the cases located inside the training domain, while the worst results located at the boundary of training domain.

This dissertation joins the works presented in attempting to use an artificial neural network to predict aerodynamic coefficients. Inspired by Suresh et al. [17], Marques et al. [92], Dmitry Ignateyv [93], Balla et al. [13], Zhang et al [19], a recurrent neural network model is proposed. Pereira et al. [103, 104] shows the early tests and attempts of the development of the proposed RNN, which led to the model and results of this dissertation. At the time of presenting, this work has already been followed up by Camacho et al. [105] for predicting loads under dynamic stall. The following chapter will explain the methodology behind the current implementation of the RNN.

This page was intentionally left blank.

Chapter 3

Methodology

Now that the concepts and previous work which lay the foundation of this dissertation have been presented, the methodology followed in this dissertation is shown. This chapter concerns the use of artificial neural networks as a modelling tool, how the proposed network is and how it will be trained and tested. The chapter is divided into sections, the first of which describes the proposed recurrent neural network. The second explains the training algorithm and validation means. The last one presents how the data will be generated and what test conditions the network will be subject to.

3.1 The Neural Network

As mentioned above, this first section describes the proposed neural network and how it will be used to model aerodynamic coefficients. This section is separated into two subsections. The first concerns how artificial neural networks can serve as modelling tools and the underlying mathematical model. The second presents the proposed recurrent network in detail and how it will be used to predict the aerodynamic coefficients. The means by which the network will be trained is presented after.

3.1.1 Neural Networks as Modelling Tools

Artificial neural networks can be used to model possibly any set of data. This is achievable from the way ANNs operate as basically a ”black-box” function with an arbitrary level of complexity. In this subsection, a closer look is taken into how artificial neural network function and can be used to model data.

As previously mentioned, there are several different kinds of artificial neural networks. An artificial neural networks is just a collection of connected artificial neurons. The network’s typology describes the away the artificial neurons are distributed and connected among themselves. Just like their biological counterparts, artificial neural networks required an external stimuli to produce an output from. This external information is the network’s external inputs, which form the input layer of the network. The most simple neural network would just take those inputs and produce an output. The outputting neurons form the output layer of the network. Such a basic network is illustrated in Figure 3.1.

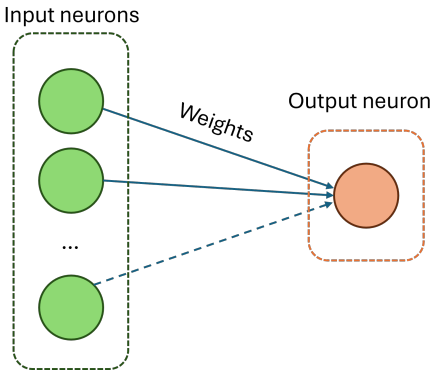

Figure 3.1: Example of a simple, single-output, artificial neural network.

One away to increase the complexity of the network would be to increase the number of inputs and/or the number of output neurons. Another away would be to add an extra layer of neurons between the input and output layers. This new, intermediate, layer is called an hidden layer. The more hidden layers are added, the deeper the network gets.

Regarding the neuron connections, a very straight forward approach would be to simply connect all neurons of a layer with all neurons of the following layer. This is the basis behind feed-forward neural networks (FNNs), like the one shown in Figure 3.2, where a layer's outputs serve as the inputs for the following layer.

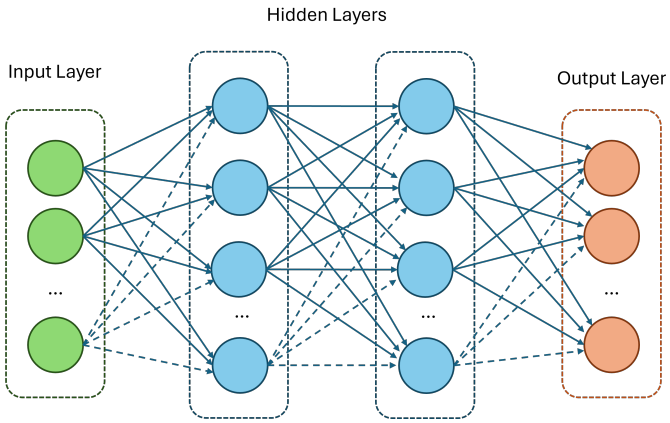

Figure 3.2: Example of a feed-forward, multilayer neural network.

These are general purpose neural networks, good for plenty of tasks. Another kind of neural network is a convolutional neural network (CNN). Convolution describes a mathematical operation in which a matrix filter, the kernel, is used to process information of a given dimension to another. These networks are extensively deployed in imaging processing and language models. They are great for pattern recognition tasks and the inputs often take the form of images. CNNs usually posses a pyramid-like typology, where information gets compressed through the neural network. The inverse is also valid, in which the convolution operation increases the dimension of the input. In this case, the network is said to be a deconvolutional neural network (DCNN). Even still, a common approach is

to combine both neural networks types such that the information first get compressed by an encoder to its main features. Then, the decoder expands those features into the output dimension. Such networks are known as convolutional-deconvolutional neural networks, or just autoencoders, like the one from Figure 3.3.

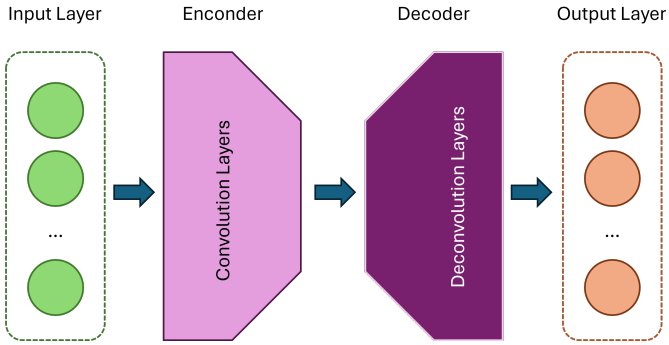

Figure 3.3: Example of an autoencoder type neural network.

It can be the case that the data one pretends to model has a strong dependency of time, such that the previous inputs are required to produce an accurate estimate of the current instant. In this case, a delay layer can be added before the inputs, where the previous values are stored and fed. Such networks are denominated time-delay neural networks. More common however are ANN where the previous outputs are also fed into the delay layer, via a feedback loop. Because in these, the current state is influenced directly by the previous state, they are called recurrent neural networks (RNN), such as the one in Figure 3.4.

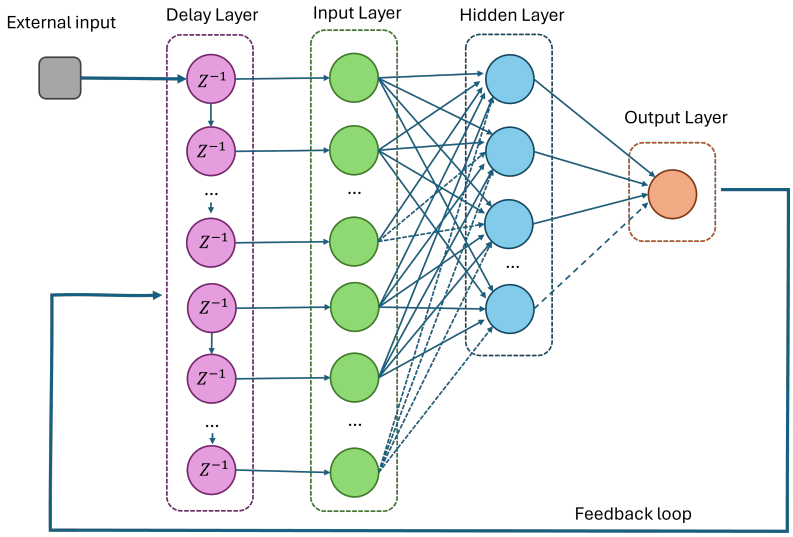

Figure 3.4: Example of a recurrent neural network.

Recurrent neural networks work for most time dependent tasks where the current output only depends on the recent history of the inputs and outputs. RNNs are by nature more unstable and, as the time complexity increases, they start to face difficulty converging.

Long-short term memory (LSTM) recurrent neural networks provide an improvement of dealing with time-dependent data. Instead of external feedback loops, these networks feature an internal memory cell with gates. LSTM-RNNs are actually an improvement made on the gated recurrent unit (GRU) architecture. These have been found to be more stable to train than standard RNNs. Nevertheless, RNNs can still be a good choice for less complex data that does not require a long time-wise context to model.

Under the hood, every ANN is nothing more than a collection of artificial neurons connected to each other. The most basic model of an artificial neuron is the perceptron, which was proposed in 1958 by Frank Rosenblatt [15]. The perceptron is a simple cell that takes in the weighted sum of its inputs and compares it to a threshold. If the weighted sum is greater than the threshold, then it outputs a 1, otherwise the output is 0, as shown in Equation (3.1). This model mimics the way biological neurons get activated or not by a stimulus.

$$output = \begin{cases} 0, & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1, & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}, \tag{3.1}$$

where x_j is the perceptron's j^{th} input and w_j the weight of its connection, as illustrated in Figure 3.5.

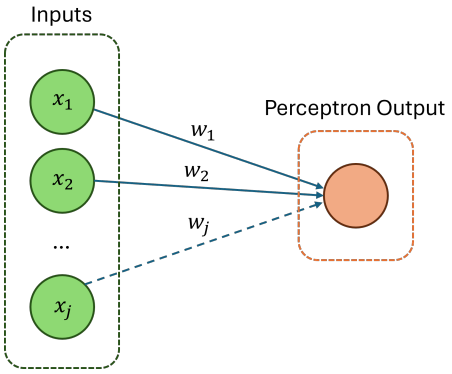

Figure 3.5: Example of a perceptron.

This notation can be improved by making the weighted sum the vectorial product of the input and weight matrices. A new parameter, the bias, can also be introduced, that is the symmetric of the threshold. This away, the perceptron model can be rewritten as:

$$output = \begin{cases} 0, & \text{if } w \cdot x + b_j \leq \text{threshold} \\ 1, & \text{if } w \cdot x + b_j > \text{threshold} \end{cases}, \tag{3.2}$$

where x is the perceptron input vector, w is the weight vector and b_j the bias of the per-

ceptron. respectively.

One way to interpret this model of the artificial neuron is to think of the weights as the neuron's sensitivity to a particular input. The bias, on the other hand, represents that neuron's overall sensitivity to activate. While the perceptron can be used to model binary logic, it has a major drawback. Because of the away a perceptron is modelled, a small change of its inputs can make the output flip. Instead, to ensure proper generalization, an artificial neuron model requires that a small change in the inputs only produces a small change in the output. This lead to the introduction of the sigmoid neuron. Instead of using a threshold, which can be represented as a step function, the output of the sigmoid neuron is the sigmoid function of its weighted sum. While the outputs of perceptron are constrained to be either 0 or 1, the sigmoid neuron can output any real number between 0 and 1. These functions are called the activation functions, since they describe the rules by which the neurons gets activated (produces an output). This functions are also responsible to introduce non-linearity into what otherwise would be a set of linear mathematical operations. With this, a generic artificial neuron model can be described as

$$a_j = \sigma(z_j) = \sigma(w \cdot x + b_j), \tag{3.3}$$

where a represents a neuron's output, $z_j = w \cdot x + b_j$ the weighted input and σ the activation function. This equation can be extended for multilayer neural networks. For a given layer l with j neurons, the activation of any of its neurons can be expressed as

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right), \tag{3.4}$$

where k is any given neuron of the previous layer, w_{jk}^l is the weight of the connection between the k^{th} neuron of the previous layer to the j^{th} neuron of the current layer. Figure 3.6 illustrates the connections to any given neuron of a generic layer.

Equation (3.4) can be further simplified in matrix form as

$$a^l = \sigma(W^l \cdot a^{l-1} + B), \tag{3.5}$$

where a^l is the activation vector of layer (l), a^{l-1} the activation of the previous layer, W^l the layer's weight matrix and B that layer's bias vector. In case of the later following the input layer, $l = 1$, the activation of the previous layer equals the network's external input vector, or $a^0 = x$. Similarly, for the output layer, $l = L$, of the network, the activation of that layer correspond to the network's outputs, or $y = a^L$.

While sigmoid neurons were very common on early neural network implementations, a vast diversity of other activation functions have been developed since. It can be seen that the activation function, for describing the way the output is produced, has direct impact

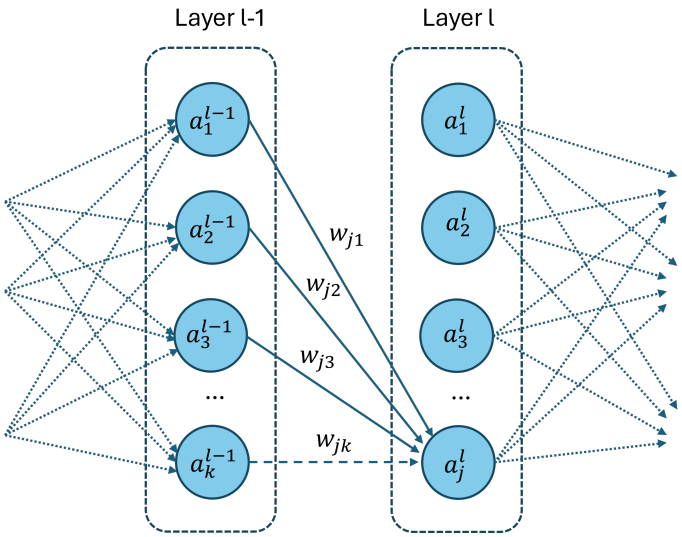

Figure 3.6: Generic position and connections of a neuron in a multi-layer FNN.

on the network's performance. Moreover, this function and its derivatives are a major contributor to the network's complexity and computational cost. While the sigmoid function allows for a smooth gradient, its gradient flattens as the input gets further away from the origin. This facts leads to the vanishing gradient problem. Another drawback is that the sigmoid only produces positive values in a bonded range.

Another popular choice is the hyperbolic tangent. While this function produces negative outputs, it is limited to the $]-1, 1[$ range, which can lead to the vanishing gradient problem. The rectified linear unit (ReLU) was developed in an attempt to fix the vanishing gradient. However, this function gives an output of zero for any negative value, not allowing for gradient information to flow on the negative input range, leading to neuron deactivation. Another problem with the ReLU is the fact that its derivative is undefined for $x = 0$. The leaky parametric rectified linear unit solves this by introducing a small slope on the negative branch, set a 0.01. The parametric rectified linear unit (PReLU) makes the negative branch's slope a parameter. Despite fixing the vanishing gradient problem, these unbounded functions are prone to suffer from exploding gradient. The exponential linear unit (ELU) was introduced as a smoother version of the ReLU. Other functions like Softplus and HardELiSH have also been proposed. A more detailed reference on activations functions has been provided by Basirat et al. in [79].

Despite its importance, the activation function is only partly responsible for an ANNs ability to model any function. In order to learn and produce the useful outputs, the weights and biases of the network need to be carefully tuned. This tuning of the network parameters is achieved from the network's training. The most common way artificial neural networks learn is by known examples, which means ANNs are data-driven methods, requiring data to be trained on. One way to teach an ANN is to subject it to a supervised learning task. In this approach, a data set of labelled data, which is data for each the

inputs and corresponding outputs are know, is prepared. The network is then presented with an input and tries to produce an output, which is compared to the target output. That comparison can be used in the training algorithm to adjust the network's weights and biases so that, with time, the network's outputs will approach the desired values. Another way to train an ANN is to proceed with an unsupervised learning task. In this case, the data as not been previously classified, so the network must find the hidden patterns and relations within the data, in order to classify it.

The process by which the weights and biases of the network are automatically adjusted is the learning rule. The most basic learning rule is based on the idea that if two neurons fire together, then their connection should be reinforced. This idea was proposed by Donald Hebb [77] in 1949. Its mathematical model was already presented in the previous chapter, in equation (2.12). The weight change can then be described in terms of the network's error by the Widrow-Hoff rule. This formula, also called the delta-rule, given in equation (2.13).

This rule applies for single layer neural networks. It is easy to define the network error for a single layer, by just comparing the outputs. On multi-layer neural networks, the error of the hidden layers must also be defined. The concept of back propagation emerges here, where the error information from the output layer is propagated backwards across the remaining layers of the neural network. The backpropagation algorithm was first introduced in the 1970s and 1980s and its most simple form relies on gradient descent. Gradient descent is a first-order optimization method in which one moves in the opposite direction of a function's gradient. For this, a cost function, J, is associated to the neural network's performance. This function is sometimes also called the objective function since the objective of training is to minimize its value. The gradient is used to define the rate of change of the cost function with respect to any weight or bias of the network. With this, one can optimize the weights and bias by descending along the gradient's slope towards a minimum. Following this, one can define the error of any neuron δ_j^l as the derivative of the cost with respect to that neuron's weighted input as

$$\delta_j^l = \frac{\partial J}{\partial z_j^l}. \tag{3.6}$$

By applying this equation to the output layer, the error of that layer is given by

$$\delta_j^L = \frac{\partial J}{\partial a_j^L}\sigma'(z_j^L), \tag{3.7}$$

where σ' is the first derivative of the objective function.

The quadratic cost is one of the most commonly used objective functions. The quadratic

cost is defined as

$$J = \frac{1}{2}\sum_{i=1}^{n} |y(i) - a^L(i)|^2, \tag{3.8}$$

where n is the total number of training examples run before the optimization. When using the quadratic cost, the gradient of the cost with respected to the activation is given as $\partial J/\partial a^L = (a^L - y)$. With this, the equation (3.7) can be formulated in matrix terms as

$$\delta_L = (a^L - y) \odot \sigma'(z^L), \tag{3.9}$$

in which $\odot$ describes element-wise multiplication, also known as the Hadamard product. The error of the output can then be propagated to find the error of the previous layer. This process is called error injection. Recursively, it is possible to define the error of any given layer in terms of the error of the previous layer as

$$\delta^l = ((W^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l), \tag{3.10}$$

where $(W^{l+1})^T$ is the transpose of the weights matrix of the following layer.

With the errors defined across the network, the rate of change of the weights and biases can be found in a similar way. In particular, the rate of change of the cost with respect to any weight can be given by

$$\frac{\partial J}{\partial w^l_{jk}} = a^{(l-1)}_k \delta^l_j, \tag{3.11}$$

or simply $\frac{\partial J}{\partial w} = a_{in}\delta_{out}$. And the gradient of the cost with respect to the bias is

$$\frac{\partial J}{\partial b^l_j} = \delta^l_j. \tag{3.12}$$

Equations (3.9), (3.10), (3.11) and (3.12) are the basis of a gradient-descent optimizer based algorithms.

However, calculating the true gradient of the cost function in parameter space for a large data set on a high-dimensional space can become difficult and computational expensive. Because of this, an alternative to gradient descent, stochastic gradient descent (SGD) was introduced to optimize the cost function. Instead of computing the true gradient for entire data set and then adjusting the parameters accordingly, SGD computes a local gradient for each individual training sample. Then, instead of taking a full step down the gradient, a learning rate is applied to it before updating the network's parameters. The learning rate translates the degree of confidence on that gradient's direction towards a minimum. When paired with a proper learning rate, is proven to eventually converge to a minimum,

global or local. The problem with this approach is that the gradient of the next data sample can point in the opposite direction, causing the gradient descent to take an indirect, oscillating path towards the minimum. This effect slows down the convergence.

For this reasons, a compromise between true gradient descent and stochastic gradient descent has been proposed. This method takes the computational efficiency of SGD by computing an ”averaged” gradient for a small subset of the complete data set, while improving the convergence rate by approximating better the direction of the true gradient. Since the gradient is computed for a batch of data samples, this method is called batch gradient descent. Moreover, many works have concluded that the use of mini-batch gradient descent, in which the batch size is kept small, can improve training performance. Although batch gradient descent promises a smoother, faster convergence of the cost function, it does not guarantee that the function will actually reach a minimum. Since the gradient is always an approximate of the true gradient, the value of the cost function can end up oscillating around the actual minimum. Another problem with both SGD and batch gradient descent is that, depending on the learning rate, there is the possibility that the gradient descent gets trapped in a local minimum.

While the methods for backpropagation presented so far have been described from the perspective of a feed-forward neural network, it is also possible to apply them to a recurrent neural network. In order to apply backpropagation the RNN needs to be unrolled in time, creating a long series of connected feed-forward neural networks. In this network, the feedback loops have been flattened to connections from a network to another, like represented in Figure 3.7. The backpropagation algorithm can then be applied to the unfolded RNN by starting the backpropagation at the last time step, feeding backwards across the network, and then injecting the error into the previous time step. A drawback of this approach is that the time-history of the errors and activations of the neural network needs to be kept in memory, instead of just saving an instant at a time.

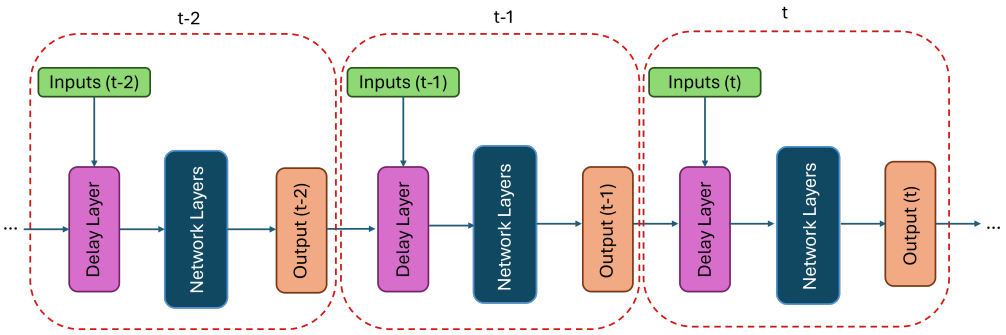

Figure 3.7: Scheme of a recurrent neural network unrolled in time.

A major problem faced by neural network is over-fitting. Modern neural networks with many weights and bias are specially prone to it. Over-fitting occurs when the ANN stops learning to model the data set, but instead starts to learn the peculiarities and variations of the data. It is characterized by a state in which the neural network achieves near 100%

accuracy on the training data, while the validation accuracy remains capped at an inferior value. This means that the ANN is memorizing the data and not learning from it. One way to help preventing over-fitting is use large amounts of data. Another simple way to deal with this is to just stop training once the classification accuracy on the validation data saturates. This method is known as early-stopping.

Regularization is another strategy that was introduced to reduce over-fitting and increase network accuracy. L2 regularization, or weight decay, is a regularization scheme that adapts the learning rule so that the network prioritizes learning small weights over large ones. This achieved by introducing a weight decay coefficient which controls how much the network should optimize the weights vs optimizing the cost. This works by making the networks evolve smaller weights, which as a consequence make it less sensitive to noise in the data. A similar approach is taken by L1 regularization, in which the cost function is directly penalized by the magnitude of the weights.

In contrast, dropout does not affect the cost function. Instead, neurons are randomly excluded from training. This strategy reduces the inter-dependency between neurons, since a given neuron cannot rely on the presence of a particular one. Because of this, the network is forced to learn more general features which can be used with a variety of subsets of neurons.

Another way to improve training of neural networks is to introduce moment. By applying a moment coefficient which interpolates between the new suggested parameters and the previous ones, the gradient descent can be made to keep moving in the previous direction. This can help smooth oscillations of the objective function and accelerate convergence. This coefficient is usually kept at 0.9. A more advanced form of moment is Nestereov's accelerated gradient (NAG). NAG works by using the previous gradient to estimate where the next parameter update should fall. It then calculates the new gradient with respect to the target parameters.

Furthermore, training can also be enhanced by implementing a form of adaptive learning rate, which tunes itself to the rate of descent of the objective function. Adagrad is one of such optimization scheme which adapts the learning rate to the sparsity of the parameters. One problem with Adagrad is the fact that in order to calculate the new learning rate, it adds the square gradients as denominator. This causes the learning rate to become increasingly small, and thus, eventually halting the network's ability to learn. Adadelta is an improved version of Adagrad which fixes that problem by taking a decaying average of past gradients, instead of summing all past squared gradients. Another popular choice of optimizer is ADAM, or adaptive moment estimation. ADAM relies on both the bast square gradient and gradients. These methods have already been presented in Section 2.2.2.

Other improvements that can be made to training include randomly shuffling the data after each epoch. Another option is to normalize all the network weights and biases to a zero-mean and unit variance after each batch.

3.1.2 Proposed Neural Network

Now that the principles of neural networks have been explained, it is time to present the proposed RNN. The major goal of the present dissertation is to implement an artificial neural network to predict the evolution of aerodynamic coefficients of a flapping airfoil. The neural network structure used follows the proposed model in Pereira et al. in references [103, 104]. A recurrent neural network typology was chosen for its ability to process temporal information while remaining relatively simple to implement. RNNs have also already been extensively used and proven that can be a robust method, despite harder to train. The idea is to build the foundation for a platform that could be used for real-time optimization and control applications.

Any given neural network can be abstracted to a *black-box* type function, which takes a set of inputs and produces a corresponding output. Analogous to a chef cooking a recipe that takes some ingredients and transforms them into a dish. The networks recipe is its set of weights and biases. In order to set up the neural network model, the first task is to define what its inputs and outputs will be. Since the goal is to model flapping aerodynamics, the aerodynamic coefficients are defined as a function of the relevant unsteady aerodynamic parameters:

- Reynolds number, Re;
- Effective angle of attack, α_{eff};
- Pitching amplitude, A_α;
- Non-dimensional frequency, k;
- Non-dimensional amplitude, h;
- Non-dimensional velocity, kh;

These inputs can be grouped into two sets, one that remains fixed in time, and another that varies with time, entering the neural network via its delay layer. The Reynolds number characterizes the flow condition, while A_α, k, h and kh describe the flapping kinematics of the airfoil. This first set of five parameters form the direct inputs of the neural network. Instead of running all the inputs through the delay layer, these inputs bypass the delay nodes and are feed directly into the RNN. In contrast, the effective angle of attack is the only time-varying external input, which goes through the delay layer. It informs about the current position of the airfoil in the flapping cycle. This flapping cycle was broken down into N discrete time steps. The presence of the delay layer allows for a reduction of input neurons, and thus simplifies the model. This comes at the expense of longer computational time, when compared to having a FNN computing the whole cycle at once.

Regarding the outputs, the RNN is going to predict the instantaneous lift, C_l and pitching moment coefficient C_m. These coefficients are then carried by the feedback loops back

into the delay layer, from where they can be used as inputs for the subsequent time steps. Together with the effective angle of attack, the previous aerodynamic coefficients form the time-dependent input of the RNN. At each time set, the network is feed a sliding window of $q+1$ points of α_{eff} and p points of each previous C_l and C_m. With this, the network's external input vector at any time step t can be defined as

$$x_{ext}(t) = [Re, A, k, h, kh, \alpha_{\text{eff}}(t), \alpha_{\text{eff}}(t-1), \alpha_{\text{eff}}(t-2), ..., \alpha_{\text{eff}}(t-q)], \tag{3.13}$$

while the output from the delay nodes Z^{-1}, which will be fed into the network at each time step t, is given as

$$\begin{aligned} Z^{-1}(t) = [&\alpha_{\text{eff}}(t), \alpha_{\text{eff}}(t-1), ..., \alpha_{\text{eff}}(t-q), \\ &C_l(t-1), C_l(t-2), ..., C_l(t-p), \\ &C_m(t-1), C_m(t-2), ..., C_m(t-p)]. \end{aligned} \tag{3.14}$$

The reason behind feeding the previous outputs back to the network is to provide the RNN with information about the flapping cycle and how each of the aerodynamic coefficients are evolving. This should improve the network's performance and aid with model generalization. This vector is then combined with the fixed inputs, forms the RNN input vector

$$x_{net}(t) = [Re, A_\alpha, k, h, kh, Z^{-1}(t)]. \tag{3.15}$$

And the RNN output vector at each time step t is simply

$$y(t) = [C_l(t), C_m(t)] \tag{3.16}$$

Finally, the aerodynamic coefficients can be modelled as a function of the network's inputs and its weights and bias. This is the function that the RNN attempts to learn as is given as

$$y(t) = f(Re, A_\alpha, k, h, kh, Z^{-1}(t), W, B) \tag{3.17}$$

where W and B represent all of recurrent neural network's weights and bias.

In the present work, the delay layer can store $3+1$ points of the effective angle of attack, and 4 points of each previous aerodynamic coefficient. A total of four points was chosen for the delays as one point corresponds to the instantaneous value, two points can inform about the *velocity*, or the rate of chance of the value, three points correspond to the *acceleration*, or the rate of change of the *velocity*, and four points can inform about the rate of change of the *acceleration*. It is thought that this should be able to fully capture the dynamics of how the system is evolving. This gives for a total of $4 \times 1 + 4 \times 2 = 12$ delay nodes (four for the effective angle of attack, four for the lift coefficient and another four for the pitching moment coefficient).

The network itself features a total of 7 layers, a delay layer, an input layer, 3 hidden layers and the output layer. The input layer features 17 input nodes, 5 for each of the fixed parameters plus the 12 inputs from the delay layer. The hidden layers have 28, 28 and 12 neurons each and the output layer contains 2 output neurons, one for each of the aerodynamic coefficients. The hidden layer are activated with a non-linear activation function while the output layers use a linear activation. The reason for this is so that the output is not bonded by the domain of the activation function, since this layer is responsible for estimating he actual aerodynamic coefficient. It is also to prevent it from getting trapped in one of the branches of the activation function.

The activation function adopted was the Leaky ReLU since it is already a very well tested activation function in both RNNs and other ANN applications. This function is also very easy to implement. Both the function and its first derivative are also very fast to compute, which is specially important since it will be run for every single neuron of the network at each time. The Leaky ReLU is defined as

$$\sigma(x) = \begin{cases} \lambda x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}, \tag{3.18}$$

where λ is the slope of the negative branch, set to 0.01 by default. Its first derivative is given by

$$\sigma'(x) = \begin{cases} \lambda & \text{if x} < 0 \\ 1 & \text{if x} \geq 0 \end{cases}. \tag{3.19}$$

This representation of this function can be seen in Figure 3.9. It is possible to implement it in code by making use of the use of the *max()* operator, making it very fast to compute. In this case, the Leaky ReLU is simply given as

$$\sigma(x) = max(x, \lambda x). \tag{3.20}$$

Figure 3.8 illustrates more clearly the structure of the proposed RNN, as described so far.

One of goals of the present work is to build a framework that can be easily adapted and extended for other test conditions. Because of this, the implementation of the RNN had to be very modular, flexible and robust. The algorithm for the RNN itself was developed in the Fortran programming language, and is compartmentalized into 4 main modules. One module handles everything related to the neural network, including its definition and procedures required for its operation. Another module includes aggregates all procedures related to the network's initialization. The third module deals with reading and writing files and data formatting. The final module controls the training process and defines the backpropagation algorithm.

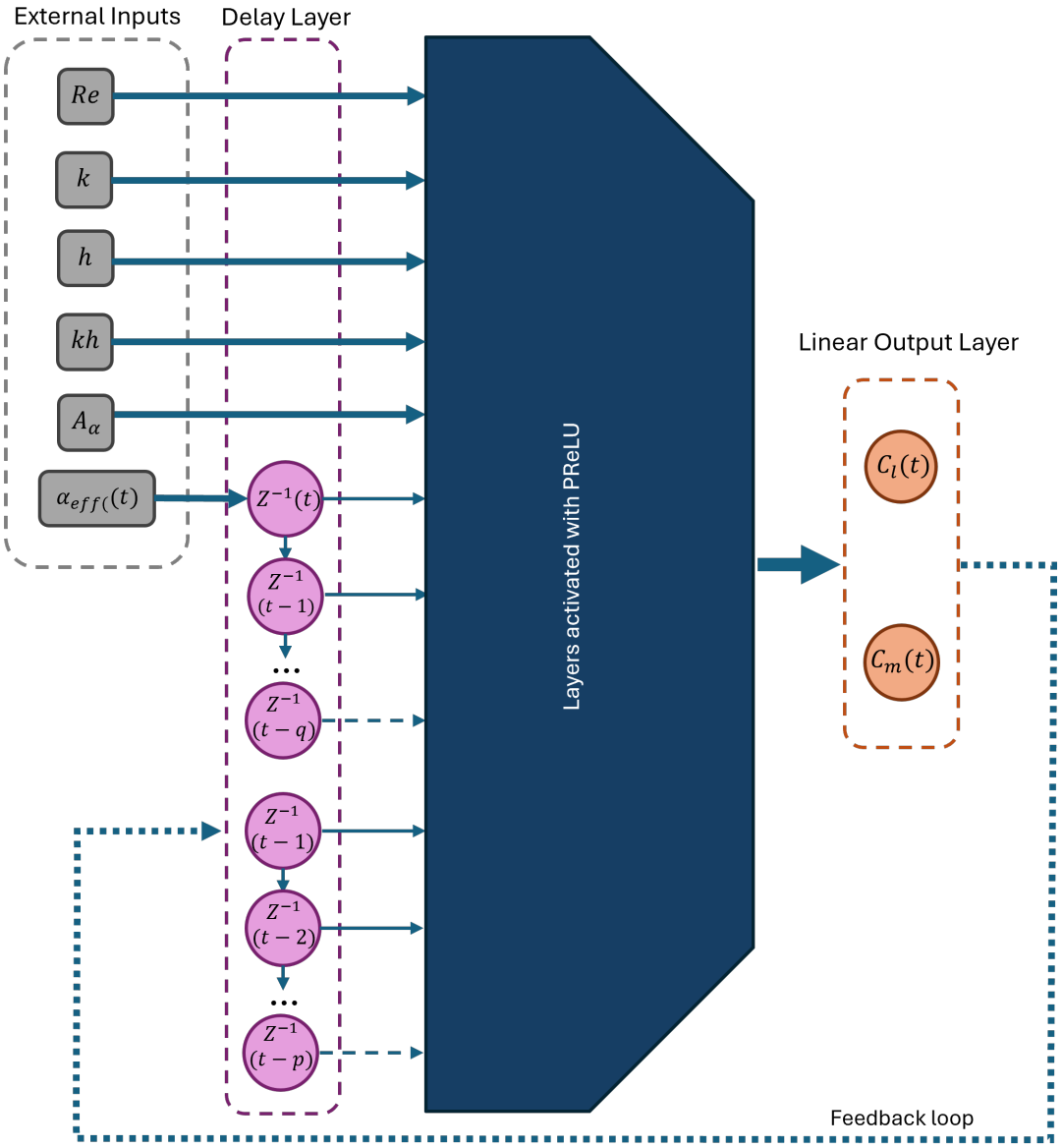

Figure 3.8: Scheme of the proposed recurrent neural network.

As mentioned, the present neural network features two types of neuron layers: linear and non-linear layer, with linear and non-linear neurons, respectively. The non-linear neuron layer follows the formulation already given in (3.5). The linear layer, which corresponds to the output layer, simply outputs the weighted sum $a^L = w^L \cdot a^{L-1} + b^L$, producing the estimate of the aerodynamic coefficients.

Each flapping period is divided into discrete time-steps, which are run in sequence in the RNN. Each time-step starts with an update of the delay nodes, in which each stored values moves forwards one delay node. The values in the last nodes are simply discarded. This is followed by in the introduction of the current time-dependent input and previous aerodynamic coefficients into the corresponding first nodes of the respective set of delays. This procedure is demonstrated in Algorithm 1.

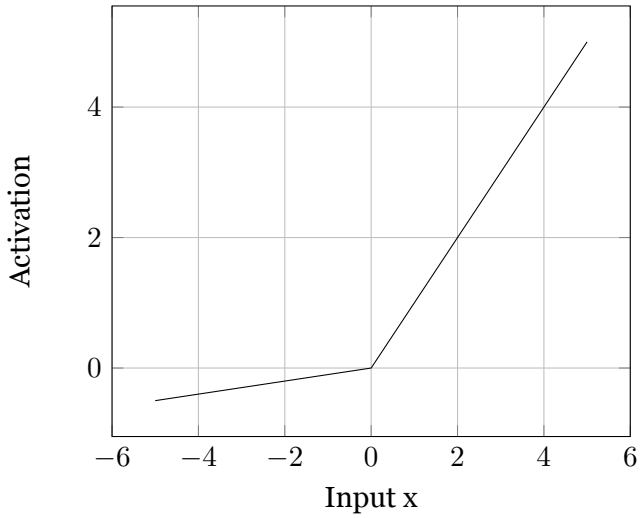

Figure 3.9: Leaky ReLU activation function for $\lambda = 0.1$, where the slope of the negative branch can be seen.

Algorithm 1: Pseudo-code for the update of the delay nodes.

for each delay set in delay layer **do**
  for delay node $Z_i^{-1}(t)$ in delay set **do**
    Push contents of delay node to the next
    $Z(t-1)_i^{-1} = Z(t)_i^{-1}$
  Now update the first input nodes
  if i is input delay **then**
    Load the input value corresponding to that time step
    $Z_{\alpha_{\text{eff}}}^{-1}(1) = \alpha_{\text{eff}}(t)$
  else
    Load the corresponding output value from the previous time step
    $Z_{Cl}^{-1}(1) = C_{l,net}(t-1)$
    $Z_{Cm}^{-1}(1) = C_{m,net}(t-1)$

Once the delay layer is updated, the input vector is constructed according to Equation (3.15). Each layer of neural network is defined recursively so that its input calls for the output of the previous layer. When the recursion reaches the first layer, the input vector is loaded. This means that only the function for the final layer needs to be called in order to run the full network. The procedure to run a single time step can be seen in Algorithm 2.

Algorithm 2: Pseudo-code for neural network layer.

if last layer **then**
  return $a^L = W^L \cdot a^{L-1} + B^L$
else if first layer **then**
  return $a^1 = \sigma(W^1 \cdot x(t) + B^1)$
else
  return $a^l = \sigma(W^l a^{l-1} + B^l)$

With this it is possible to compute the entire flapping cycle as shown in Algorithm 3.

Algorithm 3: Pseudo-code for flapping cycle.

for each time step t in flapping cycle **do**
  Update the delay layer Z^{-1}
  Create input vector $x_{net}(t) = [Re, A_\alpha, k, h, kh, Z^{-1}(t)]$
  Compute network outputs by calling the layer function
  $y(t) = a^L(t)$

The RNN algorithm starts by allocating memory and initializing the neural network according to the parameters defined. The network parameter module stores all the information required to create the neural network, such as the number of layers and how many neurons are in each one. What the inputs and outputs are, if they periodic or a fixed value. It also defines the delay nodes, how many points and what inputs/outputs it loads from.

One the network has been created, the weights, biases and delays must be initialized. There are many ways to initialize the weights and biases, but the chosen method is to set each weight and bias to a random number within a range, in this case set between $-0.1, 0.1$. The input delay nodes are easy to initialize, since the input is know and periodic, it is possible to just load the points starting from the end of the flapping cycle.

However, the initialization of the output delays poses a challenge, since it will impact the initial estimates of the neural network. Because the initial points of the delay do not correspond to actual aerodynamic coefficients, the prediction of the network is impacted. This can result in large deviations of the first estimates, as was seen by Pereira et al. [103]. The proposed solution, which is tested in the present work, is to emulate the real life operation of the system. When booting, the program starts with all delays set to zero, which would get loaded with data as the first iterations are run. The hope is that after a certain point the estimates stabilize and the system is considered to be operating normally. In order to simulate this start-up period, the network can be run for a given amount of extra time steps, η, before the actual estimate starts. During this time period the initial predictions should improve as the delay nodes get loaded with better estimates.

Furthermore, in order to help with training and network performance, the external inputs are normalized before entering the network. For this, all direct external inputs are divided by a fixed value. This value was set as the largest magnitude of the test examples, but they can be of any value as long as the neural network is trained with under that condition. Those values are stored in a separate file, together with the training examples. The goal of the normalization is to bring the external inputs closer to the -1 to $+1$ range. The values used for the normalization of the fixed inputs can be seen in Table 3.1.

The option to normalize the effective angle of attack by 2π was also implemented but not used in the present work.

Table 3.1: Parameters used for normalization of the fixed inputs.

Input	Re [–]	k [–]	h [–]	kh [–]
Parameter	1000	0.175	3.0	0.525

3.2 Network Training and Validation

So far the model by which the RNN will be used to predict the aerodynamic coefficients has been presented. In this section the training process which allows the RNN to effectively learn to predict the data is discussed. After that, the methods by which the network's predictions are evaluated are addressed.

3.2.1 The Training Algorithm

As mentioned, this section presents the training algorithm which is used to teach the RNN. RNNs are data-driven modelling methods that learn from examples. The training algorithm defines how the examples are presented to the network and how these are used to adjust the weights and biases of the network. The goal of the training process is to produce the set of weights and biases that provides the best mapping between the inputs and outputs. The algorithms presented for training are based on those described by Michael Nielsen [106] on his 2015 e-book.

An artificial neural network's training consists of a series of learning tasks. Learning can be either supervised or unsupervised. In supervised learning tasks the neural network is presented with labelled set of data, which is data in which the desired outputs for each inputs are known. The ANN then attempts to produce a prediction from the inputs, which is compared to its corresponding output. From this comparison the weights and bias are updated according to the learning rule, with hopes of closing the gap between the network and target outputs. In contrast, during unsupervised learning tasks, the desired outputs are not know, and it is up to the ANN to classify the data and unveil the underlying relationships. In the present dissertation a RNN is to be taught to estimate aerodynamic coefficients, which are known, from their respective inputs. Because it that, supervised learning was used.

Since both the network's and target outputs are known, it is possible to define the deviation between the two as

$$e(t) = r(t) - y(t), \tag{3.21}$$

in which $r(t)$ and $y(t)$ represent the desired and predicted outputs at any given time step t. From this, the total error of the neural network over a training period $[1, \tau]$ can be

expressed as

$$J^{total} = \frac{1}{2}\sum_{t=1}^{\tau} e(t)^2, \tag{3.22}$$

where J^{total} is the analogous of the linear quadratic cost function for this non-linear application. The objective of the training process is to find the weights and biases that minimize this cost function. Because of this, this function is also called the objective function. A lower value of this function implies a better approximation between the predicted and target outputs.

There are many methods to optimize this function in parameter space. The method chosen for this work is a variation of stochastic gradient descent, mini-batch gradient descent. In mini-batch gradient descent, the network is presented with a set of training examples, a mini-batch. Only after the mini-batch are the weights and biases updated, before a new batch is started. The gradients of the quadratic cost function with respect to parameter space have already been presented in equations (3.9), (3.10), (3.11) and (3.12). From the gradients it is possible to establish the learning rule, which describes the way the updates and biases are updated. For mini-batch gradient descent the update rule is as follows

$$W^l_{sugg} = W^l_{prev} - \frac{\eta}{mN}\sum_{i}^{mN} \delta(i)^l (a(i)^{l-1})^T \tag{3.23}$$

$$B^l_{sugg} = B^l_{prev} - \frac{\eta}{mN}\sum_{i}^{mN} \delta(i)^l, \tag{3.24}$$

where m is the mini-batch size and N is the number of data points (time steps) in one training example, so that the product mN equals the total number of data points presented to the network in a batch. The learning rate is represented by η, while W^l_{prev} and B^l_{prev} are the current weights and biases of layer l, respectively. The current suggestion for new weights and biases are represented by W^l_{sugg} and B^l_{sugg}. For the basic gradient descent methods, these suggested weights and biases would correspond to the new set of parameters. However, in order to improve training accuracy, moment was also introduced. Moment functions by setting the new weights and biases as a interpolation between the current and suggested parameters as

$$W^l_{new} = \beta W^l_{prev} + (1-\beta) W^l_{sugg} \tag{3.25}$$

$$B^l_{new} = \beta B^l_{prev} + (1-\beta) B^l_{sugg}, \tag{3.26}$$

where β is the moment coefficient, while W^l_{new} and B^l_{new} are the new weights and biases, respectively. The mini-batch size and moment coefficients used in the present work are

presented in Table 3.2.

Table 3.2: Fixed hyper-parameters used for training.

Mini-batch size m	Momentum coefficient β
30	0.5

With the learning rule established, it is time to define the method by which the output errors will be used to update all layers across the neural network. This is achieved by a process denominated backpropagation. It is during this process that the errors from the output layer transmitted across the remaining layers of neural network. It starts by calculating the deviation between the network's output and target value, using equation (3.21). This value is then used in equation (3.9) in order to produce the error of the output layer. This error can then be propagated across the remaining layers using equation (3.10). These errors are then used to compute the weight and biases changes according to equations (3.23) to (3.26).

In order to make this process more efficient, the deviations and derivative terms of equations (3.9) and (3.10) are computed and stored at each time step. Only at the end of the full flapping cycle are these used compute the actual errors. Then, because the parameter updates only happen after a batch, the errors are used to compute the sum terms of equations (3.25) and (3.24). This requires that the input vectors of each layer for each time step are stored in memory. These sums are accumulated at the end of each flapping cycle until the full batch is finished. The backpropagation process at the end of flapping cycle follows Algorithm 4.

Algorithm 4: Pseudo-code for backpropagation from [106].

for each time step in a flapping cycle **do**
  for each layer of the network up to the first hidden one **do**
    if last layer **then**
      calculate error of last layer $\delta^l = ((W^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$
    else
      calculate error of the layer $\delta^l = ((W^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$
    update the sums for that layer
    $\sum W^l = \sum W^l + \delta(i)^l (a(i)^{l-1})^T$
    $\sum B^l = \sum B^l + \delta(i)^l$

Once the batch ends, the suggested parameters are calculated and the weights and biases updated according to equations (3.25) and (3.26). This parameter update process follows Algorithm 5.

Each mini-batch consists of a sample of randomly chosen training examples from the data set. Then the network is presented with each case, which gets run from the start to the

Algorithm 5: Pseudo-code for parameter update from [106].

for each layer of the network up to the first hidden one **do**
compute the suggested weights and bias from the sums
$W^l_{sugg} = W^l_{prev} - \frac{\eta}{mN} \sum W^l$
$B^l_{sugg} = B^l_{prev} - \frac{\eta}{mN} \sum B^l$
compute the new weight and biases matrices
$W^l_{new} = \beta W^l_{prev} + (1 - \beta) W^l_{sugg}$
$B^l_{new} = \beta B^l_{prev} + (1 - \beta) B^l_{sugg}$

end of the flapping cycle, while the deviations and derivatives are calculated as described. The complete training sequence can be seen in Algorithm 6.

Algorithm 6: Pseudo-code for training.

Initialize neural network
for training iterations **do**
Randomly set a batch from the training data set
for flapping cycle in batch **do**
Run flapping cycle
Run the backpropagation algorithm
Perform the parameter update

Every once in a while the value of the objective function is sampled after updating the network parameters. Finally, at the end of training, a validation step is performed to assess the network's performance, as presented in the next section.

3.2.2 Network Validation

Previously, the training algorithm that allows the neural network to learn was presented. However, the convergence of the objective function alone does not ensure that RNN will produce good estimates. A validation stage is required in order to analyse the reached minimum and the generalization of the network. In the present section the process for the validation of the trained RNN is explained.

Starting with the generalization of the model, it can be easily tested with the used of two distinct sets of data. The first, the training set, is used to train the neural network, while the second is used for validation. The validation set must be made of cases different from the ones presented during training. This is important to ensure that the neural network has learnt to model the system and not just memorized the training data. This evaluation is performed once the training is complete, by running the trained network over the validation set.

Regarding the performance of the neural network itself, it can be tested using two metrics. These are the mean squared error (MSE)and the coefficient of determination (R^2). The MSE measures the average loss over the empirical data. It is a positive value that relates

the quality of an estimator with its proximity to zero. Mathematically is defined the average square difference between the and network outputs and the target values, as shown in equation (3.27).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(r_i - y_i)^2, \tag{3.27}$$

where n is the total number of data points presented, r_i is the target value and y_i is the network's prediction.

The coefficient of determination measures how good any given models fits a set of data. It describes the proportion of the variance present in the dependent variable that is predicted by the independent variable. Usually, it ranges from -1 to 1, being 1 a model that perfectly fits the observed data. It is defined in equation (3.28).

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}, \tag{3.28}$$

where SS_{res} is the residual sum of squares and SS_{tot} is the total sum of squares. The residual sum of squares is given as

$$SS_{res} = \sum_{i=1}^{n}(r_i - y_i)^2, \tag{3.29}$$

where the term $(r_i - y_i)$ corresponds to the residuals, while the total sum of squares is given by

$$SS_{tot} = \sum_{i}^{n}(r_i - \bar{y})^2, \tag{3.30}$$

being $\bar{y} = \frac{1}{n}\sum_{i}^{n} r_i$ the mean value of the target data.

The coefficient of determination is calculated for a flapping cycle as prescribed above, by setting $n = N$. The quality of the entire set, either training or validation, is expressed as the average $\bar{R^2}$ of all flapping cycles in it.

$$\bar{R^2} = \frac{1}{n_{cases}}\sum_{i=1}^{n_{cases}} R_i^2 \tag{3.31}$$

With these two metrics it is possible to evaluate the performance of the trained networks and assess the quality of the model.

3.3 Data Generation

As seen, the RNN is trained and tested using sets of data. In fact, artificial neural networks consist of data-driving models, which require large amounts of data to learn from. In this section the data generation methods are presented, together with the sets of test conditions that the network will be trained on.

3.3.1 Aerodynamic Data from Panel Method

In order to train the neural network, a set of cases has been generated for a pure plunging airfoil. Initially, it was intended to produce a data set with thousands of test cases, by including variations of Re, k, h, kh, A_α, and conditions of plunging, pitching and flapping. For this reason, a Unsteady Panel Method (UPM) approach was chosen as it would allow for the generation large quantities of data in a promptly manner, specially when compared to experimental or CFD methods.

Panel methods consist of computer codes developed to numerically solve the Prandtle-Glauert equation for linear, inviscid, irrotational flows. This equation has three known analytical solutions in the form of singularities: the source, doublet and vorticity. Surface distributions of these elements can be imposed to create a discrete representation of an aerodynamic surface, which is composed of so-called panels. Depending on the type of function used to describe the strength of the singularity distributions, the codes can be of high or low order. Low order methods, which use constant-strength distributions, were developed first. These worked for incompressible, subsonic flows.

Teng [10] extended the existing Hess-Smith Panel Method (HSPM), which was used for steady, potential flow problems, to inviscid, incompressible flows over airfoils under unsteady motions. This was achieved by imposing a vortex-shedding process into the flow wake.

The chosen panel method is an adaptation of the Hess-Smith Panel Method (HSPM) for unsteady flows proposed by Teng [10]. This UPM code was implemented and validated by Camacho et al. [107] in a collaboration between the Aeronautics Research Center (UBI) and LabDin (EESC-USP). This coded can be consulted at https://github.com/Emanuel96/PanAir-IK30. The mathematical formulations and considerations are also present in the repository.

This code represents the airfoil surface as a discrete set of panels with source singularities at each panel and a varying vorticity over the entire surface. In order to ensure Helmholtz and Kelvin's circulation theorems, the total circulation of the flow field is preserved by shedding vortices at the trailing edge. This is done by attaching an extra panel to the trailing edge with uniform vorticity. The length and orientation of this panel takes into account the assumptions of Basu and Hancock [33]. This vortex is shed into the wake and

convected as a concentrated free vortex. Furthermore, the Kutta condition must also be imposed. Unlike the steady case, the potential on the trailing edge must be included when enforcing the Kutta condition. The numerical solution takes three stages. The first calculates influence coefficients, then the Kutta condition is forced and pressure distribution is calculated at last.

As many reduced-order and numerical methods, panel methods have some limitations. First of all, panel methods are made to solve the Prandtl-Glauert equation, which is derived from the Navier-Stokes equations by neglecting all the viscous and heat-transfer terms. This solution also considers the flow to be irrotational. Another consequence of not including viscous effects is that Kutta condition must be artificially imposed on the trailing edge. Because of these assumptions, panel methods do not account for flow separation, skin-friction drag and transonic shock. Still, induced drag can be computed by panel methods. Nevertheless, the lift and pitching moment coefficients can be accurately calculated from the pressure distributions for a wide range of flows.

For this reason, the neural network is only trained to predict the lift and pitching moment coefficients, while the drag coefficient is excluded from training. Still, the implementation of the neural network allows for easy configuration of inputs and outputs. The flapping motion is assumed to be a sinusoidal motion, such that the plunging component is given by

$$y_{\text{pos}} = A' \cos(2\pi f t), \tag{3.32}$$

where y_{pos} is the airfoil's vertical position, A' is the plunging amplitude and f is the plunging frequency. And the pitching motion is expressed as

$$\alpha = A_\alpha \cos(2\pi f t + \phi), \tag{3.33}$$

where α is the airfoil's angle of attack, A_α is the pitching amplitude and ϕ is the phase angle between the pitching and plunging motion.

3.3.2 Test Conditions

As mentioned, the neural network is trained with a plunging data set for the prediction of the lift and pitching moment coefficients. This set is further divided into a training and validation subsets. Now, the flow conditions that form those sets are listed, together with the training set-ups to which the network is subject to.

All flow cases have been simulated using UPM for the NACA0012 airfoil. These cases were computed for a Re of 1.0×10^4. Each case corresponds to a flapping cycle of $100 + 1$ points (the last point is a repetition of the first used to close the plot) Data points were picked from hyperbolas of constant maximum effective angle of attack in the $k - h$ domain. Such curves are defined by equation (3.34). Points for validation have been also been picked

from lines of constant α_{eff} distinct from those in the training set, both within and outside the boundary of the training range.

$$\max \alpha_{eff} = arctan(kh) \tag{3.34}$$

The plunging data set is an expansion of the set already used in reference [104]. The training subset has a total of 240 cases, while the validation subset has 72. Each case corresponds to a flapping cycle of $100 + 1$ points (the last point is a repetition of the first one, used to close the plot). Since this set is for pure plunging, the pitching amplitude, A_α, is set to 0. The cases for training have been picked as described in the $k - h$ domain for non dimensional amplitudes h of 0.5,1.0,2.0,3.0 and 5.0. For validation, points picked from intermediate hyperbolas with values of h equal to 1.5 and 2.5. Both training and validation conditions are repeated for five mean angles of attack, $\alpha_{\text{mean}} = 0, 2.5, 5.0, 7.5$ and 10. All these cases are presented in Figure 3.10 in the $k-h$ domain. These data points can also be consulted in Appendix A, with the validation and training cases in Tables A.1 and A.2, respectively.

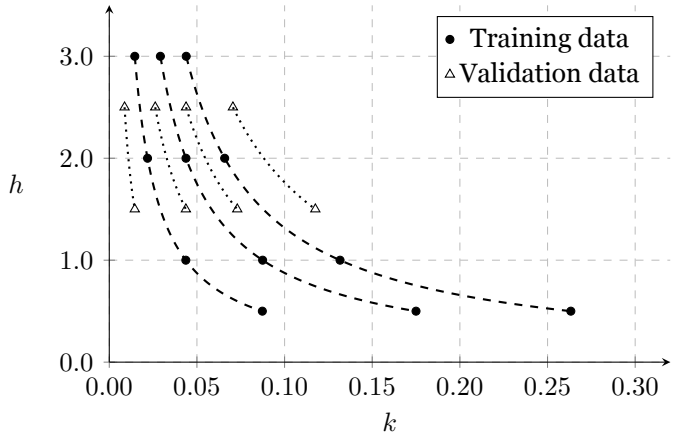

Figure 3.10: Plunging training and validation conditions in the $k - h$ domain.

It is important to note that some of the conditions presented to the neural network provide unrealistic lift coefficients due to panel method limitations. Such conditions include high mean angles of attack coupled with large effective angles of attack. Still, such conditions can be of interest to test the limits of model generalization, even thought they do not represent realistic approximations.

Now about the training set-ups. An initial training was made to find a learning rate suitable to train the neural network. In this training, the network was trained over 100000 epochs for three different learning rates,η, of 0.01, 0.1 and 1.0. The impact of other hyperparameters was not studied, as the purposed of this work is to get a working prototype of the neural network.

Once a good learning rate was found, the efficacy of running the neural network to initialize the delay nodes was tested. For this, two new networks were trained at the found learning rates, one with the extra points and another without them for control. Four extra points for initialization have been chosen to be run, as it is the same amount of points stored in the delay layers. These trainings were performed for 150000 epochs. In both cases, the delays were initially set to zero.

All results are presented and discussed in the following chapter.

This page was intentionally left blank.

Chapter 4

Results and Discussion

Now that the network and test conditions have been explained, this chapter will present the results found during the network training. The chapter is split into two sections, the first of which concerns the investigation of the learning rate to train the RNN for an airfoil undergoing pure plunging motion. This initial study aims to find a good learning rate with which the neural network can be further trained. Once found, the RNN is then trained with and without extra delay initialization cycles. The second section shows the results for those two last cases.

The results are then discussed and analysed. Plots of the evolution of the objective function during training, case distribution, model fit and network outputs are shown. Training and validation outputs, together with their respective mean squared errors and coefficients of determination, are compared and the model generalization is evaluated. Finally the best and worst predictions of each model are presented and compared.

4.1 Plunging Learning Rate Study

Training a neural network is a complex task that involves a coordination of many hyper-parameters. The learning rate is one if not the hyper-parameter with the largest and most direct impact in training. For this reason, an initial study is performed to see what would be best learning rate to conduct the trainings. In this study, three networks were trained for 100000 epochs with learning rates of 1.0, 0.1 and 0.01. Each training took about 4 hours to complete.

First, to ensure that the batch case selection follows a random uniform distribution, the cases from the first test have been recorded and are shown in Figure 4.1. Looking at the plot, it is clear the case selection approximates a random uniform distribution.

Now lets look at the evolution of the objective function for different learning rates. The objective function for the training with a learning rate of 1.0 diverged almost instantly, so it will not be considered for the remaining of this section. The plots for the tests with learning rates of 0.1 and 0.01 are shown in Figure 4.2.

From the plots it is clear that both trainings converged to a minimum, although for the case with $\eta = 0.1$ it seams that the objective function trend could continue to lower a bit more. It reached a value of 1.3×10^{-3} after the 100000 epochs. In contrast, the training with $\eta = 0.01$ found a minimum much faster, stabilizing at a value of 4.60×10^{-3}. It can

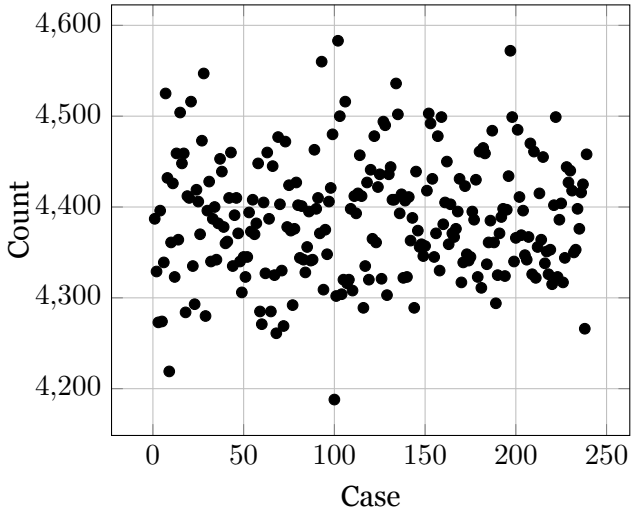

Figure 4.1: Loaded cases during 1000 first epochs of training.

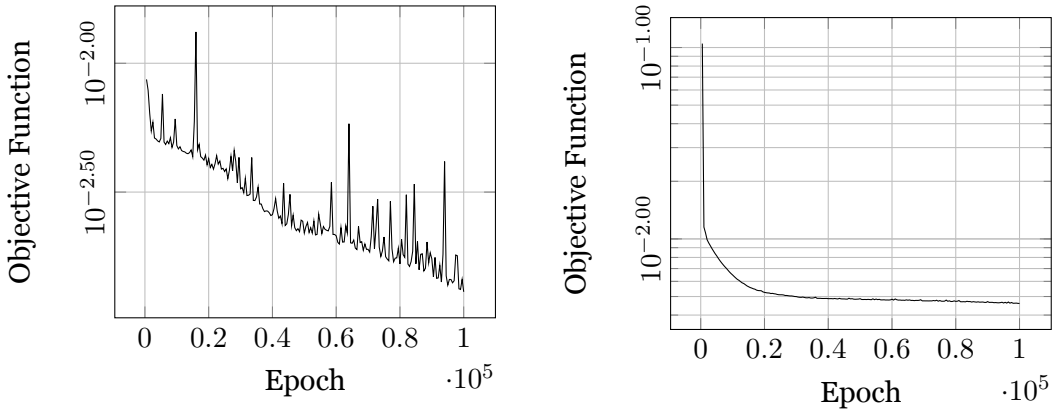

Figure 4.2: Evolution of the objective function during plunging calibration for $\eta = 0.1$ on the left and for $\eta = 0.01$ on the right.

also be seen that rate of descent of the objective function became insignificant after about 12000 epochs.

The values of the objective functions alone suggest that a learning rate of 0.1 is the most appropriate for the plunging training, since it produced the lowest minimum for the same number of epochs. The training with a learning rate of 0.01 seams to have become trapped at a larger minimum, which means that the learning rate was to low to allow the gradient of the optimization function to surpass that local minimum. In contrast, the convergence for a learning rate of 0.1 shows a significant amount of oscillations, suggesting that the learning rate could be on the verge of being too high for our function space. It also hints to the complexity of the objective function in parameter space that is being optimized, with the gradient jumping between local minimums.

Note that the value of the objective function after training only informs if the network optimization reached the vicinity of a minimum. It does not, however, represent the quality of that minimum. For this reason the results from the training and validation must be

looked at. One way to examine the accuracy of the model is to express the results in terms of their MSE and R^2, as provided in Figures 4.3 and 4.4.

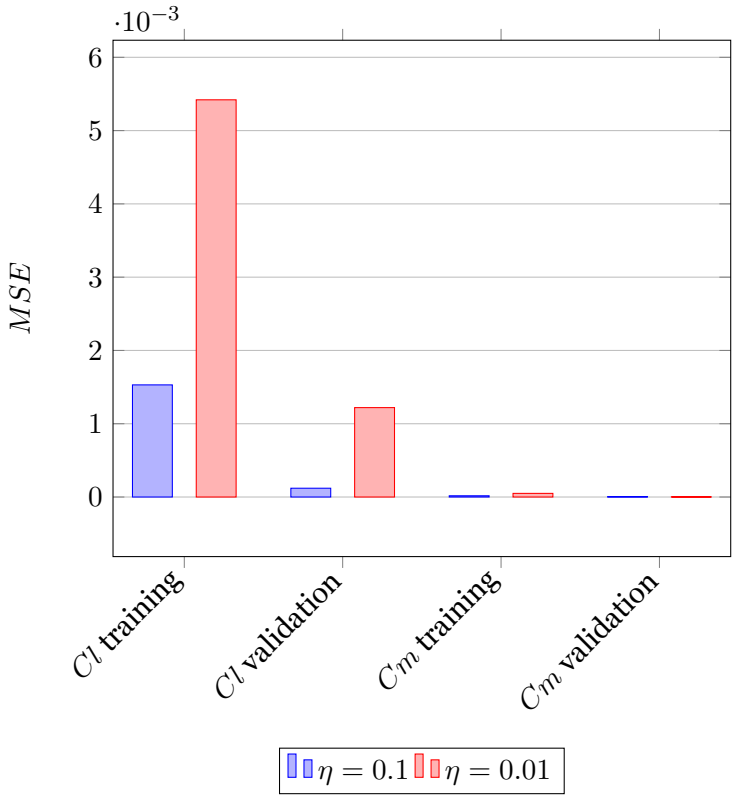

Figure 4.3: Values of MSE for the training and validation of the network with multiple learning rates.

From these results it is evident that the learning rate of 0.1 enabled the network to achieve lower scores of MSE and higher values of $\bar{R}^2$, for both the training and validation sets. It also appears that, on both cases, the lift coefficient achieved a better accuracy than the pitching moment coefficient. This difference is more remarkable for MSE in Figure 4.3, which can be due to the difference in magnitude between the C_l and C_m, so a direct comparison between coefficients can be misleading. On the other hand, since $\bar{R}^2$ is based on a quotient of squared differences, it can be used to compare between coefficients. In this case the score for the pitching moment coefficient is only slightly smaller than the lift coefficient. Still, this could hint that the neural network has more difficulty learning the dynamics of the C_m.

Also curious is how the validation set scores better than the training set, suggesting that the model could be entering a state of over-fitting. This is specially apparent for the pitching moment coefficient. and could imply that the network reached its limit in terms of learning the behaviour of the C_m faster than for the C_l. This is similar to what was found in [103], where the C_m converged faster than the C_l.

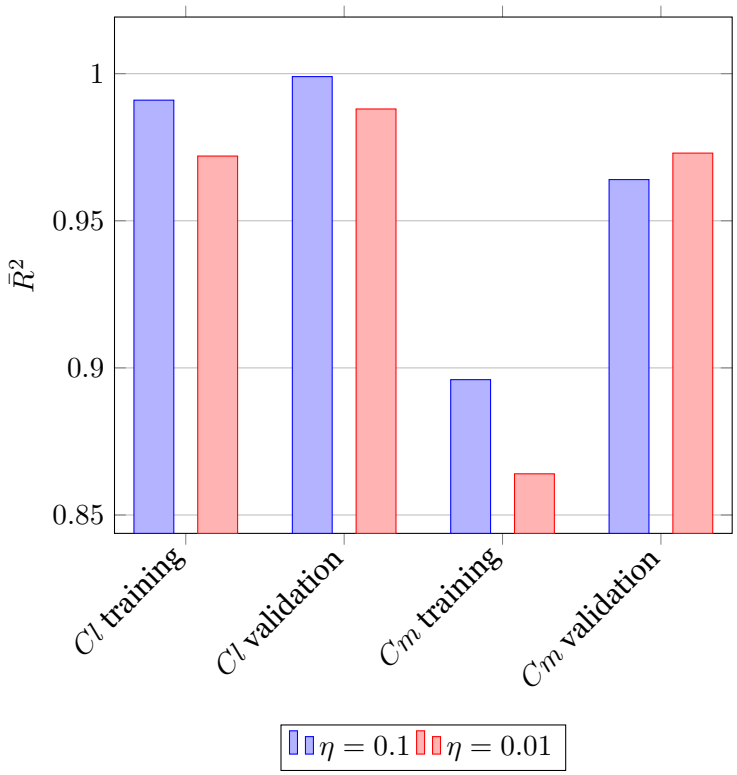

Figure 4.4: Values of $\bar{R}^2$ for the training and validation of the network with multiple learning rates.

Another and very effective way to examine the quality of the trained model is to plot the network outputs against their target values. Figures 4.5 and 4.6 provide this comparison between the predicted and target values for the validation results of C_l and C_m, respectively.

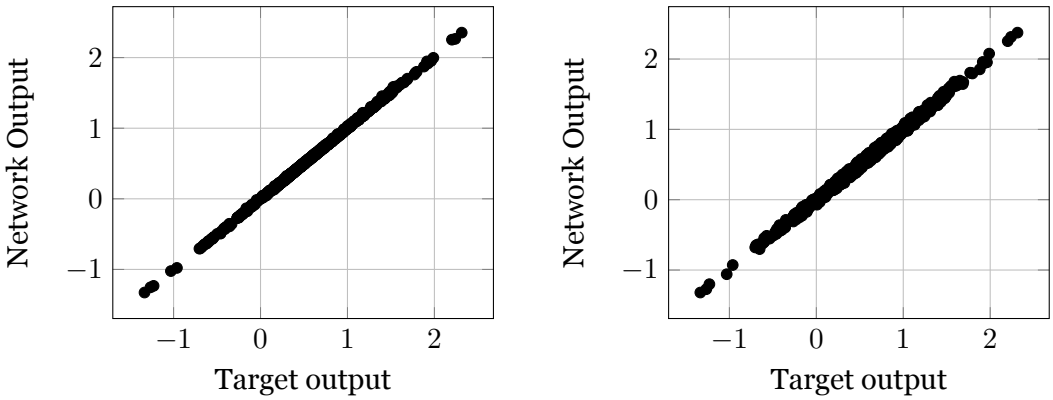

Figure 4.5: Comparison between the target and network outputs of C_l for the validation for $\eta = 0.1$ on the left and $\eta = 0.01$ on the right.

The plots for the lift coefficient, in Figure 4.5, show that the trained neural network performs as a good model for the validation data. There is good agreement between the predicted and target values for both learning rates. Regarding the two learning rates, the

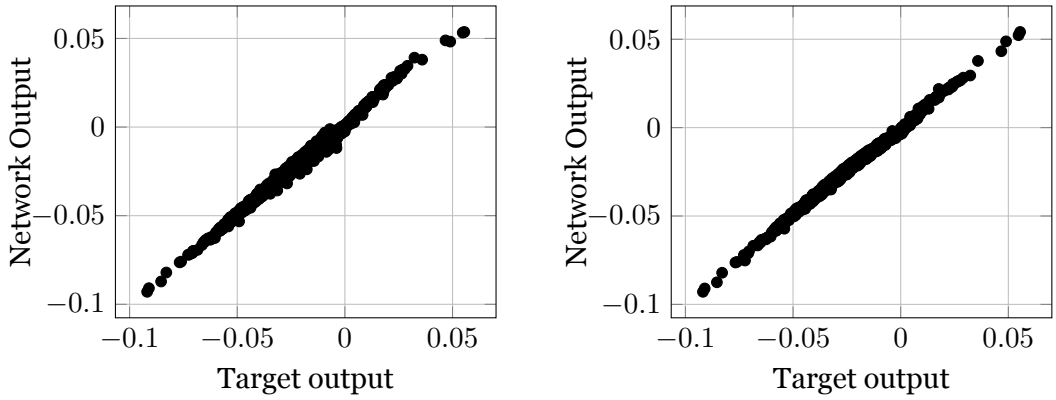

Figure 4.6: Comparison between the target and network outputs of C_m for the validation for $\eta = 0.1$ on the left and $\eta = 0.01$ on the right.

scatter of points is nearly identical in both plots. The plot for the learning rate of 0.1 appears to have only a slightly tighter distribution. This fact is also in accordance to the MSE and $\bar{R}^2$, which are very close of the lift coefficient.

Regarding the pitching moment coefficient, shown in Figure 4.6, it can be seen that the correspondence between the predicted and target values is not as good as for the lift coefficient. This also agrees with the scores from MSE and $\bar{R}^2$. However, unlike with the C_l, the model trained with the larger learning rate of 0.1 appears to have a larger dispersion of points than the model trained with $\eta = 0.01$. This agrees with the MSE and $\bar{R}^2$, where the smaller learning rate resulted in slightly better validation scores, but lower training score. Such facts could mean that the learning rate of $\eta = 0.1$ is more aggressive for the C_m.

Taking all this in consideration, a learning rate of 0.1 was chosen to conduct the final plunging trainings. With this learning rate, the network reached a lower minimum and achieved overall better training and validation scores. It is worth noting that the impacts of other hyper-parameters, such as the number of layers and neurons, or the moment coefficient β, have not been analysed, as they fall out of the scope of the present work. The results from the final trainings can be found next.

4.2 Plunging Results

With the learning rate chosen, the final set of trainings was performed. Two RNN were trained at $\eta = 0.1$ for 150000 epochs, to test if there is an improvement to prediction accuracy from the delay initialization. As control, one network was trained with the delay nodes initialized as zero. The second RNN was let to run for 4 additional time-steps in order to initialize the delays with outputs from the neural network. This section comprises of two parts, where the results from those trainings are presented and discussed. For each test, the evolution of the objective functions, MSE and $\bar{R}^2$ are analysed and the network outputs compared to their target values.

Starting with the evolution of the objective function, both trainings are presented in Figure 4.7. From the plots it can be seen that both trainings converged to similar minimums. These are of 2.3×10^{-3} and 2.1×10^{-3} for the trainings without and width the delay initialization period, respectively. It can also be seen that both descents appear to have similar levels of oscillations. The training without delay initialization also had a large peak near the end.

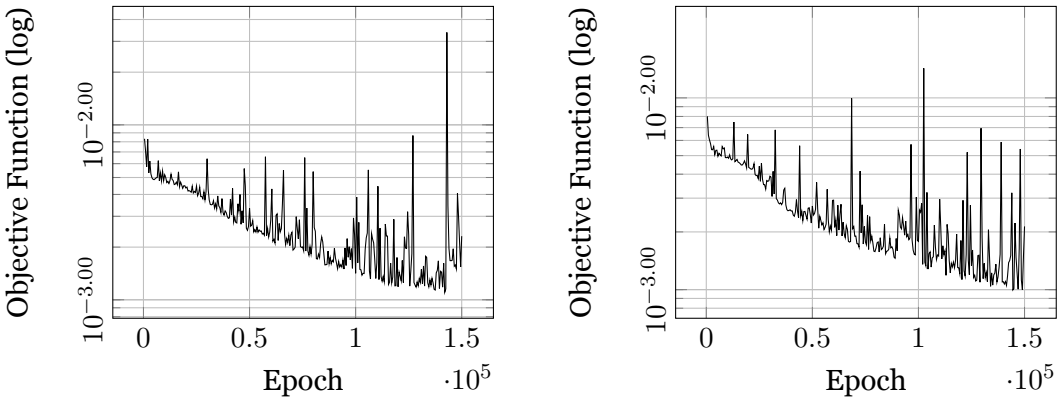

Figure 4.7: Evolution of the objective function for the plunging training with delays initialized as zero, on the left, and with delays initialized for 4 time steps on the right.

While the value of the objective function suggests the networks was learning, the quality of the trained models needs to be evaluated. For this, the MSE and $\bar{R}^2$ are shown in Figures 4.8 and 4.9. Once again it is seen that the validation set scored better than the training one, both in terms of MSE and $\bar{R}^2$. Regarding the delay initialization, it can be seen that the network with the delays initialized as zero performed better in the training set, for both C_l and C_m. In contrast, the network in which the delays are initialized by running extra time steps performed better in the validation set. However, the improvement in validation score is not substantial, with only a 0.8% and 0.1% increase for the C_l and C_m, respectively.

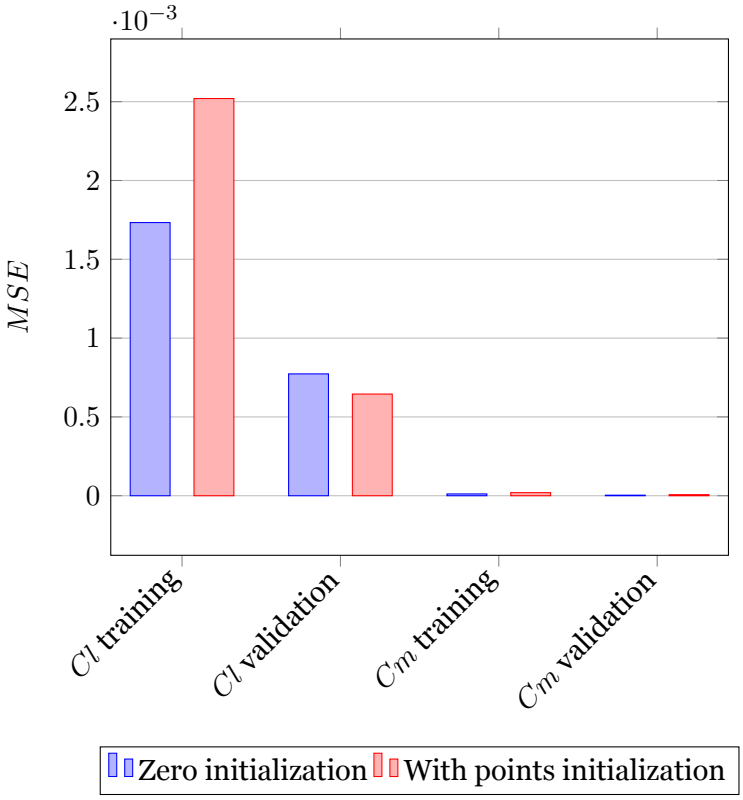

Figure 4.8: Values of MSE for the training and validation of the network with and without delay initialization steps.

While the MSE and $\bar{R}^2$ are good metrics to compare the overall network performance, plotting the network outputs against their target coefficients offers a great perspective into the quality of the model. Figure 4.10 shows the relationship between the network's predicted lift coefficients and their real values, for the validation results. The comparison between the target and predicted validation pitching moment coefficients can be found in Figure 4.11. All plots from now on concern the validation results of the model.

From looking at Figure 4.10 it is clear that both networks learned the C_l successfully. In both trainings the correlation between the predicted and the target coefficients is very good. This agrees with the MSE and $\bar{R}^2$, where the networks with and without delay initialization achieved similarly good scores.

Unlike the lift coefficient, where one cannot distinguish between both set of points, with the C_m the differences in Figure 4.11 are more apparent. While both models exhibit strong correlation between the network's predictions and the target values, these are not as good as for the C_l. There is higher scatter of points, which seams to be worst for the network with delay initialization.

Now lets have a look at the best and worst cases form each model's 48 validation cases. Table 4.1 presents the best 5 results from the validation of the model with delays initialized as zero, while Table 4.2 shows the 5 best results from the model with the delays initialized

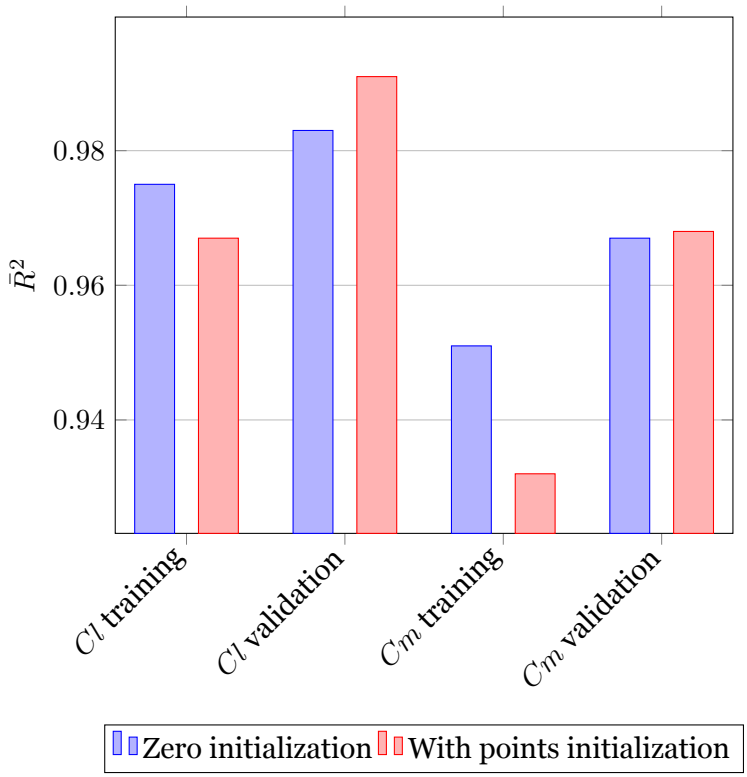

Figure 4.9: Values of $\bar{R}^2$ for the training and validation of the network with and without delay initialization steps.

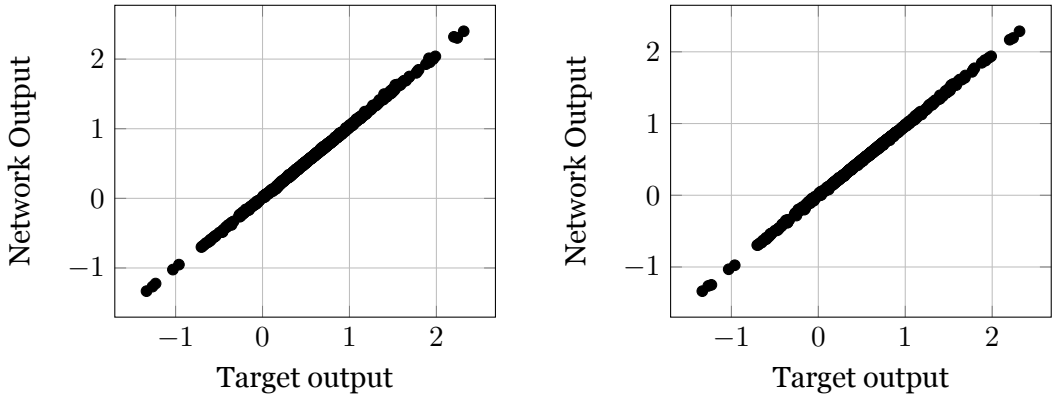

Figure 4.10: Comparison between the target and network outputs of C_l for the network without delay initialization on the left, and with delay initialization on the right.

with extra time steps. Similarly, Table 4.3 shows the worst 5 results from the validation of the network without delay initialization, while in Table 4.4 are present the 5 worst results from the model with the delay initialization steps. The complete results from both validation runs can be found in Tables B.1 and B.2 in Appendix B.

From looking at Tables 4.1 and 4.2 it can be seen that the difference between the best cases in both models is not much. This can be due to the fact that the delay initialization mostly impacts the first outputs points, as was seen by Pereira et al. [104]. The network where the delays were initialized with additional time steps score slightly better on the lift

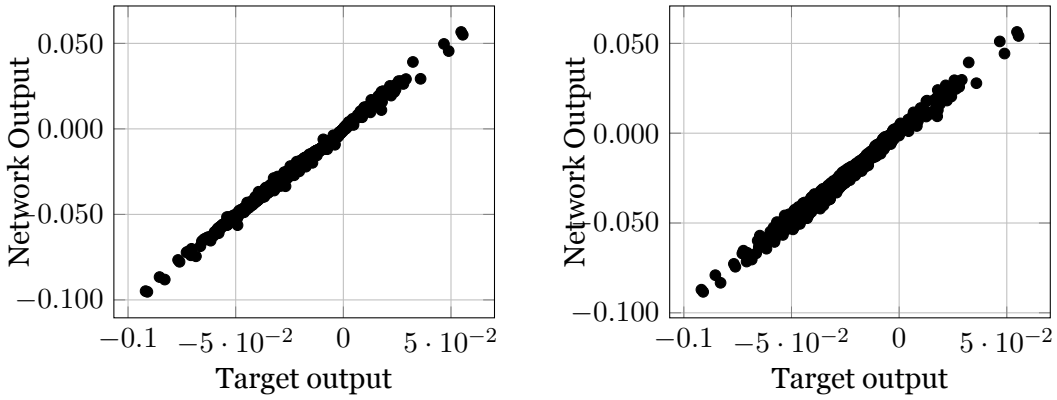

Figure 4.11: Comparison between the target and network outputs of C_m for the network without delay initialization on the left, and with delay initialization on the right.

Table 4.1: MSE and R^2 for the best validation results without delay initialization.

Case	Condition			C_l		C_m	
	k	h	α_{mean}	MSE $[\cdot 10^{-3}]$	R^2	MSE $[\cdot 10^{-6}]$	R^2
31	0.073	2.5	5.00	1.09	0.998	4.57	0.996
44	0.071	2.5	5.00	1.06	0.998	4.24	0.997
35	0.073	2.5	6.25	0.80	0.999	5.09	0.995
48	0.071	2.5	6.25	0.80	0.999	4.69	0.995
40	0.071	2.5	3.75	1.33	0.998	4.56	0.996

Table 4.2: MSE and R^2 for the best validation results with delay initialization.

Case	Condition			C_l		C_m	
	k	h	α_{mean}	MSE $[\cdot 10^{-3}]$	R^2	MSE $[\cdot 10^{-5}]$	R^2
37	0.118	1.5	3.75	0.19	0.999	0.71	0.993
41	0.118	1.5	5.00	0.64	0.999	0.67	0.993
39	0.118	2.5	3.75	0.76	0.999	2.16	0.992
43	0.118	2.5	5.00	1.93	0.999	2.14	0.992
47	0.118	2.5	6.25	3.66	0.998	2.11	0.992

coefficient, but worst on the pitching moment coefficient.

Table 4.3: MSE and R^2 for the worst validation results without delay initialization.

Case	Condition			C_l		C_m	
	k	h	α_{mean}	MSE $[\cdot 10^{-4}]$	R^2	MSE $[\cdot 10^{-6}]$	R^2
5	0.015	1.5	5.00	5.0	0.954	1.62	0.930
2	0.009	1.5	3.75	4.2	0.894	4.94	0.908
9	0.015	1.5	6.25	6.1	0.945	2.25	0.907
6	0.009	1.5	5.00	5.3	0.869	1.79	0.792
10	0.009	1.5	6.25	6.1	0.849	2.22	0.747

Again, there aren not many differences between the two trained networks, specially regarding the best results. When it comes to the worst cases, it seams that the model without the delay initialization had slightly lower scores, for both C_l and C_m. It can also be observed that the network without delay initialization has a larger range between the worst and best scores. In contrast, the model trained with delay initialization shows a narrower

Table 4.4: MSE and R^2 for the worst validation results with delay initialization.

Case	Condition			C_l		C_m	
	k	h	α_{mean}	MSE $[\cdot 10^{-4}]$	R^2	MSE $[\cdot 10^{-6}]$	R^2
2	0.009	1.5	3.75	1.1	0.973	7.05	0.933
8	0.009	2.5	5.00	3.7	0.967	6.70	0.928
4	0.009	2.5	3.75	2.2	0.980	2.16	0.908
6	0.009	1.5	5.00	2.0	0.950	2.14	0.933
10	0.009	1.5	6.25	2.3	0.944	2.11	0.880

gap from the worst to the best results. Furthermore, there seams to be a tendency for the first cases to score lower and the last cases to score better. Recalling the data's $k-h$ domain, present in Figure 3.10, it seams that the validation data points within and above the training $k-h$ domain score better. On the other hand, the validation points below the training $k-h$ domain had the worst results. This can be explained by the fact that ANNs are an interpolation functions and not an extrapolating models.

In order to further investigate this, the values of R^2 are plotted against k and h, in Figures 4.12 and 4.13, respectively. From the first figure it is clear that the network's performance increases rapidly with k. This distribution of R^2 strongly confirms the fact the the model has greater difficulty predicting cases with values of k below the training domain. This very apparent with the sudden increase of R^2 for the range of k within the training domain. Also curious is how cases above the training domain score better. This can be explain by the fact that, even though those points are above the $k-h$ hyperbolas, the hyperbolas of the training domain contain points with k larger than the validation ones. In contrast, the points with lower k from the validation domain are completely outside the range of k from the training domain. This fact can explain with those initial points score much lower than the rest, since the network never saw any similar cases during training. Nevertheless, these results strongly suggest that the model was indeed able to learn and generalized the region around its training domain.

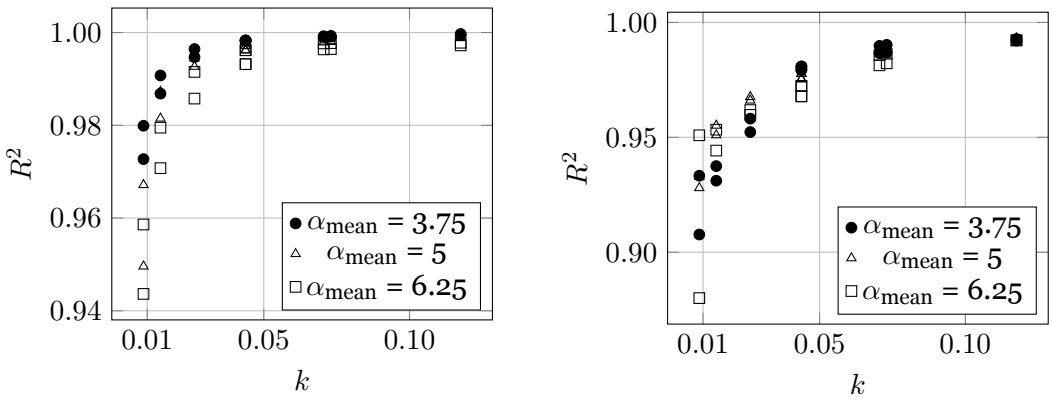

Figure 4.12: Plot of R^2 vs k for the model with delay initialization for the C_l and C_m, on the left and right, respectively.

Regarding the plots of R^2 with h in Figure 4.13, there does not seam to be any clear correlation between the model performance and h. Since the validation set only contains two values of h, it could be that there are not sufficient data points for a conclusion to be drawn. With the data available, the dispersion of R^2 seams to be affected mostly by the variation of k, as referred previously.

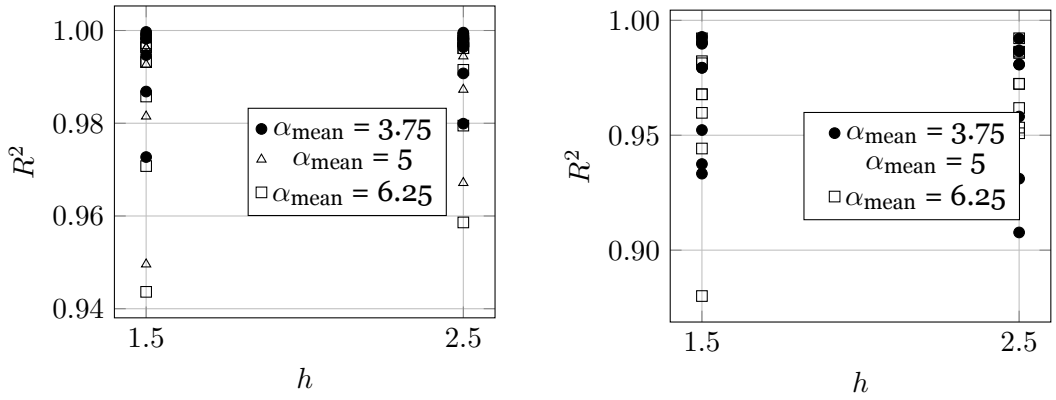

Figure 4.13: Plot of R^2 vs h for the model with delay initialization for the C_l and C_m, on the left and right, respectively.

It is also worth pointing that, although the neural network was implemented with the capability to process the Re, the study of its impact was not conducted, as it implied further expansion of the data set.

At last, the plots for the best and worst aerodynamic coefficients of both models will be presented. Due to space constraints, only the top and bottom 3 cases of each model's validation are shown. Figures 4.14 to 4.16 show the best aerodynamic predictions for the network without delay initialization, while Figures 4.20 to 4.22 show the corresponding worst cases. Simillarly, Figures 4.17 to 4.19 present the best outputs for model with the delay initialization set, and Figures 4.23 to 4.25, contain the worst cases.

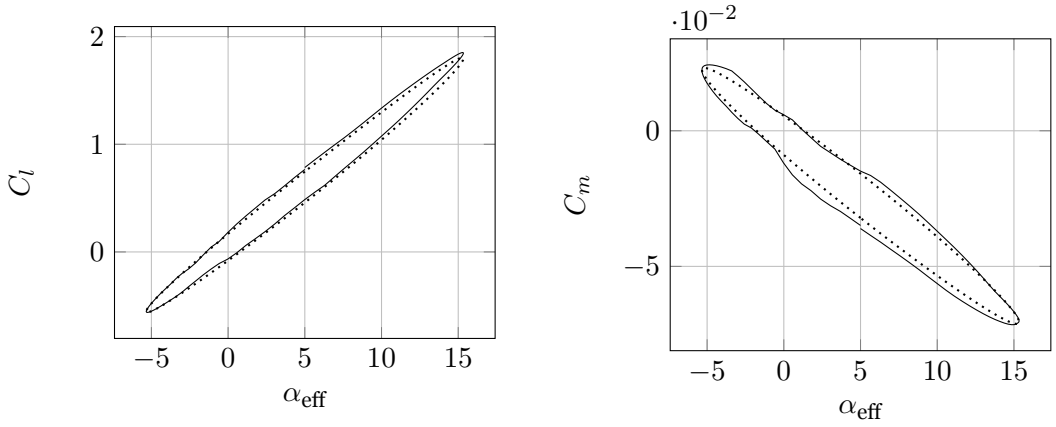

Figure 4.14: Plots of C_l (left) and C_m (right) vs α_{eff} for $k = 0.073$, $h = 2.5$ and $\alpha_{\text{mean}} = 5.0$, from the the network trained without delay initialization.

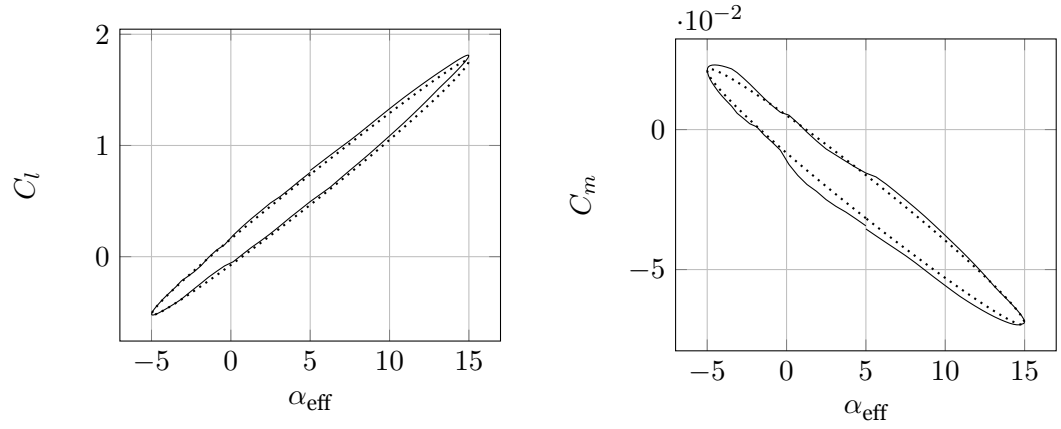

Figure 4.15: Plots of C_l (left) and C_m (right) vs α_{eff} for $k = 0.071$, $h = 2.5$ and $\alpha_{\text{mean}} = 5.0$, from the network trained without delay initialization.

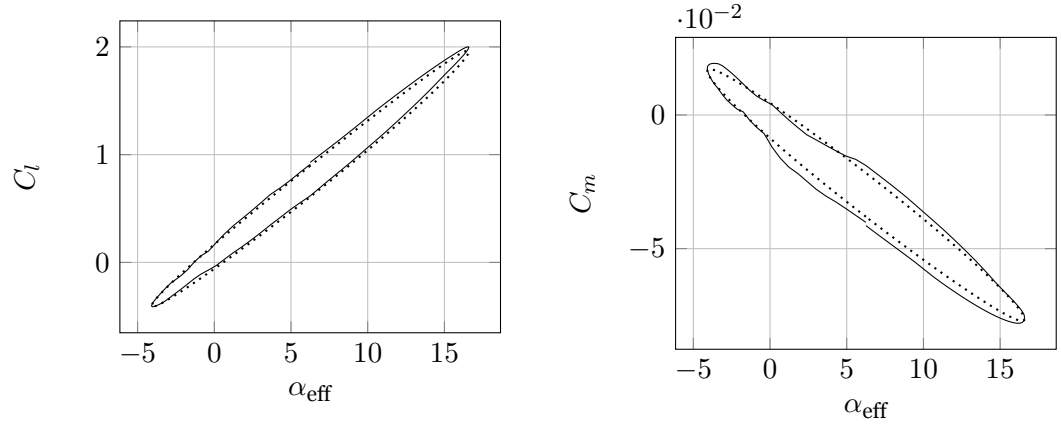

Figure 4.16: Plots of C_l (left) and C_m (right) vs α_{eff} for $k = 0.073$, $h = 2.5$ and $\alpha_{\text{mean}} = 6.25$, from the network trained without delay initialization.

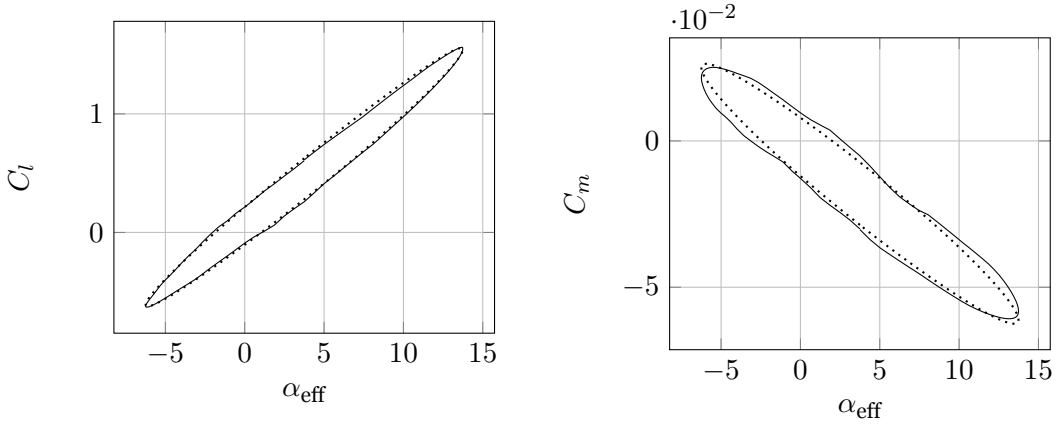

Figure 4.17: Plots of C_l (left) and C_m (right) vs α_{eff} for $k = 0.118$, $h = 1.5$ and $\alpha_{\text{mean}} = 3.75$, from the network trained with delay initialization.

From looking at the best results, it is clear that both neural networks have been able to predict the evolution of the aerodynamic coefficients to a satisfactory degree. Specially regarding the lift coefficient, it is very well approximated by both models. On the other

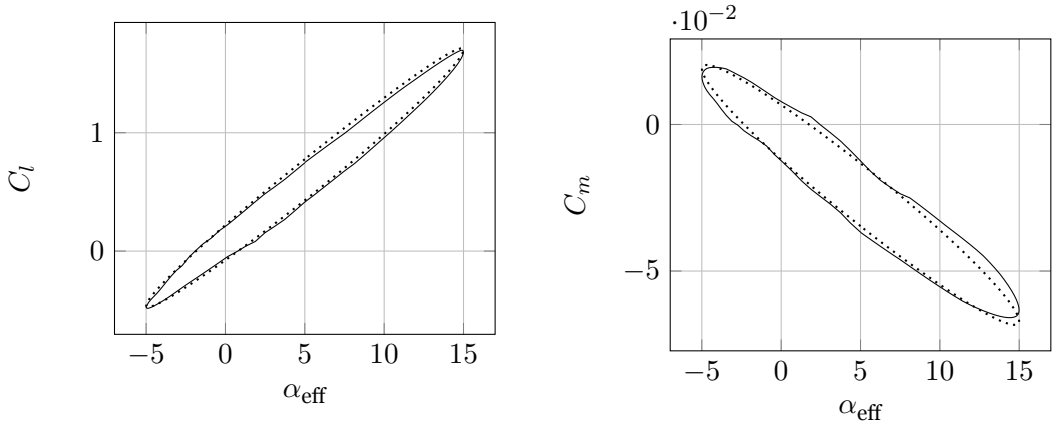

Figure 4.18: Plots of C_l (left) and C_m (right) vs α_{eff} for $k = 0.118$, $h = 1.5$ and $\alpha_{\text{mean}} = 5.0$, from the network trained with delay initialization.

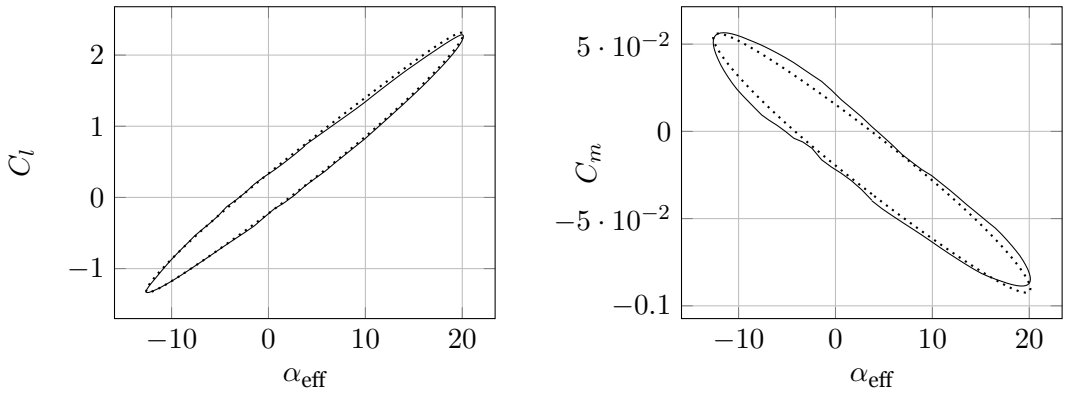

Figure 4.19: Plots of C_l (left) and C_m (right) vs α_{eff} for $k = 0.118$, $h = 2.5$ and $\alpha_{\text{mean}} = 3.75$, from the network trained with delay initialization.

hand, the distribution of the pitching moment coefficient are slightly worst, but still good. This agrees with the model scores seen so far. The points are well within the boundaries of the target outputs and the overall shape was captured. The neural networks can clearly distinguish between the two branches of the aerodynamic response. This suggests that the model is capable of capturing non-linear aerodynamic behaviour. It is also remarkable how one same neural network was able to produced very distinct outputs of aerodynamic coefficients, both in orientation and order of magnitude.

In contrast, the worst results do not exhibit such good correlations between the predicted and target outputs. In these cases the distributions of the C_l and C_m form very close ellipses, which both neural networks struggles to match. Once again, the predictions of C_m appear to be slightly worst than the estimates of C_l. Nevertheless, the predicted coefficients are still somewhat enclose the range of magnitude of the target points, appearing to be shifted from position.

It is important to note that the network never saw such conditions during training, neither did touch cases with values of k in this lower range. Because of this, the fact alone that both models were able to situate the predictions with the order of magnitude of the target values

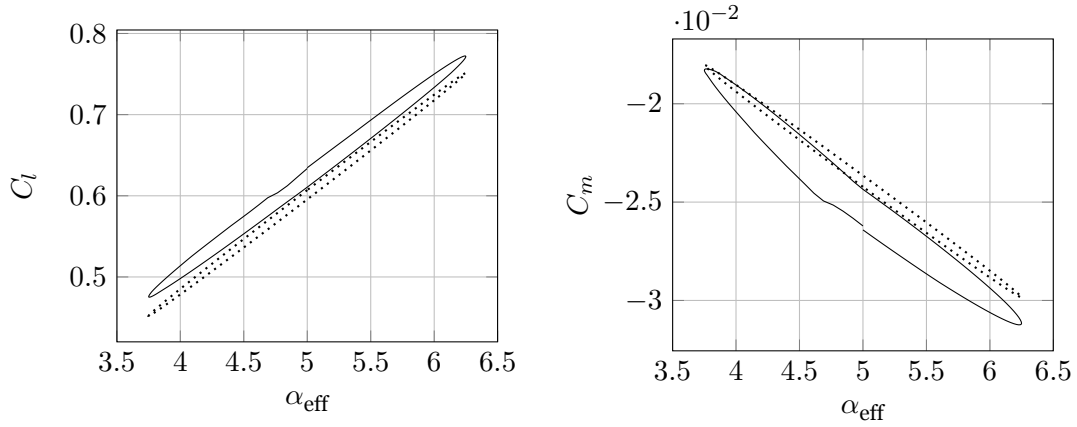

Figure 4.20: Plots of C_l (left) and C_m (right) vs α_{eff} for $k = 0.015$, $h = 1.5$ and $\alpha_{\text{mean}} = 5.0$, from the network trained without delay initialization.

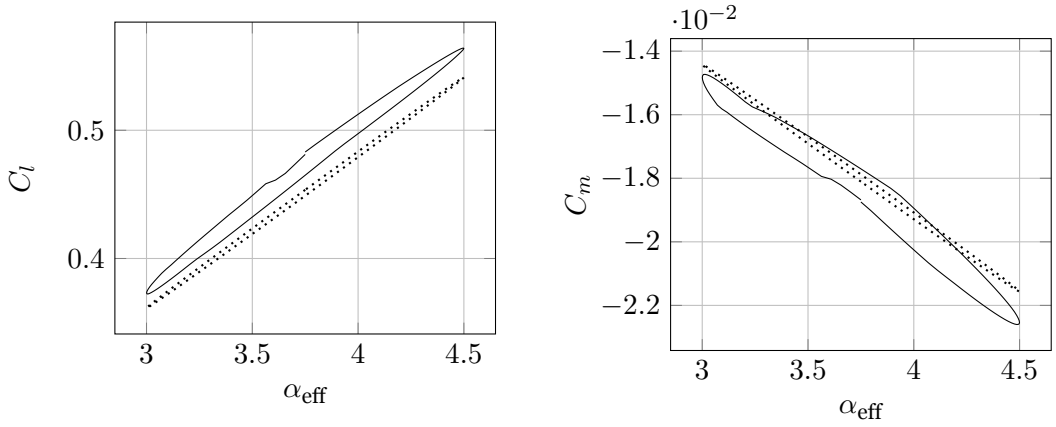

Figure 4.21: Plots of C_l (left) and C_m (right) vs α_{eff} for $k = 0.009$, $h = 1.5$ and $\alpha_{\text{mean}} = 3.75$, from the network trained without delay initialization.

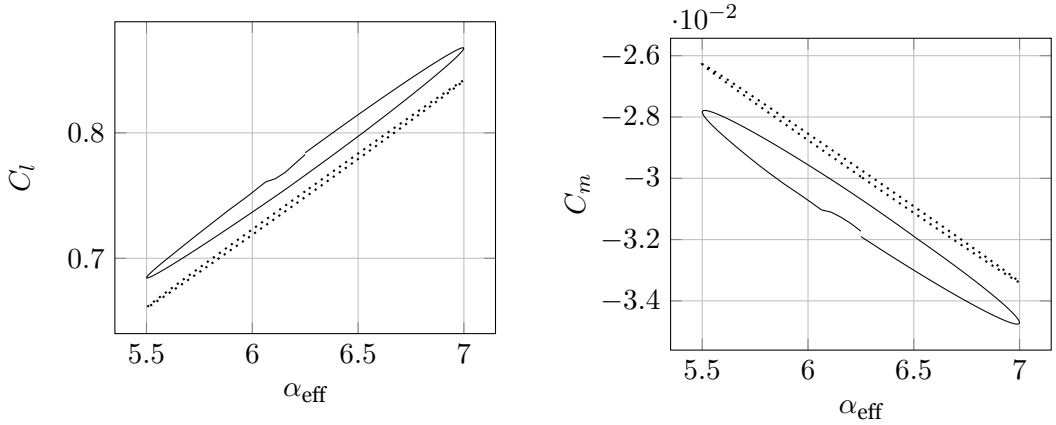

Figure 4.22: Plots of C_l (left) and C_m (right) vs α_{eff} for $k = 0.009$, $h = 1.5$ and $\alpha_{\text{mean}} = 6.25$, from the network trained without delay initialization.

confirms some degree of model generalization. This means that the neural networks can learn the dynamics of the systems and its dependency on the inputs provided.

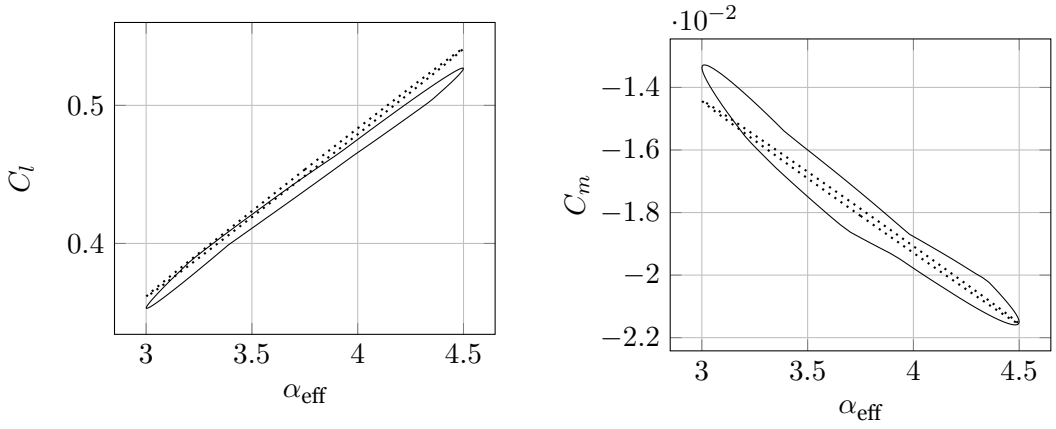

Figure 4.23: Plots of C_l (left) and C_m (right) vs α_{eff} for $k = 0.009$, $h = 1.5$ and $\alpha_{\text{mean}} = 3.75$, from the network trained with delay initialization.

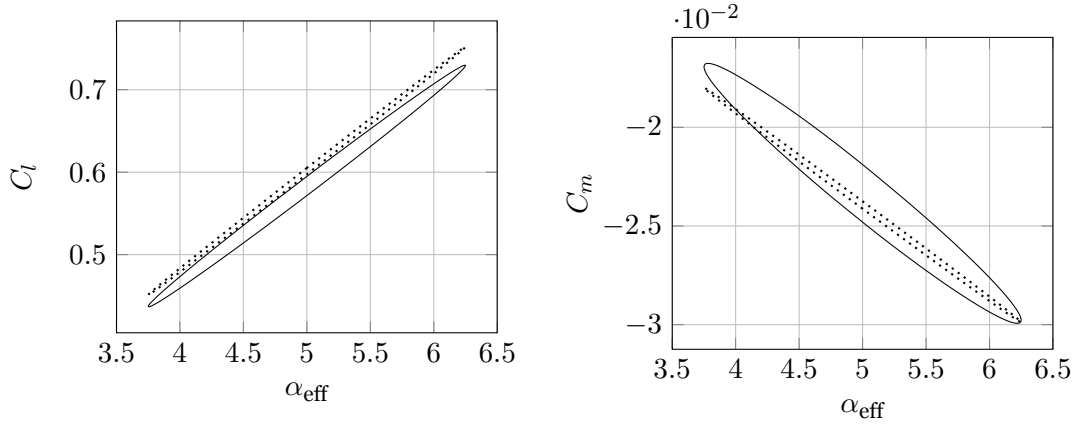

Figure 4.24: Plots of C_l (left) and C_m (right) vs α_{eff} for $k = 0.009$, $h = 2.5$ and $\alpha_{\text{mean}} = 5.0$, from the network trained with delay initialization.

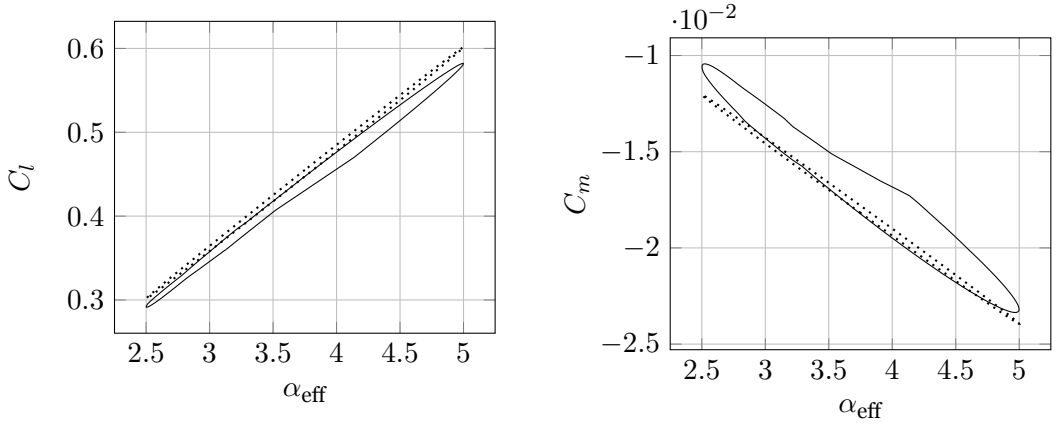

Figure 4.25: Plots of C_l (left) and C_m (right) vs α_{eff} for $k = 0.009$, $h = 2.5$ and $\alpha_{\text{mean}} = 3.75$, from the network trained with delay initialization.

Finally the only evident difference between the estimates of the network without the delay initialization step is the discontinuity that can be observed at the start of the flapping period. It can be seen how the initial prediction is slightly off and the following predictions

improve as the delay layer gets fed with the previous outputs. In fact, this initialization problem seams to mostly affect the first four to five points, after which any differences become negligible and accuracy improves. Even still„ the gap and consequent deviation cause on the first points is still contained within the deviation of the model. This explains why both networks had such similar scores in both $\bar{R}^2$ and MSE, with the major differences being found within the models, between the training and validation data.

Looking at the neural network where four extra predictions are made to initialize the delays, no discontinuity between the end and start points of the flapping period can be noted. Still, the impact on the overall model performance is negligible, as was seen. Nevertheless, this model also mimics the way a real-time prediction implementation would work on a system, which would have a start-up period, before receiving a sufficient amount of data.

Having presented and discussed the results from this work, the next and final chapter will summarize the study done and present some closing remarks, as well as future prospects.

Chapter 5

Conclusions and Future Work

As it has been presented, the flapping mechanism is a very effective mean to achieve propulsion in the natural world. From the air to water, insects, birds, fish and cetaceans deploy flapping systems to generate thrust. Through the years, this biological adaptation has captivated the minds of many who attempted to understand it, and maybe find ways to reproduce it for human applications. From micro air vehicles, wind and wave power extraction and aquatic propulsion, flapping systems have gained increased interested in recent years.

However, the study of the flapping mechanism and its inherent transient aerodynamic behaviour has always been problematic. It started with the first theories and mathematical models which laid the foundation for the study of transient aerodynamics. Experimental methods based on wind and water tunnel studies allowed for a better understating of the mechanism. These experiments also produced large amounts of aerodynamic data that would support semi-empirical models. A great leap forward was achieved with the advent of panel methods, which offered a fast way to study the flow around airfoils, but with some restrictions. More recently, the advancements in computational fluid dynamics opened a new window into solving complex flow fields.

With this, the field of artificial intelligence is gaining increasing attention. Machine learning, in special artificial neural networks, have introduced a brand new paradigm into problem solving. Instead of having to study and develop a model, it is now possible to feed a machine with data and get a trained model that fits that set of data. Moreover, neural network models can be made arbitrary complex. Because of this, applications of ANNs in the field of aerodynamics are ever more relevant. Not only can they learn very complex processes, as once trained, obtaining new results is near instantaneous.

In the present dissertation a recurrent neural network was developed and trained to predict time-wise evolution of C_l and C_m for a pure plunging airfoil. A recurrent neural network was chosen for its capability of processing time-dependent data. For each time step, the networks estimates the aerodynamic coefficients based on a set of fixed inputs and time-varying inputs. The fixed inputs include the Re, k, h, kh, A_α, while the time-varying inputs comprise of the last 4 points of α_{eff}, together with the previous 4 stored values of C_l and C_m. The time-dependent inputs where fed to the network via a delay layer, which received the previous aerodynamic coefficients from the output layer via a feedback loop. The input layer received both the values from the delay layer, as well as the fixed inputs, for a total of 17 input neurons. This layer was followed by 3 hidden layers, of 28,28 and

12 neurons each. The networks ends with two linear output neurons, one for each of the aerodynamic coefficients.

In order to train the RNN, a data set was generated using an unsteady panel code. The conditions were created for the NACA0012 airfoil subject to various pure plunging motions. To aid with the training process, the fixed inputs were normalized by their maximum values, while the effective angle of attack was normalized by 2π.

An initial study was conducted to find a good learning rate to train the neural network. In this study, the RNN was trained for 100000 epochs at several learning rates. It was found that, for the current set-up, the learning rate of 0.1 produced the best results. Using the found learning rate, two RNN were trained for 150000 epochs. One in which the delay layer was simply initialized as zero, and another where the network was run for four extra time steps to fill the delay layer. A validation subset was then used to analyse the performance of the trained models.

Regarding this last point, it was found that the use of the extra time steps resulted only in a near insignificant improvement of the prediction accuracy (less than 1%). It was seen that the delay initialization affects mostly the initial estimates of the network, being rapidly diluted. About model quality, it was seen that predictions of the C_l consistently perform better than the the estimates of C_m. This suggests that the RNN has more facility learning the behaviour of the C_l. A cause for that can be the difference in magnitude between the two aerodynamic coefficients, which leads to the error of the lift coefficient to have a heavier impact in the gradient descent.

Moreover, it was also seen that the model was able to generalize to conditions distinct from those seen during training. It was seen that the network performs best for the cases located within and slightly above the $k - h$ training domain, in contrast with cases located on the lower range of the training boundary. When plotting the R^2 against k, a very clear trend was observed, where the value of R^2 increased very abruptly from the cases below the training domain to those within it. Still regarding model quality, the best cases achieved scores of R^2 very close to 1.0, while the worst case had a R^2 of 0.75. Looking at the network outputs themselves, the best cases show very good agreement between the predicted and target coefficients. Even the worst predictions seam to be nearing the boundaries of the target values. It is still important to note that while non-linear, these conditions are far from representing the true complexity that can be seen in extreme transient flapping aerodynamic conditions.

Nevertheless, there is still much room for improvement in the present model. First of all, the training algorithm itself can be enhanced with methods, such as ADAM, weight regularization and early stopping. The size and structure of the neural network can also be investigated. More complex problems may require a larger network, but a larger model also takes longer to compute and increases the sensitivity to over-fitting. The choice of parameter to be used as inputs for the model can be further studied, as well as the impact

of the number of data points to be stored in the delay layer. It can also be interesting to study the use of an individual neural network to estimate each aerodynamic coefficient, to see if it can improve accuracy and model convergence. Finally, there is the need to keep expanding the training data set and increase model generalization. Pitching conditions can be included and tested, then also combined pitching and plunging motions. Also the variation of other parameters like the Re. Eventually, an experimental data base of real, highly non-linear, transient aerodynamic conditions can be created and used to train a more accurate model.

To conclude, the present work joins many others that explore the prospects of applying artificial intelligence to the study of aerodynamics. The implemented model, even if still very limited, shows the capability of using a RNN for fast time-wise prediction of aerodynamic coefficients. Once trained, such a model could be easily deployed in real-time aerodynamic optimization schemes, like the ones required for morphing aerodynamics. Despite all efforts, a long way remains before neural networks can be come full transient aerodynamic prediction tools and real time control applications emerge.

This page was intentionally left blank.

Bibliography

[1] A. Vuruskan, I. Fenercioglu, and O. Cetiner, "A study on forces acting on a flapping wing," in *EPJ Web of Conferences*, vol. 45. EDP Sciences, 2013, p. 01028. 1

[2] X. Kai, L. Abbas, C. Dongyang, Y. Fufeng, and R. Xiaoting, "Numerical investigations on dynamic stall of a pitching-plunging helicopter blade airfoil," *International Journal of Mechanical and Mechatronics Engineering*, vol. 11, no. 10, pp. 1566–1571, 09 2017. 1, 23

[3] M. F. Platzer, K. D. Jones, J. Young, and J. C. Lai, "Flapping wing aerodynamics: progress and challenges," *AIAA journal*, vol. 46, no. 9, pp. 2136–2149, 2008. xiii, 1, 7, 14, 15

[4] J. Young, J. C. Lai, and M. F. Platzer, "A review of progress and challenges in flapping foil power generation," *Progress in aerospace sciences*, vol. 67, pp. 2–28, 2014. 1, 15

[5] M. S. Triantafyllou, G. Triantafyllou, and D. K. Yue, "Hydrodynamics of fishlike swimming," *Annual review of fluid mechanics*, vol. 32, no. 1, pp. 33–53, 2000. 1, 12

[6] R. Knoller and Ö. F. Verein, *Die Gesetze des Luftwiderstandes*. Verlag des Österreichischer Flugtechnischen Vereines, 1909, vol. 3. [Online]. Available from: https://books.google.pt/books?id=bRNDnQEACAAJ (accessed on 01-10-23). 1, 7, 9

[7] A. Betz, "Ein beitrag zur erklaerung segelfluges," *Z Flugtech Motorluftschiffahrt*, vol. 3, pp. 269–272, 1912. 1, 7, 9

[8] R. Katzmayr, "Effect of periodic changes of angle of attack on behavior of airfoils," NACA, Tech. Rep. NACA-TM-147, 10 1922. [Online]. Available from: https://ntrs.nasa.gov/citations/19930083152 (accessed on 01-09-23). 1, 9

[9] T. Von Kármán and J. M. Burgers, *General aerodynamic theory: Perfect fluids*. Berlin: Julius Springer, 1935, vol. 2. 1, 9

[10] N.-H. Teng, "The development of a computer code (u2diif) for the numerical solution of unsteady, inviscid and incompressible flow over an airfoil," Ph.D. dissertation, Naval Postgraduate School Monterey, CA, 1987. 2, 10, 66

[11] L. L. Erickson, "Panel methods: An introduction," NASA, CA, United States, Tech. Rep. A-89266, 12 1990. [Online]. Available from: https://ntrs.nasa.gov/citations/19910009745 (accessed on 01-07-23). 2, 10

[12] X. Wu, X. Zhang, X. Tian, X. Li, and W. Lu, "A review on fluid dynamics of flapping foils," *Ocean Engineering*, vol. 195, p. 106712, 2020. 2, 5, 17

[13] K. Balla, R. Sevilla, O. Hassan, and K. Morgan, "An application of neural networks to the prediction of aerodynamic coefficients of aerofoils and wings," *Applied Mathematical Modelling*, vol. 96, pp. 456–479, 2021. 2, 39, 43

[14] H. Moin, H. Zeeshan Iqbal Khan, S. Mobeen, and J. Riaz, "Airfoil's aerodynamic coefficients prediction using artificial neural network," in *2022 19th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*. IEEE, aug 2022, pp. 175–182. [Online]. Available from: http://dx.doi.org/10.1109/IBCAST54850.2022.9990112 (accessed on 01-06-2023). 2, 42

[15] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65 6, pp. 386–408, 1958. 2, 48

[16] T. Rajkumar, C. Aragon, J. Bardina, and R. Britten, "Prediction of aerodynamic coefficients for wind tunnel data using a genetic algorithm optimised neural network," *WIT transactions on information and communication technologies*, vol. 28, pp. 1–15, 02 2002. [Online]. Available from: https://www.witpress.com/elibrary/wit-transactions-on-information-and-communication-technologies/28/1253 (accessed on 01-11-2023). 2, 37

[17] S. Suresh, S. Omkar, V. Mani, and T. Guru Prakash, "Lift coefficient prediction at high angle of attack using recurrent neural network," *Aerospace Science and Technology*, vol. 7, no. 8, pp. 595–602, 2003. [Online]. Available from: https://www.sciencedirect.com/science/article/pii/S1270963803000531 (accessed on 01-07-2023). 2, 37, 43

[18] J.-Z. Peng, S. Chen, N. Aubry, Z.-H. Chen, and W.-T. Wu, "Time-variant prediction of flow over an airfoil using deep neural network," *Physics of Fluids*, vol. 32, no. 12, p. 123602, 12 2020. 2, 38

[19] B. Zhang, J. Han, T. Zhang, and R. Ma, "Unsteady aerodynamic identification based on recurrent neural networks," *Journal of Vibroengineering*, vol. 23, no. 2, pp. 449–458, 2021. 2, 40, 43

[20] B. W. Mersha, D. N. Jansen, and H. Ma, "Angle of attack prediction using recurrent neural networks in flight conditions with faulty sensors in the case of f-16 fighter jet," *Complex & Intelligent Systems*, vol. 9, no. 3, pp. 2599–2611, 2023. 2, 42

[21] A. V. Altena, J. J. van Beers, and C. C. de Visser, "Loss-of-control prediction of a quadcopter using recurrent neural networks," *Journal of Aerospace Information Systems*, vol. 20, no. 10, pp. 648–659, 2023. 2, 42

[22] T. Kinsey and G. Dumas, "Parametric study of an oscillating airfoil in a power-extraction regime," *AIAA journal*, vol. 46, no. 6, pp. 1318–1330, 2008. xiii, 7, 8

[23] E. A. R. Camacho, “Numerical analysis of a plunging naca0012 airfoil,” Master’s thesis, in Aeronautical Engineering. Universidade da Beira Interior (Portugal), 2019. xiii, 9, 10

[24] I. E. Garrick, “Propulsion of a flapping and oscillating airfoil,” NACA, Langley Field, VA, United States, Tech. Rep. NACA-TR-567, 01 1936. [Online]. Available from: https://ntrs.nasa.gov/citations/19930091642 (accessed on 01-09-23). 10

[25] A. Silverstein and U. T. Joyner, “Experimental verification of the theory of oscillating airfoils,” NACA, Langley Field, VA, United States, Tech. Rep. NACA-TR-673, 01 1939. [Online]. Available from: https://ntrs.nasa.gov/citations/19930091748 (accessed on 01-10-23). 10

[26] J. Bratt, “Flow patterns in the wake of an oscillating aerofoil,” Aeronautical Research Council, Tech. Rep., 1950. 10

[27] P. Freymuth, “Propulsive vortical signature of plunging and pitching airfoils,” *AIAA journal*, vol. 26, no. 7, pp. 881–883, 1988. 10

[28] I. H. Tuncer and M. F. Platzer, “Thrust generation due to airfoil flapping,” *AIAA Journal*, vol. 34, no. 2, pp. 324–331, 1996. [Online]. Available from: https://doi.org/10.2514/3.13067 (accessed on 01-09-2023). 11

[29] K. D. Jones, C. M. Dohring, and M. F. Platzer, “Experimental and computational investigation of the knoller-betz effect,” *AIAA Journal*, vol. 36, no. 7, pp. 1240–1246, 1998. [Online]. Available from: https://doi.org/10.2514/2.505 (accessed on 01-09-13). 11, 13

[30] I. H. Tuncer and M. F. Platzer, “Computational study of flapping airfoil aerodynamics,” *Journal of aircraft*, vol. 37, no. 3, pp. 514–520, 2000. 11

[31] G. C. Lewin and H. Haj-Hariri, “Modelling thrust generation of a two-dimensional heaving airfoil in a viscous flow,” *Journal of Fluid Mechanics*, vol. 492, p. 339–362, 2003. 12

[32] J. Young and J. C. S. Lai, “Oscillation frequency and amplitude effects on the wake of a plunging airfoil,” *AIAA Journal*, vol. 42, no. 10, pp. 2042–2052, 2004. [Online]. Available from: https://doi.org/10.2514/1.5070 (accessed on 01-10-2023). xiii, 12, 13

[33] B. C. Basu and G. J. Hancock, “The unsteady motion of a two-dimensional aerofoil in incompressible inviscid flow,” *Journal of Fluid Mechanics*, vol. 87, no. 1, p. 159–178, 1978. 13, 66

[34] J. H. Wu and M. Sun, “Unsteady aerodynamic forces of a flapping wing,” *Journal of Experimental Biology*, vol. 207, no. 7, pp. 1137–1150, 2004. 13

[35] M. Sun and J. Tang, "Unsteady aerodynamic force generation by a model fruit fly wing in flapping motion," *Journal of experimental biology*, vol. 205, no. 1, pp. 55–70, 2002. 13

[36] M. H. Dickinson, F.-O. Lehmann, and S. P. Sane, "Wing rotation and the aerodynamic basis of insect flight," *science*, vol. 284, no. 5422, pp. 1954–1960, 1999. 13

[37] W. Shyy and H. Liu, "Flapping wings and aerodynamic lift: The role of leading-edge vortices," *Aiaa Journal - AIAA J*, vol. 45, pp. 2817–2819, 12 2007. 13

[38] D. Kurtulus, L. David, A. Farcy, and H. Alemdaroğlu, "Aerodynamic characteristics of flapping motion in hover," *Experiments in Fluids*, vol. 44, pp. 23–36, 01 2008. 14

[39] K. Lu, Y. Xie, and D. Zhang, "Nonsinusoidal motion effects on energy extraction performance of a flapping foil," *Renewable energy*, vol. 64, pp. 283–293, 2014. 15

[40] N. Arora, A. Gupta, H. Aono, and W. Shyy, "Propulsion of a plunging flexible airfoil using a torsion spring model," in *33rd AIAA Applied Aerodynamics Conference*, Dallas, TX, United States, 06 2015, p. 3295. 15

[41] B. Zhu, P. Xia, Y. Huang, and W. Zhang, "Energy extraction properties of a flapping wing with an arc-deformable airfoil," *Journal of Renewable and Sustainable Energy*, vol. 11, no. 2, p. 023302, 03 2019. 16

[42] I. Gursul and D. Cleaver, "Plunging oscillations of airfoils and wings: progress, opportunities, and challenges," *AIAA Journal*, vol. 57, no. 9, pp. 3648–3665, 2019. 16

[43] H. Bao, W. Yang, D. Ma, W. Song, and B. Song, "Numerical simulation of flapping airfoil with alula," *International Journal of Micro Air Vehicles*, vol. 12, pp. 1–15, 08 2020. [Online]. Available from: https://journals.sagepub.com/doi/10.1177/1756829320977989 (accessed on 01-10-2023). 16

[44] W. Tao, S. Bifeng, S. Wenping, Y. Wenqing, X. Dong, and H. Zhonghua, "Lift performance enhancement for flapping airfoils by considering surging motion," *Chinese Journal of Aeronautics*, vol. 35, no. 9, pp. 194–207, 2022. 17

[45] E. A. Camacho, F. D. Marques, and A. R. Silva, "Influence of a deflectable leading-edge on a flapping airfoil," *Aerospace*, vol. 10, no. 7, p. 615, 2023. 17

[46] R. Lopes, E. Camacho, F. Neves, A. Silva, and J. Barata, "Numerical and experimental study of a plunging airfoil," in *4th Thermal and Fluids Engineering Conference*, Las Vegas, NV, United States, 04 2019, pp. 1869–1872. 18

[47] D. Rodrigues, E. A. Camacho, F. Neves, J. Barata, and A. R. Silva, "Plunging airfoil motion: Effects of unequal ascending and descending velocities," in *AIAA*

AVIATION 2020 FORUM, vol. 1. AIAA, 2020, pp. 1–11. [Online]. Available from: https://arc.aiaa.org/doi/abs/10.2514/6.2020-3042 (accessed on 05-08-2025). 18

[48] G. Torres, E. A. Camacho, F. D. Marques, and A. R. Silva, "Theoretical and numerical analysis of oscillating airfoil including viscous effects," in *AIAA SCITECH 2022 Forum*. AIAA 2022, 12 2021, p. 2026. [Online]. Available from: https://arc.aiaa.org/doi/abs/10.2514/6.2022-2026 (accessed on 05-8-2025). 18

[49] J. G. Silva, E. A. Camacho, and A. R. Silva, "Investigation of asymmetric plunging of a naca0012 airfoil," in *AIAA SCITECH 2023 Forum*. AIAA 2023, 01 2023, p. 1028. [Online]. Available from: https://arc.aiaa.org/doi/abs/10.2514/6.2023-1028 (accessed on 05-08-2025). 18

[50] E. A. Camacho, F. Neves, J. Barata, and A. R. Silva, "Plunging airfoil: Reynolds number and angle of attack effects," in *AIAA AVIATION 2020 FORUM*, vol. 1. AIAA 2020, 06 2020, p. 3040. [Online]. Available from: https://arc.aiaa.org/doi/abs/10.2514/6.2020-3040 (accessed on 05-08-2025). 18

[51] E. Camacho, F. Neves, A. Silva, and J. Barata, "Numerical investigation of frequency and amplitude influence on a plunging naca0012," *Energies*, vol. 13, no. 8, p. 1861, 2020. [Online]. Available from: https://www.mdpi.com/1996-1073/13/8/1861 (accessed on 05-08-2025). 18

[52] E. A. Camacho, F. M. Neves, F. D. Marques, J. M. Barata, and A. R. Silva, "Effects of a dynamic leading edge on a plunging airfoil," in *AIAA AVIATION 2021 FORUM*. AIAA 2021, 07 2021, p. 2839. [Online]. Available from: https://arc.aiaa.org/doi/abs/10.2514/6.2021-2839 (accessed on 05-08-2025). 18

[53] E. A. Camacho, F. D. Marques, A. R. Silva, and J. M. Barata, "Leading-edge parametric study of the naca0012-ik30 airfoil," in *AIAA AVIATION 2022 Forum*. AIAA, 06 2022, p. 3632. [Online]. Available from: https://arc.aiaa.org/doi/abs/10.2514/6.2022-3632 (accessed on 05-08-2025). 18

[54] E. A. Camacho, A. R. Silva, and F. D. Marques, "Optimal operation of the naca0012-ik30 airfoil," in *AIAA AVIATION 2023 Forum*. AIAA, 06 2023, p. 4439. [Online]. Available from: https://arc.aiaa.org/doi/abs/10.2514/6.2023-4439 (accessed on 05-08-2025). 18

[55] E. A. Camacho, A. R. Silva, and F. D. Marques, "Optimal leading-edge deflection for flapping airfoil propulsion," *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, vol. 237, no. 16, pp. 3640–3653, 2023. [Online]. Available from: https://doi.org/10.1177/09544100231201553 (accessed on 05-08-2025). 18

[56] E. A. R. Camacho, M. M. da Silva, A. R. R. Silva, and F. D. Marques, "Real-time optimization of wing drag and lift performance using a movable leading edge,"

Physics of Fluids, vol. 36, no. 1, p. 016128, 01 2024. [Online]. Available from: https://doi.org/10.1063/5.0185716 (accessed on 05-08-2025). 18

[57] E. A. R. Camacho, A. R. R. Silva, and F. D. Marques, "Dynamic stall mitigation using a deflectable leading edge: The ik30 mechanism," *Journal of Aerospace Engineering*, vol. 38, no. 2, p. 04024127, 2025. [Online]. Available from: https://ascelibrary.org/doi/abs/10.1061/JAEEEZ.ASENG-6040 (accessed on 05-08-2025). 18

[58] S. B. Gonçalves, E. A. Camacho, and A. R. Silva, "Influence of trailing-edge shape on the propulsive performance of a plunging flat plate," in *AIAA AVIATION 2022 Forum*. AIAA 2022, 06 2022, p. 4042. [Online]. Available from: https://arc.aiaa.org/doi/abs/10.2514/6.2022-4042 (accessed on 05-08-2025). 18

[59] J. Pinho, E. A. Camacho, and A. R. Silva, "Design and testing of a wing with a morphing trailing edge," in *AIAA AVIATION 2023 Forum*. AIAA 2020, 06 2023, p. 3752. [Online]. Available from: https://arc.aiaa.org/doi/abs/10.2514/6.2023-3752 (accessed on 05-08-2025). 18

[60] M. M. da Silva, E. A. Camacho, A. R. Silva, and F. D. Marques, "Thrust force assessment of a mfc-actuated tail-like robotic fish using unsteady panel method," *Mechatronics*, vol. 107, p. 103308, 2025. [Online]. Available from: https://www.sciencedirect.com/science/article/pii/S0957415825000170 (accessed on 05-08-2025). 18

[61] W. J. Mccroskey, K. W. Mcalister, L. W. Carr, S. L. Pucci, O. Lambert, and R. F. Indergrand, "Dynamic stall on advanced airfoil sections," *Journal of The American Helicopter Society*, vol. 26, pp. 40–50, 1981. 18

[62] M. Costes and H. Jones, "Computation of transonic potential flow on helicopter rotor blades," ONERA, Chatillon Cedex, France, Tech. Rep. ONERA, TP NO. 1987-136, 01 1987. 19

[63] A. Baron and M. Boffadossi, "Numerical simulation of unsteady rotor wakes," in *17th European Rotorcraft Forum*, Berlin, Germany, 1991, pp. 565–583. [Online]. Available from: https://dspace-erf.nlr.nl/items/16b9e6b7-6473-4600-8719-33c498779b3b (accessed on 01-10-2023). 19

[64] J. A. Ekaterinaris and M. F. Platzer, "Computational prediction of airfoil dynamic stall," *Progress in aerospace sciences*, vol. 33, no. 11-12, pp. 759–846, 1998. [Online]. Available from: https://www.sciencedirect.com/science/article/pii/S0376042197000122 xiii, 19, 20, 21

[65] J. Majhi and R. Ganguli, "Modeling helicopter rotor blade flapping motion considering nonlinear aerodynamics," *CMES - Computer Modeling in Engineering and Sciences*, vol. 27, pp. 25–36, 01 2008. [Online]. Available from:

https://www.researchgate.net/publication/251244776_Modeling_Helicopter_Rotor_Blade_Flapping_Motion_Considering_Nonlinear_Aerodynamics (accessed on 01-09-2023). 20

[66] M. R. Visbal, "Numerical investigation of deep dynamic stall of a plunging airfoil," *AIAA journal*, vol. 49, no. 10, pp. 2152–2170, 2011. 21

[67] B. Heine, K. Mulleners, G. Joubert, and M. Raffel, "Dynamic stall control by passive disturbance generators," *AIAA Journal*, vol. 51, no. 9, p. 3371, 09 2013. [Online]. Available from: https://www.researchgate.net/publication/225007489_Dynamic_Stall_Control_by_Passive_Disturbance_Generators (accessed on 01-10-2023). 22

[68] K. Gharali and D. A. Johnson, "Dynamic stall simulation of a pitching airfoil under unsteady freestream velocity," *Journal of Fluids and Structures*, vol. 42, pp. 228–244, 2013. 22

[69] B. Vieira and M. Maughmer, "Consideration of dynamic stall in rotorcraft airfoil design," *Annual Forum Proceedings - AHS International*, vol. 1, pp. 387–400, 01 2015. 22

[70] O. Perdomo and F.-S. Wei, "On the flapping motion of a helicopter blade," *Applied Mathematical Modelling*, vol. 46, pp. 299–311, 2017. [Online]. Available from: https://www.sciencedirect.com/science/article/pii/S0307904X17300616 (accessed on 01-09-2023). 22

[71] K. V. Truong, "Modeling aerodynamics, including dynamic stall, for comprehensive analysis of helicopter rotors," *Aerospace*, vol. 4, no. 2, 2017. [Online]. Available from: https://www.mdpi.com/2226-4310/4/2/21 (accessed on 01-09-2023). 23

[72] W. Geissler and B. G. van der Wall, "Dynamic stall control on flapping wing airfoils," *Aerospace Science and Technology*, vol. 62, pp. 1–10, 2017. 24

[73] Q. Wang and Q. Zhao, "Unsteady aerodynamic characteristics simulations of rotor airfoil under oscillating freestream velocity," *Applied Sciences*, vol. 10, no. 5, p. 1822, 2020. 24

[74] F. Samara and D. A. Johnson, "Dynamic stall on pitching cambered airfoil with phase offset trailing edge flap," *AIAA Journal*, vol. 58, no. 7, pp. 2844–2856, 7 2020. 24

[75] E. Costa and A. Simões, *Inteligência artificial: fundamentos e aplicações*, 2nd ed. FCA, 09 2008. 25

[76] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943. 25

[77] D. O. Hebb, "The first stage of perception: growth of the assembly," *The Organization of Behavior*, vol. 4, no. 60, pp. 78–60, 1949. 27, 51

[78] B. Widrow and M. E. Hoff, "(1960) bernard widrow and marcian e. hoff, "adaptive switching circuits," 1960 ire wescon convention record, new york: Ire, pp. 96-104," in *Neurocomputing, Volume 1: Foundations of Research*. The MIT Press, 04 1988. [Online]. Available from: https://doi.org/10.7551/mitpress/4943.003.0012 28

[79] M. Basirat and P. M. Roth, "The quest for the golden activation function," 2018. [Online]. Available from: https://arxiv.org/abs/1808.00783 (accessed on 01-06-2023). xiii, 29, 35, 50

[80] R. Rojas, *Weighted Networks—The Perceptron*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 55–76. [Online]. Available from: https://doi.org/10.1007/978-3-642-61068-4_3 (accessed on 01-07-23). 29

[81] R. J. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent networks and their computational complexity," in *Backpropagation*. Psychology Press, 2013, pp. 433–486. 30

[82] W.-F. Chang and M.-W. Mak, "A conjugate gradient learning algorithm for recurrent neural networks," *Neurocomputing*, vol. 24, no. 1-3, pp. 173–189, 1999. 30

[83] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," 2012. [Online]. Available from: https://arxiv.org/abs/1206.5533 31, 32

[84] R. Kiros, "Training neural networks with stochastic hessian-free optimization," *arXiv preprint arXiv:1301.3641*, 2013. 32

[85] J. Martens and I. Sutskever, "Learning recurrent neural networks with hessian-free optimization," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 1033–1040. 32

[86] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014. 32

[87] J. Bradbury, S. Merity, C. Xiong, and R. Socher, "Quasi-recurrent neural networks," *arXiv preprint arXiv:1611.01576*, 2016. 33

[88] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *arXiv preprint arXiv:1609.04836*, 2016. 33

[89] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016. 33, 34

[90] R. C. Staudemeyer and E. R. Morris, "Understanding lstm–a tutorial into long short-term memory recurrent neural networks," *arXiv preprint arXiv:1909.09586*, 2019. xiii, 35, 36

[91] J. Lederer, "Activation functions in artificial neural networks: A systematic overview," *arXiv preprint arXiv:2101.09957*, 2021. 36

[92] F. D. Marques, L. d. F. Souza, D. C. Rebolho, A. S. Caporali, E. M. Belo, and R. L. Ortolan, "Application of time-delay neural and recurrent neural networks for the identification of a hingeless helicopter blade flapping and torsion motions," *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, vol. 27, pp. 97–103, 2005. 37, 43

[93] D. Ignatyev, "Simulation of unsteady aerodynamic characteristics of aircraft at high angles of attack using neural networks," in *5-th European Conference For Aeronautics And Space Sciences (EUCASS)*, 2013. 38, 43

[94] R. Han, Y. Wang, Y. Zhang, and G. Chen, "A novel spatial-temporal prediction method for unsteady wake flows based on hybrid deep neural network," *Physics of Fluids*, vol. 31, no. 12, 2019. 38

[95] X. Hui, J. Bai, H. Wang, and Y. Zhang, "Fast pressure distribution prediction of airfoils using deep learning," *Aerospace Science and Technology*, vol. 105, p. 105949, 2020. 39

[96] M. I. Zafar, M. M. Choudhari, P. Paredes, and H. Xiao, "Recurrent neural network for end-to-end modeling of laminar-turbulent transition," *Data-Centric Engineering*, vol. 2, p. e17, 2021. 39

[97] J. Zhou, H. Xu, Z. Li, S. Shen, and F. Zhang, "Control of a tail-sitter vtol uav based on recurrent neural networks," *arXiv preprint arXiv:2104.02108*, 2021. 39

[98] M.-Y. Wu, Y. Wu, X.-Y. Yuan, Z.-H. Chen, W.-T. Wu, and N. Aubry, "Fast prediction of flow field around airfoils based on deep convolutional neural network," *Applied Sciences*, vol. 12, no. 23, p. 12075, 2022. 40

[99] W. Peng, Y. Zhang, E. Laurendeau, and M. C. Desmarais, "Learning aerodynamics with neural network," *Scientific Reports*, vol. 12, no. 1, p. 6779, 2022. 40

[100] M. Yan, Z. Zhang, S. Gao, and S. Cao, "Predicting aerodynamic pressure on a square cylinder from wake velocity field by masked gated recurrent unit model," *Physics of Fluids*, vol. 34, no. 11, 2022. 41

[101] B. Lan, Y.-J. Lin, Y.-H. Lai, C.-H. Tang, and J.-T. Yang, "A neural network approach to estimate transient aerodynamic properties of a flapping wing system," *Drones*, vol. 6, no. 8, 2022. [Online]. Available from: https://www.mdpi.com/2504-446X/6/8/210 (accessed on 01-06-2023). 41

[102] S. Ahmed, K. Kamal, T. A. H. Ratlamwala, S. Mathavan, G. Hussain, M. Alkahtani, and M. B. M. Alsultan, "Aerodynamic analyses of airfoils using machine learning as an alternative to rans simulation," *Applied Sciences*, vol. 12, no. 10, p. 5194, 2022. 42

[103] J. A. Pereira, E. A. Camacho, F. D. Marques, and A. R. Silva, "Fast flapping aerodynamics prediction using a recurrent neural network," *Engineering Proceedings*, vol. 56, no. 1, p. 219, 2023. [Online]. Available from: https://doi.org/10.3390/ASEC2023-16272 (accessed on 15-11-2023). 43, 55, 60, 73

[104] J. A. Pereira, E. A. Camacho, F. D. Marques, and A. R. Silva, "Flapping airfoil aerodynamics using recurrent neural network," in *AIAA SCITECH 2024 Forum*, 2024, p. 1982. [Online]. Available from: https://arc.aiaa.org/doi/abs/10.2514/6.2024-1982 (accessed on 01-02-2024). 43, 55, 68, 78

[105] E. A. Camacho, A. R. Silva, and F. D. Marques, "Predicting airfoil dynamic stall loads using neural networks," *Aerospace Science and Technology*, vol. 165, p. 110466, 2025. [Online]. Available from: https://www.sciencedirect.com/science/article/pii/S1270963825005371 (accessed on 05-08-2025). 43

[106] M. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015. [Online]. Available from: http://neuralnetworksanddeeplearning.com/ (accessed on 2020-03-02). xix, 61, 63, 64

[107] E. A. R. Camacho, F. D. Marques, and A. R. R. Silva, "Predicting the naca0012-ik30 airfoil propulsive capabilities with a panel method," *Engineering Proceedings*, vol. 56, no. 1, p. 193, 2023. [Online]. Available from: https://www.mdpi.com/2673-4591/56/1/193 (accessed on 01-09-2023). 66

Appendix A

Training and Validation Cases

Table A.1: Validation set cases for pure plunging

Case [−]	Re [−]	h [−]	k [−]	A_α [−]	α_{mean} [$degrees$]	Case [−]	Re [−]	h [−]	k [−]	A_α [−]	α_{mean} [$degrees$]
1	10000	1.5	0.015	0.0	3.75	25	10000	1.5	0.073	0.0	3.75
2	10000	1.5	0.009	0.0	3.75	26	10000	1.5	0.044	0.0	3.75
3	10000	2.5	0.015	0.0	3.75	27	10000	2.5	0.073	0.0	3.75
4	10000	2.5	0.009	0.0	3.75	28	10000	2.5	0.044	0.0	3.75
5	10000	1.5	0.015	0.0	5.00	29	10000	1.5	0.073	0.0	5.00
6	10000	1.5	0.009	0.0	5.00	30	10000	1.5	0.044	0.0	5.00
7	10000	2.5	0.015	0.0	5.00	31	10000	2.5	0.073	0.0	5.00
8	10000	2.5	0.009	0.0	5.00	32	10000	2.5	0.044	0.0	5.00
9	10000	1.5	0.015	0.0	6.25	33	10000	1.5	0.073	0.0	6.25
10	10000	1.5	0.009	0.0	6.25	34	10000	1.5	0.044	0.0	6.25
11	10000	2.5	0.015	0.0	6.25	35	10000	2.5	0.073	0.0	6.25
12	10000	2.5	0.009	0.0	6.25	36	10000	2.5	0.044	0.0	6.25
13	10000	1.5	0.044	0.0	3.75	37	10000	1.5	0.118	0.0	3.75
14	10000	1.5	0.026	0.0	3.75	38	10000	1.5	0.071	0.0	3.75
15	10000	2.5	0.044	0.0	3.75	39	10000	2.5	0.118	0.0	3.75
16	10000	2.5	0.026	0.0	3.75	40	10000	2.5	0.071	0.0	3.75
17	10000	1.5	0.044	0.0	5.00	41	10000	1.5	0.118	0.0	5.00
18	10000	1.5	0.026	0.0	5.00	42	10000	1.5	0.071	0.0	5.00
19	10000	2.5	0.044	0.0	5.00	43	10000	2.5	0.118	0.0	5.00
20	10000	2.5	0.026	0.0	5.00	44	10000	2.5	0.071	0.0	5.00
21	10000	1.5	0.044	0.0	6.25	45	10000	1.5	0.118	0.0	6.25
22	10000	1.5	0.026	0.0	6.25	46	10000	1.5	0.071	0.0	6.25
23	10000	2.5	0.044	0.0	6.25	47	10000	2.5	0.118	0.0	6.25
24	10000	2.5	0.026	0.0	6.25	48	10000	2.5	0.071	0.0	6.25

Table A.2: Training set cases for pure plunging.

Case [−]	Re [−]	h [−]	k [−]	A_α [−]	α_{mean} [$degrees$]	Case [−]	Re [−]	h [−]	k [−]	A_α [−]	α_{mean} [$degrees$]
1	10000	0.5	0.175	0.0	0.0	51	10000	0.5	0.044	0.0	7.5
2	10000	0.5	0.087	0.0	0.0	52	10000	0.5	0.029	0.0	7.5
3	10000	0.5	0.044	0.0	0.0	53	10000	1.0	0.175	0.0	7.5
4	10000	0.5	0.029	0.0	0.0	54	10000	1.0	0.087	0.0	7.5
5	10000	1.0	0.175	0.0	0.0	55	10000	1.0	0.044	0.0	7.5
6	10000	1.0	0.087	0.0	0.0	56	10000	1.0	0.029	0.0	7.5
7	10000	1.0	0.044	0.0	0.0	57	10000	2.0	0.175	0.0	7.5
8	10000	1.0	0.029	0.0	0.0	58	10000	2.0	0.087	0.0	7.5
9	10000	2.0	0.175	0.0	0.0	59	10000	2.0	0.044	0.0	7.5
10	10000	2.0	0.087	0.0	0.0	60	10000	2.0	0.029	0.0	7.5
11	10000	2.0	0.044	0.0	0.0	61	10000	3.0	0.175	0.0	7.5
12	10000	2.0	0.029	0.0	0.0	62	10000	3.0	0.087	0.0	7.5
13	10000	3.0	0.175	0.0	0.0	63	10000	3.0	0.044	0.0	7.5
14	10000	3.0	0.087	0.0	0.0	64	10000	3.0	0.029	0.0	7.5
15	10000	3.0	0.044	0.0	0.0	65	10000	0.5	0.175	0.0	10.0
16	10000	3.0	0.029	0.0	0.0	66	10000	0.5	0.087	0.0	10.0
17	10000	0.5	0.175	0.0	2.5	67	10000	0.5	0.044	0.0	10.0
18	10000	0.5	0.087	0.0	2.5	68	10000	0.5	0.029	0.0	10.0
19	10000	0.5	0.044	0.0	2.5	69	10000	1.0	0.175	0.0	10.0
20	10000	0.5	0.029	0.0	2.5	70	10000	1.0	0.087	0.0	10.0
21	10000	1.0	0.175	0.0	2.5	71	10000	1.0	0.044	0.0	10.0
22	10000	1.0	0.087	0.0	2.5	72	10000	1.0	0.029	0.0	10.0
23	10000	1.0	0.044	0.0	2.5	73	10000	2.0	0.175	0.0	10.0
24	10000	1.0	0.029	0.0	2.5	74	10000	2.0	0.087	0.0	10.0
25	10000	2.0	0.175	0.0	2.5	75	10000	2.0	0.044	0.0	10.0
26	10000	2.0	0.087	0.0	2.5	76	10000	2.0	0.029	0.0	10.0
27	10000	2.0	0.044	0.0	2.5	77	10000	3.0	0.175	0.0	10.0
28	10000	2.0	0.029	0.0	2.5	78	10000	3.0	0.087	0.0	10.0
29	10000	3.0	0.175	0.0	2.5	79	10000	3.0	0.044	0.0	10.0
30	10000	3.0	0.087	0.0	2.5	80	10000	3.0	0.029	0.0	10.0
31	10000	3.0	0.044	0.0	2.5	81	10000	0.5	0.087	0.0	0.0
32	10000	3.0	0.029	0.0	2.5	82	10000	0.5	0.044	0.0	0.0
33	10000	0.5	0.175	0.0	5.0	83	10000	0.5	0.022	0.0	0.0
34	10000	0.5	0.087	0.0	5.0	84	10000	0.5	0.015	0.0	0.0
35	10000	0.5	0.044	0.0	5.0	85	10000	1.0	0.087	0.0	0.0
36	10000	0.5	0.029	0.0	5.0	86	10000	1.0	0.044	0.0	0.0
37	10000	1.0	0.175	0.0	5.0	87	10000	1.0	0.022	0.0	0.0
38	10000	1.0	0.087	0.0	5.0	88	10000	1.0	0.015	0.0	0.0
39	10000	1.0	0.044	0.0	5.0	89	10000	2.0	0.087	0.0	0.0
40	10000	1.0	0.029	0.0	5.0	90	10000	2.0	0.044	0.0	0.0
41	10000	2.0	0.175	0.0	5.0	91	10000	2.0	0.022	0.0	0.0
42	10000	2.0	0.087	0.0	5.0	92	10000	2.0	0.015	0.0	0.0
43	10000	2.0	0.044	0.0	5.0	93	10000	3.0	0.087	0.0	0.0
44	10000	2.0	0.029	0.0	5.0	94	10000	3.0	0.044	0.0	0.0
45	10000	3.0	0.175	0.0	5.0	95	10000	3.0	0.022	0.0	0.0
46	10000	3.0	0.087	0.0	5.0	96	10000	3.0	0.015	0.0	0.0
47	10000	3.0	0.044	0.0	5.0	97	10000	0.5	0.087	0.0	2.5
48	10000	3.0	0.029	0.0	5.0	98	10000	0.5	0.044	0.0	2.5
49	10000	0.5	0.175	0.0	7.5	99	10000	0.5	0.022	0.0	2.5
50	10000	0.5	0.087	0.0	7.5	100	10000	0.5	0.015	0.0	2.5

Appendix B

Validation Scores

Table B.1: Validation results for the network without delay initialization.

Case	C_l		C_m		Case	C_l		C_m	
	MSE $[\cdot 10^{-3}]$	R^2	MSE $[\cdot 10^{-6}]$	R^2		MSE $[\cdot 10^{-3}]$	R^2	MSE $[\cdot 10^{-6}]$	R^2
1	0.357	0.968	0.971	0.944	25	0.973	0.996	2.996	0.992
2	0.424	0.894	0.494	0.922	26	0.420	0.996	1.889	0.987
3	0.320	0.990	1.831	0.962	27	1.392	0.998	4.998	0.995
4	0.299	0.973	1.065	0.939	28	0.566	0.998	2.430	0.994
5	0.504	0.954	1.620	0.906	29	0.815	0.997	2.039	0.995
6	0.526	0.869	1.793	0.715	30	0.432	0.995	1.929	0.987
7	0.379	0.988	1.217	0.975	31	1.085	0.998	4.573	0.996
8	0.443	0.960	0.858	0.951	32	0.549	0.998	2.201	0.995
9	0.611	0.945	2.248	0.869	33	0.654	0.997	1.584	0.996
10	0.614	0.849	2.221	0.645	34	0.455	0.995	1.363	0.991
11	0.575	0.981	1.163	0.976	35	0.802	0.999	5.086	0.995
12	0.605	0.946	0.591	0.966	36	0.571	0.998	2.600	0.994
13	0.419	0.996	1.893	0.987	36	0.571	0.998	2.600	0.994
14	0.333	0.990	1.792	0.968	37	2.356	0.996	4.443	0.995
15	0.564	0.998	2.417	0.994	38	0.927	0.996	2.860	0.992
16	0.337	0.997	2.102	0.986	39	2.642	0.998	21.986	0.992
17	0.431	0.995	1.926	0.987	40	1.332	0.998	4.565	0.996
18	0.423	0.988	1.651	0.970	41	1.876	0.997	3.858	0.996
19	0.547	0.998	2.204	0.995	42	0.776	0.997	2.037	0.995
20	0.403	0.996	2.658	0.983	43	1.852	0.999	20.592	0.992
21	0.455	0.995	1.364	0.991	44	1.061	0.998	4.236	0.996
22	0.550	0.985	1.767	0.968	45	1.470	0.998	4.530	0.995
23	0.570	0.998	2.610	0.994	46	0.640	0.997	1.640	0.996
24	0.496	0.995	2.165	0.986	47	1.464	0.999	19.756	0.993
25	0.973	0.996	2.996	0.992	48	0.803	0.999	4.689	0.995

Table B.2: Validation results for the network with delay initialization.

Case	C_l		C_m		Case	C_l		C_m	
	MSE [$\cdot 10^{-3}$]	R^2	MSE [$\cdot 10^{-6}$]	R^2		MSE [$\cdot 10^{-3}$]	R^2	MSE [$\cdot 10^{-6}$]	R^2
1	0.115	0.987	1.085	0.937	25	0.218	0.999	3.882	0.990
2	0.109	0.973	0.421	0.933	26	0.159	0.998	3.083	0.979
3	0.283	0.991	3.320	0.931	27	0.495	0.999	14.534	0.987
4	0.224	0.980	1.618	0.908	28	0.469	0.998	7.973	0.981
5	0.205	0.981	0.848	0.951	29	0.492	0.998	5.022	0.987
6	0.203	0.950	0.423	0.933	30	0.361	0.996	3.675	0.975
7	0.392	0.987	2.156	0.955	31	0.899	0.999	14.370	0.987
8	0.368	0.967	1.259	0.928	32	0.755	0.997	9.245	0.978
9	0.327	0.971	0.961	0.944	33	0.892	0.996	7.022	0.982
10	0.229	0.944	0.751	0.880	34	0.651	0.993	4.776	0.968
11	0.636	0.979	2.233	0.953	35	1.589	0.998	14.979	0.986
12	0.467	0.959	0.855	0.951	36	1.011	0.996	11.395	0.972
13	0.158	0.998	3.081	0.979	37	0.185	0.999	7.045	0.993
14	0.185	0.995	2.635	0.952	38	0.221	0.999	3.802	0.990
15	0.468	0.998	7.961	0.981	39	0.760	0.999	21.584	0.992
16	0.345	0.996	6.421	0.958	40	0.506	0.999	13.903	0.987
17	0.360	0.996	3.668	0.975	41	0.644	0.999	6.697	0.993
18	0.258	0.993	1.786	0.968	42	0.490	0.998	4.942	0.987
19	0.756	0.997	9.222	0.978	43	1.937	0.999	21.358	0.992
20	0.542	0.994	5.187	0.966	44	0.893	0.999	13.842	0.987
21	0.650	0.993	4.759	0.968	45	1.650	0.997	7.580	0.992
22	0.505	0.986	2.206	0.960	46	0.854	0.996	6.907	0.981
23	1.009	0.996	11.369	0.972	47	3.664	0.998	21.089	0.992
24	0.836	0.992	5.808	0.962	48	1.531	0.998	14.613	0.986