Deep learning applied to data-driven discovery of partial differential equations in fluid mechanics.

Jean Jonathan Schuster

Post-graduation in Robotics and Artificial Intelligence
Universidad Tecnológica

Rivera – Uruguay
December of 2021

Deep learning applied to data-driven discovery of partial differential equations in fluid mechanics.

Jean Jonathan Schuster

Project of Post-graduation submitted to the course of Robotics and Artificial Intelligence, from the Technologic University of Uruguay, as part of the requirements necessary to obtain the title of Epecialist in Robotics and Artificial Intelligence.

Tutor:
Ph.D. Prof. Emanuel da Silva Diaz Estrada

Rivera – Uruguay
December of 2021

Schuster, Jean Jonathan

Deep learning applied to data-driven discovery of partial differential equations in fluid mechanics. / Jean Jonathan Schuster. - Rivera: Universidad Tecnológica,

xi, 37 p.: il.; 29, 7cm.

Tutor:

Emanuel da Silva Diaz Estrada

Project de Post-graduation – Universidad Tecnológica, Post-graduation in Robotics and Artificial Intelligence, 2021.

Bibliographic references: p. 34 – 37.

1. Red neuronal informada por la física, 2. Descubrimiento de ecuaciones diferenciales parciales basado en datos, 3. Problemas inversos.

I. da Silva Diaz Estrada, Emanuel, . II. Universidad Tecnológica, Post-graduation in Robotics and Artificial Intelligence. III. Título.

MEMBERS OF THE DEFENSE COURT OF THE PROJECT

_____

Ph.D. Prof. Emanuel da Silva Diaz Estrada

_____

Ph.D. Prof. Paulo Drews-Jr

_____

Ph.D. Prof. Elizaldo Domingues dos Santos

Rivera – Uruguay
December of 2021

ABSTRACT

The concept of Physics-Informed Neural Networks (PINNs) is applied for solving 2D incompressible flow over a bluff body with a square shape, forming a von Kármán street vortex. We create the dataset through Computational Fluid Dynamics simulation for training the PINNs, and aim to discover latent solutions through supervised learning tasks while respecting any given laws of physics described by Navier-Stokes and continuity equations. We were able to create our dataset, and correctly apply the methodology to discover parameters of the Navier-Stokes equations and the pressure field without feeding the algorithm with pressure data. Noisy was introduced in the data to study the performance of the PINNs in this situation, and even with sparse and scarce data the algorithm was able to identify the unknown parameter with great precision.

Keywords:
Physical-Informed Neural Network, Data-driven discovery of partial differential equations, Inverse problems.

RESUMEN

Se aplica el concepto de Redes Neuronales Informadas por la Física (PINN) para resolver un flujo incompresible 2D sobre un cuerpo bruto no aerodinámico, formando una calle de vórtices von Kármán. Creamos el conjunto de datos a través de la simulación de dinámica de fluidos computacional para entrenar el PINN, y nuestro objetivo es descubrir soluciones latentes a través de tareas de aprendizaje supervisadas, respetando las leyes de la física descritas por las ecuaciones de Navier-Stokes y de la continuidad. Pudimos crear nuestro conjunto de datos y aplicar correctamente la metodología para descubrir los parámetros de las ecuaciones de Navier-Stokes, y el campo de presión sin alimentar el algoritmo con datos de presión. Se introdujo ruido en los datos para estudiar el desempeño de los PINN en esta situación, e incluso con datos escasos y esparcidos el algoritmo fue capaz de identificar el parámetro desconocido con gran precisión.

Palabras claves:
Red neuronal informada por la física, Descubrimiento de ecuaciones diferenciales parciales basado en datos, Problemas inversos.

List of Figures

List of Tables

Acronyms

AAAI Association for the Advancement of Artificial Intelligence 3

AI Artificial Intelligence 2, 11, 17

CFD Computational Fluid Dynamics 20, 21, 23, 26, 32

CNN Convolutional Neural Network 3

DL Deep Learning 5, 11, 17, 18, 19, 20, 22, 23, 24

DNN Deep Neural Network 11

DNS Direct Numerical Simulation 1

FEM Finite Elements Method 32

HFM Hidden Fluid Mechanics 4

L-BFGS-B Limited-memory Broyden–Fletcher–Goldfarb–Shanno B 25

LES Large Eddy Simulation 2

ML Machine Learning 11, 17, 18, 19

NN Neural Network 11, 12, 13, 14, 15, 16, 19, 22, 23

NS Navier-Stokes 1, 10, 21, 22, 23, 24, 28, 29

PDE Partial Differential Equation 4, 20, 24, 32

PINN Physical-Informed Neural Network 22, 26, 27, 28, 30, 32

RANS Reynolds-Averaged Navier-Stokes 1

Contents

1 Introduction

In a time of profound transformations in society at speeds never seen before, arising from a connected and technological world, where society is influenced by codes of artificial intelligence and machine learning, humanity requires ever faster responses. For example, in large conurbations, we may have entire cities, or large regions of agriculture, affected by a local microclimate, called an urban heat island, which can be generated due to both its geographical characteristics and human activity, affecting directly the daily lives of the population in these areas, and is a growing problem due to the increase in urbanization on the Planet.

The dynamics of these microclimates, and weather in general, can be modeled by a set of equations defined by conservation laws, like conservation of mass, energy and momentum. From those laws arise the well known Navier-Stokes (NS) equations that describe the fluid motion (Pope, 2000).

NS equations smooth solution were obtained only for the case of simple two-dimensional flows, whereas three-dimensional turbulent flows, that have complex vorticity and chaotic behavior, have proven intractable to any analytic solution but numerical approximation analysis methods , like Direct Numerical Simulation (DNS) where al spatial and temporal turbulence scales are resolved (Phillips, 2018).

Due to extreme computational cost of Direct Numerical Simulation (DNS), other methods arise, like the idea proposed by Osborne Reynolds (1985), whereby an instantaneous quantity is decomposed into its time-averaged and fluctuating quantities, today well known as Reynolds-Averaged Navier-Stokes (RANS) equations. Other approaches try to remove the small scales from the numerical solution by a low-pass filter of the NS equations, once its information is not relevant and represent the major part of the computational cost. However, those small scales cannot be ignored because they play an important role, mainly in energy dissipation, and therefore must be modeled. This small

scales model was first proposed by Smagorinsky, 1963, on atmospheric air currents simulation, and is known as Large Eddy Simulation (LES). After that many other LES models arise, like those proposed by Germano et al., 1991, and Meneveau and Katz, 2000.

Those numerical experiments, so called numerical flow simulations, are used in weather forecast, aerodynamic problems: from automobiles, airplanes, buildings, wind turbine blades; flow control and optimization and other engineering studies applied to fluid mechanics.

However, due to the complexity of the equations that describe the flow of a fluid, these are problems where the applied modeling techniques imply a very high processing time, or overly in simplified models that may not represent many aspects of the physical reality of the problem.

Thus, it is very important that the numerical models for fluid flows are able to combine the physical properties and also techniques that reduce the machine time of the simulations. For example, in a numerical forecast of the weather it is necessary to obtain its results in advance, because it is useless for a model to correctly forecast the weather conditions after they have occurred.

In fact, Brunton et al., 2020, state that the extremely wide range of spatio-temporal scales nature of turbulence, makes it to computational expensive to resolve all scales in a numeric experiment, even with Moore's law, there are several decades away from resolving all turbulence scales for simulations that interest to humanity activities.

In front of this scenario, efforts to develop methods to minimize computational cost in fluid mechanics have been made since the 60'. Much of those methods were around since the 50' in many names like Cybernetics, automata theory, and complex information processing, all of them related to machine thinking (McCorduck & Cfe, 2004).

So, in 1956, all those disciplines related to machine thinking gave rise to a new science in a workshop at Dartmouth College, the Artificial Intelligence (AI), conceived by John McCarthy and completely separate from the Cybernetics field (Crevier, 1993; McCorduck & Cfe, 2004; Russell & Norvig, 2021).

Among the many proposed methods of model and treat turbulence, one with especial interest to this work, trace back from the early 60's, when Rechenberg, 1964, used a branch of machine learning, closely related to Cybernetics (Wiener, 2019), where evolutionary algorithms were used to find a configuration of angles of a corrugated structure composed of 5 linked plates, that

would have the minimum drag possible in a wind tunnel experiment. The five angles of the plates were generated randomly by a Galton board, producing a Gaussian distribution that introduced stochasticity into the optimization, and the increase or decrease of the angles was learned based on the success rate of the experiments. The experiment shows that the optimal drag configuration of the plates was a smooth convex upwards shape. Rechenberg experiment remotes, maybe, the first application of artificial intelligence to a fluid mechanics problem.

Despite all the euphoria about AI in the 50', 60' and early 70', in 1973, James Lighthill made a report evaluating the current state of AI written for the British Science Research Council.

The report, know later as the Lighthill report, concluded that the potential promises of AI was exaggerated, and there was no discovery so far that result in the promises made by researches, it even pointed out that the most disappointing research area had been machine translation, where great amounts of resources ware been spent with practically no fruitful results obtained, mainly by the lack of computational power (Schuchmann, 2019).

The chronological development of AI, according to some authors, are classified as the first AI boom, between 50'an 60', starting with Alan Turing, when he arbitrarily set the standard for what could be regarded as a "thinking machine", later one know as the Turing test. In the 70', the decade is mark as the first AI winter, mainly after the Lighthill report was released. The 80' are classified by many authors as the second AI boom, but at this time a lot more effort was focused on creating commercial products, and the emerge of large conferences like Association for the Advancement of Artificial Intelligence (AAAI), that experienced a rapid growth, refreshing general industry and government interest in AI technology. The next decade, 90', is knows as the the second AI winter, where the expectation about AI decreased until the turn of the century, where computational power and the data availability provided by the internet, started to renew the interest in this area, until in 2012, when a new architecture of NN arise, the Convolutional Neural Network (CNN), called AlexNet, designed by Alex Krizhevsky in collaboration with Ilya Sutskever and Geoffrey Hinton in a Ph.D. study, given place to a AI revolution that is still on going (Lim, 2019; Ray, 2021; Schuchmann, 2019).

In the last decades, many researches in NN's used to model dynamic systems have been published, some of them applied to transport phenomena

problems that is of special interest in this work, many of them with Physics-informed Machine Learning in the last years, given place to accelerated growth in this field (Almajid & Abu-Al-Saud, 2022; Mao et al., 2020; Ramabathiran & Ramachandran, 2021; Sun et al., 2020; Yu et al., 2017). In 90' Dissanayake and Phan-Thien, 1994 and Lagaris et al., 1998 used NN's to learn solutions of differential equations, and most recently Raissi and Karniadakis, 2018 and Raissi et al., 2018 proposed a framework for models that can mix physical and conservation principles, or phenomenological behaviors represented by partial differential equations with the data sets available in science, engineering and technology. Where they addressed the problem of inferring solutions of time dependent and nonlinear partial differential equations, even in noisy observations, to deal with the problem of learning, system identification, or data-driven discovery of partial differential equations in discrete and continuous in time networks.

As a demonstration of the early work cited above, Raissi et al., 2020 developed an approach applied on fluid mechanics problems, which they called Hidden Fluid Mechanics (HFM), that aims to use simultaneously information available in images from the flow visualization and the NS equations combined, given place to a physics-informed deep learning approach.

Most of the efforts mentioned above came due the fact that solutions of Partial Differential Equation (PDE) can be challenging, or have no analytic solution at all, like NS equations in a turbulent flow, as mentioned early, requiring an enormous computational effort to solve it numerically, making it impossible to use in real time applications, like a feedback control system, or in hour accurate weather forecast, to cite a couple of examples (Brunton et al., 2020).

So, the importance of new strategies using some machine learning approach rests in the fact that many applications require real time results, in special in control applications, where using a mathematical model is not possible, for example, a drone nowadays can not control the air flow around it to hovering in the air only by moving its aerodynamic control surfaces, for this to be possible it is necessary to use actuators such as propellers, and it demands lots of energy, while some birds may stand still in the air controlling the air flow around its wings, stretching, shrinking and rotating them.

As far as is known, birds do not know the NS equations, but somehow they have learned to control the airflow around them to their advantage. As we are

far away from obtain real-time result from NS equations solutions in flows of interest, as mentioned early, learn some features without solving the entire flow field in a short window of time could improve manned and unmanned aquatic and aerial vehicles autonomy and efficiency, by using those learned quantities in control applications, to name and example, since a feedback loop could be closed in real time.

Thus, this work intends to answer the following question: is it possible to obtain similar results as Raissi et al., 2019 by creating a dataset through CFD simulation in a slightly more complex bluff body, like a column with square profile, learning the flow field, pressure field and estimate parameter of the PDE's that describe the flow?

1.1 Objectives

Therefore, this work has as general objective to create a dataset composed from a velocity flow field of a two dimensional (2D) classical problem of a flow over a solid corpus through a numerical simulation, generating a von Kármán vortex street, and then apply a Deep Learning (DL) framework to discover PDEs coefficients through data-driven inference.

1.1.1 Specific objectives

- Infer the hidden states of the system, in this case the pressure field (no training pressure data will be feed), from partial knowledge of the velocity field flow by leveraging the known underlying dynamics of the system.
- Train the DL algorithm to learn the entire velocity flow field from sparse and scarce velocity training data.

2 Theoretical foundations

The next sections will shed light on fundamental concepts of fluid mechanics an artificial intelligence that will be covered in this work.

2.1 Field equations for Newtonian fluid flow

The three-dimensional movement of a flow field is given by the velocity vector $\vec{v} = u\vec{i} + v\vec{j} + w\vec{k}$, being the tree component of the velocity field u, v and w parallel to the axes $\vec{i}$, $\vec{j}$ and $\vec{k}$, in a Cartesian plane respectively. Also, the pressure p and temperature T field can be necessary to determine flow field in certain condition, like in a presence of a pressure driven flow, like Poiseuille Flow (Chicone, 2017), or in a presence of a temperature gradient, like happen in a natural convection flow (Balaji et al., 2021).

In order to determine these five variables, five equations are required, which are the continuity equation (mass conservation), the three momentum conservation equations and the energy equation (energy conservation). These equations are obtained by Newton's laws of motion and by the first and second laws of thermodynamics, joined through the Reynolds transport theorem for a fixed and immutable control volume. These five conservation equations contain elements that are physical properties dependent on temperature and pressure, such as density $\rho(T, p)$, specific heat at constant pressure $C_p(T, p)$, viscosity $\mu(T, p)$ and the thermal conductivity $K(T, p)$ (White, 2021).

In this section, the basic equations for Newtonian fluid flow fields are presented. Here the fluid is considered as continuous, isotropic, and in accordance whit the Fourier law of heat conduction.

2.1.1 Continuity equation

The continuity equation or mass conservation equation, expresses the fact that the sum of all mass per volume unit entering or leaving a control volume per unit of time, must be equal to the change of mass due to the change in density per unit of time. This law can be expresses in a general manner as Equation 2.1 (Young et al., 2010).

$$\frac{D\rho}{Dt} + \rho\vec{\nabla} \cdot \vec{v} = \frac{\partial\rho}{\partial t} + \vec{\nabla} \cdot (\rho\vec{v}) = 0 \tag{2.1}$$

Where the first term in the above equation, $\frac{D\rho}{Dt}$, is the total derivative/-material of the density in time, where it can be split in tow terms: $\frac{\partial\rho}{\partial t}$, that represents the rate of density change with respect to time ins transient incompressible flows; $\vec{v} \cdot \vec{\nabla}\rho$, that represents the change in position in time. This parcel can be seen in the Equation 2.2.

$$\frac{D\rho}{Dt} = \frac{\partial\rho}{\partial t} + \vec{v} \cdot \vec{\nabla}(\rho) \tag{2.2}$$

In flows where the assumption of incompressible fluid can be made, ie the particle density does not change as it travels, the total derivative can be set to zero ($\frac{D\rho}{Dt} = 0$). The continuity equation for incompressible flows whit this assumption, and divided by ρ can be write as the Equation 2.3 (Schlichting & Gersten, 2003).

$$\vec{\nabla} \cdot \vec{v} = \frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} + \frac{\partial u_z}{\partial z} = 0 \tag{2.3}$$

2.1.2 Momentum equations

These equations come from Newton's second law, which states that the product of mass and acceleration is directly proportional to the resultant of all forces acting on the body. These forces are body forces (due to the acceleration of mass by gravity, or electromagnetic forces in case of ferrofluid) and surface forces (forces of pressure or friction). Body forces are external forces, while surface forces depend on the fluid's state of motion (deformation state).

If the body force per unit volume is the product of density and the acceleration of gravity ($\vec{f} = \rho\vec{g}$) and $\vec{P}$ the force of surface per volume unit, we have the Equation 2.4 representing the momentum equation (Kundu et al., 2015).

$$\rho \frac{D\vec{v}}{Dt} = \frac{\partial \vec{v}}{\partial t} + \frac{d\vec{v}}{dt} = \vec{f} + \vec{P} \tag{2.4}$$

Where the material derivative of acceleration $\frac{D\vec{v}}{Dt}$ is composed of the local acceleration $\frac{\partial \vec{v}}{\partial t}$ plus the convective acceleration $\frac{d\vec{v}}{dt}$, the latter as a consequence of the change in position and is represented by the Equation 2.5 (Schlichting & Gersten, 2003).

$$\frac{d\vec{v}}{dt} = \vec{\nabla}\left(\frac{1}{2}\vec{v}^2\right) - \vec{v} \times \vec{\nabla} \times \vec{v} \tag{2.5}$$

The surface forces vector $\vec{P}$ in the Equation 2.4, can be seen in the Equation 2.6, and are obtained by a balance of the surface forces of the state of stresses in Figure 2.1, here shown only in x direction for simplicity.

$$\vec{P} = \frac{\partial \vec{p}_x}{\partial x} + \frac{\partial \vec{p}_y}{\partial y} + \frac{\partial \vec{p}_z}{\partial z} \tag{2.6}$$

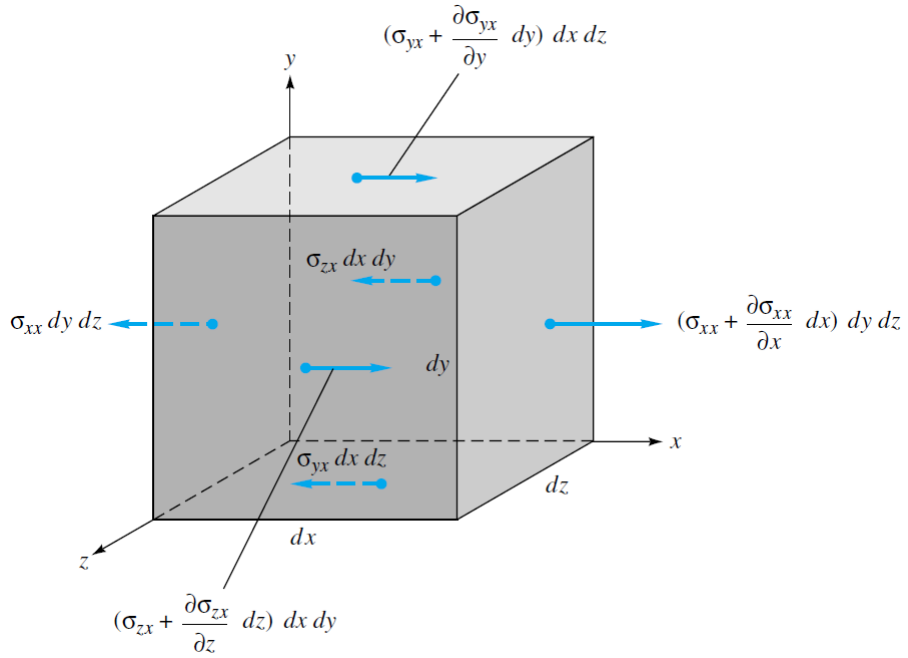


Figure 2.1: Elemental cartesian fixed control volume showing the surface forces in the x direction only (White, 2021).

Thus, we can define the surface force for the Cartesian plane according to Equations 2.7, where the shear stresses $\tau_{ij}\vec{k}$ acts tangential to the plane ij in the $\vec{k}$ axis direction.

$$\begin{aligned}\vec{p_x} &= \sigma_x\vec{i} + \tau_{xy}\vec{j} + \tau_{xz}\vec{k} \\ \vec{p_y} &= \tau_{yx}\vec{i} + \sigma_y\vec{j} + \tau_{yz}\vec{k} \\ \vec{p_z} &= \tau_{zx}\vec{i} + \tau_{zy}\vec{k} + \sigma_z\vec{k}\end{aligned} \tag{2.7}$$

Those surface forces can be represented as a symmetric tensor know as stress tensor, and can be seen en the Equation 2.8. Since the tensor is symmetrical, the shear stresses are equal in magnitude, but act in the opposite direction, for example $\tau_{xy} - \tau_{yx} = 0 \rightarrow \tau_{xy} = \tau_{yx}$.

$$\boldsymbol{\sigma} = \begin{pmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{pmatrix} \tag{2.8}$$

Replacing Equations 2.7 in the Equation 2.6,and separating the pressure from the normal stresses, we have that: $\tau_{xx} = \sigma_x + p$, $\tau_{yy} = \sigma_y + p$, $\tau_{zz} = \sigma_z + p$. Decomposing the stresses into the normal part, which is equal in all directions, and the shear part and replacing it in Equation 2.6, we have the three momentum equations for one cartesian coordinate system, Equations 2.9 White, 2021.

$$\begin{aligned}\rho\frac{Du}{Dt} &= f_x - \frac{\partial p}{\partial x} + \left(\frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z}\right) \\ \rho\frac{Dv}{Dt} &= f_y - \frac{\partial p}{\partial y} + \left(\frac{\partial \tau_{yx}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y} + \frac{\partial \tau_{yz}}{\partial z}\right) \\ \rho\frac{Dw}{Dt} &= f_z - \frac{\partial p}{\partial z} + \left(\frac{\partial \tau_{zx}}{\partial x} + \frac{\partial \tau_{zy}}{\partial y} + \frac{\partial \tau_{zz}}{\partial z}\right)\end{aligned} \tag{2.9}$$

The latter one can be represented in vector form as can be seen in the Equation 2.10

$$\rho\frac{D\vec{v}}{Dt} = \vec{f} - \vec{\nabla}(p) + \vec{\nabla}\cdot(\boldsymbol{\tau}), \tag{2.10}$$

where $\boldsymbol{\tau}$ is called the viscous stress tensor, and is composed of the last terms in parentheses on the right side of the equality in Equations 2.9 (Kundu et al., 2015).

2.1.3 Navier-Stokes equations

The Navier-Stokes (NS) equations together with the continuity equation, Equation 2.3, are capable to describe a flow field without heat transfer, and are enough for this work. When heat transfer is involved an extra energy equation is need to be derived.

By inserting a set of transport equations that relate the shear stress τ_{ij} with a proportionality constant multiplied by the velocity vector divergent, which is useful in compressible flows (in imcompressible flow this term is equal to zero), and a linear expansion term, or change in shape of the volume element, which is related to viscosity μ, into the momentum Equations 2.9, and taking into account Stokes' hypothesis, the following equations of motion in Cartesian coordinates according to Equations 2.11. The reader can see the sections 3.4 to 3.8 of the book Boundary-Layer Theory by Schlichting and Gersten, 2003, to get the complete derivation of those equations and hypothesis, and how they are put together through the Reynolds transport theorem.

$$
\begin{aligned}
\rho\frac{Du}{Dt} &= f_x - \frac{\partial p}{\partial x} + \frac{\partial}{\partial x}\left[\mu\left(2\frac{\partial u}{\partial x} - \frac{2}{3}\vec{\nabla}\cdot\vec{v}\right)\right] + \\
&\qquad \frac{\partial}{\partial y}\left[\mu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\right] + \frac{\partial}{\partial z}\left[\mu\left(\frac{\partial w}{\partial x} + \frac{\partial u}{\partial z}\right)\right] \\
\rho\frac{Dv}{Dt} &= f_y - \frac{\partial p}{\partial y} + \frac{\partial}{\partial y}\left[\mu\left(2\frac{\partial v}{\partial y} - \frac{2}{3}\vec{\nabla}\cdot\vec{v}\right)\right] + \\
&\qquad \frac{\partial}{\partial z}\left[\mu\left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\right)\right] + \frac{\partial}{\partial x}\left[\mu\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\right] \\
\rho\frac{Dw}{Dt} &= f_z - \frac{\partial p}{\partial z} + \frac{\partial}{\partial z}\left[\mu\left(2\frac{\partial w}{\partial z} - \frac{2}{3}\vec{\nabla}\cdot\vec{v}\right)\right] + \\
&\qquad \frac{\partial}{\partial x}\left[\mu\left(\frac{\partial w}{\partial x} + \frac{\partial u}{\partial z}\right)\right] + \frac{\partial}{\partial y}\left[\mu\left(\frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}\right)\right]
\end{aligned}
\tag{2.11}
$$

These are the NS differential equations that describe the flow of a Newtonian fluids, allowing to determine the velocity and pressure fields in these flows. Using vector notation, these equations can be given in a form that will work for any coordinate system, as shown by Equation 2.12:

$$
\rho\frac{D\vec{v}}{Dt} = \vec{f} - \vec{\nabla}(p) + \vec{\nabla}\cdot(\boldsymbol{\tau}), \text{ where } \boldsymbol{\tau} = \mu\left(2\dot{\boldsymbol{\varepsilon}} - \frac{2}{3}\boldsymbol{\delta}\vec{\nabla}\cdot\vec{v}\right), \tag{2.12}
$$

and $\boldsymbol{\delta}$ is the Kronecker unit tensor, also know as the Delta of Kronecker ($\delta_{ij} = 1$ for $i = j$ and $\delta_{ij} = 0$ for $i \neq j$) (Schlichting & Gersten, 2003).

2.2 Artificial intelligence

Artificial Intelligence (AI) science is experiencing a fast growth rate in the contemporary world, being present in the human being's daily life through the internet and electronic devices, like cell phone applications accessed daily by hundreds of millions of people.

But what after all is AI? Despite some disagreement among researchers, a well-accepted definition can be found in the article called "What is artificial intelligence?", by McCarthy, 2007, that provide the following definition:

> It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable.

In the 80' Nilsson, 1982, cited some applications of AI, between them natural language processing, intelligent retrieval from databases, expert consulting systems, theorem proving, robotics, automatic programming, combinatorial and scheduling problems and finally perception problems, a list that after 40 years is still incredibly current.

A more contemporaneous list of AI applications could be speech recognition, customer service, computer vision, recommendation engines and automated stock trading (IBM, 2020).

Some sub-fields of AI of special interest in this work are Machine Learning (ML) and Deep Learning (DL), where DL is a sub field of ML, Figure 2.2, and the word deep comes from the fact that this technique uses deep neural networks, that are Neural Network (NN) comprised of more than three layers, as we will see ahead in the next sections.

Until now some definitions and applications of AI are gave, so its time do investigate how it can be done. For that task, a branch of AI will be used, and the focus in this section will be on artificial neural networks, from now on cited as NN, specifically on the multi-layer perceptron NN, also know as Deep Neural Network (DNN) (Rosenblatt, 1957), that is a configuration of NN among many others, as can be seen in the taxonomy of NN architectures in the Figure 2.3.

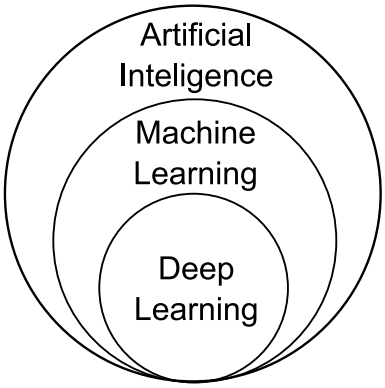


Figure 2.2: Location of Machine Learning within Artificial Intelligence, adapted from IBM, 2020.

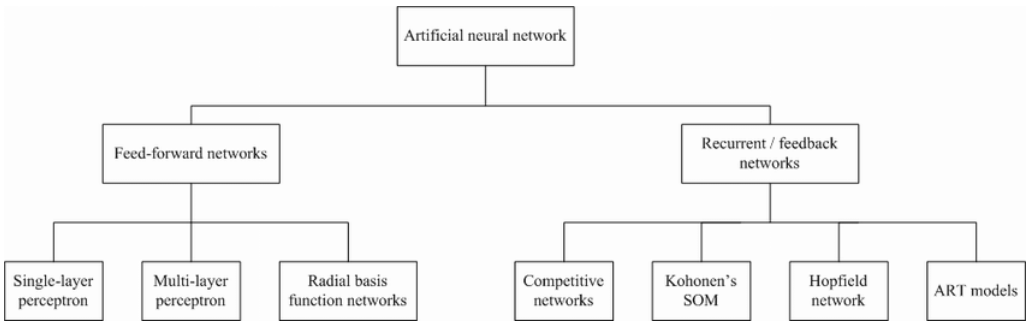


Figure 2.3: A taxonomy of neural network architectures (Mohd Yusof, 2005).

2.2.1 Perceptron neural network

Perceptron NN is a concept introduced by Rosenblatt, 1957, that made use of neuron idea proposed by McCulloch and Pitts, 1943, that described such a artificial nerve cell, or artificial neuron, as a simple logic gate with binary outputs through a Heaviside (step) activation function (Russell & Norvig, 2021). Such artificial neuron consist in a mathematical function, where each neuron have inputs, weighs each of those inputs at a time, sums them up and passes the sum result through a function, called activation function, to produce output. In the Figure 2.4 schematic representation of such a neuron can be seen.

The idea behind an artificial neuron is, given a numerical value of the inputs (X_i) and the weights (W_i), a output will be produced through a function inside the neuron, like a weighted sum, as can be seen in the Equation 2.13, that sum the weight and inputs for a simple case with only three inputs. The values of weights, by general, are initialized randomly.

$$Y = X1W1 + X2W2 + X3W3 \tag{2.13}$$

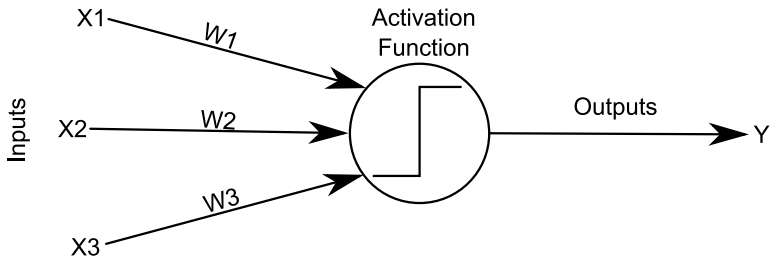


Figure 2.4: Artificial neuron schematics adapted from Bhardwaj, 2020.

As a linear regression, each neuron also has a bias which can be interpreted as an input that always has the value 1 and it too must be weighted, e.g., a neuron that have two inputs requires three weights, one for each input and one for the bias, were the Equation 2.13 summing the weights can be write in sum notation as Equation 2.14 (Brownlee, 2020). The function of the bias is to shift the result of activation function towards the positive or negative side (Malik, 2019).

$$Y = \sum_{i=1}^{n} X_i W_i + W_{bias} \tag{2.14}$$

The weighted inputs are then summed and send through an activation function, like a simple step activation functions used in the original idea of a neuron (McCulloch and Pitts, 1943), where if the summed input was above a threshold valued it would output 1, else outputs 0.

Activation functions are available in many flavors and traditionally non-linear activation functions are most used today. Those functions allows the NN to combine the inputs in a complex way, providing a richer capability in the problems they can model.

The most common non-linear functions is the logistic also called the sigmoid function, Equation 2.15, that is used to output a value between 0 and 1 with an s-shaped distribution, in a similar manner the hyperbolic tangent function, also called tanh, that outputs the same distribution as the sigmoid function but over the range -1 to +1, an that represents and advantage over sigmoid, since the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph (Sharma, 2017; Sutton & Barto, 2018).

$$\sigma(j) = \frac{1}{1 + e^{\sum_i w_{ij} x_i}} \tag{2.15}$$

In the Figure 2.5 can be seen a hyperbolic tangent activation function over a sigmoid activation function graphic. Both functions are differentiable, monotonic while its derivative are not monotonic. The tanh function is mainly used classification between two classes, an the two of them are used in feed-forward nets, as can be seen in the following texts, that explains why they are so popular (Sharma, 2017).

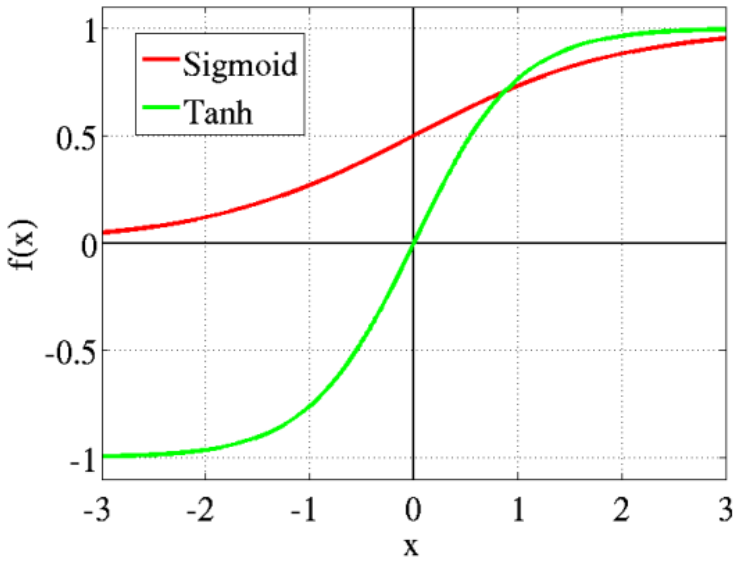


Figure 2.5: Sigmoid vs Hyperbolic Tangent activation function (Sharma, 2017).

Neurons can be associated in many forms and have many layers, those association are know as NN. Maybe the most used association of neuron is a Perceptron NN, that is a feed-forward network in is original approach, that can be classified accordingly to the number of layers, as can be seen in Figure 2.3. A single layer perceptron is formed by a input layer, a hidden layer of neurons, and a output layer, while a multi layer have at least two hidden layer of neurons (Mohd Yusof, 2005).

In the Figure 2.6 a general diagram of a perceptron NN is showed. This perceptron NN is a feed-forward NN, that means that the update of the weights and bias occur from the first hidden layer to the last one by error propagation, unlike a backpropagation NN, where the update occurs backwards, i.e., from the last layer to the first.

As weight and bias are now defined, a method to adjust they values is be needed. The action of adjusting weights and bias is called training of a NN,

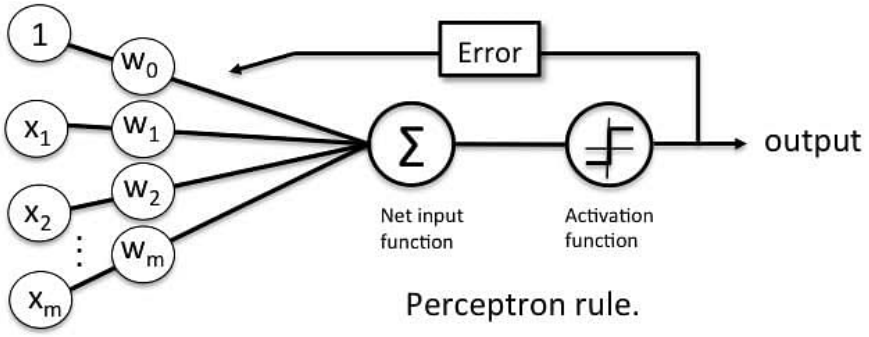


Figure 2.6: General diagram of a perceptron NN for supervised learning (Simplilearn, 2021).

and in a multilayer perceptron is the procedure by which the values for the individual weights and bias are calculated, in a way that the relationship the NN is modelling is accurately resolved.

Maybe the most common technique to training a NN are the methods based on the minimization of a sum-of-square errors function, and is given by a sum over all patterns in the dataset used in the training process, and over all outputs, as can be seen in Equation 2.16 (Bishop et al., 1995):

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{c}\{y_k(\mathbf{X}^n;\mathbf{w}) - t_k^n\}^2\,, \tag{2.16}$$

in this equation $y_k(\mathbf{X}^n;\mathbf{w})$ represents the output of unit k as a function of the inputs vector $\mathbf{X^n}$ and the weights vector $\mathbf{w}$, N is the number of training patterns, c is the number of outputs, and t_k^n represents the target value for the output unit k when the input vector is $\mathbf{X^n}$.

The error function mentioned above is differentiable and can be minimized through many techniques, one of them being the gradient descent method, that is discussed further in this section.

As an introduction to the gradient descent method, consider a simple multilayer perceptron that contains only two weights (two degrees of freedom), so it can be visualized easy on a 3D plot. For any combination of weights the network error for a given pattern can be defined, varying the weights through all possible values, and plotting the errors in three-dimensional space, a error surface plot is obtained, Figure 2.7. The objective of training is to find the combination of weights which result in the smallest error (Bishop et al., 1995; Gardner & Dorling, 1998).

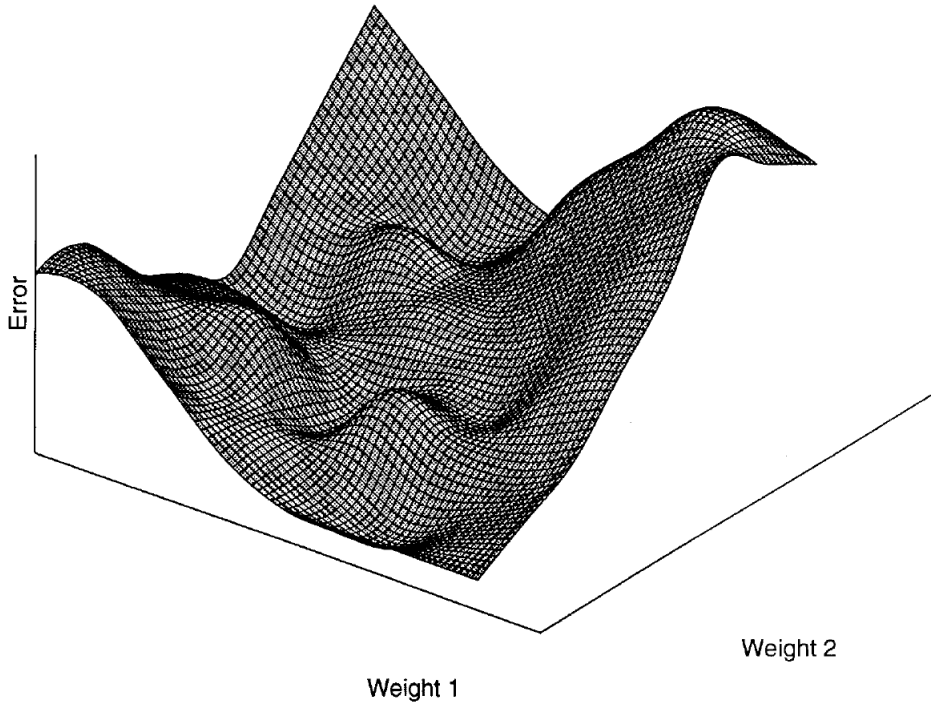


Figure 2.7: Error surface of a two degrees of freedom perceptron (Gardner & Dorling, 1998).

In general, NN can have hundreds or thousands of inputs/degrees of freedom, due to the high number of dimensions, it is not possible to plot a error surface due to the multitude of weights. So, a method to find the minimum point of the error surface (error function) is required.

One way to successfully find the minimum in the error function, as mentioned early, is the gradient descent method.

A gradient simply measures the change in all weights with regard to the change in error, and can be think as the slope of a function, since a gradient is a partial derivative with respect to it's inputs. The higher the gradient, more abrupt the slope will be, and the faster a model can be trained. But if the slope is zero, a maximum or a minimum as been found on the error/loss function, and the model stops learning.

In the gradient descent method, the path opposite to maximum variation is followed, because the gradient operation takes a negative sign ($-\nabla f(x)$), so a minimum of the error/loss function is seeked.

To apply the gradient descent method, error or loos function must be differentiable and an a good practice is to group the weights and biases (both NN parameters) together, forming a single weight vector $\mathbf{w}$ allowing to represent

the error/loss function as $E = E(\mathbf{w})$. As mentioned early, at first the parameters (weights an biases) vector $\mathbf{w}$ is initialized randomly, an then the start to update the parameter vector by moving a small distance in the $\mathbf{w}$ space in the direction in which E decrease most rapidly, that is, in the $-\nabla_{\mathbf{w}} E$ direction. Making this process interactive by Equation 2.17, a sequence of weighted vectors $\mathbf{w}^{(\tau)}$ is generated (Bishop et al., 1995; Russell & Norvig, 2021; Sutton & Barto, 2018),

$$\omega_{kj}^{(\tau+1)} = \omega_{kj}^{(\tau)} + \eta \left(\frac{\partial E}{\partial \omega_{kj}} \right)_{\mathbf{w}^{(\tau)}}, \tag{2.17}$$

the parameter η is know as learning rate and can be interpreted as the steps that gradient descent takes into the direction of the local minimum. η must be chosen properly, in general start whit a small step, 10^{-4} order for example, but a good practice is plot the learning rate an see hoe it evolves.Id the problem is properly set, the parameters vector will converge to a point which E is minimized (Donges, 2021).

There is no guarantee that the minimum found by gradient descent method is a global minimum, it is possible that the minimum found is a local minimum, so the results must be checked when possible.

2.2.2 Deep learning

As can be seen in the Figure 2.2, IBM, 2020, ranks Deep Learning (DL) as a sub-field of Machine Learning (ML), and the last one as a sub-field of Artificial Intelligence (AI) area.

ML enables the creation of algorithms to teach a specific machine to perform a task from a set of data, and from these data, explore the correlation between them, discovering patterns, applying algorithms, and generating models that can be generalized for a specific task (Russell & Norvig, 2021).

When the model is trained, it is able to generalize to new data that were not presented in the training stage, finding correlations and generating predictions to accomplish the specified task. Generally speaking, ML is divided into three main areas (Russell & Norvig, 2021; Sutton & Barto, 2018):

- Supervised Learning: consists of labeled data. The algorithm receives a set of labeled data, that is, data with the corresponding correct outputs, and the algorithm learns by comparing the model's output with the ex-

pected output, readjusting its parameters until reaching an acceptable and predetermined threshold a priori.

- Unsupervised Learning: Consists of unlabeled data. The algorithm receives a set of unlabeled data and seeks to find similarities between groups of data, generating clusters, or groups of data.
- Reinforcement Learning:The agent learns to achieve a goal in an uncertain and potentially complex environment. In reinforcement learning, the artificial intelligence system faces a situation,the algorithm uses trial and error to find a solution to the problem. In order for the machine to do what the programmer wants, artificial intelligence receives rewards or penalties for the actions it performs, were the objective is to maximize the total reward.

Another categorization of ML gave by Brunton et al., 2020, can be seen in the Figure 2.8, were applications are given for the categories. In this classification Reinforcement Learning falls in the Semi-supervised Learning.

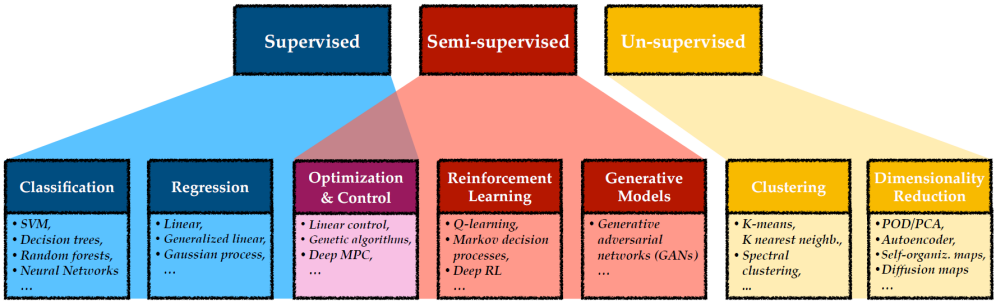


Figure 2.8: Machine learning algorithms categorization (Brunton et al., 2020).

So, DL, a subfield of ML, is focused in a depth analysis of data in larger volume than usual. These algorithms aims to find patterns and correlations in a large amount of information. That means that the datasets can be much larger then the usual, many of then fitting in a classification called BIG Data, the last one is beyond the scope of this work, but some information can be found in Camargo Vega et al., 2015, and Davenport et al., 2012, work.

For this, the foundation of DL are its algorithms that help to identify data, since they are designed to attempt to mime the functioning of the neural networks in the human brain. Thus, such algorithms are able to repeatedly perform analyzes of this data, which generates a greater and deeper learning capacity when compared to traditional ML (Russell & Norvig, 2021).

The main difference between DL and ML, roughly speaking, is the structure of the NN. While, in general, in ML we have a few layers in the NN or a very simple structure, like linear regression or a decision tree, a DL NN can have hundreds or thousands of layers.

Therefore, DL algorithms require much less human intervention. For example shown in the Figure 2.9, the upper half show a ML structure, were a software engineer would manually choose the features and a classifier to classify the images, verify that the output is accurate, and adjust the algorithm if it was not. Wile a DL algorithm, features are automatically extracted and the algorithm learns from its own mistakes (Wauke, 2020; Zhang et al., 2017).

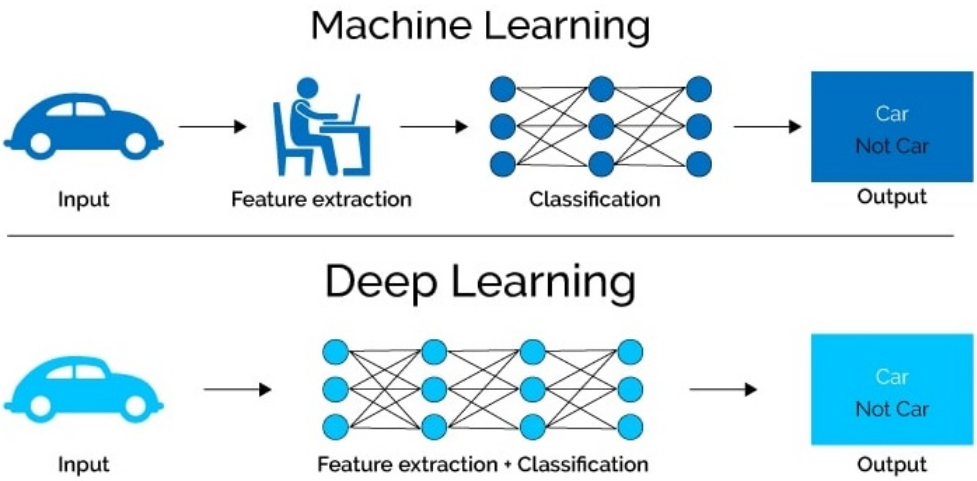


Figure 2.9: Machine Learning vs Deep Learning (Wauke, 2020).

3 Methodology

As a first step toward the goal of infer parameters and flow fields of Partial Differential Equation PDE's in fluid mechanics, a dataset for training the DL algorithm using the method proposed by Raissi et al., 2019, needs to be created.

For that task, as slightly different case will be adopted do obtain the dataset when compared to the reference work of Raissi et al., 2019. In the mentioned work , a two dimension Computational Fluid Dynamics (CFD) was used to simulate a flow over a cylinder till a periodic von Kármán vortex street is obtained. The option for this problem is based in the fact that it is a transient problem that exhibits rich dynamics in steady state behavior, characterized by a asymmetrical vortex shedding pattern, being a classical prototype case in fluid mechanics.

The CFD simulation parameters can be seen in the Table 3.1 for the reference work (Raissi et al., 2019) and the present work. For the present work the parameter are chosen to match Wissink, 1997, work as a manner to validate simulation results.

Table 3.1: Simulation parameters as Reynolds Number (Re), free stream velocity (U_∞), characteristic length (D for cylinder diameter an L for the square edge), kinematic viscosity (ν), discrete domain dimensions in the xy plane and boundary condition (BC).

Case	(Raissi et al., 2019)	Present work
$Re = U_\infty D/\nu$	100	100
U_∞	$1\ m/s$	$1\ m/s$
Characteristic length	$D = 1\ m$	$L = 1\ m$
ν	$0.01\ m^2/s$	$0.01\ m^2/s$
Discrete domain $(x \times y)$	$[-15, 25]m \times [-8, 8]m$	$[-5.5, 15]m \times [-5.7, 5.7]m$
Inlet BC	$Uniform\ x\ vel. = U_\infty$	$Uniform\ x\ vel. = U_\infty$
Outlet BC	*Zero pressure*	*Zero pressure*
Top/bottom wall BC	*Periodic*	*Periodic*
Around cylinder/square BC	*No slip*	*No slip*
Flow type	*Incompressible*	*Incompressible*

In the Figure 3.1 a snapshot of the simulation used by Raissi et al., 2019, can be seen. The figure shows the vorticity, that is a pseudovector field that describes the local spinning motion, being the von Kármán vortex street easily seen in the cylinder downstream.

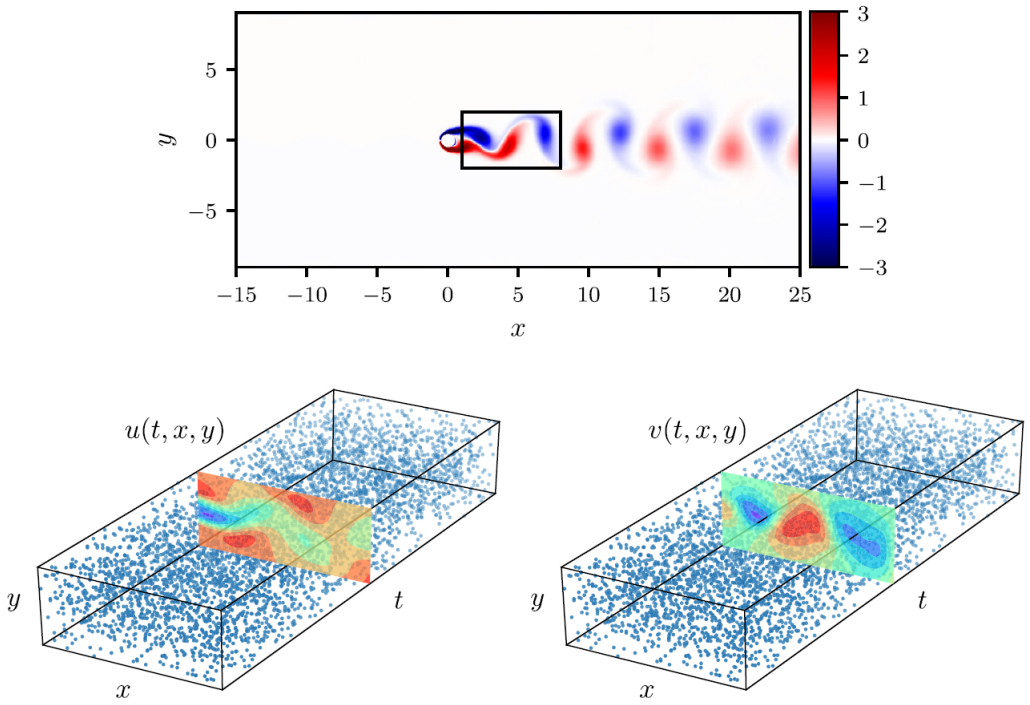


Figure 3.1: Top:Incompressible flow and dynamic vortex shedding at Re =100. The spatio-temporal training data correspond to the depicted rectangular region in the cylinder wake. Bottom:Locations of training data-points for the stream-wise and transverse velocity components, $u(t, x, y)$ and $v(t, x, t)$, respectively (Raissi et al., 2019).

In order to obtain a dataset to answer the question made in the Introduction section, a CFD simulation is conducted with the conditions shown in the right column in the Table 3.1. The coordinate system is set in the geometric center of the square, Figure 3.2, and the x axis runs positive horizontally to the right, while the y axis runs vertically positive upwards.

To discretize the $x[-5.5, 15]m \times y[-5.7, 5.7]m$ domain a quadrilateral mesh composed of 18815 discrete elements is used, where in each of them the Navier-Stoques (NS) equations will be approximated. In the Figure 3.2 the mesh can be seen, where in the outer rectangle the biggest elements nearby the boundaries have a maximum edge size of $0,3\ m$, while the elements inside the green lines on the square wake have a maximum edge size of $0,075\ m$ or $75\ mm$.

The mesh in contact whit the square (first layer around the square) have a first layer width of $2,5\ mm$ and a height of $2\ mm$, and is inflated by a growth rate of $1,1$ over 40 layers. A sphere of influence with $1\ m$ of diameter around the cylinder is set to have maximum element edge size of $30\ mm$.

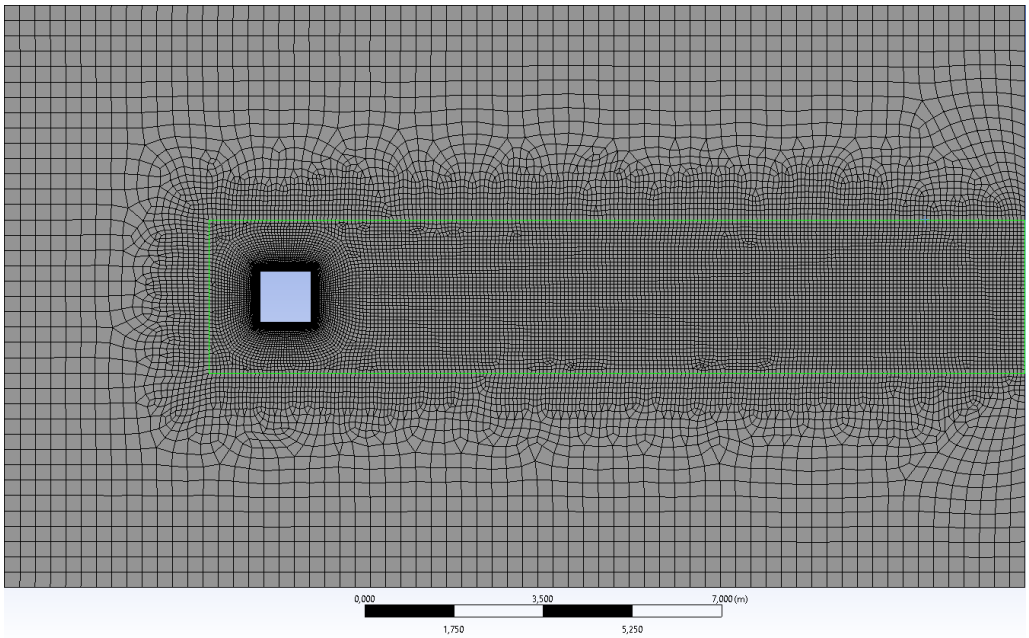


Figure 3.2: Discretized domain for CFD simulation. Vertically, on the left, the square obstacle to the flow is showed. Around it, the mesh is finer, and the same occurs in the region of the domain downstream of the square.

For the simulation ANSYS(R) Fluent(R) 2021 will be used. The flow will be laminar, so no model will be used. Solution methods for Pressure-Velocity coupling will use a SIMPLE scheme, wile spatial discretization will use a Least Squares Cell Based method for gradient calculation, a second order method for the Pressure and a second order upwind scheme for Momentum equation. The transient formulation will be done through a second order implicit scheme. The solution will be approximated till residual reach a order of 10^{-6}.

To predict flow components, a DL algorithm will be used, where the NN for that task will use a fully connected feed-forward perceptron layout.

A Physical-Informed Neural Network (PINN) algorithm will be used, were the physics constraints to respect any symmetries, invariances, or conservation laws will be added by the known NS differential equations directly into the loss function when training the neural network, as can be seen in the Figure 3.3 (Raissi et al., 2019). The PINN will have a feedforward multilayer perceptron layout, with 8 hidden layers, were automatic Differentiation will be used to

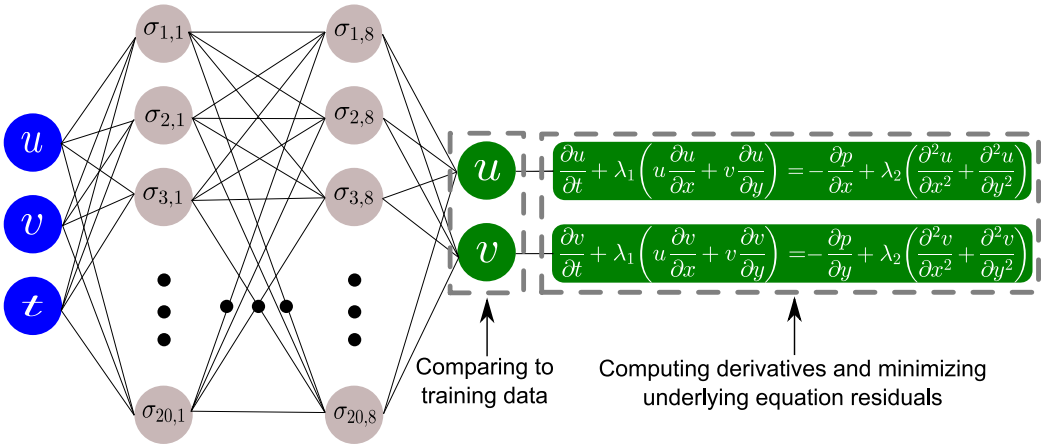


Figure 3.3: Schematic structure of the PINN.

differentiate NN outputs gradients with respect to their input coordinates and model parameters. As a last step, the residual of the underlying differential equation will be computed using these gradients, and add as an extra term in the error/loss function.

Three inputs in the input layers will be used, one for the stream-wise flow component u, another for the span-wise flow component v, and the last one for the time step t. The output layer will consist in two outputs, one for the predicted streamwise flow component $u_{pred.}$, and one for the predicted spanwise component of the flow $v_{pred.}$. Between the input and output layer, eight hidden layers are used, each one whit twenty neurons (Raissi et al., 2019).

For the activation function a Hyperbolic Tangent ($tanh$) function will be used. Weights and biases parameters will be initialized by a Xavier method and set to zero, respectively. To optimize the NN parameters, a Adam optimization algorithm will be used in order to find a minimum in the error/lost function.

After the dataset is obtained by CFD simulation, with a velocity field of N data points: $\{x^i, y^i, t^i, u^i, v^i\}_{i=1}^N$, and the DL framework is set, it will be focused in the NS equations for a 2D incompressible flow with a set of unknown parameters, λ_1 and λ_2, as can be seen in the Equation 3.1.

$$\begin{aligned} \frac{\partial u}{\partial t} + \lambda_1 \left(u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} \right) &= -\frac{\partial p}{\partial x} + \lambda_2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \\ \frac{\partial v}{\partial t} + \lambda_1 \left(u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} \right) &= -\frac{\partial p}{\partial y} + \lambda_2 \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) \end{aligned} \tag{3.1}$$

The question to be answered is: What are the values of λ_1 and λ_2 the best describe the data?

In order to automatically satisfy the continuity Equation 2.3, the latent solution is defined as the stream function $\psi(x, y, t)$, ans can be seen in the Equation 3.3.

$$u = \frac{\partial \psi}{\partial y},\ v = -\frac{\partial \psi}{\partial x} \tag{3.2}$$

Once the stream function is continuous, continuity equation is unconditionally satisfied, and then 2D incompressible NS and continuity equation can be condense into one equation, resulting in the Equation 3.3.

$$\frac{\partial}{\partial t}(\nabla^2\psi) + \frac{\partial \psi}{\partial y}\frac{\partial}{\partial x}(\nabla^2\psi) - \frac{\partial \psi}{\partial x}\frac{\partial}{\partial y}(\nabla^2\psi) = \frac{\partial}{\partial t}(\nabla^2\psi) + \frac{\partial(\psi, \nabla^2\psi)}{\partial(y, x)} \tag{3.3}$$

So, the residuals for each PDE can be defined as Equation 3.4.

$$\begin{aligned} f(x, y, t) &= \frac{\partial u}{\partial t} + \lambda_1 \left(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} \right) = -\frac{\partial p}{\partial x} + \lambda_2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \\ g(x, y, t) &= \frac{\partial v}{\partial t} + \lambda_1 \left(u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} \right) = -\frac{\partial p}{\partial y} + \lambda_2 \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) \end{aligned} \tag{3.4}$$

As a next step, the parameters $\lambda_{1,2}$ of the Navier–Stokes operator as well as the parameters of the neural network $[\psi(t, x, y)\ \ p(t, x, y)]$ and $[f(t, x, y)\ g(t, x, y)]$ will be approximated by the mean square error method, as can be seen in the Equation 3.5.

$$\begin{aligned} E = \frac{1}{N}\sum_{i=1}^{N}\Big(|u(x^i, y^i, t^i) - u^i|^2 + |v(x^i, y^i, t^i) - v^i|^2 \Big) \\ + \frac{1}{N}\sum_{i=1}^{N}\Big(|f(x^i, y^i, t^i)|^2 + |g(x^i, y^i, t^i)|^2 \Big) \end{aligned} \tag{3.5}$$

With the dataset and DL framework set, for scattered and scarce training data (N), say a few percent of the total learning data available on the streamwise $u(t, x, y)$, and transverse $v(t, x, y)$ velocity components, the objective will be find the unknown parameters λ_1 and λ_2 in the Equation 3.1, to test the ability to predict the flow field in u and v directions, as well as the reconstructions of the pressure field in the square wake without providing any pressure

training data.

The network will be trained using the Adam stochastic gradient descent algorithm (Kingma & Ba, 2014) whit a initial learning rate $\eta = 1 \times 10^{-3}$. And then the error will be minimized by a Limited-memory Broyden–Fletcher–Goldfarb–Shanno B (L-BFGS-B) method (Zhu et al., 1997), where a random set of continuous points over the full input space are sampled to compute the physics loss according to the Equation 3.5.

4 Results

In order to obtain a date set to train Physical-Informed Neural Network (PINN), a Computational Fluid Dynamics (CFD) simulation was conducted following the Methodology described in the previous section. For that 1000 iterations whit a time step 0.1 was conducted, resulting ins 100 s of simulation.

From the 1000 iterations, the last 200 was taken to create the dataset after certificate that a steady-state and periodic von Kármán vortex street was obtained, as can be seen in the Figure 4.1, were the vorticity field for the last time step (1000 that corresponds to 100 s of simulation) can be seen.

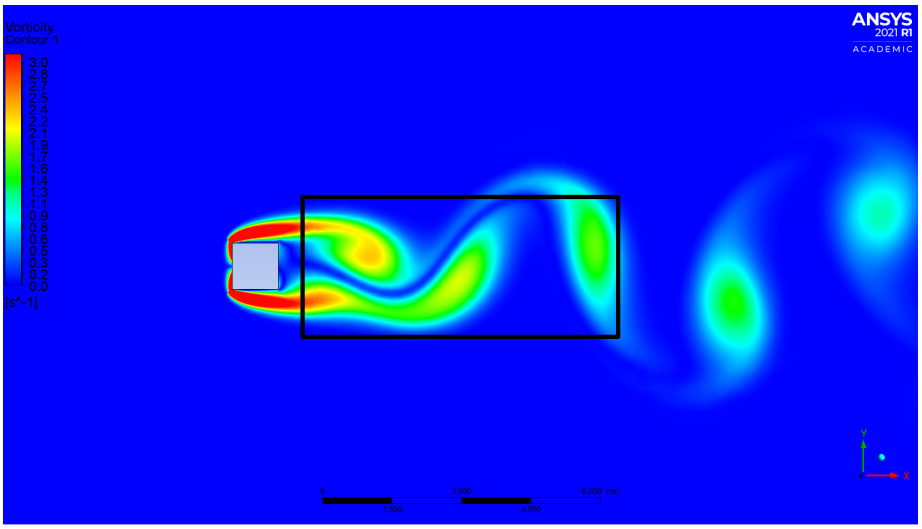


Figure 4.1: Vorticity field demonstrating the von Kármán vortex street for the last time step of simulation. The spatio-temporal dataset was obtained from data in the depicted rectangular region in the square wake

From the black rectangle in the square wake in the Figure 4.1, the dataset was obtained. So, the velocity field took a tensor form whit two columns, fist for u velocity and second for v velocity, and 4799 lines representing the scattered velocity data points for a time-step simulation. The third dimension can be thought of as the tensor depth, representing the velocity field for the

4799 points in each of the 200 time steps.

For the first use of the Methodology described, following Raissi et al., 2019, work, we have chosen $N = 4799$, corresponding to 1% of the total available training data seen in the Figure 4.2. To train the PINN model, 200000 iterations on the dataset has been set.

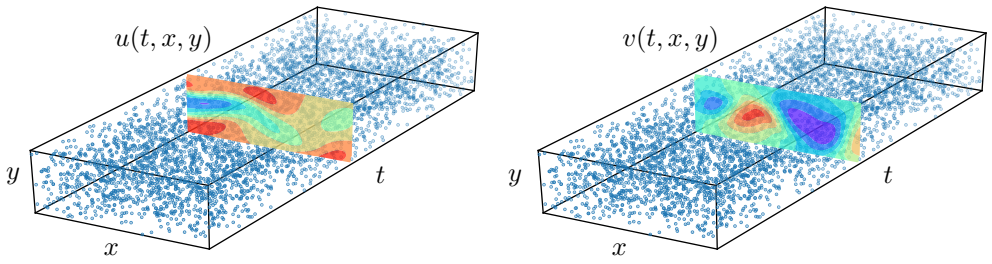
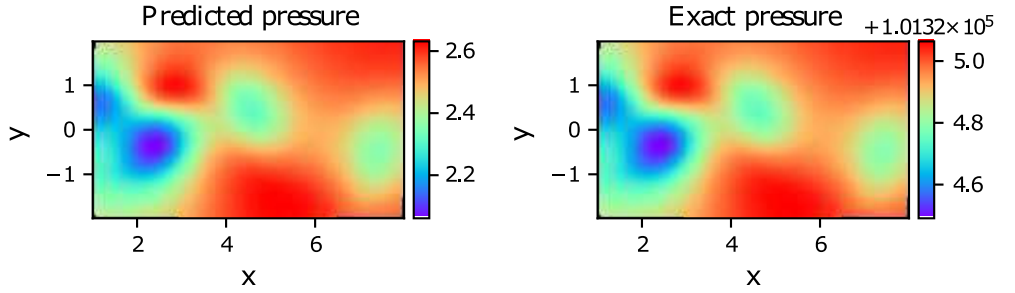


Figure 4.2: Spatio-temporal training data obtained by CFD simulation corresponding to the depicted rectangular region in the square wake. Contours: in the left predicted x velocity field (u), in the right predicted y velocity field (v) by PINN.

The result was obtained after 12 hours of parallel processing on 1664 CUDA(R) cores. The error in the predicted velocity field was 2.018933×10^{-3} and 5.382336×10^{-3} for u and v velocity field respectively.

Also, the algorithm was able to predict the entire pressure field ($p(x, y, t)$) without any pressure training data due constraints imposed by continuity equation, as can be seen in the Figure 4.3.



Correct PDE	$u_t + (uu_x + vu_y) = -p_x + 0.01(u_{xx} + u_{yy})$ $v_t + (uv_x + vv_y) = -p_y + 0.01(v_{xx} + v_{yy})$
Identified PDE (clean data)	$u_t + 0.994(uu_x + vu_y) = -p_x + 0.01058(u_{xx} + u_{yy})$ $v_t + 0.994(uv_x + vv_y) = -p_y + 0.01058(v_{xx} + v_{yy})$
Identified PDE (1% noise)	$u_t + 0.994(uu_x + vu_y) = -p_x + 0.01073(u_{xx} + u_{yy})$ $v_t + 0.994(uv_x + vv_y) = -p_y + 0.01073(v_{xx} + v_{yy})$

Figure 4.3: TOP: at left predicted pressure field, as right exact pressure field. BOTTOM: Table 1st row exact values for λ, 2nd row λ values for clean training data and 3th row values of λ for 1% noise inserted in the data.

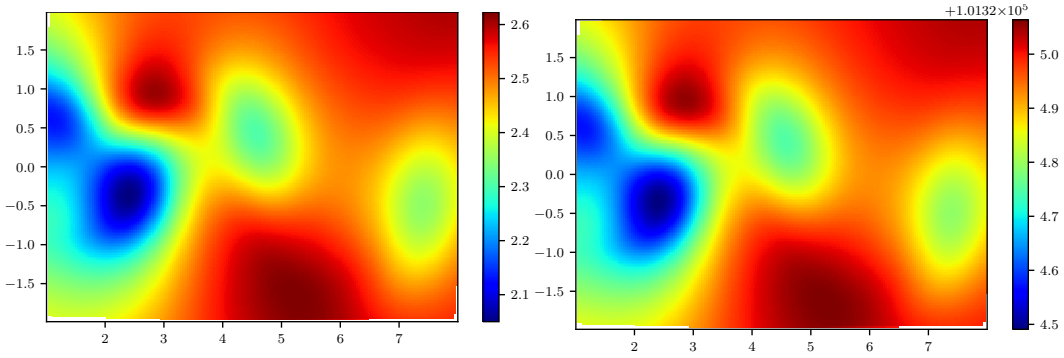
The pressure utilized in our simulation, from where the dataset was produced, is in terms of absolute pressure, so the difference in magnitude from the predicted pressure field an the exact pressure fild obtained by CFD simulation, top of the Figure 4.3, is $2.4 + P_{atm} \approx 2.4 + 1.0132 \times 10^5\ Pa$ between the exact and the predicted pressure. The pressure field can be retrieved only by a constant from the velocity field data by definition, and it is so by the very nature of the incompressible NS equations.

As highlighted by Raissi et al., 2019, this result of inferring a continuous quantity of interest from auxiliary measurements, by leveraging the underlying physics is a great example of the enhanced capabilities that PINN can offer, and exposes their potential in solving high-dimensional inverse problems.

Still in the Figure 4.3, it is possible to see the values found for λ_1 and λ_2 in the center line of the table, were the error found was 0.62262% and 5.75713%, respectively. To be clear, in the way the Equations 3.1 was written, the exact values for λ_1 and λ_2 must be 1 and $0.001 = \nu$, respectively. λ_2 is equal to the kinematic viscosity ν, as can be seen in the Equations 2.11, the difference is that here $\nu = \rho/\mu$ is used, so the exact values to be expected for the λs are know. Has Raissi et al., 2019, did in their work, to test the performance of the PINN algorithm, a 1% white noise was inserted into the data, and another run of the algorithm was take. The result for λ_1 was 0.994, practically no change till the third decimal when compared with clean data λ_1, while the λ_2 for noisy data was 7.34432%, about 27% higher.

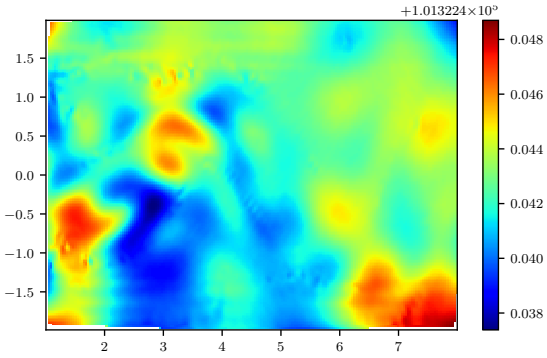
Another analysis that can be done is subtract the predicted pressure field from the simulated pressure field, in order to highlight the regions where the error is greater, for example. For that we subtracted the predicted pressure, Figure 4.4 (a), from simulate pressure Figure 4.4 (b), $P_{simulated} - P_{predicted}$ as can be seen in the Figure 4.4 (c). The error ranges between 0.038 an 0.048, values two orders of magnitude less than predicted value. Being a very small error given the circumstances, where the regions that presented the greatest error are coincident with the vortex formation regions downstream of the square.

To analyze the evolution values of the parameters λ for the clean training data of the PINN, the Figure 4.5 was made, as Figure 4.5a shows the evolution of λ_1, whereas Figure 4.5b shows λ_2. It can be seen that that λ_1 converged to 0.992 and λ_2 to 0.01118. It is important to clarify that up to this point the restrictions imposed by the NS equations have not yet been used to minimize the error, and the values obtained are purely optimizing the input data with



(a) Predicted Pressure field.

(b) Simulated Pressure field.



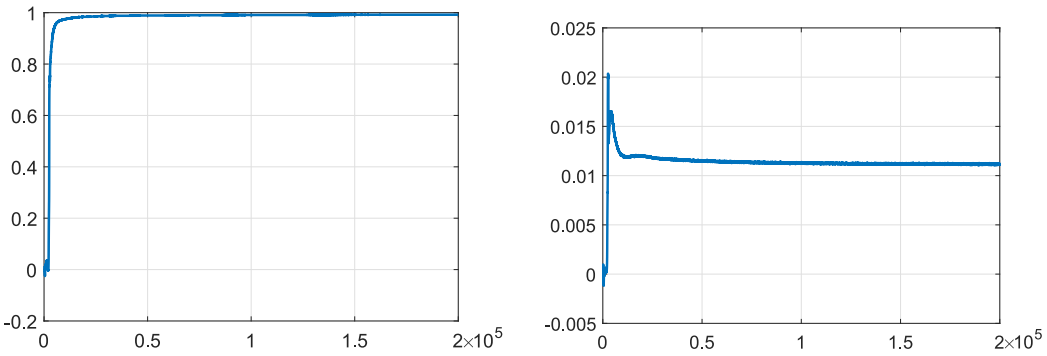
(c) Predicted Pressure subtracted from Simulated Pressure.

Figure 4.4: Red regions represent the places where the error between Predicted and Simulated Pressure is greater (c). Figure (b) corresponds of the Simulated pressure subtracted from the Atmospheric Pressure.

the output data, since the model will be submitted to a new error minimization process, but this time taking into account the restrictions imposed by physical laws (last term in the right of the NS Equations 3.1). Another detail is that to generate this figure the first 20 iterations were discarded, as in some of them the approximate values for λ were too high or too low (3 to 6 orders of magnitude greater or smaller than the converged values) in order to produce a graph where it was possible to see the learning evolution.

In the Figure 4.6 it can be seen the loss/error evolution thought the 200000 iterations, reaching 0.6685 for the clean data, and 1.023 for the noisy data, both in the last iteration. For the same motive as in the Figure 4.5, the first 20 iterations were discarded because of high loss values, an so enable the visualization of the loss evolution over iteration.

Now, the result of the final values of λ, Figure 4.7, and loss, Figure 4.8, that was obtained by the mean square error taking into account the underlying



(a) λ_1 evolution. **(b)** λ_2 evolution.

Figure 4.5: $\lambda_{1,2}$ convergence after 200000 iteration for clean dataset.

physics laws (final optimization process), shown in the Equation 3.5, can be seen. The stop criterion is 50000 iterations or when the machine epsilon (a float precision) is achieved. Where the clean data and noisy data took around 37000 iterations to achieve machine epsilon. It can be seen that λ_1, Figure 4.7a, was already optimized, that explains the weird behavior in the graph. The λ_2 got a bit more optimized, as can be seen in the Figure 4.7b.

For evaluation of the final training process, a plot of the loss function against the iteration is shown in the Figure 4.8. It can be seen a much more smooth convergence, that is explained because the early training process that the PINN got through.

Even whit noisy data, the PINN was able to identify λ_1 and λ_2 with good precision, taking in the account the scarce and sparse data provided for the training process, a fact that shows great potential in real world applications.

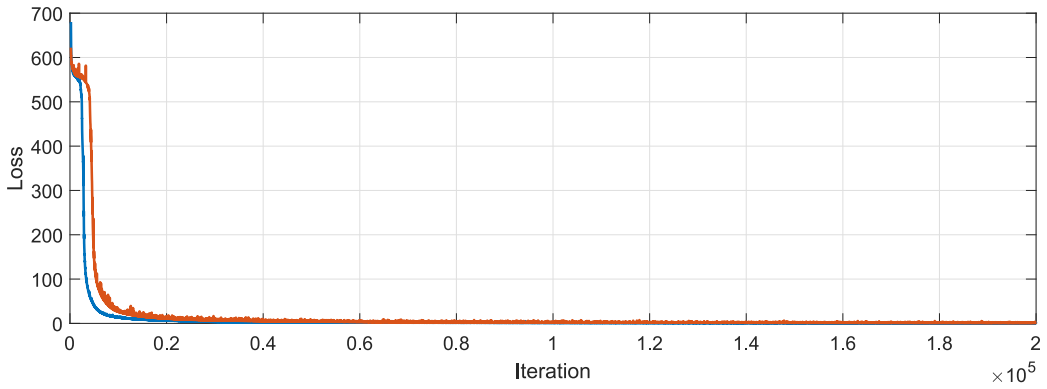
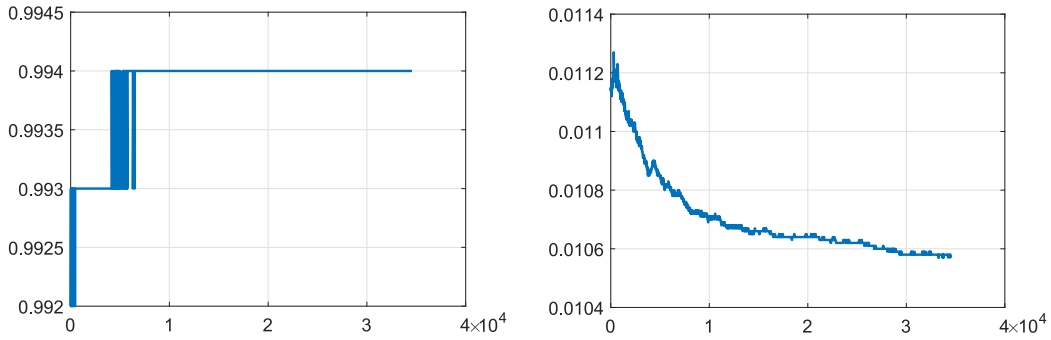


Figure 4.6: Training error (loss) by iteration first loss minimization, blues curve for clean training data, red for 1% noisy training data.



(a) λ_1 evolution for final minimization. **(b)** λ_2 evolution for final minimization.

Figure 4.7: $\lambda_{1,2}$ convergence after around 37000 iteration in the final loss minimization for clean data.

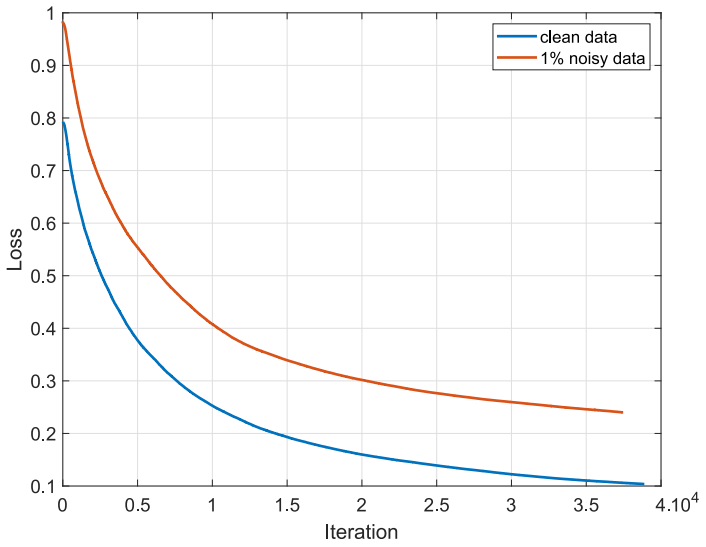


Figure 4.8: Mean square error (loss) by iteration, blues curve for clean training data, red for 1% noisy training data.

5 Conclusions

As closing remarks of this work a few conclusions can be made. First of all, the general objective was reached, where a dataset was created from a CFD simulation, and the PINN was successfully applied from these data, obtaining the coefficients λ fitted to the simulation dataset with great precision, even whit scarce, sparse and noisy data.

The specific objectives of obtain the entire pressure field, without any training pressure data provided, was achieved in its entirety, as well as the prediction of velocity fields in the directions of u and v. This result of inferring a continuous quantity of interest from auxiliary measurements, by leveraging the underlying physics, demonstrates the enhanced capabilities that PINN can offer.

It was proven, that with scarce and sparse data, or even noisy data, excellent results can be obtained, with errors around 5%, in high-dimensionality non linear problems like fluid dynamics presented here.

Another application of data-driven discovery problems is to help to derive more accurate closure turbulence models or develop more efficient surrogate models for specialized applications or optimization studies. PINN can contribute by increasing the quality and speed of obtaining results from problems involving non-linear PDE, and can replace traditional methods like Finite Elements Method (FEM) and CFD in some situations, but not in all.

Despite the good results obtained, there is much room for improvement, like the loss function used in this work and used by Raissi et al., 2019. Starting by how the loss is calculated, it would give a more accurate result just by introducing a L1 or L2 norm regularization, for example. This is an easy and direct improvement, requiring only a few lines of code.

As future work, it is intended to continue deepening the knowledge in data-driven problems, seeking implementation in real and theoretical problems, applying the methodology described here in more challenging situations, such

as turbulent flows in three dimensions, or heat and mass transfer.

Bibliography

Almajid, M. M., & Abu-Al-Saud, M. O. (2022). Prediction of porous media fluid flow using physics informed neural networks. *Journal of Petroleum Science and Engineering*, *208*, 109205.

Balaji, C., Srinivasan, B., & Gedupudi, S. (2021). Chapter 6 - natural convection. In C. Balaji, B. Srinivasan, and S. Gedupudi (Eds.), *Heat transfer engineering* (pp. 173–198). Academic Press. https://doi.org/https://doi.org/10.1016/B978-0-12-818503-2.00006-X

Bhardwaj, A. (2020). What is perceptron: A beginners guide for perceptron [Accessed: 13 dec. 2021]. https://towardsdatascience.com/what-is-a-perceptron-basics-of-neural-networks-c4cfea20c590

Bishop, C. M. et al. (1995). *Neural networks for pattern recognition*. Oxford university press.

Brownlee, J. (2020). Crash course on multi-layer perceptron neural networks [Accessed: 13 dec. 2021]. https://machinelearningmastery.com/neural-networks-crash-course/

Brunton, S. L., Noack, B. R., & Koumoutsakos, P. (2020). Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, *52*, 477–508.

Camargo Vega, J. J., Camargo Ortega, J. F., & Joyanes Aguilar, L. (2015). Conociendo big data. *Facultad de Ingeniería*, *24*(38), 63–77.

Chicone, C. (2017). Chapter 12 - flow in a pipe. In C. Chicone (Ed.), *An invitation to applied mathematics* (pp. 321–326). Academic Press. https://doi.org/https://doi.org/10.1016/B978-0-12-804153-6.50012-9

Crevier, D. (1993). *Ai: The tumultuous history of the search for artificial intelligence*. Basic Books, Inc.

Davenport, T. H., Barth, P., & Bean, R. (2012). How "big data" is different.

Dissanayake, M., & Phan-Thien, N. (1994). Neural-network-based approximations for solving partial differential equations. *communications in Numerical Methods in Engineering*, *10*(3), 195–201.

Donges, N. (2021). Neural networks bias and weights [Accessed: 14 dec. 2021]. https://builtin.com/data-science/gradient-descent

Gardner, M. W., & Dorling, S. (1998). Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment*, *32*(14-15), 2627–2636.

Germano, M., Piomelli, U., Moin, P., & Cabot, W. H. (1991). A dynamic subgrid-scale eddy viscosity model. *Physics of Fluids A: Fluid Dynamics*, *3*(7), 1760–1765.

IBM, C.-E. (2020). Artificial intelligence (ai) [Accessed: 12 dec. 2021]. https://www.ibm.com/cloud/learn/what-is-artificial-intelligence

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kundu, P., Cohen, I., & Dowling, D. (2015). *Fluid mechanics*. Elsevier Science. https://books.google.com.br/books?id=EehDBAAAQBAJ

Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, *9*(5), 987–1000.

Lim, M. (2019). History of ai winters [Accessed: 10 dec. 2021]. https://www.actuaries.digital/2018/09/05/history-of-ai-winters/

Malik, F. (2019). Neural networks bias and weights [Accessed: 13 dec. 2021]. https://medium.com/fintechexplained/neural-networks-bias-and-weights-10b53e6285da

Mao, Z., Jagtap, A. D., & Karniadakis, G. E. (2020). Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, *360*, 112789.

McCarthy, J. (2007). What is artificial intelligence?

McCorduck, P., & Cfe, C. (2004). *Machines who think: A personal inquiry into the history and prospects of artificial intelligence*. CRC Press.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*(4), 115–133.

Meneveau, C., & Katz, J. (2000). Scale-invariance and turbulence models for large-eddy simulation. *Annual Review of Fluid Mechanics*, *32*(1), 1–32.

Mohd Yusof, N. (2005). *Time series modeling and designing of artificial neural network (ann) for revenue forecasting* (Ms.C. dissertation). Universiti Teknologi Malaysia. Malaca, Malaysia.

Nilsson, N. J. (1982). *Principles of artificial intelligence*. Springer Science & Business Media.

Phillips, L. (2018). Turbulence, the oldest unsolved problem in physics [Accessed: 08 dec. 2021]. https://arstechnica.com/science/2018/10/turbulence-the-oldest-unsolved-problem-in-physics/?comments=1

Pope, S. B. (2000). *Turbulent flows*. Cambridge University Press.

Raissi, M., & Karniadakis, G. E. (2018). Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, *357*, 125–141.

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, *378*, 686–707.

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2018). Numerical gaussian processes for time-dependent and nonlinear partial differential equations. *SIAM Journal on Scientific Computing*, *40*(1), A172–A198.

Raissi, M., Yazdani, A., & Karniadakis, G. E. (2020). Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, *367*(6481), 1026–1030.

Ramabathiran, A. A., & Ramachandran, P. (2021). Spinn: Sparse, physics-based, and partially interpretable neural networks for pdes. *Journal of Computational Physics*, *445*, 110600.

Ray, A. (2021). Ai winter: The highs and lows of artificial intelligence [Accessed: 10 dec. 2021]. https://www.historyofdatascience.com/ai-winter-the-highs-and-lows-of-artificial-intelligence/

Rechenberg, I. (1964). Kybernetische lösungsansteuerung einer experimentellen forschungsaufgabe. *Ann. Conf. WGLR Berlin, vol. 35*.

Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton project para*. Cornell Aeronautical Laboratory.

Russell, S., & Norvig, P. (2021). *Artificial intelligence: A modern approach, global edition 4th*. Pearson Series.

Schlichting, H., & Gersten, K. (2003). *Boundary-layer theory*. Springer Science & Business Media.

Schuchmann, S. (2019). History of the first ai winter [Accessed: 10 dec. 2021]. https://towardsdatascience.com/history-of-the-first-ai-winter-6f8c2186f80b

Sharma, S. (2017). Activation functions in neural networks [Accessed: 13 dec. 2021]. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Simplilearn. (2021). What is perceptron: A beginners guide for perceptron [Accessed: 12 dec. 2021]. https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron

Smagorinsky, J. (1963). General circulation experiments with the primitive equations: I. the basic experiment. *Monthly weather review*, *91*(3), 99–164.

Sun, L., Gao, H., Pan, S., & Wang, J.-X. (2020). Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Computer Methods in Applied Mechanics and Engineering*, *361*, 112732.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Wauke, J. (2020). Aprendizado profundo (deep learning) vs aprendizado de máquina (machine learning) - qual é a diferença? [Accessed: 15 dec. 2021]. https://jobu.com.br/2020/10/24/aprendizado-profundo-deep-learning-vs-aprendizado-de-maquina-machine-learning-qual-e-a-diferenca/

White, F. M. (2021). *Fluid mechanics* (9th ed.). McGaw-Hill.

Wiener, N. (2019). *Cybernetics or control and communication in the animal and the machine*. MIT press.

Wissink, J. G. (1997). Dns of 2d turbulent flow around a square cylinder. *International journal for numerical methods in fluids*, *25*(1), 51–62.

Young, D. F., Munson, B. R., Okiishi, T. H., & Huebsch, W. W. (2010). *A brief introduction to fluid mechanics*. John Wiley & Sons.

Yu, B. et al. (2017). The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. *arXiv preprint arXiv:1710.00211*.

Zhang, L., Tan, J., Han, D., & Zhu, H. (2017). From machine learning to deep learning: Progress in machine intelligence for rational drug discovery. *Drug discovery today*, *22*(11), 1680–1685.

Zhu, C., Byrd, R. H., Lu, P., & Nocedal, J. (1997). Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on mathematical software (TOMS)*, *23*(4), 550–560.