People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

Blida 1 University

Institute of Aeronautics and Space Studies

MASTER REPORT

In Partial Fulfillment of the Requirements for the Degree of Master in Aeronautics

Specialty: Propulsion

A Machine Learning Approach for Simulation of Fluid Flow

by

Anis TCHERAK

Supervisors: **Mr. Tahar REZOUG**

Mr. Abderrahmane BELKALLOUCHE

Academic Year 2020 - 2021

Acknowledgements

In the Name of Allah, the Most Merciful, the Most Compassionate all praise be to Allah, the Lord of the world, and prayers and peace be upon Mohamed His servant and messenger.

First and foremost, I must acknowledge my limitless thanks to Allah, the Ever Magnificent, the Ever Thankful, for His help and bless. I am totally sure that this work would have never become truth, without His guidance.

I thank you very much my supervisor, Professor ***Tahar REZOUG*** *who helped me and being with me during my work, who guides me while I'm working by making the hardest steps easier on me and brought my work to a higher level by teaching me all what I need and make me learn from my mistakes by his precious advices. That Professor who believes in my talents and makes it real in my project.*

I thank you very much too, my co-supervisor, Professor ***Abderrahmane BELKALLOUCHE*** *for encouraging and helping me in my research and for allowing me to take a step forward and for your precious advices. I also want to acknowledge you for your patient support and for your valuable guidance during this project.*

I would like to thank the members of the jury for agreeing to judge my work while assuring them of my sincere gratitude and deep respect. I wish, however, their patience and indulgence in my modest work.

My thanks go also to all my teachers who shared their knowledge and experiences with me throughout my university studies and who allowed me to reach the level of expertise necessary for the realization of this thesis and the accomplishment of myself. I address my thanks to all those who directly or indirectly contributed to the development of this thesis without forgetting my parents, all my relatives and friends, who have always supported and encouraged me during all these years.

Thank's everyone.

Anis TCHERAK

Dedications

With a blissful joy and a big happiness, I dedicate this humble work, fruit of moments of pain and glory,

To all those who are dear to me:

To the memory of my dear uncle LAMINE SALIM

To my dear parents "Samia" and "M'hamed" for their unwavering love, their support, their encouragement, their enormous sacrifices, they have always helped and supported me during all these years.

To my brother "Amine" for their encouragement, support, help and availability.

To all my family: Grandparents, aunts and uncles, cousins and cousins and all my friends.

Anis TCHERAK

Abstract

Solving fluid dynamics problems mainly rely on experimental methods and CFD (Computational Fluid Dynamics) based numerical simulations. However, in experimental methods it is difficult to simulate the physical problems in reality and there is also a high-cost to the economy, while CFD simulation are sensitive about meshing a complicated structure. It is also time-consuming due to the billion degrees of freedom in relevant spatial-temporal flow fields. Therefore, constructing a cost-effective model to settle fluid dynamics problems is of significant meaning. Physics-informed machine learning has drawn tremendous interest in recent years to solve computational physics problems, whose basic concept is to embed physical laws to constrain/inform neural networks, with the need of less data for training a reliable model. This can be achieved by incorporating the residual of physics equations into the loss function. Through minimizing the loss function, the network could approximate the solution. In this report, we propose a physics-informed neural network (PINN) to solve fluid flows problems governed by the Navier-Stokes equations. We apply the proposed PINN to simulate steady incompressible laminar flows at low Reynolds numbers. The predicted velocity and pressure fields by the proposed PINN approach are also compared with the reference numerical solutions. Simulation results demonstrate great potential of the proposed PINN for fluid flow simulation with a high accuracy.

Keywords: Machine Learning, neural network, Physics-informed neural network (PINN), Computational Fluid Dynamics (CFD), Navier-Stokes equations, numerical simulation, fluid flow.

Résumé

La résolution de problèmes de dynamique des fluides repose principalement sur des méthodes expérimentales et des simulations numériques basées sur la CFD (*Computational Fluid Dynamics*). Cependant, dans les méthodes expérimentales, il est difficile de simuler les problèmes physiques dans la réalité et il y a aussi un coût élevé pour l'économie, tandis que la simulation CFD est sensible au maillage d'une structure complexe. Cela prend également du temps en raison des milliards de degrés de liberté dans les champs d'écoulement spatio-temporels pertinents. Par conséquent, la construction d'un modèle rentable pour résoudre les problèmes de dynamique des fluides est d'une importance significative. L'apprentissage automatique basé sur la physique a suscité un vif intérêt ces dernières années pour résoudre des problèmes de physique computationnelle, dont le concept de base est d'intégrer des lois physiques pour contraindre/informer les réseaux de neurones, avec le besoin de moins de données pour former un modèle fiable. Ceci peut être réalisé en incorporant le résidu des équations physiques dans la fonction de perte. En minimisant la fonction de perte, le réseau pourrait se rapprocher de la solution. Dans ce rapport, nous proposons un réseau neuronal basé sur la physique (PINN) pour résoudre les problèmes d'écoulement de fluides régis par les équations de Navier-Stokes. Nous appliquons le PINN proposé pour simuler des écoulements laminaires incompressibles stationnaires à de faibles nombres de Reynolds. Les champs de vitesse et de pression prédits par l'approche PINN proposée sont également comparés aux solutions numériques de référence. Les résultats de simulation démontrent le grand potentiel du PINN proposé pour la simulation d'écoulement de fluide avec une grande précision.

Mots clés : apprentissage automatique, réseau de neurones, réseau de neurones informés par la physique (PINN), *Computational Fluid Dynamics* (CFD), équations de Navier-Stokes, simulation numérique, écoulement des fluides.

ملخص

يعتمد حل مشاكل ديناميكا الموائع بشكل أساسي على الطرق التجريبية والمحاكاة العددية القائمة على CFD (ديناميكا السوائل الحسابية). ومع ذلك، في الأساليب التجريبية، من الصعب محاكاة المشكلات الفزيائية في الواقع وهناك أيضًا تكلفة عالية للاقتصاد، في حين أن محاكاة CFD حساسة لشبكة البنية المعقدة. كما أنها تستغرق وقتًا طويلاً نظرًا لمليار درجة من الحرية في مجالات التدفق المكاني والزماني ذات الصلة. لذلك، فإن بناء نموذج فعال من حيث التكلفة لحل مشاكل ديناميكيات الموائع له أهمية كبيرة. لقد ولّد التعلم الآلي المستنير بالفيزياء قدراً كبيراً من الاهتمام في السنوات الأخيرة لحل مشاكل الفيزياء الحسابية، والمفهوم الأساسي لها هو دمج القوانين الفيزيائية لتقييد / إعلام الشبكات العصبية، مع الحاجة إلى بيانات أقل لتشكيل نموذج موثوق. يمكن تحقيق ذلك من خلال دمج بقايا المعادلات الفزيائية في دالة الخسارة. بتقليل دالة الخسارة، يمكن للشبكة أن تقترب من الحل. في هذا التقرير، نقترح شبكة عصبية قائمة على الفيزياء (PINN) لحل مشاكل تدفق السوائل التي تحكمها معادلات نافيير-ستوكس. نطبق PINN المقترح لمحاكاة التدفقات الصفائحية الثابتة غير القابلة للضغط عند أرقام رينولدز المنخفضة. تتم أيضًا مقارنة مجالات السرعة والضغط التي تنبأ بها نهج PINN المقترح بالحلول المرجعية الرقمية. توضح نتائج المحاكاة الإمكانات الكبيرة لـ PINN المقترح لمحاكاة تدفق السوائل بدقة عالية.

الكلمات المفتاحية : التعلم الآلي ، الشبكة العصبية ، الشبكة العصبية المستنيرة للفيزياء (PINN) ، ديناميكيات السوائل الحاسوبية (CFD) ، معادلات نافيير ستوكس ، المحاكاة العددية ، تدفق السوائل.

Table of Contents

List of Figures

CHAPTER 1

Introduction

1.1. Thesis Overview

Fluid dynamics is an engineering field and a subdiscipline of fluid mechanics that describes the behaviour of fluid (liquid as water or gas as air) flow. This field is omnipresent in many disciplines such as aerospace, mechanical, chemistry, environmental, biology, biomedical, nuclear, etc. Let us take examples to illustrate the ubiquity of fluid dynamics, from the air flow around a wing that provides lift for the airplane to the flow of a coolant in an automobile radiator or air-conditioning system to the unsteady, turbulent, compressible flow of air or steam in turbomachinery used to produce aeronautical propulsion and electrical power [18]. All of these examples show the importance of fluid dynamics and the need to understand it.

Any physical phenomena (e.g. fluid flow) can be described with a mathematical model, for example, by using a set of partial differential equations (PDEs), and then the model can be solved analytically to obtain the quantities of interest (e.g. velocity and pressure fields).

Fluid dynamics problems such as cavity flow, pipeline flow, and flow around bluff body, is typically governed by the Navier-Stokes (N-S) equations, which is a highly nonlinear PDEs system [15]. Due to this nature of N-S equations, finding an analytical solution for them remains a daunting and complicated task. That is why, solving fluid flow problems mainly rely on experimental methods and numerical simulations [40].

An experimental study begins by reproducing the physical problem in a laboratory, and then using instrumentation (e.g. sensors), the physical quantities can be determined. Wind tunnel experiments are an example, where aerodynamicists test models of proposed aircraft and engine

components [56]. During a test, the model is placed in the test section of the tunnel and air is made to flow past the model. In this way, aerodynamicists can visualize the air flow, visualizing structure of the boundary layer, measure pressures and velocity, determine forces and moments (lift, drag, and pitch) on the model, etc.

Despite the usefulness of the experimental methods, the latter have major drawbacks: the difficulty to simulate the physical problems in reality, high-cost to the economy, time-consuming process, etc. These constraints forced engineers to find a new method of studying physical problems.

After years of research and with the considerable development of computers, a new field has emerged, which is Computational Fluid Dynamics (CFD).

CFD simulation is based on discretization and approximation of the underlying differential equations that govern the behaviour of fluids (i.e. N-S equations) using a computational grid (discretization of computational domain) and numerical methods such as finite difference, finite volume, or finite element methods [16], then, linear algebra is used to solve the discretized equations [6].

Nowadays, CFD is heavily applied in many engineering areas and play an important role in modelling various physical phenomena, such as weather, climate, aerodynamics, etc [44]. It can also replace cost and time intensive experiments and even reduce the need for prototypes. However, CFD simulations are often computationally cumbersome, especially for the flows with turbulence and complex geometries, and when dealing with billion degrees of freedom in relevant spatial-temporal flow fields. Moreover, mesh generation also usually incurs a huge burden, in particular when moving boundary or large geometric variation is considered [15]. Furthermore, the CFD method has significant limitations in dealing with special mesh (e.g., moving mesh) which is difficult to converge.

Faced with constraints encountered with both experimental study and CFD approach, it is no wonder that researchers are always looking for new and better ways to solve physical problems. In the last years, they are interested with a field that is currently getting a lot of attention which is Machine Learning (ML), more specific, Artificial Neural Networks (ANNs).

ML is a type of Artificial Intelligence (AI) that provides computers with the ability to learn (through experience and by the use of data) without being explicitly programmed [10]. It facilitates automation of tasks and augment human domain knowledge [30].

In the past decade, ML has given us self-driving cars, practical speech recognition, effective web search, and a vastly improved understanding of the human genome. ML is so pervasive today that you probably use it dozens of times a day without knowing it [45].

ML is based on several algorithms, one of particular interest is the Artificial Neural Networks (ANNs) which are a computing system inspired by the biological neural networks that constitute human brain [10]. ANNs are one of the core technologies in the rapidly growing field of ML and are widely used in various complex cognitive tasks such as visual object recognition, video analysis, mathematical optimization, etc [8].

All of these successes of ML, in particular ANNs, explain the increasing attention from the scientific community on the capabilities of such methods, and their possible applications to diverse research fields.

In recent years, the field of numerical computation known a major breakthrough thanks to ANNs, especially with the creation of the field of Physics-Informed Neural Networks (PINNs) by the authors Raissi et al. in 2019 [1]. These latter had an ingenious idea by combining ANNs and physical principle constraints in the form of PDEs, thus forming the PINNs field.

PINNs are a new class of numerical methods for solving PDEs [29]. Compared to conventional methods (finite difference, finite volume, finite element), they have enormous advantages: PINNs are totally mesh-free methods which allow him to solve accurately and efficiently differential equation defined on complex domain where classical methods are inefficient [7]. PINNs have potential to solve high-dimensional equations while the classical methods have rapidly increasing complexity with increasing dimension of problem [3] (and many others advantages that we will cite in Chapter 2).

In the CFD field, the appearance of the new PINNs solvers triggered a real enthusiasm, *"we even think of replacing the commercial solvers based on classical numerical methods with Machine Learning solvers"*.

1.2. Thesis Objectives

The main objective of this thesis is to employ PINNs for solving fluid dynamics problems governed by the N-S equations. The fluid problems for which the PINNs framework will be tested on, are the steady two-dimensional incompressible laminar viscous flow around NACA 0012 airfoil with different angle of attack.

To validate the PINNs results, we will make a comparative study with reference solutions based on finite volume method (given by CFD solvers).

1.3. Thesis Outline

Chapter 2 Machine Learning Applied to CFD

Presents a theoretical background and fundaments of both CFD, Machine Learning approach, and subsequently Artificial Neural Network, then, the step to Physics-Informed Neural Networks is made.

Chapter 3 Preliminary Examples

Presents two preliminary examples to understand the Physics-Informed Neural Networks approach.

Chapter 4 Implementation and Results

An implementation of Physics-Informed Neural Networks framework to solve the Navier-Stokes equations is presented, then, to validate the PINNs results, we compare them with a reference solution.

Chapter 5 Summary and Conclusions

Presents the summary and conclusions found during the present study, along with future works.

CHAPTER 2

Machine Learning Applied to CFD

2.1. Fundamentals of CFD

2.1.1. Introduction

Methods in Computational Fluid Dynamics (CFD) aim to find numerical solutions to the differential equations governing the behavior of fluids. Depending on the concrete problem, this includes finding valid solution to field quantities such as fluid velocity, pressure, density and inner energy [16]. Since analytical solutions to the governing equations can only be found for a very small subset of problems, the goal of a CFD simulation, in general, is to find approximate solutions to those physical quantities using numerical methods. In theory, with unlimited computing power, numerically obtained solutions can be arbitrarily accurate [18]. In practice, however, the limits of even today's most performant supercomputers limit the accuracy of CFD simulations [38].

2.1.2. Equations of fluid flow

The motion of fluids can be described by three conservation laws: continuity equation, momentum equation, and the energy equation. In combination, these equations describe the state of the fluid in its entirety. In order to find a closed solution to these equations, additional constraints are needed. These constraints can be enforced by an additional equation of state (i.e. the ideal gas law) [63].

The continuity equation (conservation of mass) in differential form is defined as follows:

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho u_i)}{\partial x_i} = 0 \tag{2.1}$$

The variable ρ denotes the fluid density, and u_i denotes the velocity component in direction x_i. The continuity equations enforce that the rate of change of mass within a control volume must be equal to the mass flux through the surfaces of the control volume [64].

For incompressible fluids (i.e. liquid water can be approximated to be incompressible) the density ρ is constant and does not change over time, which simplifies the continuity equation to:

$$\frac{\partial u_i}{\partial x_i} = 0$$

The fluid flow is divergence free and contains no sinks or sources.

The momentum equation (conservation of momentum) enforces that the rate of change in momentum of a control volume must be equal to the momentum flux through the surfaces of the control volume [64]. The conservation of momentum equation can be expressed as follows:

$$\frac{\partial(\rho u_i)}{\partial t} + \frac{\partial(\rho u_i u_j)}{\partial x_j} + \frac{\partial(p\delta_{ij})}{\partial x_j} - \frac{\partial \sigma_{ij}}{\partial x_j} - \rho f_i = 0 \qquad (2.2)$$

Note that equation (2.2) yields three independent equations for the three spatial dimensions. The variable ρ denotes the density, u_i denotes the flow velocity component in direction x_i, p denotes the pressure and σ_{ij} denotes the viscous stress tensor. The viscous stress tensor has 9 entries for three-dimensional fluid flow. The variable δ_{ij} denotes the Kronecker delta which equals 1 for $i = j$, and equals 0 otherwise. The variable f_i denotes the external forces acting on the control volume in direction i [74]. In this work, transient flow phenomena will be neglected, meaning that time derivatives of field quantities can be assumed to be equal to zero. The remaining equations describe the behavior of a steady-state fluid flow problem.

An additional equation, the energy equation measures the flux of energy through the control volume. For non-reacting flows without phase changes and without heat conduction, this equation can be omitted [63].

In general, the values of the stress tensor are not known rendering finding solutions to equations (2.1) and (2.2) impossible. However, for many liquid and gaseous fluids the stress tensor can be expressed in terms of partial derivatives of the velocity field [64]. Such fluids are called Newtonian fluids. For such fluids, the stress tensor can be defined as follows:

$$\sigma_{ij} = 2\mu\left[\frac{1}{2}\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right)-\frac{1}{3}\frac{\partial u_k}{\partial x_k}\delta_{ij}\right] \tag{2.3}$$

Equation (2.3) is a so-called constitutive equation that relates the stress tensor to the velocity field and a material parameter μ, the shear viscosity, which is specific to each kind of liquid.

Combining the continuity, momentum and the material equation for Newtonian fluids yields the infamous Navier-Stokes (N-S) equations. Solving a CFD problem means finding valid solutions to the N-S equations.

- **Note:** we can also express the N-S equations using the gradient or the projected forms as follow:

- **Gradient-Form:**

Conservation of mass:

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

Conservation of momentum:

$$\frac{\partial \rho\mathbf{u}}{\partial t}+\nabla\cdot(\rho\mathbf{u}\mathbf{u})+\nabla p-\mu\nabla\cdot\left(\left(\nabla\mathbf{u}\right)^T+\nabla\mathbf{u}-\frac{1}{3}\nabla\cdot\mathbf{u}\right)=0$$

For an incompressible flow, these equations become:

$$\nabla\cdot\mathbf{u}=0$$

$$\rho\frac{\partial \mathbf{u}}{\partial t}+\rho\left(\mathbf{u}\cdot\nabla\right)\mathbf{u}+\nabla p-\mu\nabla^2\mathbf{u}=0$$

Where $\mathbf{u}$ denotes the velocity vector $\mathbf{u}=\{u,v,w\}$.

- **Projected-Form:** for steady two-dimensional incompressible flow, the N-S equations are written:

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0$$

$$\rho\left(u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}\right)=-\frac{\partial p}{\partial x}+\mu\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)$$

$$\rho\left(u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}\right)=-\frac{\partial p}{\partial y}+\mu\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right)$$

2.1.3. CFD Process

The use of CFD techniques to solve a fluid flow and heat transfer problem is split into three discrete parts: pre processing, processing, and post processing. In general, different computer programs that form the CFD code must undertake each of the three tasks [62].

- **Definition of a CFD Problem (Preprocessor)**

The first stage in solving a CFD problem is to define all the relevant parameters required by the CFD code prior to the numerical solution process, as follows:

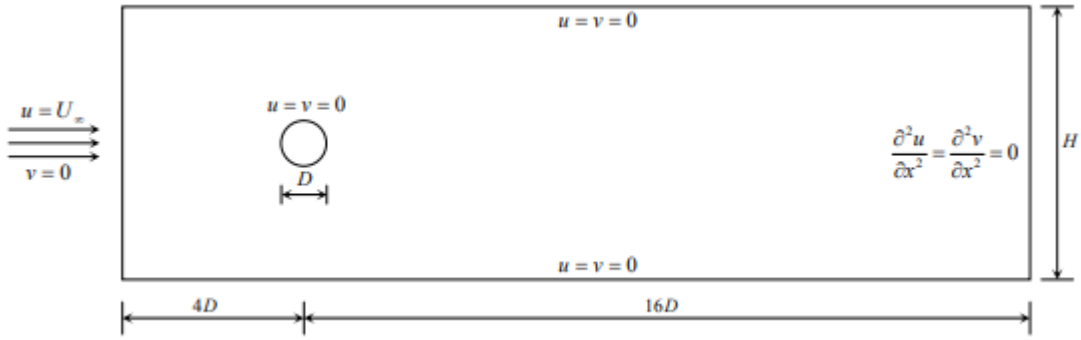


Figure 2.1 Definition of computational domain and boundary conditions [53].

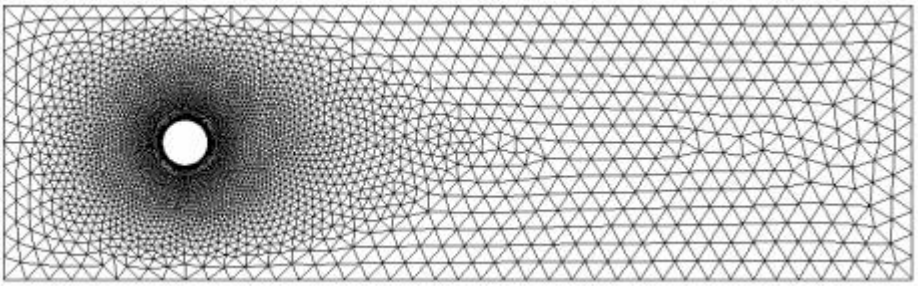


Figure 2.2 Construction of a computational grid [53].

a. Definition of the physical geometry of the environment in which the fluid flows, which is normally done by building up a geometric representation of the environment. "Create the shape of the problem domain that needs to be analysed".

b. Definition of flow parameter (such as the density, viscosity of the fluid flow).

c. Declaration of the boundary conditions of the physical environment. These boundary conditions will include defining certain areas such as the inlets and outlets for the fluid flow and the boundary areas of solids where heat transfer from or to the fluid can occur.

d. Construction of a mesh or grid as a geometric representation of the physical environment. This mesh or grid will form the computational grid that will be used in the solution of the problem by the powerful mathematical techniques around which CFD is based.

- **Solution of the Problem (Processor)**

The solution of the problem by the CFD code is where a host of mathematical techniques is used to approximate the differential equations into algebraic form, which can be solved directly or iteratively [16]. Different CFD codes employ different solution techniques, but the physics is the same if it can be well defined and understood. The solution of the transport equations for the geometry under study is not a trivial matter and cannot be solved readily, if at all, by analytical techniques. CFD uses numerical techniques (such as finite difference, finite volume, and finite element) to solve discretized representations of the transport equations [68].

Direct or explicit numerical methods, which can be both extremely accurate and rapid, may be used if sufficient computing power is available. Many codes use iterative methods to solve the equations because they tend to be more robust, although they can take longer to converge [64].

- **Analysis of the Results (Postprocessor)**

The results can be analyzed both numerically and graphically. The postprocessor takes the numerical results and displays them as a visual representation. It displays a visual image of the physical geometry through which the fluid flows, with the option of printing a hard copy of all the results as tables of numbers and other means [62].

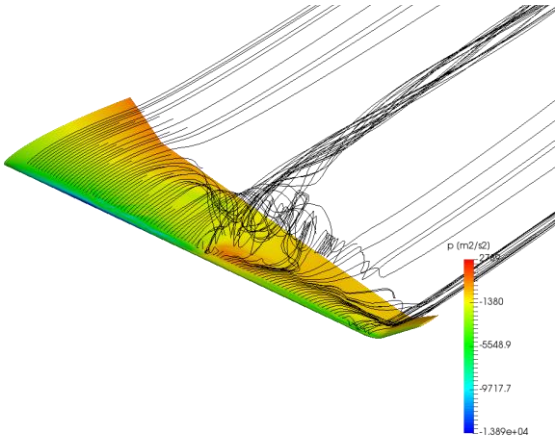


Figure 2.3 Example of Post processing results [62].

2.1.4. Summary

The momentum equation and the continuity equation in combination with the Newtonian material equation leads to the Navier-Stokes equations, which describes the motion of fluids.

Computational Fluid Dynamics (CFD) is the simulation of fluids engineering systems using modeling (mathematical physical problem formulation) and numerical methods (discretization methods, solvers, numerical parameters, and grid generations, etc.). The process is as figure 2.4.

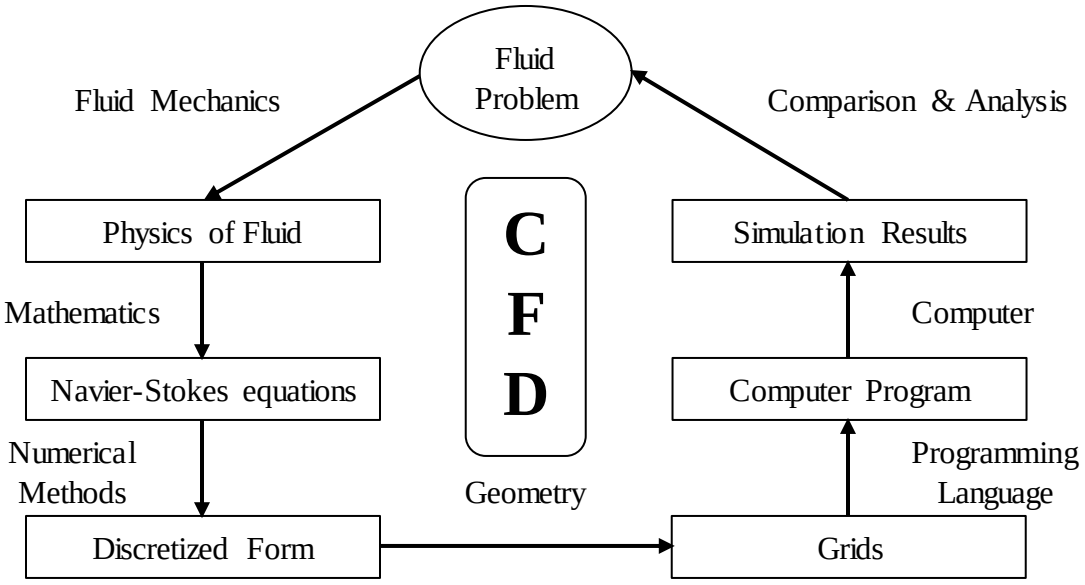


Figure 2.4 Process of Computational Fluid Dynamics [55].

A great amount of research has been conducted to find simplified versions the equations under certain conditions and to build software to solve them efficiently and robustly. Highly optimized and parallelized software libraries for Linear Algebra and partial differential equations in combination with high-performance computer hardware allows engineers to find solutions to practical problems in the area of computational fluid dynamics. While effects such as turbulence cannot be modeled explicitly in all detail even with today's high-performance computing systems, CFD solvers are essential in trying to understand the complex behavior of fluid under many conditions and deliver numerical results very close to experimental measurements [26].

CFD solvers and the software implementing the solvers are complex but powerful tools for solving the equations governing the behavior of fluids. Due to their complexity, CFD solvers remain difficult to use, even for engineers with experience in the field of numerical simulations.

Despite the advances of CFD solvers, a tradeoff between solution accuracy, computational requirements and time-to-solution must be made.

2.2. Machine Learning Fundamentals

2.2.1. Introduction

Machine Learning (ML) is a field of research that concern techniques and algorithms that allow computers to "learn" how to solve specific problems, rather than having the solution explicitly programmed [66]. To learn and develop, however, machine need data to analyze, understand, extract knowledge (or information) from them. In other words, ML uses data to feed an algorithm that can understand the relationship between the input and the output [72]. When the machine finished learning, it can predict the value or the class of new data point.

Within ML, a commonly employed model is the Artificial Neural Network (ANN). Just like a biological brain, the idea behind the ANN is to arrange multiple artificial neurons (Section 2.2.2.2) in layers to form a neural network (Section 2.2.2.3). Once the network structure built, it will be trained (Section 2.2.2.4) to perform a specific task.

2.2.2. Artificial Neural Networks (ANNs)

2.2.2.1. Biological Inspiration

The human brain is built up with neurons (figure 2.5). The neurons found in human brains consist out of three components: the dendritic tree, the cell body, and the axon [6]. The dendritic tree forms the connection with other neurons and collects signals from them. The cell body integrates the signals and generates an output signal. Subsequently, the output is passed on to other neurons through the branching axon [8]. A human neuron is depicted in figure 2.6.

Figure 2.5 Human brain [10].

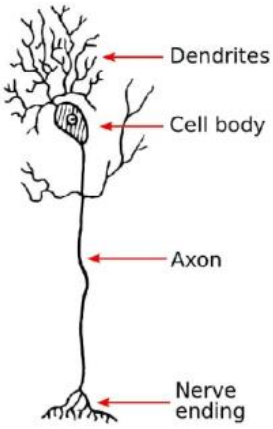
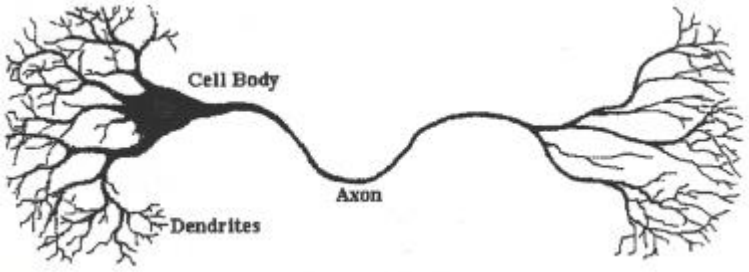


Figure 2.6 Human neuron [6].

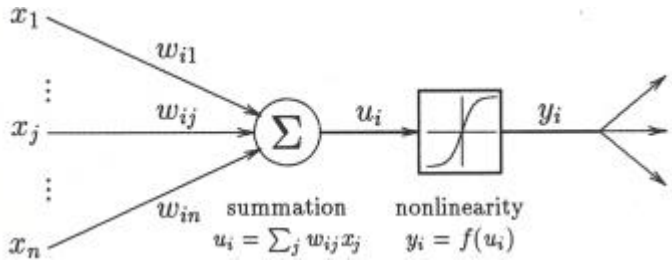
The most prominent theory on how learning happens in the human brain is known as Hebbian learning [31]. This is a theory which postulates that learning is the result of connections between biological neurons strengthening and weakening with use, often summarized as "neurons that fire together, wire together" [10]. Inspired by this, ANNs consist of artificial neurons that imitate the functioning of biological neurons. They receive an input in the form of an electrical signal from other neurons or sensory cells and if the sum of these inputs is sufficiently strong, the neuron fires its own action potential.

2.2.2.2. Artificial Neurons

The artificial neuron or perceptron works similarly to the biological one. Inputs are fed through the computing node through connections with other neurons, which are then processed through summation and an activation function to form an output [21]. This output can then be distributed to other neurons (Figure 2.7).



(a)



(b)

Figure 2.7 (a) Biological neuron. (b) Artificial neuron [18].

The neuron receives inputs x_i that are weighted with weights w_i and an additional bias term w_0. The inner state of the neuron is calculated by adding all weighted inputs and the bias term:

$$s = \sum_{i=1}^{N} x_i w_i + w_0$$

Based on the inner state of the neuron, the neuron's activation is calculated with an activation function f; maps the inner state of the neuron to an activation state. The activation state of a

neuron in one layer is used as an input for a neuron in the subsequent layer [31]. Thus, a neuron's activation can be calculated by the following equation:

$$y = f(s) = f\left(\sum_{i=1}^{N} x_i w_i + w_0\right)$$

An overview of common activation functions can be found in figure 2.8.

Hyperbolic tangent function

$$f(s) = \tanh(s)$$

Sigmoid function

$$f(s) = \frac{1}{1+e^{-s}}$$

Step function

$$f(s) = \begin{cases} 1 & s \geq 0 \\ 0 & s < 0 \end{cases}$$

Sign function

$$f(s) = \begin{cases} 1 & s \geq 0 \\ -1 & s < 0 \end{cases}$$

Figure 2.8 Common artificial neuron activation functions [9].

Some functions are non-linear or discontinuous. Generally, the magnitude of the output is $|f(s)| \leq 1$. Activation functions are sometimes also referred to as squashing functions or limiters since they have the ability to limit/convert large input values to smaller output values. The most common activation functions are the sigmoid function and the hyperbolic tangent (tanh), due to their smooth and non-decreasing properties. Because of their inherent smoothness, these activation functions are also occasionally referred to as being soft limiters, where discontinuous activation functions are referred to as hard limiters [23].

The next step is to add neurons in layers and connect them to form an artificial neural network.

2.2.2.3. Network

Neural network consists of multiple connected layers. There are three types of layers, namely, the input layer, hidden layer, and output layer. There can be multiple hidden layers, depending on the complexity of the neural network. Each layer contains a certain number of neurons, as

for each layer, all the artificial neurons are connected to all the ones in the next layer (figure 2.9), where each connection contains a weight that tells the neuron how much it should take over from the previous neurons, and each artificial neuron has a bias and a predefined activation function [18].

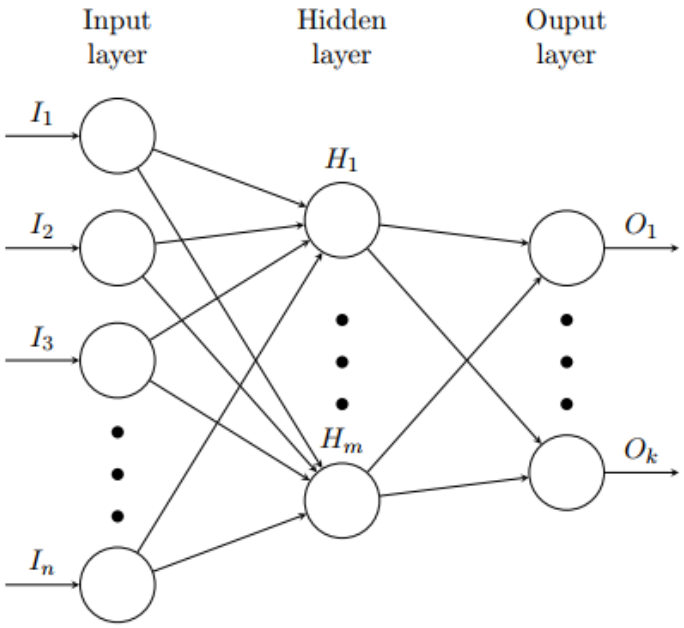


Figure 2.9 Neural network with an input layer with n neurons, a hidden layer with m neurons, and an output layer with k neurons. Each neuron is indicated with a circle, and each connection between neurons is indicated with an arrow [27].

The input layer is just for the inputs and does not contain any activation (it takes the data that is fed into the network). The hidden layers contain weights and biases and the artificial neurons have activation functions $f(s)$ [22]. After performing all the calculations layer by layer, the neural network outputs prediction through the output layer. The output nodes have also activation functions and contain weights and biases [20].

The mathematical expressions for calculations process in the ANN are shown in equations below, and are in vector-form, where W are tensors, the inputs are vectors $\mathbf{x}$ as well as the outputs $\bar{\mathbf{y}}$ and the biases $\mathbf{b}$. The superscript in front of the symbols denotes the layer l. The weight tensor entries represent the weight between connecting nodes. The subscript indices of ${}^{(l)}W_{ij}$, i and j denote the node it is pointing to and coming from respectively. So, for the connection between the input node for x_2 and the first node in the first hidden layer, the weight is ${}^{(1)}W_{12}$. For the biases, the subscript just denotes the bias for their respective nodes. Furthermore, for each layer, the summation and bias step yield $\mathbf{s}$. The application of the

activation function f to $\mathbf{s}$ (element-wise), will yield $\mathbf{a}$, which is the output of the artificial neurons in that layer and is what will be used for weighting and addition of the bias for the next layer.

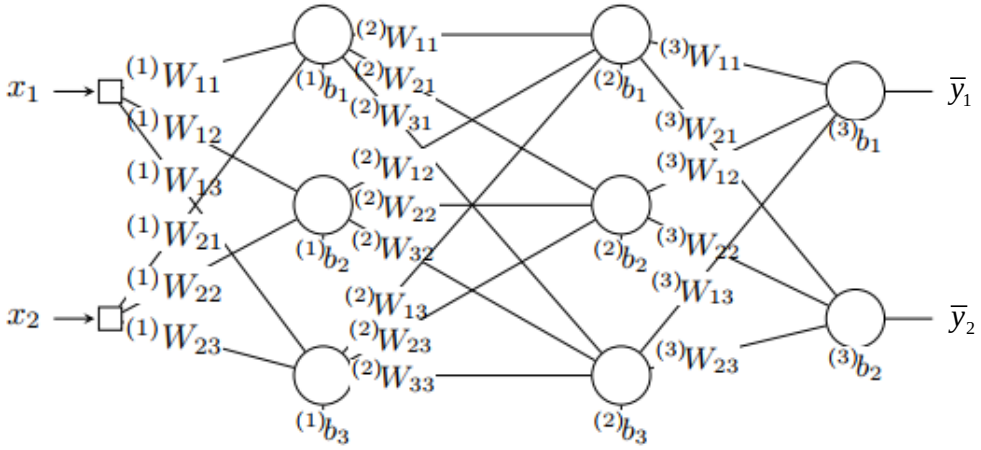


Figure 2.10 Detailed architecture of ANN [27].

The process calculation in the ANN is formalised as follow:

- **Input layer:**

This layer receives inputs $\mathbf{x} = \begin{pmatrix} x_1 & x_2 \end{pmatrix}$ and then fed them to the 1st hidden layer.

- **1st hidden layer:**

- **Weighting and Summation:** incoming signals $\mathbf{x}$ from input neurons are each weighted by $^{(1)}W$, then, the weighted linear combination of incoming signals is formed, as follow:

$$^{(1)}\mathbf{s} = \mathbf{x}\left(^{(1)}W\right)^T$$

- **Biasing:** an additional constant factor is added to the linear combination:

$$^{(1)}\mathbf{s} = {}^{(1)}\mathbf{s} + {}^{(1)}\mathbf{b}$$

- **Activation:** the scalar quantity resulting is then fed through the activation function which then forms the output:

$$^{(1)}\mathbf{a} = f\left(^{(1)}\mathbf{s}\right)$$

- **2nd hidden layer:**

- **Weighting and Summation:** incoming signals $^{(1)}\mathbf{a}$ from 1st hidden layer neurons are each weighted by $^{(2)}W$, then, the weighted linear combination of incoming signals is formed:

$$^{(2)}\mathbf{s} = {}^{(1)}\mathbf{a}\left(^{(2)}W\right)^T$$

- **Biasing:**

$$^{(2)}\mathbf{s} = {}^{(2)}\mathbf{s} + {}^{(2)}\mathbf{b}$$

- **Activation:**

$$^{(2)}\mathbf{a} = f\left({}^{(2)}\mathbf{s}\right)$$

- **<u>Output layer</u>:**

- **Weighting and Summation:** incoming signals $^{(2)}\mathbf{a}$ from 2nd hidden layer neurons are each weighted by $^{(3)}W$, then, the weighted linear combination of incoming signals is formed:

$$\bar{\mathbf{y}} = {}^{(2)}\mathbf{a}\left({}^{(3)}W\right)^{T}$$

- **Biasing:**

$$\bar{\mathbf{y}} = \bar{\mathbf{y}} + {}^{(3)}\mathbf{b}$$

- **Activation:**

$$\bar{\mathbf{y}} = f(\bar{\mathbf{y}})$$

Where:

- $^{(1)}W = \begin{pmatrix} {}^{(1)}W_{11} & {}^{(1)}W_{12} \\ {}^{(1)}W_{21} & {}^{(1)}W_{22} \\ {}^{(1)}W_{31} & {}^{(1)}W_{32} \end{pmatrix}, {}^{(2)}W = \begin{pmatrix} {}^{(2)}W_{11} & {}^{(2)}W_{12} & {}^{(1)}W_{13} \\ {}^{(2)}W_{21} & {}^{(2)}W_{22} & {}^{(2)}W_{23} \\ {}^{(2)}W_{31} & {}^{(2)}W_{32} & {}^{(2)}W_{33} \end{pmatrix}, {}^{(3)}W = \begin{pmatrix} {}^{(3)}W_{11} & {}^{(3)}W_{12} & {}^{(3)}W_{13} \\ {}^{(3)}W_{21} & {}^{(3)}W_{22} & {}^{(3)}W_{23} \end{pmatrix}$ represent respectively the weights of connections between: input and 1st hidden layer, 1st and 2nd hidden layer, output and 2nd hidden layer.

- $^{(1)}\mathbf{b}=\left({}^{(1)}b_1 \quad {}^{(1)}b_2 \quad {}^{(1)}b_3\right), {}^{(2)}\mathbf{b} = \left({}^{(2)}b_1 \quad {}^{(2)}b_2 \quad {}^{(2)}b_3\right), {}^{(3)}\mathbf{b} = \left({}^{(3)}b_1 \quad {}^{(3)}b_2 \quad {}^{(3)}b_3\right)$, represent respectively the biases added in 1st, 2nd hidden layers and output layer.

- $^{(1)}\mathbf{s} = \left({}^{(1)}s_1 \quad {}^{(1)}s_2 \quad {}^{(1)}s_3\right), {}^{(2)}\mathbf{s} = \left({}^{(2)}s_1 \quad {}^{(2)}s_2 \quad {}^{(2)}s_3\right), {}^{(3)}\mathbf{s} = \left({}^{(3)}s_1 \quad {}^{(3)}s_2 \quad {}^{(3)}s_3\right)$, vectors resulting from weighting, summation, and biasing process, at respectively 1st, 2nd hidden layers and output layer.

- $^{(1)}\mathbf{a} = \left({}^{(1)}a_1 \quad {}^{(1)}a_2 \quad {}^{(1)}a_3\right), {}^{(2)}\mathbf{a} = \left({}^{(2)}a_1 \quad {}^{(2)}a_2 \quad {}^{(2)}a_3\right), {}^{(3)}\mathbf{a} = \left({}^{(3)}a_1 \quad {}^{(3)}a_2 \quad {}^{(3)}a_3\right)$, vectors resulting from activation process at 1st, 2nd hidden layers and output layer.

Now that we have determined the structure of the neural network and how it works, we need to train it on a set of data, so it can learn from them, and then, make accurate predictions for a new giving examples.

2.2.2.4. Training the Network

Suppose we assign random values to all the weights. Then with a certain input, the output will most likely be nothing like the output we desire. This is why we first need to train the neural network. For this training, training data is required. This training data is comprised of inputs, that we already know the answer (output) for [6].

During the training of the neural network, all weights and biases of all neurons are being updated for every iteration using an optimizer algorithm. This is done by iteratively changing the weights and biases of the neurons starting in the last layer and ending with the first layer in order to minimize an error measurement of the network (How wrong was the prediction of the network) called loss, which is typically measured by the squared distance between the network output $\bar{y}$ and the expected output y [17]. The goal of the learning algorithm is to minimize this loss by changing the weights and biases according to the gradient of the loss, which is dependent on all weights and biases of all neurons in the network [6]. This algorithm is called the backpropagation algorithm since errors are propagated backward through the network in order to iteratively find optimal values for the weights and biases in the network [14].

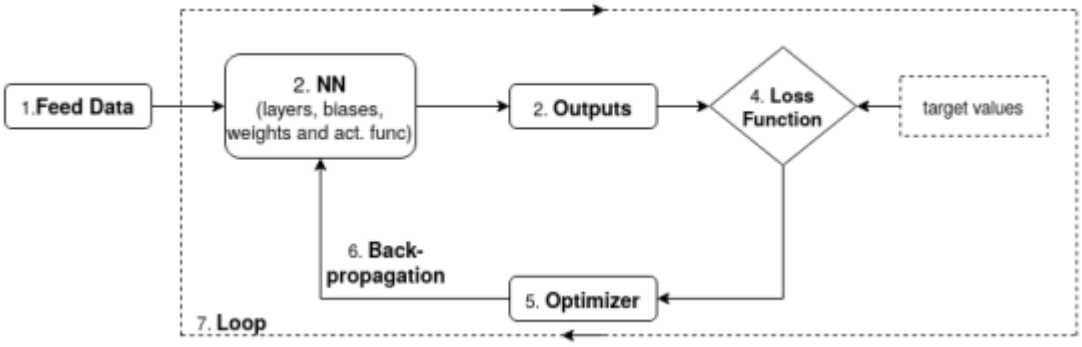


Figure 2.11 Training process [8].

2.2.2.4.1. Loss function

To train the model, a scalar variable that quantifies the fitness of the model is required. The functional that governs the performance or fit of the artificial neural network to some target data is referred to as the loss function. An artificial neural network can represent a function or a mapping $N : x \mapsto \bar{y}$. The mapping is dependent on the artificial neural network parameters (weights and biases). By computing the difference between the output of the neural network $\bar{y}$

and the data y and summing those, a scalar function can be obtained, that represents the discrepancy of the neural network to the data. There are different ways for representing the discrepancy, or the loss function. The most common one is the mean-squared error (MSE) [6]. This loss function is shown in equation:

$$\text{Mean-squared error:} \quad MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \bar{y}_i\right)^2$$

For which $\bar{y}_i$ and y_i is the ANN output and the target output respectively for $i = 1,\ldots,N$ data points. Each output of the artificial neural network is governed by the combination of weights and biases of the artificial neurons in the preceding layers that form the particular output. Therefore, the weights and biases have to be trained in order to minimize the loss function. Moreover, the weights and biases form the parameter space and the loss function defines the error surface. The error surface can contain global, local minima and saddle points wherein the first one is sought for and the last two can cause difficulty when training an ANN [3]. To make the idea clearer behind the loss function and how it can be minimized, we give the mathematical formulation of the literature above:

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \bar{y}_i\right)^2$$

Here θ is the set of all parameters in the network, which is the set of all weights along with the set of all biases. The parameters are initialized in some random manner. Let Θ be the parameter space. The problem at hand is simply to find the values for θ such that the loss function is minimal, that is, to approximate $\arg\min_{\theta\in\Theta} L(\theta)$ [3].

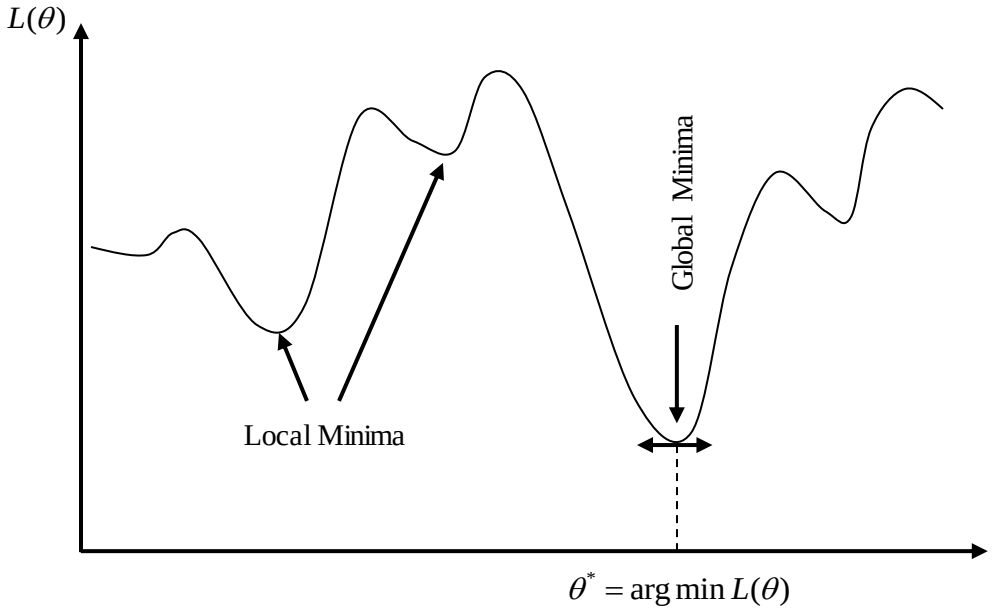


Figure 2.12 Loss function minima.

2.2.2.4.2. Backpropagation Process

In Calculus, when confronted with the task to determine a minimum of a function, the task is clear. Simply calculate the gradient and determine where it is equal to zero.

$$\theta^* = \arg\min L(\theta)$$
$$\left.\frac{\partial L}{\partial \theta}\right|_{\theta=\theta^*} = 0$$

The backpropagation algorithm operates as computing the gradient of loss function with respect to parameters (weights and biases) of a neural network based on the chain rule, that is, computing the gradient at one layer and recursively backward from the last layer. An optimizer algorithm adjusts each parameter of the neural network until we satisfy the equations above and then obtain the optimum parameters θ^* of the ANN (at this stage, the network makes accurate predictions) [21].

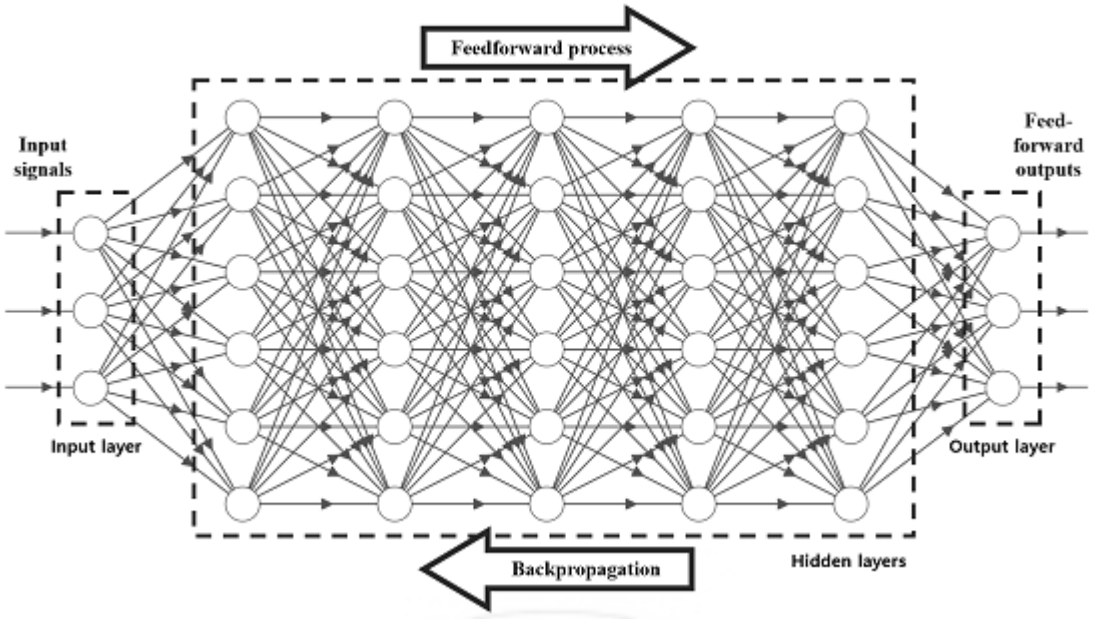


Figure 2.13 Feedforward and Backpropagation process [8].

2.2.2.4.3. Optimizer Algorithms

As illustrated in figure 2.11 optimizer is used after the comparison between target values and outputs of the network and the goal is to minimize the loss function by adjusting model parameters (weights and biases). The optimization is crucial for the learning process. The choice of optimizer over another one can lead to a better optimization. This can result in a faster learning and/or in a better final prediction [10].

There are different types of optimisers. Each type of optimiser has its own merits. The most common optimizers are Adam and the L-BFGS-B (Limited-memory Broyden-Fletcher-Goldfarb-Shanno Bound) [66]. When training the network, we prefer starting with Adam optimizer rather than L-BFGS-B, because this latter have more probability to stuck on a local minimum.

2.2.2.4.4. Model Hyper-parameter

Machine learning models and the algorithms to train them on data have so-called hyperparameters which are parameters set (by programmer) before optimizing the model's parameters (the network weights and biases). Network hyper-parameters describe the concrete model of the neural network and have a profound impact on the actual performance of the model [9]. The parameters can be chosen manually, based on experience or guidelines.

Model hyper-parameters are the following [6]:

- **Number of hidden layers**
- **Number of units (neurons) in each layer**
- **Activation function**
- **Optimizer**
- **Loss function**
- **Learning rate:** controls model's update step size with respect to the accuracy achieved with the present model. It's an important parameter. A value too small may result in a slow training and danger of getting stuck in a local. Instead a large value may result in risk of non-converting to any minimum, since the optimizer jumps out a minimum rather than descending to it.
- **Number of iterations:** refers to the number of times that the whole data-set arranged for the training phase is feed into the model. For example, if a number of iterations is equal to 10, during the training phase the model "sees" 10 times the whole data-set.

We now know the basics of how an artificial neural network works. The next step is to make the neural network a physics-informed neural network: get the physical model in the neural network.

2.3. Physics-Informed Neural Networks

In this section, we introduce the physics-informed neural networks (PINNs) and related settings in this study. Traditional neural networks are based entirely on a data-driven approach that does not take into account the physical laws (i.e. governing PDEs and initial/boundary conditions I/BCs). Therefore, a large amount of data is often required to train the neural networks to obtain a reasonable model. In contrast, PINNs introduce physical information into the network by forcing the network output to satisfy the corresponding physics equations (PDEs, I/BCs) [3]. Specifically, by encoding these equations in the loss function, the model is made to consider physical laws during the training process. This processing makes the training process require less data and speeds up the training process.

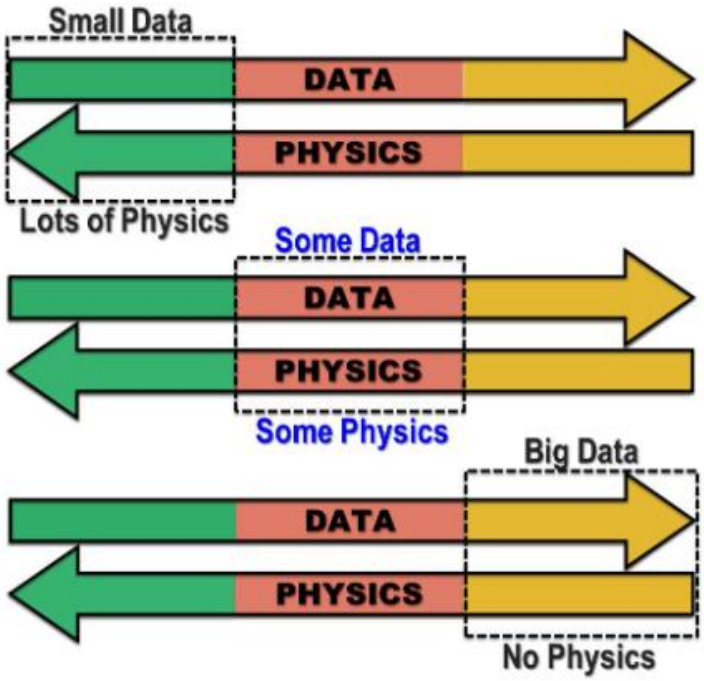


Figure 2.14 Physics-driven, Data-driven Neural Network [39].

PINNs can be used to solve not only the forward problem, i.e., obtaining approximate solutions to PDEs, but also the inverse problem, i.e., obtaining the parameters of PDEs from training data. In the following, the PINNs modified and used in this study is introduced for the forward problem of PDEs.

In this study, consider the partial differential equation defined on the domain Ω with the boundary $\partial\Omega$.

$$Du = 0 \quad \text{in } \Omega$$
$$Bu = 0 \quad \text{on } \partial\Omega$$

where $u = u(\mathrm{x}, t)$ (for $\mathrm{x} \in \mathbb{R}^n$, $t \in \mathbb{R}^+$) is the unknown solution and D denotes a linear or nonlinear N differential operator (e.g., $\partial / \partial x$, $\partial / \partial t$, $u \circ \partial / \partial x$, $u \circ \partial^2 / \partial x^2$, etc.), and the operator B denotes the boundary condition of a partial differential Equation (e.g., Dirichlet boundary condition, Neumann boundary condition, Robin boundary condition, etc.). Let $\mathrm{X} = (\mathrm{x}, t) \in \Omega$ for convenience. At this point, the initial condition can be treated as a special type of Dirichlet boundary condition on the spatio-temporal domain [6].

First, we construct a neural network for approximating the solution $u(\mathrm{X})$ of a partial differential equation. This neural network is denoted by $\hat{u}(\mathrm{X};\theta)$, which takes the X as input and outputs a vector of the same dimension as $u(\mathrm{X})$. θ represent the neural network parameters (weights and biases) [11]. These parameters will be continuously optimized during the training phase. The neural network $\hat{u}$ should satisfy the physics equations, thus, we fulfill this requirement by defining a residual network:

$$f(\mathrm{X};\theta) := N\left[\hat{u}(\mathrm{X};\theta)\right]$$

To build this neural network, we need to use automatic differentiation (AD) [15]. This represent all the differential operators (e.g. ∇, ∇^2) in the PDEs; then the equations can be formulated by the neural network.

In this study, for the surrogate network $\hat{u}$, we derive the neural network by the AD. Moreover, since the network f has the same parameters as the network $\hat{u}$, both networks are trained by minimizing a loss function. Specifically, Figure 2.15 shows a schematic diagram of a physics-informed neural network.

The next main task is to find the best neural network parameters that minimize the defined loss function [3]. In a physics-informed neural network, the loss function is defined, as follows:

$$J(\theta) = MSE_u + MSE_f \tag{2.4}$$

Where:

$$MSE_u = \frac{1}{N_u}\sum_{i=1}^{N_u}\left|\hat{u}^i - u\left(x_u^i, t_u^i\right)\right|^2 \tag{2.5}$$

$$MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f}\left|f\left(x_f^i, t_f^i\right)\right|^2 \tag{2.6}$$

Where $(t_u^i,\ x_u^i,\ u\left(x_u^i,t_u^i\right))$ are the given initial and boundary conditions. Equation 2.5 is taken over all the boundary and initial points and equation 2.6 is taken over all the points in the domain. So MSE_u ensures the boundary and initial conditions are met and MSE_f ensures the given differential equations are satisfied [6].

It should be noted here that while MSE_u can be calculated fairly easily, determining $f\left(x_f^i,t_f^i\right)$ and thus MSE_f is more difficult. However, as discussed above, automatic differentiation is one of the qualities of a neural network. So, we can use this same technique to compute the derivatives, without the need of making grids as in classical numerical methods. So, f can also be determined relatively easily (albeit with a longer computing time then MSE_u).

Next, the optimization problem for equation 2.4 is addressed by optimizing the parameters in order to find the minimum value of the loss function, i.e., we seek the following parameters:

$$w^* = \arg\min_{w\in\theta}\left(J\left(w\right)\right)$$
$$b^* = \arg\min_{b\in\theta}\left(J\left(b\right)\right)$$

In the last step, we use gradient optimizers to minimize the loss function, such as Adam, and L-BFGS-B. It is found that, for smooth PDE solutions, L-BFGS-B can find a good solution faster than Adam, using fewer iterations. This is because Adam optimizer relies only on the first order derivative, whereas L-BFGS-B uses the second order derivative of the loss function [10]. However, one problem with L-BFGS-B is that it is more likely to get stuck on a bad local minimum [31]. Considering their respective advantages, in this study we end up using a combination of L-BFGS-B and Adam optimizer to minimize the loss function. By the above method, we will obtain trained neural networks that can be used to approximate the solutions of partial differential equations.

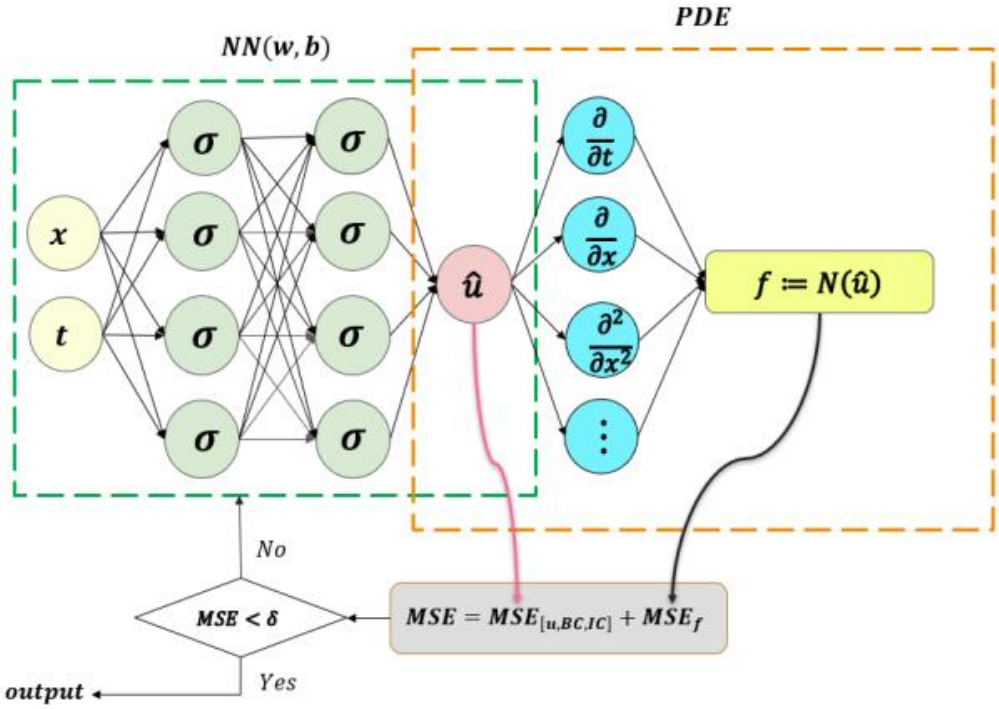


Figure 2.15 The schematic of physics-informed neural network (PINN) for solving partial differential equations [3].

In the next chapter, we will use the above method to study two important partial differential equations: the one-dimensional wave equation and the Burgers equation.

2.3.1. Advantages of PINNs

The classical numerical methods for PDEs, such as finite differences, finite volumes, and finite element methods, are very commonly used for flow simulation and also other computational physics. These methods require discretizing computational domain into many small meshes, and evaluate approximation of function value at each mesh point. The mesh-based processes are computationally expensive and difficult for large systems, high-dimensional equations, or complex geometries, and evaluating the value at each point has some limits in terms of post-calculation.

On the other hand, numerical methods based on neural networks have several advantages compared to the classical methods. The advantages are:

- **Memory complexity:** the neural network methods require low memory cost due to less parameters to be calculated [9].

- **Closed form:** the solutions obtained by neural networks have a closed form. They have capabilities to perform subsequent calculation, such as differentiation or integration of the solutions [7].
- **Transfer learnability:** by the generalization property of a neural network, the obtained solution of a problem is reusable and able to be generalized, which means that the solution may be applied to a class of similar problems with the given problem [31].
- **Complex shape:** if the shape of domain where a differential equation is defined on is complex, the neural network method are relatively more efficient than the classical method because neural network methods are totally mesh-free methods [28].
- **Dimensionality:** the neural network method has a potential to efficiently solve high-dimensional equations. In other words, the computational complexity of neural network method is linear with the dimension of problem, while the classical methods have rapidly increasing complexity with increasing dimension of problem [40].

CHAPTER 3

Preliminary Examples

In this Chapter, we study the one-dimensional wave equation and the Burgers equation using PINNs. The neural network models are constructed for these two equations, respectively, based on the given initial and boundary conditions. The approximation results of the neural networks are compared with the true solutions to test the PINNs. We will present the experimental design and results of these two equations, respectively.

3.1. Wave Equation

This section presents an experimental study of the wave equation using the PINN. The wave equation is a typical hyperbolic PDE and it contains second-order partial derivatives about the independent variable. In physics, the wave equation describes the path of a wave propagating through a medium and is used to study the various types of wave propagation phenomena. It appears in many fields of science, such as acoustic wave propagations, radio communications, and seismic wave propagation. The study of wave equations is of great importance, as they are widely used in many fields. In this study, we choose a one-dimensional wave equation for our experiments. In mathematical form, this wave equation is defined, as follows:

$$u_{tt} - cu_{xx} = 0, \quad x \in [0,\ 1], \quad t \in [0,\ 1]$$

where u is a function of the spatial variables x and time t. In the equation, the value of c represents the wave propagation velocity, which is given as 1 in this study. Besides, for this wave equation, its initial conditions and the homogeneous Dirichlet boundary conditions are given, as follows:

$$u(0,x)=\frac{1}{2}\sin(\pi x)$$
$$u_t(0,x)=\pi\sin(3\pi x)$$
$$u(t,0)=u(t,1)=0$$

The true solution of the above equation is:

$$u(t,x)=\frac{1}{2}\sin(\pi x)\cos(\pi t)+\frac{1}{3}\sin(3\pi x)\sin(3\pi t),$$

The initial conditions, boundary conditions, and some random data in the space-time domain are used as training data to train the neural network model. In order to test the performance of the training model, we use the neural network model to make multiple predictions and compare it with the true solution of the PDE. The specific experimental setup and procedure are as follows.

First, a neural network is designed for approximating the solutions of PDEs, denoted as $\hat{u}(t,x)$. For the architecture of the neural network, it contains six hidden layers, each with 100 neurons, and a hyperbolic tangent tanh is chosen as the activation function. Besides, a physics-informed neural network $f(t,x)$ is constructed for introducing control information of the equation:

$$f(t,x):=u_{tt}-u_{xx}$$

The next main task is to train the parameters of the neural network $\hat{u}(t,x)$ and $f(t,x)$. We continuously optimize the parameters by minimizing the mean square error loss to obtain the optimal parameters:

$$J(\theta)=MSE_u+MSE_f$$

Where:

$$MSE_u=\frac{1}{N_u}\sum_{i=1}^{N_u}\left|\hat{u}^i-u\left(x_u^i,t_u^i\right)\right|^2$$

$$MSE_f=\frac{1}{N_f}\sum_{i=1}^{N_f}\left|f\left(x_f^i,t_f^i\right)\right|^2$$

MSE_u is a loss function constructed using observations of initial and boundary conditions. MSE_f is a loss function that is based on partial differential equations for introducing physical

information. Specifically, $\{t_u^i, x_u^i, u^i\}_{i=1}^{N_u}$ corresponds to the initial and boundary training data of $u(t,x)$, and N_u is the number of data provided. In addition, $u(t_f, x_f)$ and $\{t_f^i, x_f^i\}_{i=1}^{N_f}$ corresponds to the training data of the spatio-temporal domain, and N_f is the corresponding number of training data. In this work, to fully consider the physical information embedded in the equations, we select the data in the spatio-temporal domain to train the neural network. The training data of the spatio-temporal domain is selected randomly, and the amount of training data N_f is 40,000. Besides, the total number of training data of the initial and boundary conditions is relatively small, and the expected effect can be achieved when N_u is 300. Similarly, the selection of training data for the initial and boundary conditions is also random. During the optimization procedure, we set the learning rate to 0.001, and in order to balance convergence speed and global convergence, we ran L-BFGS 30,000 epochs and then continued the optimization using Adam until convergence. In addition, we used the Glorot normal initializer for initialization. In this experiment, the time to train the model was approximately fifteen minutes. We tested the effect of the model after completing the training of the neural network model. Figure 3.1 is the prediction of the neural network model obtained from the training, and it can be seen that the prediction obtained is quite complex. We choose different moments to compare the prediction with the exact solution to test the accuracy of this prediction. Figure 3.2 shows the comparison between the exact solution and the prediction at different times $t = 0.2, 0.5, 0.8$. From Figure 3.2, it can be seen that the predictions of the neural network model and exact solutions are very consistent, indicating that the constructed neural network model has a good ability to solve partial differential equations. In addition, the relative L2 error of this example was calculated to be $5.16 \cdot 10^{-4}$, which further validates the effectiveness of this method. Although the solution of the selected partial differential equations is complex, the neural network model can still approximate a result very close to the true solution from the training data, indicating that the neural network with physical information has great potential and value, and is worthy of further research.

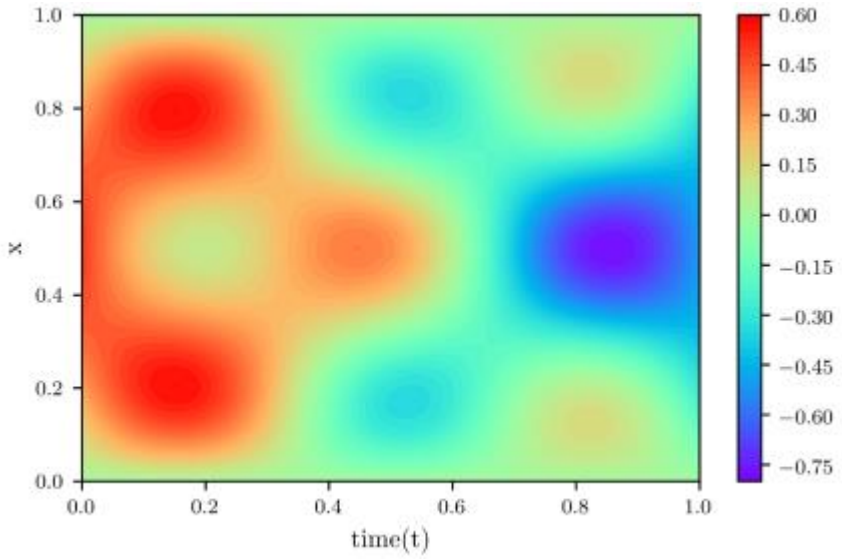


Figure 3.1 Solution of the wave equation given by physics-informed neural networks [3].

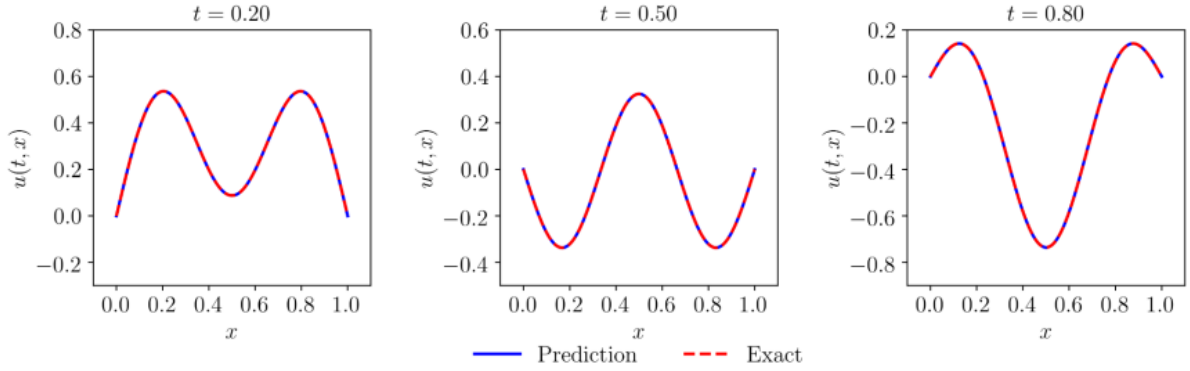


Figure 3.2 Comparison of the prediction given by physics-informed neural networks with the exact solution [3].

3.2. Burgers Equation

Let us consider the Burgers equation. This equation arises in various areas of applied mathematics, including fluid mechanics, nonlinear acoustics, gas dynamics, and traffic flow. It is a fundamental partial differential equation and can be derived from the Navier-Stokes equations for the velocity field by dropping the pressure gradient term. For small values of 4the viscosity parameters, Burgers equation can lead to shock formation that is notoriously hard to resolve by classical numerical methods. In one space dimension, the Burger's equation along with Dirichlet boundary conditions reads as:

$$u_t + uu_x - (0.01/\pi)u_{xx} = 0, \quad x \in [-1,1], \quad t \in [0,1]$$
$$u(0,x) = -\sin(\pi x)$$
$$u(t,-1) = u(t,1) = 0$$

Let us define $f(t,x)$ to be given by:

$$f := u_t + uu_x - (0.01/\pi)u_{xx}$$

Similar to the previous experiment, we train the parameters of the neural network $\hat{u}(t,x)$ and $f(t,x)$. We continuously optimize the parameters by minimizing the mean square error loss to obtain the optimal parameters:

$$J(\theta) = MSE_u + MSE_f$$

Where:

$$MSE_u = \frac{1}{N_u}\sum_{i=1}^{N_u}\left|\hat{u}^i - u\left(x_u^i, t_u^i\right)\right|^2$$

$$MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f}\left|f\left(x_f^i, t_f^i\right)\right|^2$$

The training data of the spatio-temporal domain is selected randomly, and the amount of training data N_f is 10,000. Besides, the total number of training data of the initial and boundary conditions is relatively small, and the expected effect can be achieved when N_u is 100. Similarly, the selection of training data for the initial and boundary conditions is also random. During the optimization procedure, we set the learning rate to 0.001, and in order to balance convergence speed and global convergence, we ran L-BFGS 20,000 epochs and then continued the optimization using Adam until convergence. In addition, we used the Glorot normal initializer for initialization. In this experiment, the time to train the model was approximately fifteen minutes. We tested the effect of the model after completing the training of the neural network model. Figure 3.3 is the prediction of the neural network model obtained from the training, and it can be seen that the prediction obtained is quite complex. We choose different moments to compare the prediction with the exact solution to test the accuracy of this prediction. Figure 3.4 shows the comparison between the exact solution and the prediction at different times t = 0.25, 0.5, 0.75. From Figure 3.4, it can be seen that the predictions of the neural network model and exact solutions are very consistent, indicating that the constructed neural network model has a good ability to solve partial differential equations. In addition, the relative L2 error of this example was calculated to be $6.7\cdot10^{-4}$, which further validates the effectiveness of this method.

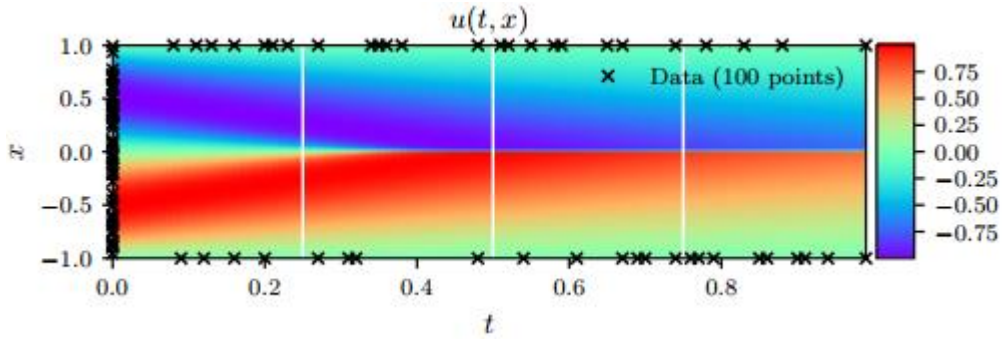


Figure 3.3 Solution of the Burgers equation given by physics-informed neural networks.

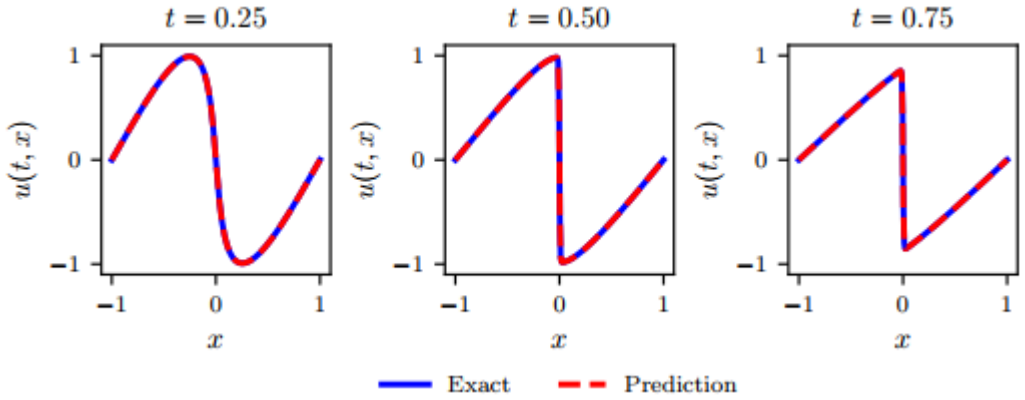


Figure 3.4 Comparison of the prediction given by PINNs with the exact solution [2].

Although the solution of the selected partial differential equations is complex, the neural network model can still approximate a result very close to the true solution from the training data, indicating that the neural network with physical information has great potential and value, and is worthy of further research.

CHAPTER 4

Implementation and Results

In this Chapter, we implement a physics-informed neural network (PINN) scheme to solve fluid dynamics problems governed by the Navier-Stokes equations. We first introduce the methodology of the proposed PINN and the mathematical formulation for fluid dynamics (Section 4.1). Subsequently in Section 4.2, the steady two-dimensional viscous incompressible flow at low Reynolds number passing a NACA 0012 airfoil will be modeled using the proposed PINN scheme. A comparison study with a reference numerical solutions is made to validate our PINN results. We summarize our finding in Section 4.3.

4.1. Solution Methodology

Navier–Stokes equations describe the physics of many phenomena of scientific and engineering interest. They may be used to model the weather, ocean currents, air flow around a wing, etc. The Navier–Stokes equations in their full and simplified forms help with the design of aircrafts and cars, the study of blood flow, and many other applications.

Let us consider the steady incompressible Newtonian flow governed by the following Navier-Stokes equations (in the velocity-pressure form):

$$\rho(\mathbf{v}\cdot\nabla)\mathbf{v} = -\nabla p + \mu\nabla^2\mathbf{v} \quad in\ \Omega \tag{4.1a}$$

$$\nabla\cdot\mathbf{v} = 0 \quad in\ \Omega \tag{4.1b}$$

$$\mathbf{v} = \mathbf{v}_\Gamma \quad on\ \Gamma_D \tag{4.1c}$$

$$\frac{\partial\mathbf{v}}{\partial n} = 0 \quad on\ \Gamma_N \tag{4.1d}$$

Where ∇ is the Nabla operator, $\mathbf{v} = (u, v)$ is the velocity vector, p is the pressure, μ is the viscosity of the fluid, ρ is the density of fluid. The boundary conditions are required in order

to solve Eq. (4.1). Here, Γ_D and Γ_N denote the Dirichlet and Neumann boundaries, respectively.

In this study, instead of using conventional computational fluid dynamics (CFD) methods, we investigate the possibility of using neural networks (NNs) for solving the aforementioned partial differential equations (PDEs) (see Eqs. 4.1a and 4.1b). In other words, the solutions of Navier-Stokes equations are approximated by a neural network, which takes spatial coordinates as inputs and predicts the corresponding velocity and pressure fields, i.e., $(x, y) \mapsto (\mathbf{v}, p)$. To ensure the divergence free condition of the flow we use the stream function ψ. In this way, the continuity equation (Eq. 4.1b) will be satisfied automatically. For a two-dimensional problem, the velocity components can be computed by $[u, v, 0] = \nabla \times [0, 0, \psi]$, i.e., $u = \frac{\partial \psi}{\partial y}$, $v = -\frac{\partial \psi}{\partial x}$.

First, we construct the architecture of the proposed PINN for fluid dynamics simulation (Figure 4.1), which consist of a fully-connected network and the residual networks. Here, the nonlinear activation function σ is the hyper tangent function $\tanh$. For the residuals, these include the errors of the momentum equations (4.1a). In order to compute these residuals res_1 and res_2, the partial differential operators are computing by using automatic differentiation (AD), which leads to very high computational efficiency compared to numerical differentiation. However, it does not require grids (mesh), and avoids the classical artificial dispersion and diffusion errors. AD can be directly formulated in the machine learning framework, e.g., using "tf.gradients()" in TensorFlow.

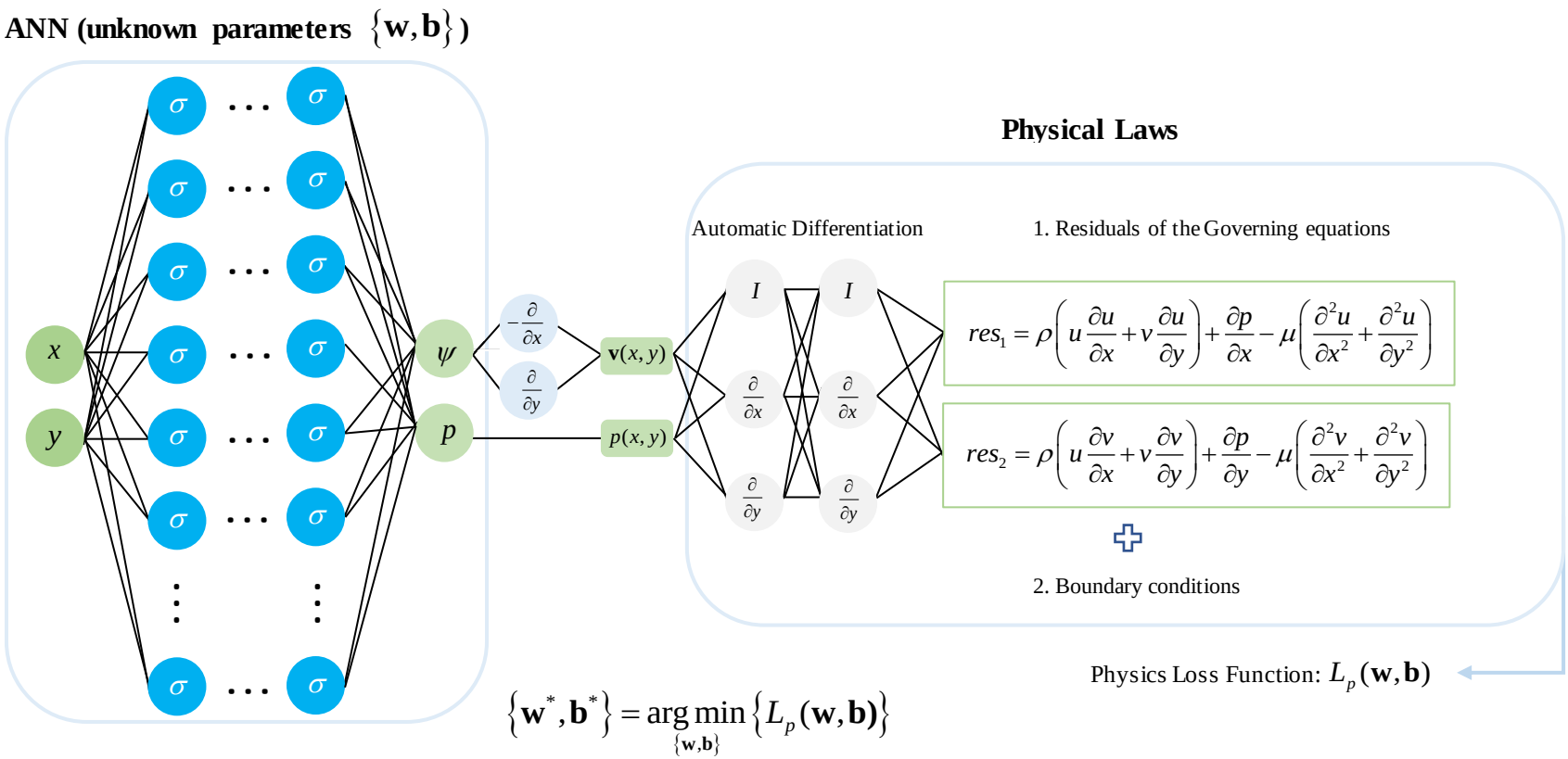


Figure 4.1 Architecture of the physics-informed neural network for fluid dynamics. Note that **w** and **b** are weights and biases for the ANN. The left part of the NN is an uninformed network, while the right part implements the physical laws using Automatic Differentiation (AD). The constraint of governing equations and boundary conditions can be converted as residuals adding to the loss function.

Now, we apply the main idea of PINNs method, i.e., embed physics laws (governing equations and boundary conditions) into the loss function of the PINNs. This can be achieved by incorporating the residual of physics equations into the loss function.

Let define $L_p(\mathbf{w},\mathbf{b})$ a physics loss function for training the parameters of PINN to obtain the solutions of Eq. (4.1) as follow:

$$L_p = L_e + L_b \tag{4.2a}$$

$$L_e = \frac{1}{N_e}\sum_{i=1}^{2}\sum_{n=1}^{N_e}\left|res_i^n\right|^2 \tag{4.2b}$$

$$L_b = \frac{1}{N_b}\sum_{n=1}^{N_b}\left|\mathbf{v}^n - \mathbf{v}_b^n\right|^2 \tag{4.2c}$$

Where L_e and L_b represent loss function components corresponding to the residual of the momentum equations and the boundary conditions, respectively; N_b and N_e denote the number of training data for different terms; $\mathbf{v}_b^n = \left[u_b^n, v_b^n\right]^T$ is the given velocity for the n th data point on the boundaries; res_i^n represents the residual of the i th equation at the n th data point. We consider the boundary conditions as supervised data-driven parts, and the residual of the momentum equations as the unsupervised physics-informed part in the loss function.

The next task is to find the best neural network parameters that minimize the defined loss function, for that, we use two optimization algorithms, Adam and L-BFGS-B.

$$\left\{\mathbf{w}^*,\mathbf{b}^*\right\} = \arg\min_{\{\mathbf{w},\mathbf{b}\}}\left\{L_p(\mathbf{w},\mathbf{b})\right\}$$

The solutions are obtained when the training of the PINN converges, i.e., the total loss function L_p reaches some very small value.

4.2. Results

In this section, we apply the proposed PINN to model the steady two-dimensional viscous incompressible flow at low Reynolds number passing a NACA 0012 airfoil at different angle of attack ($0^\circ, 3^\circ, 9^\circ, 12^\circ$). We present comparisons between the PINN solutions and reference solution obtained from the ANSYS Fluent 19.0 package (finite volume-based) in order to investigate the accuracy of the solutions inferred by the PINN framework.

We consider a computational domain of width $L = 2m$ and height $H = 1m$ with an NACA 0012 airfoil of chord length $c = 1m$, the leading edge of the airfoil is placed at the spatial coordinate (0.2, 0.5) (Fig. 4.2). For the boundary conditions of the problem, we define a parabolic velocity

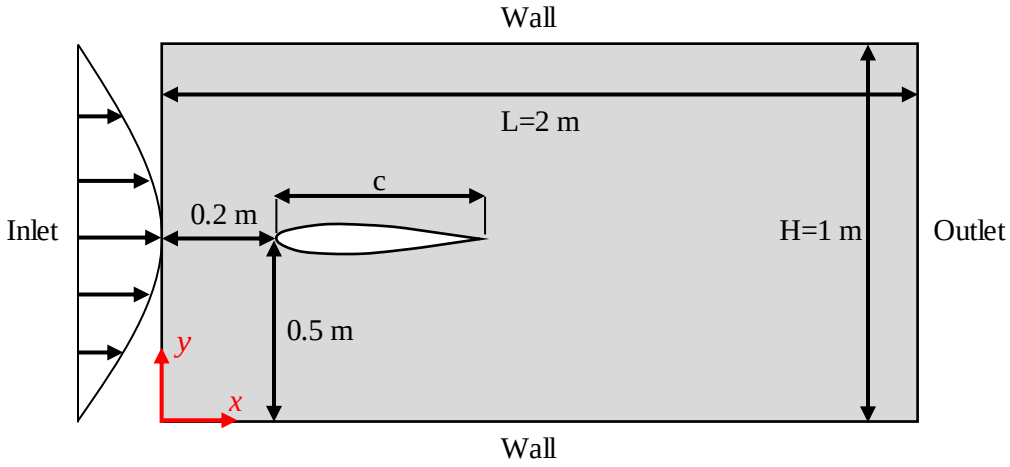


Figure 4.2 Diagram of the computation model.

profile on the inlet with the following expression:

$$\mathbf{U}_{Inlet} = \begin{Bmatrix} u_{Inlet} \\ v_{Inlet} \end{Bmatrix}$$

with:

$$u_{Inlet} = \left[4\mathrm{U}_{\max}(\mathrm{H} - y)y / \mathrm{H}^2 \right] \cos\alpha$$
$$v_{Inlet} = \left[4\mathrm{U}_{\max}(\mathrm{H} - y)y / \mathrm{H}^2 \right] \sin\alpha$$

where α is the angle of attack and $\mathrm{U}_{\max} = 1m/s$ which results in a small Reynolds number so that the flow is dominated by laminar flow. On the outlet the zero pressure condition is applied. Nonslip conditions are enforced on the wall and airfoil boundaries. The gravity is ignored. For the material properties of the fluid flow, the dynamic viscosity and density is $10^{-3} kg/(m \cdot s)$ and $1 kg/m^3$ respectively.

Now, we solve the defined problem using the PINN scheme, following the methodology made in the previous section. We define a computational domain (same as seen in Figure 4.2) by sampling spatial points in the domain, as shown in Figure 4.3. A total number of 50000 spatial scattered points are generated in the whole domain using Latin hypercube sampling (LHS) for the training the network. This include 3085 Dirichlet boundary (airfoil, wall, inlet) points and 201 Neumann boundary (outlet) points, such that we have 3286 training data for the boundary

conditions, i.e., $N_b = 3286$. Inside the domain, we use 46714 points to compute the residuals (or the equation loss), i.e., $N_e = 46714$. It should be noted that the collocation points are refined near and behind the airfoil to better capture the details of the flow.

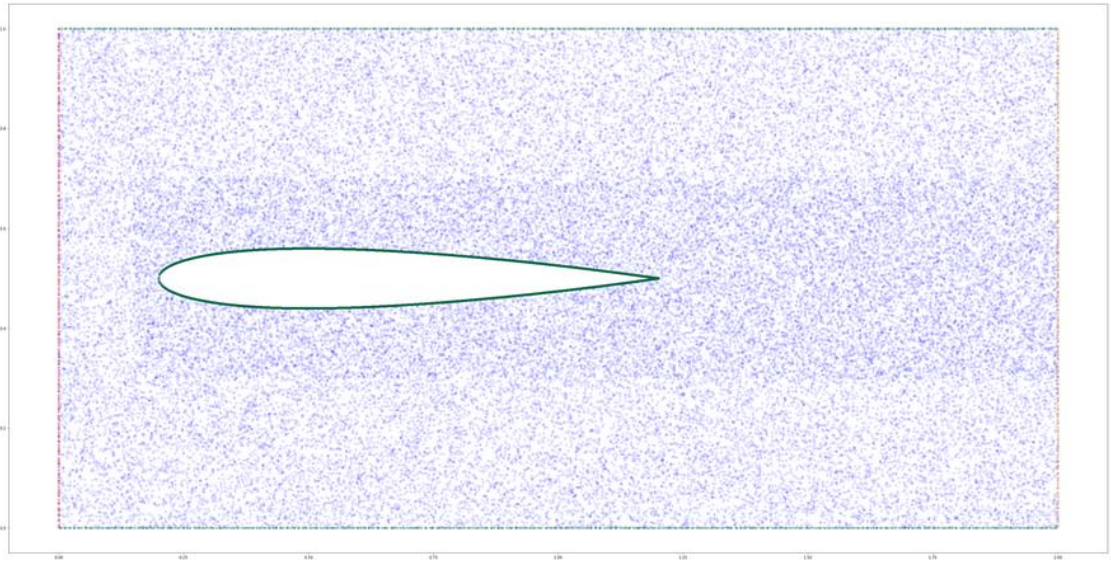


Figure 4.3 Sampling points using LHS.

The PINN is assessed after a two-step training: we first use the Adam optimizer for 30000 iterations with learning rate of 1×10^{-3}, then apply the L-BFGS-B to finetune the results. The training process of L-BFGS-B is terminated automatically based on the increment tolerance.

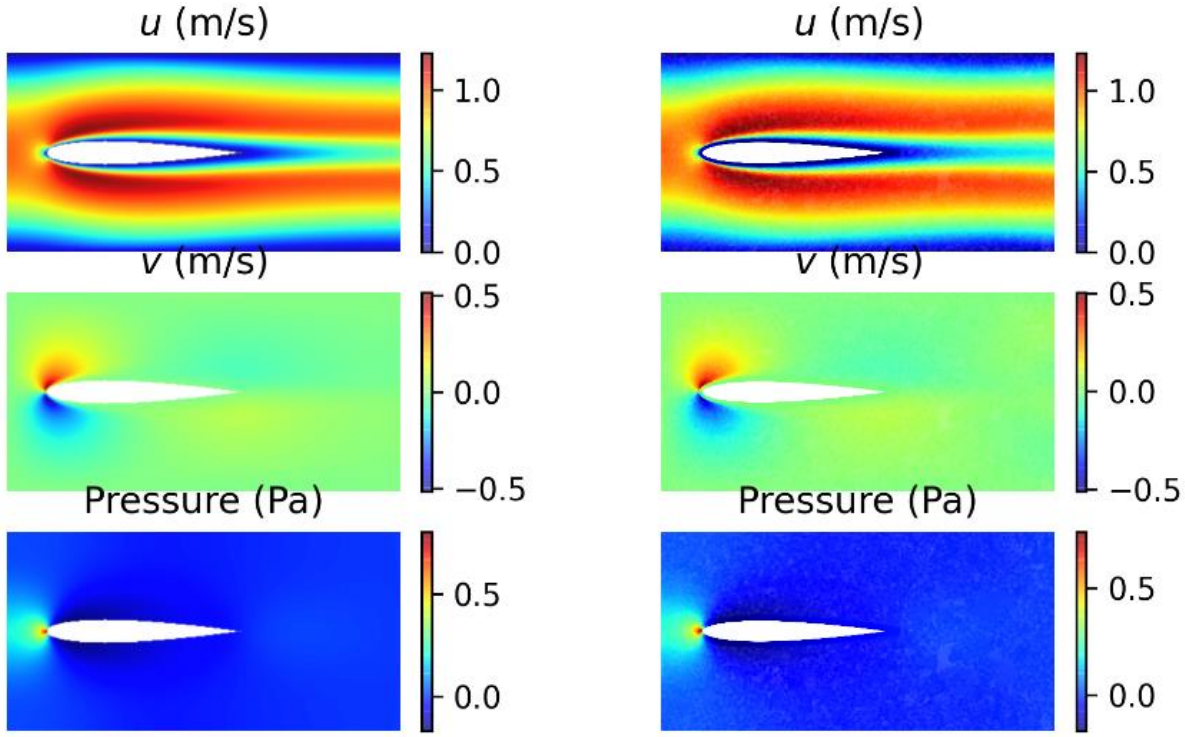
We first investigate the influence of the neural network architecture. We employ different sizes of network by varying the number of hidden layers and the number of neurons per layer. This strategy allows to find an optimal combination of depth and width for the network. The loss (error) function is used as the metric for comparison (Table 4.1).

NN size	**Loss L_p**
4×50	1.8×10^{-1}
6×40	5.3×10^{-2}
7×40	9.7×10^{-3}
8×40	4.4×10^{-3}
8×100	3.7×10^{-5}

Table 4.1 Loss value L_p for PINN with different sizes (NN size is the number of hidden layers × the number of neurons per layer).

We can observe from Table 4.1 that the performance of the PINN is improved as the network size increases, in other words, PINN provides more accurate solutions when using large networks. PINN are able to attain the solutions with high accuracy using a deep neural network of 8×100. The loss value (error) are in order of 10^{-5}. The network of 8×100 achieves the best result among all the configurations.

After this assessment, we select the more accurate PINN architecture, i.e., 8×100 to compare it with a reference solution obtained from the ANSYS Fluent 19.0 package (finite volume-based). The predicted velocity and pressure fields by the PINN are shown in Fig. 4.4(a), 4.7(a), 4.10(a) and 4.13(a) for $\alpha = 0^{\circ}$, $\alpha = 3^{\circ}$, $\alpha = 9^{\circ}$ and $\alpha = 12^{\circ}$, respectively. The reference solution is obtained from the CFD solver ANSYS Fluent (see Fig. 4.4(b), 4.7(b), 4.10(b) and 4.13(b) for $\alpha = 0^{\circ}$, $\alpha = 3^{\circ}$, $\alpha = 9^{\circ}$ and $\alpha = 12^{\circ}$, respectively).



(a) (b)

Figure 4.4 Velocity and pressure fields of the flow around an NACA 0012 airfoil at $\alpha = 0^{\circ}$: (a) PINN solution with 8×100 network; (b) Reference solution from ANSYS Fluent.

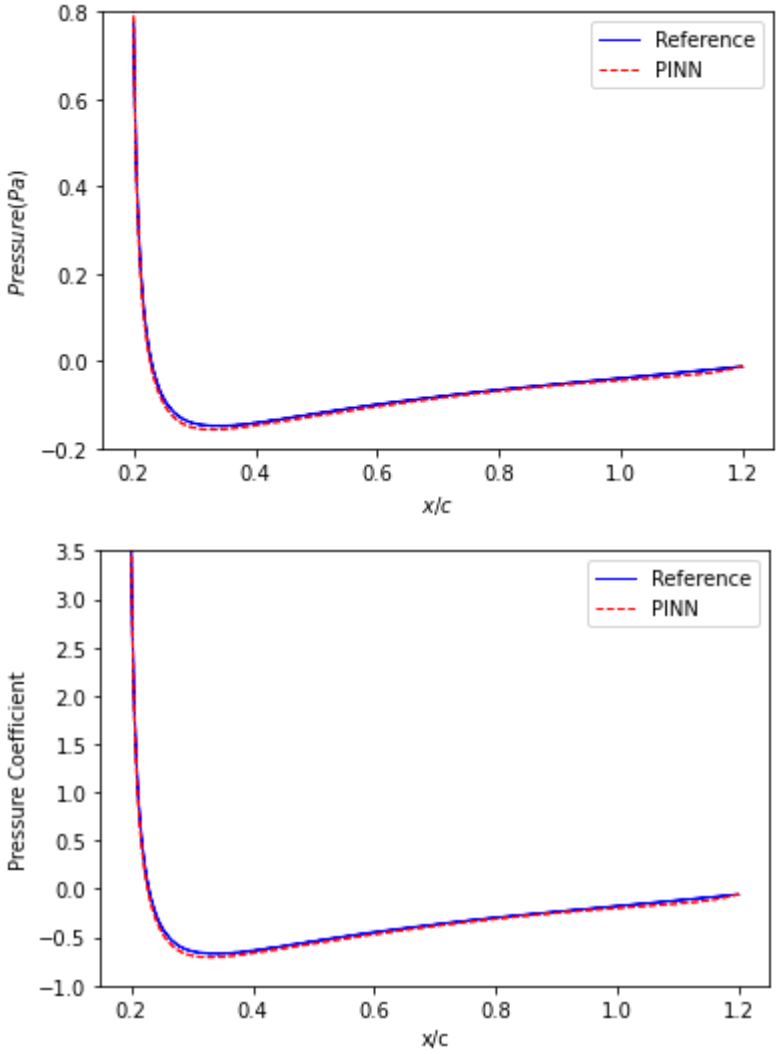


Figure 4.5 Distribution of p and C_p on an airfoil at $\alpha = 0^\circ$. Network of 8 × 100 are used.

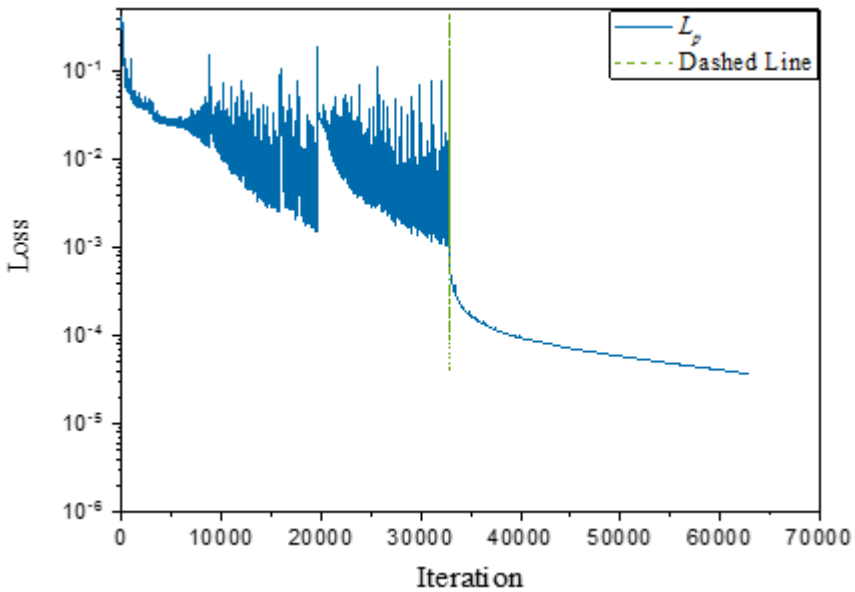
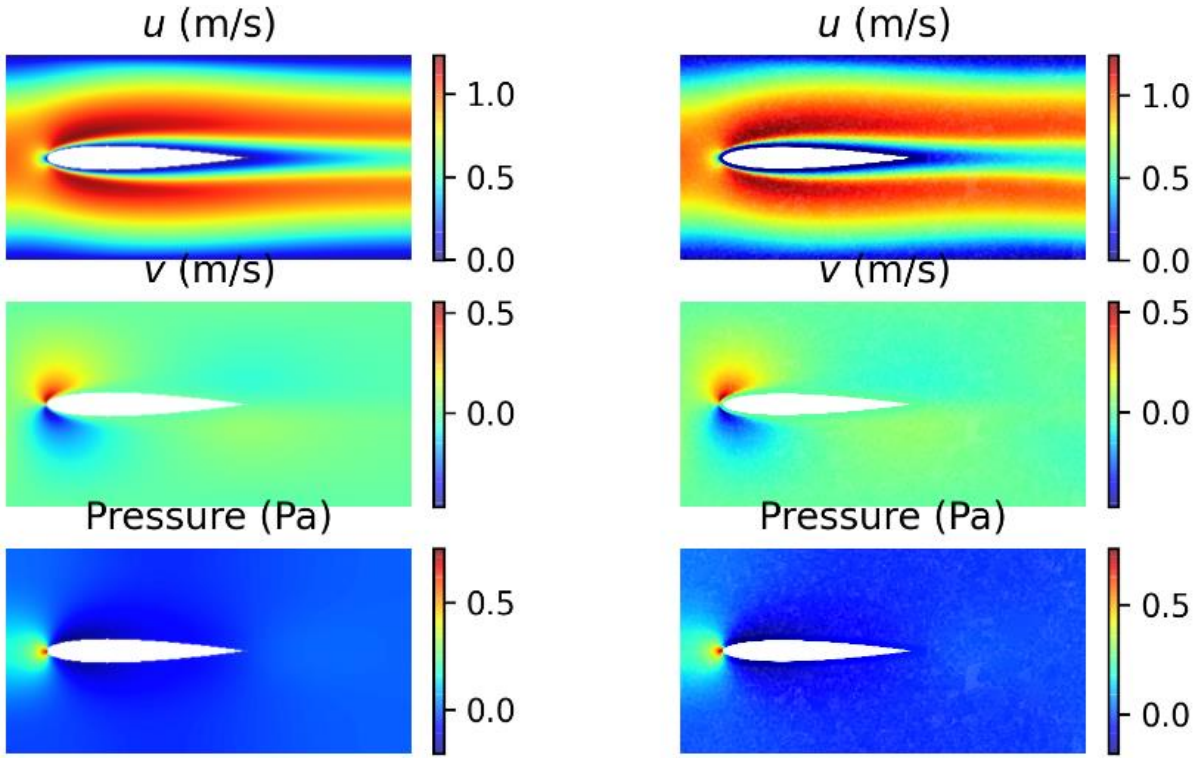


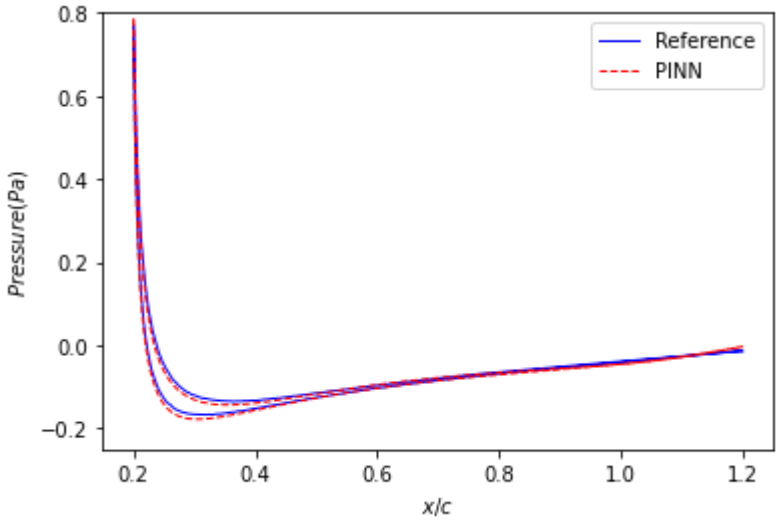
Figure 4.6 Convergence of the physics loss function curve L_p. Adam optimizer is used before the dashed green line, and L-BFGS-B optimizer is used after the dashed green line. The NN size is 8 × 100. (case $\alpha = 0^\circ$)

We observe from the Figures 4.4, 4.7, 4.10 and 4.13 that the steady velocity and pressure fields are well reproduced by the PINN. We obtain good accuracy of the PINN simulation results upon the convergence of the loss function (see Fig. 4.6, 4.9, 4.12 and 4.15). From these figures, we also observe that applying a two-step optimization yields more consistent results.



(a) (b)

Figure 4.7 Velocity and pressure fields of the flow around an NACA 0012 airfoil at $\alpha = 3^{\circ}$: (a) PINN solution with 8 × 100 network; (b) Reference solution from ANSYS Fluent.



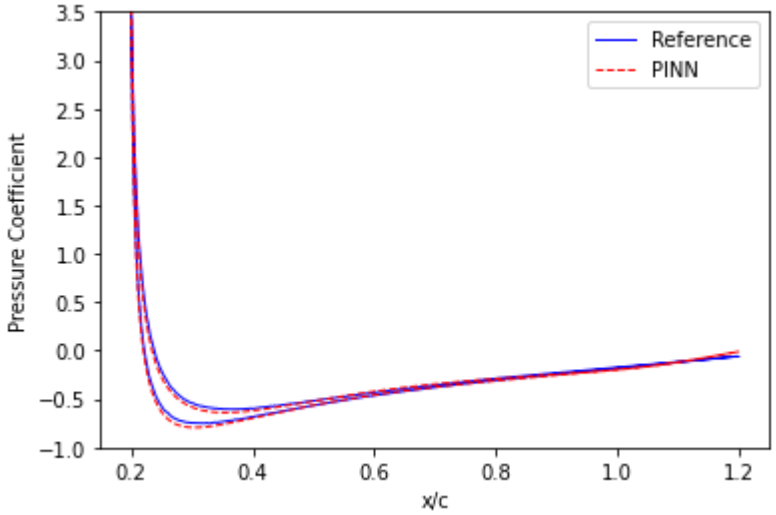


Figure 4.8 Distribution of p and C_p on an airfoil at $\alpha = 3^\circ$. Network of 8 × 100 are used.

It is worth mentioning that the pressure distribution on the airfoil surface is typically of interest for computing the resultant drag and lift forces and subsequently the aerodynamics coefficients (lift C_L and drag C_D coefficients). Therefore, we compare the pressure distributions and the pressure coefficient distributions obtained by PINN and ANSYS Fluent as shown in Fig. 4.5, 4.8, 4.11 and 4.14 for $\alpha = 0^\circ$, $\alpha = 3^\circ$, $\alpha = 9^\circ$ and $\alpha = 12^\circ$, respectively. The overall agreement between the PINN and ANSYS Fluent is very good.

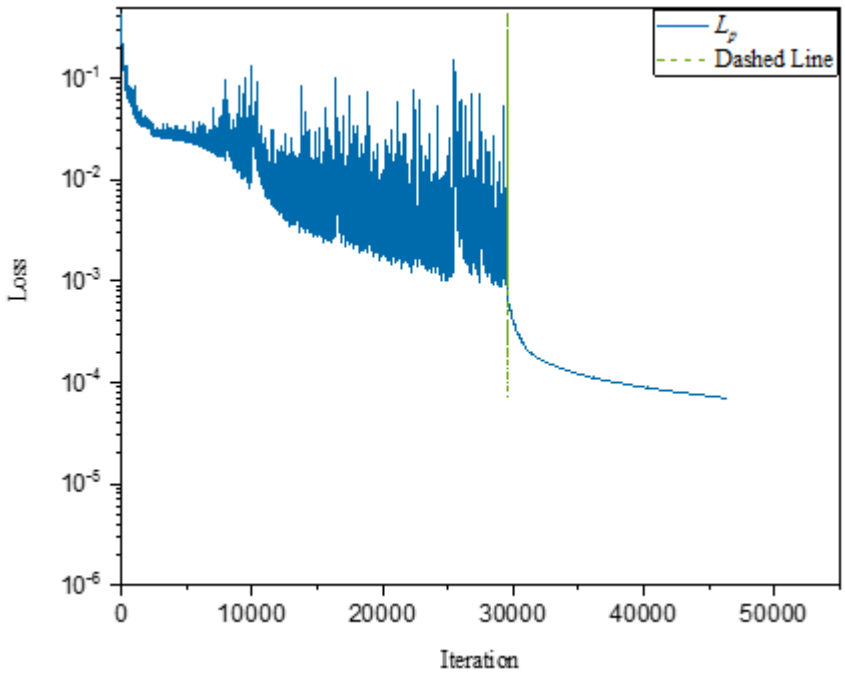
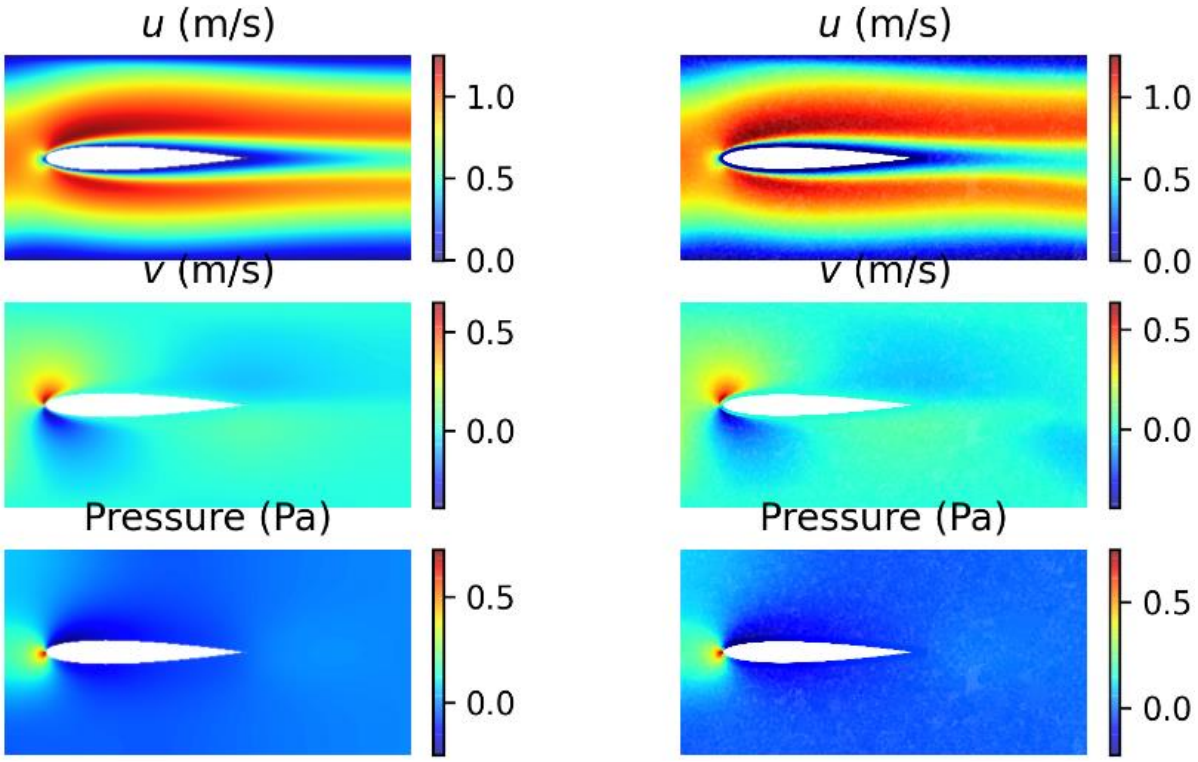
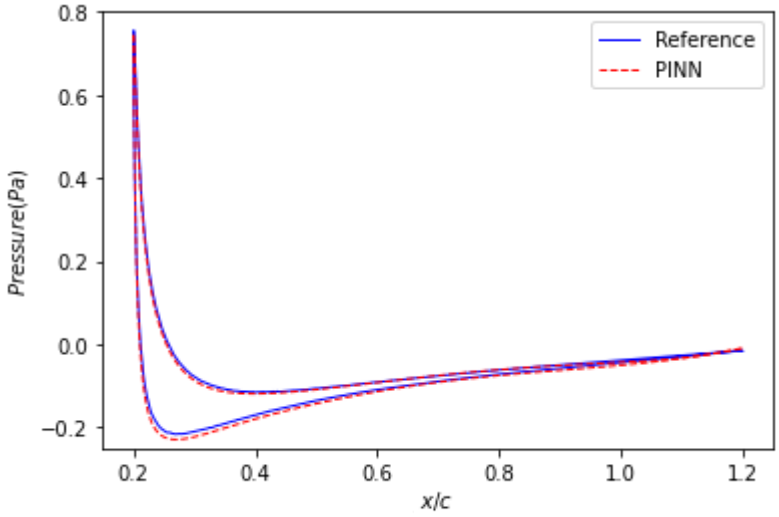


Figure 4.9 Convergence of the physics loss function curve L_p. Adam optimizer is used before the dashed green line, and L-BFGS-B optimizer is used after the dashed green line. The NN size is 8 × 100. (case $\alpha = 3^\circ$)



(a) (b)

Figure 4.10 Velocity and pressure fields of the flow around an NACA 0012 airfoil at $\alpha = 9^{\circ}$ (a) PINN solution with 8×100 network; (b) Reference solution from ANSYS Fluent.



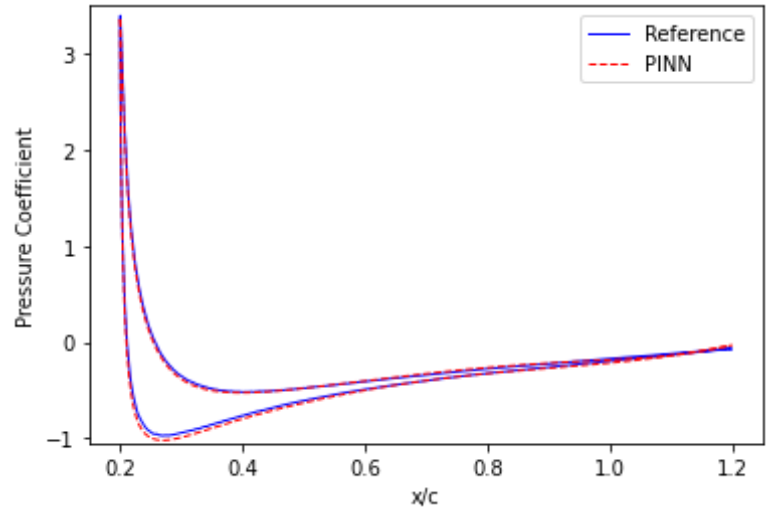


Figure 4.11 Distribution of p and C_p on an airfoil at $\alpha = 9^{\circ}$. Network of 8×100 are used.

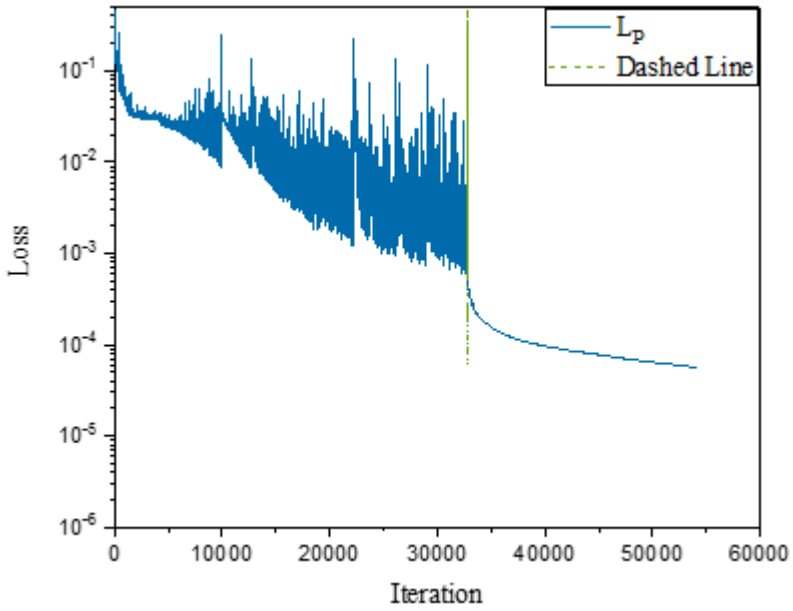
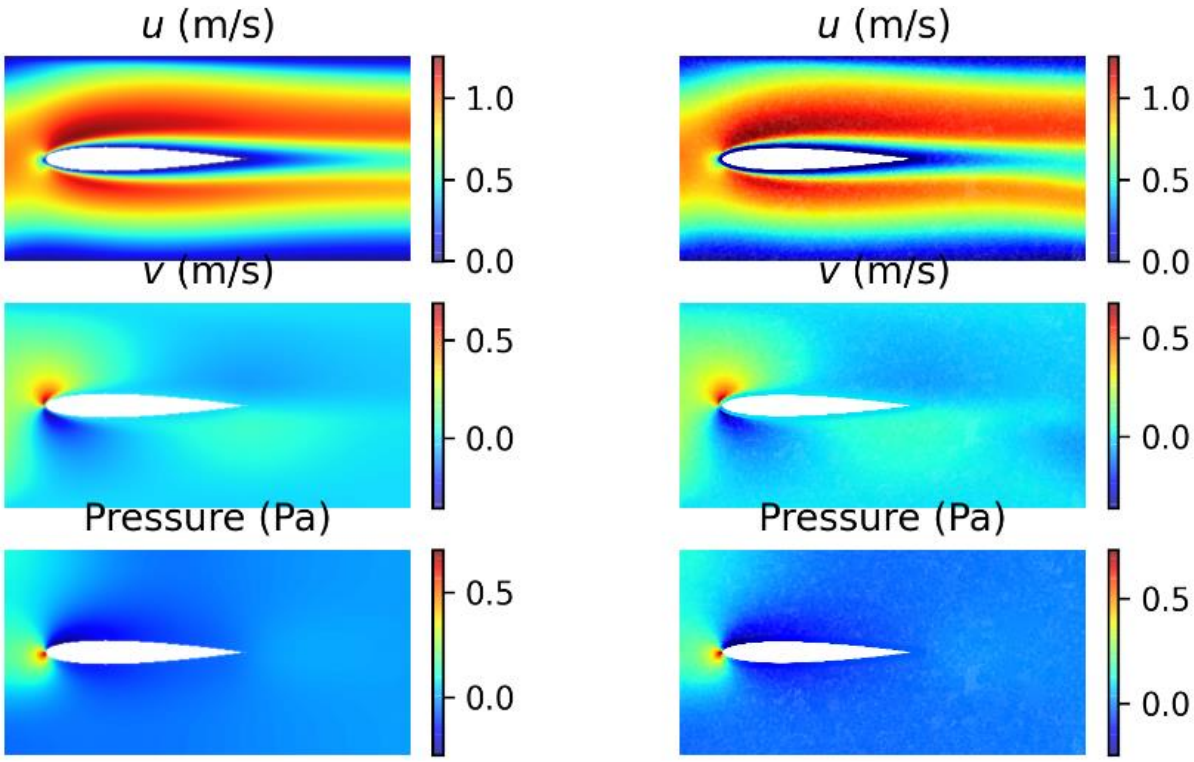


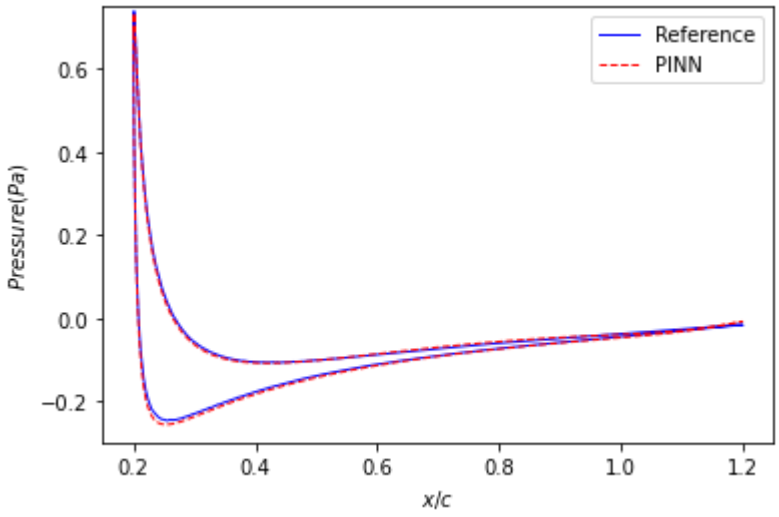
Figure 4.12 Convergence of the physics loss function curve L_p. Adam optimizer is used before the dashed green line, and L-BFGS-B optimizer is used after the dashed green line. The NN size is 8×100. (case $\alpha = 9^{\circ}$)

We obtained good agreement between the Fluent results and the PINN simulation results upon convergence of the loss function.



(a) (b)

Figure 4.13 Velocity and pressure fields of the flow around an NACA 0012 airfoil at $\alpha = 12^{\circ}$ (a) PINN solution with 8×100 network; (b) Reference solution from ANSYS Fluent.



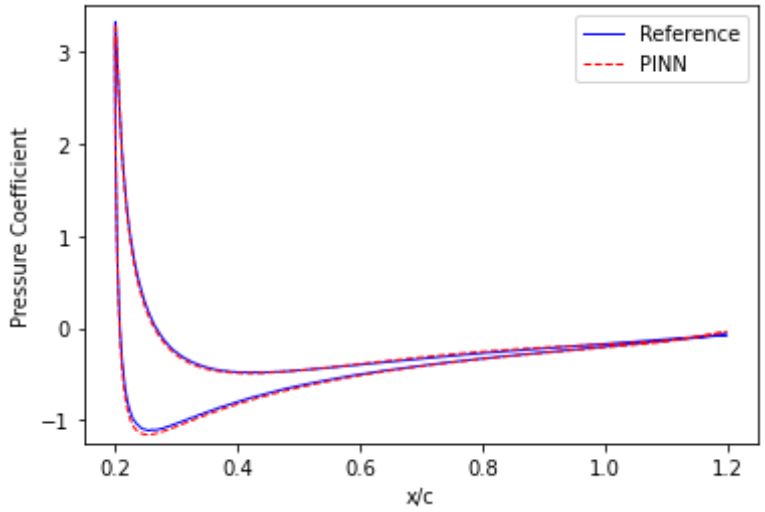


Figure 4.14 Distribution of p and C_p on an airfoil at $\alpha = 12^{\circ}$. Network of 8 × 100 are used.

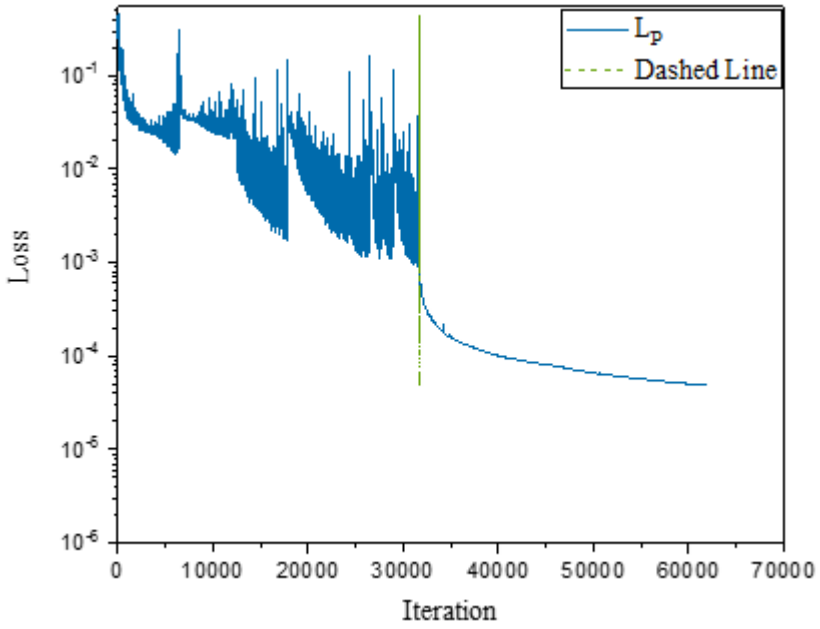


Figure 4.15 Convergence of the physics loss function curve L_p. Adam optimizer is used before the dashed green line, and L-BFGS-B optimizer is used after the dashed green line. The NN size is 8 × 100. (case $\alpha = 12^{\circ}$)

We obtain good accuracy of the PINN simulation results upon convergence of the loss function, verifying that PINN can effectively simulate complex incompressible flows. Thus, the PINN approach enables us to develop Navier-Stokes solvers that do not require mesh generation and achieve higher accuracy.

4.3. Summary

In this study, we explored the effectiveness of PINNs to simulate fluid dynamics problems. We have formulated PINN scheme based on the governing Navier-Stokes equations. The spatial coordinates are the inputs of the PINN, and the velocity and pressure fields are the outputs. We used automatic differentiation to represent all the differential operators in the momentum equations; then the equations can be formulated by the neural networks. We regard the boundary conditions as supervised data-driven part, and the residual of the momentum equations as the unsupervised physics-informed part in the loss function of PINN. Convergence of PINN was monitored using the loss function. We used the PINN framework to simulate the steady two-dimensional viscous incompressible flow at low Reynolds number around an NACA 0012 airfoil at different angle of attack. We obtained good agreement between the ANSYS Fluent results and the PINN simulation results upon convergence of the loss function.

Summary and Conclusions

In this thesis, we explored a new method for solving computational physics problems to surrogate existing Computational Fluid dynamics (CFD) solvers that they present major drawbacks: mesh-generation is complex, cannot tackle high-dimensional problems, time-consuming, etc. This new method as known as Physics-Informed Neural Network (PINN) is based on Machine Learning approach, more specific, artificial neural network. The basic concept of PINN is to embed physical laws to constrain/inform neural networks, with the need of less data for training a reliable model. This can be achieved by incorporating the residual of physics equations into the loss function. Through minimizing the loss function, the network could approximate the solution. PINN has shown great abilities to tackle limitations encountered with CFD solvers. In the implementation part of the thesis, we developed a PINN framework to solve fluid dynamics problems governed by the Navier-Stokes equations. We tested our PINN on the two-dimensional steady incompressible viscous laminar flow around an NACA 0012 airfoil at different angle of attack. We also compared the predicted velocity and pressure fields by the proposed PINN approach with the reference numerical solutions. Simulation results demonstrate great potential of the proposed PINN for fluid flow simulation with a high accuracy.

To conclude, we have shown the efficiency and robustness of the PINN approach by developing our framework (solver) to solve the Navier-Stokes equations. Our future work aims to extend the application of the PINN approach to:

- Solve the compressible Navier-Stokes equations that model several fluid flows, e.g., the flow through converging-diverging nozzles;
- Model turbulence and simulate flows at high Reynolds number;
- Solve acoustics and aeroacoustics problems.

Bibliography

[1] RAISSI, Maziar, PERDIKARIS, Paris, et KARNIADAKIS, George E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 2019, vol. 378, p. 686-707.

[2] RAISSI, Maziar, PERDIKARIS, Paris, et KARNIADAKIS, George Em. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.

[3] GUO, Yanan, CAO, Xiaoqun, LIU, Bainian, *et al.* Solving partial differential equations using deep learning and physical constraints. *Applied Sciences*, 2020, vol. 10, no 17, p. 5917.

[4] RAISSI, Maziar, WANG, Zhicheng, TRIANTAFYLLOU, Michael S., *et al.* Deep learning of vortex-induced vibrations. *Journal of Fluid Mechanics*, 2019, vol. 861, p. 119-137.

[5] KADEETHUM, Teeratorn, JØRGENSEN, Thomas M., et NICK, Hamidreza M. Physics-informed neural networks for solving nonlinear diffusivity and Biot's equations. *PloS one*, 2020, vol. 15, no 5, p. e0232683.

[6] BOUMA, Jort. Using a Physics-Informed Neural Network to solve the Ideal Magnetohydrodynamic Equations. 2020.

[7] JIN, Xiaowei, CAI, Shengze, LI, Hui, *et al.* NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations. *Journal of Computational Physics*, 2021, vol. 426, p. 109951.

[8] YADAV, Neha, YADAV, Anupam, KUMAR, Manoj, *et al. An introduction to neural network methods for differential equations*. Berlin : Springer, 2015.

[9] CHAKRAVERTY, Snehashish et MALL, Susmita. *Artificial neural networks for engineers and scientists: solving ordinary differential equations*. CRC Press, 2017.

[10] RASHID, Tariq. *Make your own neural network*. CreateSpace Independent Publishing Platform, 2016.

[11] KARALI, Hasan, DEMIREZEN, Umut M., YUKSELEN, Mahmut A., *et al.* A Novel Physics Informed Deep Learning Method for Simulation-Based Modelling. In : *AIAA Scitech 2021 Forum*. 2021. p. 0177.

[12] LAGARIS, Isaac E., LIKAS, Aristidis, et FOTIADIS, Dimitrios I. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 1998, vol. 9, no 5, p. 987-1000.

[13] BAI, Xiaodong et ZHANG, Wei. Machine Learning for Vortex Induced Vibration in Turbulent Flow. *arXiv preprint arXiv:2103.05818*, 2021.

[14] SHIN, Yeonjong, DARBON, Jerome, et KARNIADAKIS, George Em. On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs. *arXiv preprint arXiv:2004.01806*, 2020.

[15] SUN, Luning, GAO, Han, PAN, Shaowu, *et al.* Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Computer Methods in Applied Mechanics and Engineering*, 2020, vol. 361, p. 112732.

[16] ZAWAWI, Mohd Hafiz, SALEHA, A., SALWA, A., *et al.* A review: Fundamentals of computational fluid dynamics (CFD). In : *AIP Conference Proceedings*. AIP Publishing LLC, 2018. p. 020252.

[17] WIGHT, Colby. Numerical Approximations of Phase Field Equations with Physics Informed Neural Networks. 2020.

[18] TERLETH, Niels. Artificial Neural Networks for Flow Field Inference: A machine learning approach. 2019.

[19] BRUNTON, Steven L. et KUTZ, J. Nathan. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2019.

[20] EL-AMIR, Hisham et HAMDY, Mahmoud. *Deep learning pipeline: building a deep learning model with TensorFlow*. Apress, 2019.

[21] RUNGTA, Krishna. *TensorFlow in 1 Day: Make your own Neural Network*. 2018.

[22] CHOLLET, Francois. *Deep learning with Python*. Simon and Schuster, 2017.

[23] HOPE, Tom, RESHEFF, Yehezkel S., et LIEDER, Itay. *Learning tensorflow: A guide to building deep learning systems*. " O'Reilly Media, Inc.", 2017.

[24] SINGH, Pramod et MANURE, Avinash. *Learn TensorFlow 2.0: Implement Machine Learning and Deep Learning Models with Python*. Apress, 2020.

[25] JI, Weiqi, QIU, Weilun, SHI, Zhiyu, *et al.* Stiff-pinn: Physics-informed neural network for stiff chemical kinetics. *arXiv preprint arXiv:2011.04520*, 2020.

[26] SIRIGNANO, Justin, MACART, Jonathan F., et FREUND, Jonathan B. DPM: A deep learning PDE augmentation method with application to large-eddy simulation. *Journal of Computational Physics*, 2020, vol. 423, p. 109811.

[27] MENG, Xuhui, LI, Zhen, ZHANG, Dongkun, *et al.* PPINN: Parareal physics-informed neural network for time-dependent PDEs. *Computer Methods in Applied Mechanics and Engineering*, 2020, vol. 370, p. 113250.

[28] RAO, Chengping, SUN, Hao, et LIU, Yang. Physics-Informed Deep Learning for Computational Elastodynamics without Labeled Data. *Journal of Engineering Mechanics*, 2021, vol. 147, no 8, p. 04021043.

[29] WASEI, E. A. A. Investigating Physics-Informed Neural Networks for solving PDEs. 2020.

[30] BRUNTON, Steven L., NOACK, Bernd R., et KOUMOUTSAKOS, Petros. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 2020, vol. 52, p. 477-508.

[31] ELTVIK, Audun. *Deep Learning for the Classification of EEG Time-Frequency Representations*. 2018. Thèse de maîtrise. NTNU.

[32] WESSELS, Henning, WEIßENFELS, Christian, et WRIGGERS, Peter. The neural particle method–an updated Lagrangian physics informed neural network for computational fluid dynamics. *Computer Methods in Applied Mechanics and Engineering*, 2020, vol. 368, p. 113127.

[33] NASCIMENTO, Renato G., FRICKE, Kajetan, et VIANA, Felipe AC. A tutorial on solving ordinary differential equations using Python and hybrid physics-informed neural network. *Engineering Applications of Artificial Intelligence*, 2020, vol. 96, p. 103996.

[34] BEKELE, Yared W. Deep learning for one-dimensional consolidation. *arXiv preprint arXiv:2004.11689*, 2020.

[35] MISYRIS, George S., VENZKE, Andreas, et CHATZIVASILEIADIS, Spyros. Physics-informed neural networks for power systems. In : *2020 IEEE Power & Energy Society General Meeting (PESGM)*. IEEE, 2020. p. 1-5.

[36] KHARAZMI, Ehsan, ZHANG, Zhongqiang, et KARNIADAKIS, George Em. Variational physics-informed neural networks for solving partial differential equations. *arXiv preprint arXiv:1912.00873*, 2019.

[37] YUCESAN, Yigit A. et VIANA, Felipe AC. A physics-informed neural network for wind turbine main bearing fatigue. *International Journal of Prognostics and Health Management*, 2020, vol. 11, no 1.

[38] MAO, Zhiping, JAGTAP, Ameya D., et KARNIADAKIS, George Em. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 2020, vol. 360, p. 112789.

[39] PANG, Guofei, LU, Lu, et KARNIADAKIS, George Em. fPINNs: Fractional physics-informed neural networks. *SIAM Journal on Scientific Computing*, 2019, vol. 41, no 4, p. A2603-A2626.

[40] RAO, Chengping, SUN, Hao, et LIU, Yang. Physics-informed deep learning for incompressible laminar flows. *Theoretical and Applied Mechanics Letters*, 2020, vol. 10, no 3, p. 207-212.

[41] RABAULT, Jean et KUHNLE, Alexander. Accelerating deep reinforcement learning strategies of flow control through a multi-environment approach. *Physics of Fluids*, 2019, vol. 31, no 9, p. 094105.

[42] VIQUERAT, Jonathan et HACHEM, Elie. A supervised neural network for drag prediction of arbitrary 2D shapes in laminar flows at low Reynolds number. *Computers & Fluids*, 2020, vol. 210, p. 104645.

[43] ISKHAKOV, Arsen S. et DINH, Nam T. Physics-integrated machine learning: embedding a neural network in the Navier-Stokes equations. Part I. *arXiv preprint arXiv:2008.10509*, 2020.

[44] KOCHKOV, Dmitrii, SMITH, Jamie A., ALIEVA, Ayya, *et al.* Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 2021, vol. 118, no 21.

[45] SADREHAGHIGHI, Ideen. Artificial Neutral Networks (ANNs) Applied as CFD Optimization Techniques.

[46] LYE, Kjetil O., MISHRA, Siddhartha, et RAY, Deep. Deep learning observables in computational fluid dynamics. *Journal of Computational Physics*, 2020, vol. 410, p. 109339.

[47] BAYMANI, Mojtaba, EFFATI, Sohrab, NIAZMAND, Hamid, *et al.* Artificial neural network method for solving the Navier–Stokes equations. *Neural Computing and Applications*, 2015, vol. 26, no 4, p. 765-773.

[48] CHENG, Chen et ZHANG, Guang-Tao. Deep learning method based on physics informed neural network with resnet block for solving fluid flow problems. *Water*, 2021, vol. 13, no 4, p. 423.

[49] PANWAR, Vishwanath, VANDRANGI, Seshu Kumar, et EMANI, Sampath. Artificial intelligence-based computational fluid dynamics approaches. In : *Hybrid Computational Intelligence*. Academic Press, 2020. p. 173-190.

[50] MCCRACKEN, Megan F. Artificial neural networks in fluid dynamics: A novel approach to the navier-stokes equations. In : *Proceedings of the Practice and Experience on Advanced Research Computing*. 2018. p. 1-4.

[51] BAI, Xiao-dong, WANG, Yong, et ZHANG, Wei. Applying physics informed neural network for flow data assimilation. *Journal of Hydrodynamics*, 2020, vol. 32, no 6, p. 1050-1058.

[52] FALLER, William E. et SCHRECK, Scott J. Unsteady fluid mechanics applications of neural networks. *Journal of aircraft*, 1997, vol. 34, no 1, p. 48-55.

[53] LUNDSTRÖM, Robin. Machine Learning for Air Flow Characterization: An application of Theory-Guided Data Science for Air Fow characterization in an Industrial Foundry. 2019.

[54] HARYANTO, Ismoyo, UTOMO, Tony Suryo, SINAGA, Nazaruddin, *et al.* Optimization of maximum lift to drag ratio on airfoil design based on artificial neural network utilizing genetic algorithm. In : *Applied Mechanics and Materials*. Trans Tech Publications Ltd, 2014. p. 123-128.

[55] OBIOLS-SALES, Octavi, VISHNU, Abhinav, MALAYA, Nicholas, *et al.* CFDNet: A deep learning-based accelerator for fluid simulations. In : *Proceedings of the 34th ACM International Conference on Supercomputing*. 2020. p. 1-12.

[56] SECCO, Ney R. et MATTOS, Bento S. Artificial neural networks applied to airplane design. In : *53rd AIAA Aerospace Sciences Meeting*. 2015. p. 1013.

[57] ZHU, Linyang, ZHANG, Weiwei, KOU, Jiaqing, *et al.* Machine learning methods for turbulence modeling in subsonic flows around airfoils. *Physics of Fluids*, 2019, vol. 31, no 1, p. 015105.

[58] WANG, Zheng, XIAO, Dunhui, FANG, Fangxin, *et al.* Model identification of reduced order fluid dynamics systems using deep learning. *International Journal for Numerical Methods in Fluids*, 2018, vol. 86, no 4, p. 255-268.

[59] XIE, Chenyue, XIONG, Xiangming, et WANG, Jianchun. Artificial neural network approach for turbulence models: A local framework. *arXiv preprint arXiv:2101.10528*, 2021.

[60] PAWAR, Suraj, SAN, Omer, AKSOYLU, Burak, *et al.* Physics guided machine learning using simplified theories. *Physics of Fluids*, 2021, vol. 33, no 1, p. 011701.

[61] HARYANTO, Ismoyo, UTOMO, Tony Suryo, SINAGA, Nazaruddin, *et al.* Optimization of maximum lift to drag ratio on airfoil design based on artificial neural network utilizing genetic algorithm. In : *Applied Mechanics and Materials*. Trans Tech Publications Ltd, 2014. p. 123-128.

[62] ASHGRIZ, Nasser et MOSTAGHIMI, Javad. An introduction to computational fluid dynamics. *Fluid flow handbook*, 2002, vol. 1, p. 1-49.

[63] LOMAX, Harvard, PULLIAM, Thomas H., et ZINGG, David W. *Fundamentals of computational fluid dynamics*. Springer Science & Business Media, 2013.

[64] HIRSCH, Charles. *Numerical computation of internal and external flows: The fundamentals of computational fluid dynamics*. Elsevier, 2007.

[65] KELLEHER, John D., MAC NAMEE, Brian, et D'ARCY, Aoife. *Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies*. MIT press, 2020.

[66] DU, Ke-Lin et SWAMY, M. N. S. Fundamentals of machine learning. In : *Neural Networks and Statistical Learning*. Springer, London, 2014. p. 15-65.

[67] FRANK, Michael, DRIKAKIS, Dimitris, et CHARISSIS, Vassilis. Machine-learning methods for computational science and engineering. *Computation*, 2020, vol. 8, no 1, p. 15.

[68] EROĞLU, Muhammet Yusuf. PREDICTION OF DRAG FORCE USING COMPUTATIONAL FLUID DYNAMICS BASED ARTIFICIAL NEURAL NETWORK MODEL. 2019.

[69] MIYANAWALA, Tharindu P. et JAIMAN, Rajeev K. An efficient deep learning technique for the Navier-Stokes equations: Application to unsteady wake flow dynamics. *arXiv preprint arXiv:1710.09099*, 2017.

[70] RANADE, Rishikesh, HILL, Chris, et PATHAK, Jay. DiscretizationNet: A machine-learning based solver for Navier–Stokes equations using finite volume discretization. *Computer Methods in Applied Mechanics and Engineering*, 2021, vol. 378, p. 113722.

[71] SEONG, Yongho, PARK, Changhyup, CHOI, Jinho, *et al.* Surrogate Model with a Deep Neural Network to Evaluate Gas–Liquid Flow in a Horizontal Pipe. *Energies*, 2020, vol. 13, no 4, p. 968.

[72] BRUNTON, Steven L., HEMATI, Maziar S., et TAIRA, Kunihiko. Special issue on machine learning and data-driven methods in fluid dynamics. 2020.

[73] PEREIRA, Francisco Câmara et BORYSOV, Stanislav S. Machine learning fundamentals. In : *Mobility Patterns, Big Data and Transport Analytics*. Elsevier, 2019. p. 9-29.

[74] ANDERSON, J. D. Basic philosophy of CFD. In : *Computational Fluid Dynamics*. Springer, Berlin, Heidelberg, 2009. p. 3-14.

[75] ZHAO, Yaomin, AKOLEKAR, Harshal D., WEATHERITT, Jack, *et al.* RANS turbulence model development using CFD-driven machine learning. *Journal of Computational Physics*, 2020, vol. 411, p. 109413.