



A state-of-the-art of physics-informed neural networks in engineering

Pedro Henrique da Silva Singue Cerqueira

Advisors: D.Sc Fábio Pereira dos Santos and
Eng. Lucas Henrique Queiroz dos Reis

Rio de Janeiro

August 2021

A state-of-the-art of physics-informed neural networks in engineering

Pedro Henrique da Silva Singue Cerqueira

A bachelor thesis submitted to the Chemical Engineering department of the School of Chemistry at the Federal University of Rio de Janeiro in fulfillment of the requirements for the degree of Chemical Engineer

Author:

Pedro Henrique da Silva Singue Cerqueira

Advisor professor I:

D.Sc Fábio Pereira dos Santos

Advisor II:

Eng. Lucas Henrique Queiroz dos Reis

Thesis Defence Committee:

Prof., D.Sc Heloísa Lajas Sanches Fernandes

Thesis Defence Committee:

Eng., M.Sc Victor Corcino de Albuquerque

Rio de Janeiro

August 2021

Cerqueira, Pedro Henrique da Silva Singue.

A state-of-the-art of physics-informed neural networks in engineering / Pedro Henrique da Silva Singue Cerqueira. Rio de Janeiro: UFRJ/EQ, 2021.

xv, 26 p.; il.

(Monografia) - Universidade Federal do Rio de Janeiro, Escola de Química, 2021.

Orientadores: Fábio Pereira dos Santos e Lucas Henrique Queiroz dos Reis.

1. Machine learning. 2. Neural networks. 3. Physics-informed neural networks.
4. Monografia (Graduação UFRJ/EQ). 5. Fábio Pereira dos Santos e Lucas Henrique Queiroz dos Reis. I. A state-of-the-art of Physics-informed neural networks in engineering.

To my father (in memoriam)

“It is our choices, Harry, that show what we truly are, far more than our abilities.”

- Albus Dumbledore

ACKNOWLEDGEMENTS

A Deus e todas as boas energias do Universo. À minha família, que são como amigos: Carla, Ivonete, Renata, Ivana, Valéria, Juarez, Renato, Celso e primos. Ao meu pai que nunca deixou de estar presente me vigiando de outro plano.

Aos meu orientadores, Fábio Pereira dos Santos e Lucas Henrique Queiroz dos Reis, por todo apoio, conselho e direcionamento. Aos membros da banca, Heloísa Lajas Sanches Fernandes e Victor Corcino de Albuquerque pela leitura e pelas considerações.

Aos meus amigos, que são como uma família: Stephanie, Victor, Aluan, Morgana, Paula, Gabriel e Caroline. Flávio, meu irmão, um agradecimento especial para você, amigão.

À tous les amis que j'ai connus en France et qui restent dans mon coeur même au Brésil. Mes professeurs Centraliens: Valérie Hamel, Cécile Loubet, Pierrette Guichardon, Pascal Denis, Christophe Pouet, Fabien Anselmet et Françoise Duprat ; Saulo, Mateus, Raffael, Bortolini, Nobrega et Julia, ma marraine, un gros merci à vous. Merci également à M. Lardeux et F-X Pasquet, mes tuteurs chez Total, et Adriano Motta. À mon parrain M. Pardigon. Je tiens à remercier aussi mon ami Nathan, celui qui m'a présenté les reseaux des neurones.

Aos meus colegas de estágio, Amanda, Murilo, Marina, Larissa, Maria Eduarda, Marco e Maurício. Ao Rafael Aislan e à Gaziela Cerveira pelas primeiras orientações ainda na IC.

A todos os professores que contribuíram ativamente para minha formação na UFRJ. Agradecimento também à Tânia, Virginia, Mônica, Ray, Karla e tantos outros da EPSJV.

**Resumo da Monografia apresentada à Escola de Química como parte
dos requisitos necessários para obtenção do grau de Bacharel em
Engenharia Química**

A state-of-the-art of Physics-informed neural networks in engineering

Pedro Henrique da Silva Singue Cerqueira

August, 2021

Orientadores: Prof. Fábio Pereira dos Santos, D.Sc

Eng. Lucas Henrique Queiroz dos Reis

Técnicas de machine learning vêm ganhando cada vez mais espaço no cenário industrial no intuito de converter o crescente fluxo de informação (data) em melhorias de processos. Entre tais técnicas, as redes neurais se destacam devido à sua capacidade de aproximador universal de funções, cuja performance pode ser enriquecida ao se fornecer conhecimentos físicos prévios: tem-se, então, o desenvolvimento das Physics-informed neural networks (PINN). Nesse contexto e observando-se um “gap” na produção de trabalhos relacionados ao tema e da difusão dessa temática na grade de formação dos cursos da Escola de Química, essa trabalho se propõe a realizar um estado da arte da técnica mencionada. Observou-se interesse particular das PINN para aplicações em mecânica dos fluidos e transferência de calor. Ademais, as PINN se mostram ferramentas importantes tanto para a resolução de problemas ditos “diretos” quanto “indiretos”. Por fim, através de exemplos práticos, constatou-se a capacidade de se aproximar funções de interesse particular na indústria química usando-se redes neurais sem nenhuma informação física do problema (obtenção do fator de atrito) e utilizando-se a equação diferencial que descreve o problema (resolução da equação de difusão em 1D).

Palavras-chave: 1. Machine learning. 2. Neural networks. 3. Physics-informed neural networks.

ABSTRACT

A state-of-the-art of Physics-informed neural networks in engineering

Pedro Henrique da Silva Singue Cerqueira

Agosto, 2021

Supervisors: Prof. Fábio Pereira dos Santos, D.Sc

Eng. Lucas Henrique Queiroz dos Reis

Machine learning techniques have gained space in the industrial scenario as a tool to convert the increasing flux of information (data) in process improvement. Among these techniques, neural networks has got much attention due to their universal approximators capacity, of which performance can be improved by providing previous physical knowledge: one has, therefore, the development of the so called Physics-informed neural networks (PINN). In such context and having noticed a “gap” in the works related on this topics and in the diffusion of this theme in the School of Chemistry, this work proposes a state-of-the-art of the mentioned technique. Particular interesting concerning PINN in fluid mechanics and heat transfer has been noticed. Moreover, PINN have been pointed as important tools for solving forward and inverse problems. Finally, through practical examples, this work has shown the use of neural networks for solving one particular example in chemical engineering without informing the physics of the problem (obtaining the friction factor) and using the differential equation that describes it (solving the 1D heat diffusion equation).

Key-words: 1. Machine learning. 2. Neural networks. 3. Physics-informed neural networks.

Contents

List of Figures	xii
List of Tables	xvi
1 Introduction	1
1.1 Motivation	1
1.2 Objectives	2
1.2.1 Specific objectives	2
1.3 Methodology	3
2 Machine learning and examples in chemical engineering	4
2.1 Machine learning	4
2.2 Supervised learning	6
2.2.1 Neural networks	6
2.2.2 Support vector machines and random forests	7
2.3 Unsupervised learning	11
2.3.1 Proper orthogonal decomposition and autoencoders	11
2.3.2 Clustering	15
2.4 Semisupervised learning	17
2.4.1 Generative adversarial networks	17
2.4.2 Reinforcement learning	19
2.5 Conclusion of the chapter	22
3 Neural networks	23
3.1 Nodes, activation functions and networks	25
3.2 Some considerations on NN optimization	28

3.3	Deep learning	29
3.3.1	Feedforward networks: some examples	29
3.3.2	Convolutional networks	31
3.3.3	Recurrent networks	34
3.4	Practical example: Colebrook-White equation	35
4	Physics-informed neural networks	39
4.1	Fundamentals of PINN: integrating physics to the model	40
4.1.1	Loss function guided PINN	40
4.1.2	Loss function guided PINN - solving PDE	41
4.1.3	Guided initialization PINN	49
4.2	Additional thoughts	49
4.2.1	Data generation	50
5	Practical application of PINN	51
5.1	Addressed problem: 1D diffusion equation	51
5.2	Solving the 1D heat equation with NN - forward problem	52
5.3	Solving the 1D heat equation with NN - inverse problem	54
6	Conclusion	57
6.1	Conclusion	57
6.2	Further researches and works	58
6.3	Suggestions	58
A	Modeling a NN to predict friction factor	60
B	Solving the 1D heat equation using PINN in Python	63
	Bibliography	70

Nomenclature

ϵ Pipe roughness

CNN Convolutional neural network

D Diffusion coefficient

d Pipe internal diameter

f Friction factor

GAN Generative adversarial network

ML Machine learning

MMP Minimal miscible pressure

NN Neural network

PDE Partial differential equation

PINN Physics-informed neural network

POD Proper orthogonal decomposition

Re Reynolds number

RL Reinforcement learning

RNN Recurrent neural network

SVM Support vector machines

List of Figures

2.1	Graph representation of a neural network that performs a $\mathbb{R}^8 \rightarrow \mathbb{R}^4$ transformation with 3 hidden layers (left) and the mathematical operation that the inputs undergoes at each neuron (right). Source: adapted from Medium.com and Towardsdatascience.com websites.	7
2.2	Several possible hyperplanes (left) and the one that maximizes the margins (right). Source: adapted from [7]	8
2.3	Analogy on how do decision trees work. Source: [5].	8
2.4	Decision boundary of tree with depth one (upper) and boundaries for depth two (bottom) in a 2D dataset. Source: adapted from [5]	9
2.5	POD (left), autoencoder (middle) and deep autoencoder (right) structures [3].	11
2.6	Process case study (left) and autoencoder structure (right). Source: adapted from [17]	13
2.7	(a) Original velocity field, (b) reconstructed field from the 3-dimension latent variables and (c) reconstruction error. Source: adapted from [18].	13
2.8	Stacked autoenconders for order reduction prior to model training. Source: adapted from [19].	14
2.9	Representation of the k-means clustering algorithm for a 2D dataset and 3 clusters. Source: [5].	15
2.10	Risk zoning map of the Nanjing Industrial Park proposed by Shi and Zeng. Source: [22].	16
2.11	Representations of the generator and discriminator in a GAN. Source: [27].	17

2.12	Block representation of RL agent-environment interaction at each interaction t . Source: [31].	19
2.13	Environment showing pressure field without active control. Black dots represent the velocity probes while red ones represent control jets. Source: [34].	20
2.14	Velocity magnitude snapshot without actuation (top) and with active flow control (bottom). Source: [34].	21
2.15	Outlet temperature setpoint (black) compared with outlet air temperature obtained with the reinforcement learning controller (red) and the baseline ones (green and blue). Source: [36].	21
3.1	Rosenblatt and the Perceptron (left) and its training for image recognition (right). Notice that the training process required labeled data (“Right/wrong” and “Man/woman”). Sources: Cornell University and YouTube’s “The Thinking Machine”.	23
3.2	Diagram of the Perceptron. Source: [41].	24
3.3	A multi-layer neural network with d inputs, q outputs and k hidden layers (containing each an undefined number of nodes). Source: adapted from Quora.com website.	27
3.4	The feedforward model for total organic carbon removal and sludge production prediction (HRT stands for “hydraulic retention time”). Source: [56].	31
3.5	Convolution operation. Source: Quora.com website.	32
3.6	An example of different kernels after convolution and activation: (a) original image, (b) horizontal edge kernel and (c) vertical edge kernel. Source: [58].	33
3.7	Two main examples of pooling: max and average. Source: Analyticsvidhya.com website.	33
3.8	Simplified diagram of a CNN with two pairs of convolution-pooling layers and two feed-fowards ones. Source: Analyticsvidhya.com website.	34
3.9	Simple recurrent units and Long short-term memory ones. Source: [3].	35
3.10	Model loss for the training dataset (blue) and the test dataset (orange). Loss got stuck in 0.001.	37

3.11	New model loss (in log scale) after normalization.	37
3.12	Comparison between friction values used for testing and their respective predicted values. Data ordered in crescent order for better visualization.	38
4.1	Schematic of a PINN framework in representing the equation, initial condition and boundary condition losses in a heat transfer problem. Source: [56].	42
4.2	Training points (top) and predict and exact solutions for three different time values (bottom). Source: [71].	43
4.3	Distribution of training points randomly distributed (upper-left) and clustered (uper-right). Below the flow density prediction. Source: [80].	44
4.4	(a) Scheme of the PINN, (b) loss function with weights, (c) the PDE equations describing the flow and (d) the terms of the loss function. For better understanding, θ is the networks learning parameters (weights and biases). Source: adapted from [73].	45
4.5	Comparison of density with exact solution at various x locations using randomly distributed training points (upper left) and clustered (lower left) alongside with the errors and loss for the clustered problem (right) (“epochs” are steps in the optimization algorithm). Source: [80].	46
4.6	Schema of the PINN proposed. Source: [83].	48
4.7	Imaging set to obtain temperature field around an espresso cup. Source: [88].	48
5.1	Schema of the PINN used for solving the forward 1D diffusion problem. Source: [96].	52
5.2	Losses for the model trained using only boundary and initial conditions of the 1D diffusion transfer problem.	53
5.3	Predicted surface for $C(x, t)$ with training points in red (left) and the module error (right).	53
5.4	Predicted surface for $C(x, t)$ with training points in red (upper) and the module error surface (bottom) for case i (left) and case ii (right).	54

5.5	Schema of the PINN used for solving the reverse 1D diffusion problem. Source: adapted from [96].	55
5.6	(a) Surface predicted solution for the case where a 3.19 D was obtained, (b) surface for predicted $D = 1.00$ and (c) real solution. In red the points used for training	56

List of Tables

2.1	Risk management measures and emergency responses table as proposed by Shi and Zeng. Source: [22].	16
2.2	Comparison of real leakage parameters and the ones generated by GAN. Source: [29].	19
3.1	Activation functions (the notation $\phi(z)$ is the equivalent of $\sigma(z)$ in this work). Source: Simplilearn.com website.	26
5.1	Table with the number of domain data (real solutions), configuration of the hidden layer and the obtained D coefficient.	55

Chapter 1

Introduction

1.1 Motivation

The new so-called Industry 4.0 is changing the way products are manufactured, distributed, and improved. It incorporates new technologies and software and new techniques of machine learning to transform the new flux of data (also called the “new oil”) into valuable information for process improvement.

Among all the possible machine learning techniques, neural network is a subset that has gained much attention. The development of new optimization algorithms have permitted neural networks to be essential tools in different fields in engineering, mainly because neural networks are universal approximators.

However, it is not surprising that the number of dimensions is too high and the function-to-be-approximate too complex. The networks would require a large amount of training data and may be too complex to be optimized even by modern computers.

In such scenarios, the use of already known physics (e.g., mass conservation) could be explored alongside the neural networks in order to limit the possible solutions, making the modeled network more reliable and requiring fewer data and time for training. That is a new field that has recently emerged called Physics-informed neural networks.

The approach toward a data model of phenomena or even the hybrid approach of physics-informed neural networks are, nevertheless, still little explored in the Chemical Engineering bachelor program of the School of Chemistry (EQ - UFRJ). The need of a paper that introduces and discussed the techniques previously mentioned in therefore evident. Not only it could have internal uses in order to introduce such subject in the bachelor program, but also for external uses as a reference that aggregates many useful topics in only one simple and objective work.

1.2 Objectives

This bachelor thesis has for objective to introduce and discuss the recent advances in the state-of-the-art of Physics-informed neural networks.

1.2.1 Specific objectives

As specific objectives, this work firstly discusses the most important and applied machine learning methods in engineering along with literature examples on how they have been employed. Then, the neural networks are introduced and discussed in the same way of the previous techniques, but a more mathematical explanation and underlying the three principal types of networks in a deep learning scenario. With that theoretical basis, this work proposes the state-of-the-arts of the physics-informed neural networks, showing how physics can be integrated in the modeling process and why to do so.

Along with the literature review proposed, two practical examples of the applications of neural networks are also discussed. The first one aims to model the Colebrook-White equation for the friction factor using a standard feedforward neural network. Finally, a Physics-informed neural network is used for solving the 1D diffusion equation.

Taking into account both general and specific objectives, this work aims to propose the bases to contribute for diffusing machine learning, more specifically neural networks and physics-informed neural networks, in the engineering program of the UFRJ, with particular attention to the Chemical Engineering bachelor course.

1.3 Methodology

The literature review has been conducted using several scientific data base, such as Sciencedirect ¹. The choice of papers was usually based on the most relevant ones as well as the ones cited in it.

For the experimental parts, codes have been written in Python using the Keras (neural network) and DeepXDE (physics-informed neural networks) libraries. Plots have been made using the Matplotlib library in Python as well.

¹<https://www.sciencedirect.com/>

Chapter 2

Machine learning and examples in chemical engineering

2.1 Machine learning

With the flux of data and advances in computational performance, a new approach has emerged: machine learning (ML). While engineering has been, for many years, tackling problems by studying it in detail and generating a mathematical model that captures the physics of it, the ML approach is based on feeding a learning algorithm with enough data (so called “training set”) in order to produce a trained machine capable of carrying out the desired task [1]. These desired tasks may vary, for example, from understanding and/or generating a model of the phenomena under study to predict future values of the phenomena and detect anomalous behaviors.

As stated by [2] “the goal of ML is to design general purpose methodologies to extract valuable patterns from data”. The same authors highlight that three concepts are in the core of this technique: data, a model, and learning. Valuable pattern from **data** is extracted through an appropriate **model**; finally, the parameters of the model are optimized in a **learning** process with the aim that the model can perform well not only on the training data, but also on data not used for training.

According to [3] the learning process can be summarized as the minimization of a risk function:

$$R(w) = \int L[y, \phi(x, y, w)]p(x, y)dx dy, \quad (2.1)$$

where:

x is the data input

y is the output

$p(\cdot)$ is the probability distribution

$\phi(\cdot, w)$ is the structure of the learning machine with parameter w

$L(\cdot)$ is the Loss function

Different types of learning tasks have been proposed and used in the literature. ML tasks may be classified in 4 different ways [4]:

- Supervised and unsupervised learning: this classification is based according to the nature of the interaction between the learner and the environment. In a supervised process, the training data ¹ contains significant information that is missing in the unseen set, while that information is not present even in the training set of unsupervised algorithms. A typical example is training a ML algorithm capable of identifying spam/not-spam e-mail using as inputs a set where the subsets spam/not-spam are well defined. An hybrid classification, semisupervised, is also encountered;
- Active and passive learners: a so-called active learner interacts with the environment during training by posing queries or performing experiments, while a passive learner “observes” the information (data) provided without directing or influencing it. Using the spam identification example, while passive learners wait until the user informs them if an e-mail is considered a spam (“mark as spam” option), an active one would ask to the user if a suspicious e-mail (identified as “suspicious” after an initial training) is indeed a spam or not;
- Helpfulness of the “teacher”: ML algorithms may be trained when fed - by a “helpful teacher” - with information useful to achieve the learning goal.

¹The training data is part of the dataset and is used for training the algorithm to perform a given task. Once trained, its performance can be verified using a training with unseen data (training or validation set).

A passive teacher is possible in scenarios where training data is considered generated by a random process. Finally, an “adverse teacher” is also possible: in the spam example, it can be understood as the spammer making an effort to mislead the spam filtering design;

- Online and batch learning: in some situations, the learner has to respond online (during the training process), while in others it can process a large amount of data (batch) prior to taking a decision/generating an output.

Although 4 classifications has been proposed, the first one is the most widely discussed in the literature (see [5] for instance). Therefore, in the next section, it will be better discussed.

2.2 Supervised learning

As introduced, this type of ML implies that corrective information - labeled data - is available to the learning machine. Interpolation methods, used for centuries, are good examples. A widely employed loss function is [3]:

$$L[\mathbf{y}, \Phi(x, y, w)] = \|y - \Phi(x, y, w)\|_2. \quad (2.2)$$

The notation $\|\cdot\|_2$ is used hereafter for the norm 2, defined as $\|x\|_2 = \sqrt{\sum |x_i|^2}$.

2.2.1 Neural networks

Neural networks (NN) are the most recognized supervised learning method [3]. In this subsection it will only be introduced, since next chapter will be dedicated entirely to this method.

Recognized as fundamental nonlinear function approximators (see conclusion of the paper [6] in chapter 3), NN are powerful and flexible tools that are based on neurons as building elements. Each one of these sub-unities receives an input that is processed through an activation function, and produces an output [3].

NN building blocks are the neurons. Each input value is weighted, summed (bias are also added) and passes through an activation function before serving as an input in the next layer neuron. By doing so, neural networks can be understood as composed functions. By informing labeled data to the network it can then adjust its parameters (weights and biases) in a way to minimize the loss function through an optimization process.

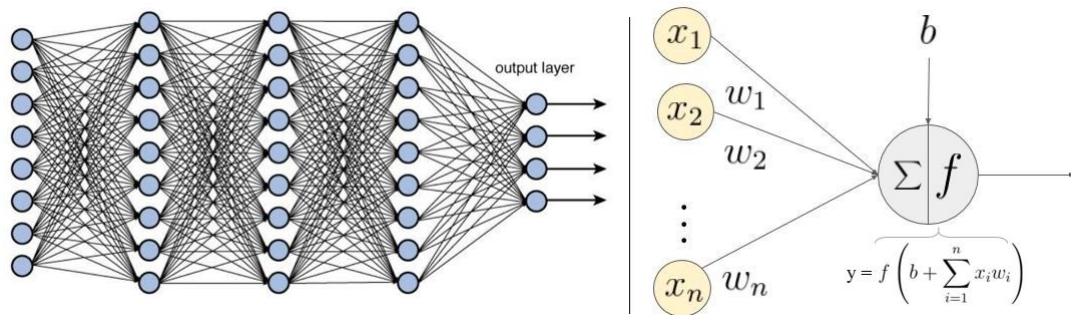


Figure 2.1: Graph representation of a neural network that performs a $\mathbb{R}^8 \rightarrow \mathbb{R}^4$ transformation with 3 hidden layers (left) and the mathematical operation that the inputs undergoes at each neuron (right). Source: adapted from Medium.com and Towardsdatascience.com websites.

Once trained, the modeled network is usually tested with unseen data in order to verify its performance.

2.2.2 Support vector machines and random forests

Support vector machines (SVM) and random forests are classification supervised ML algorithms. As it indicates, their objective is to determine the label or category of a dataset [3].

SVM linearly separates (classify) the data in order to indicate the classes to which it belongs. When not possible, a high-dimensional transformation is applied prior to linear classification, where a hyperplane is used to segregate the groups and then classify untrained data.

But considering two separable datasets, it is evident that multiple hyperplanes can be chosen for separating the data. Therefore, the algorithm must choose the

one that maximizes the distance to the closest points from each group. It can be understood as a “safe” hyperplane that reduced underfitting even when a particular point is not necessarily close to the main classified dataset [7].

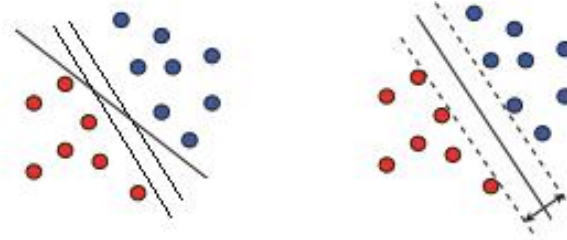


Figure 2.2: Several possible hyperplanes (left) and the one that maximizes the margins (right). Source: adapted from [7]

On the other hand, random forests algorithms use decision trees that hierarchically split the data and classify it. A good analogy on how does decision trees work is shown below [5]. Those may be the involuntary questions to distinguish 4 animals, for instance.

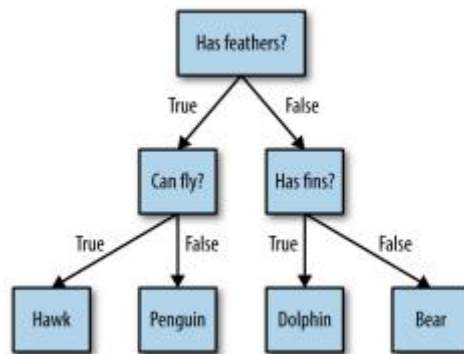


Figure 2.3: Analogy on how do decision trees work. Source: [5].

Data does not come usually in the form of “yes/no” features, but instead they are represented as continuous features. The test (question) in such cases must be: “is feature i greater than value a ?”. Depending on the result (yes/no) the data is split among two nodes. Notice that the algorithm searches for different values of i and a that are most informative, i.e., that better splits data [5].

In figure 2.4, two depths of a decision tree for a 2D dataset is illustrated. Notice how data is split in a way to reduce missclassification (i.e., blue dots in red regions

or red triangles in the blue ones).

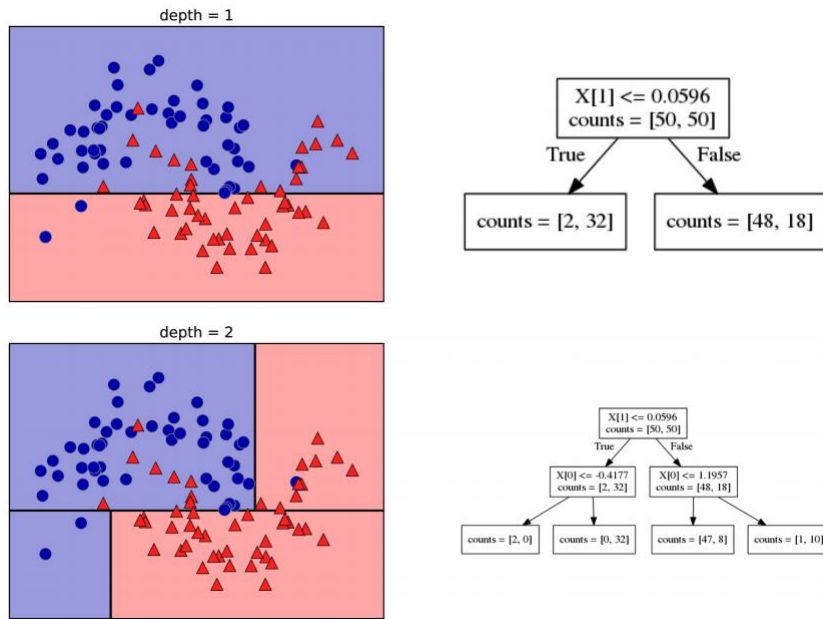


Figure 2.4: Decision boundary of tree with depth one (upper) and boundaries for depth two (bottom) in a 2D dataset. Source: adapted from [5]

A random forest algorithm is a modification of normal decision trees that aims to reduce overfitting of the data. The idea is to generate different decision trees, all of which working well and overfitting data in different ways; the idea is to average the results so reduction in overfitting is obtained, while retaining the predictive power of the trees [5].

It is important to notice that, since the output takes only discrete values, the loss function can be expressed, for both techniques, in the case where only two classes are considered, as:

$$L[y, \phi(x, y, w)] = \begin{cases} 0, & \text{if } y = \phi(x, y, w) \\ 1, & \text{if } y \neq \phi(x, y, w) \end{cases} . \quad (2.3)$$

Chen et al. [8], interested in CO₂ Minimal miscible pressure (MMP) in enhanced oil recovery applications, recently applied a supported vector machine algorithm. Using 147 sets of pressure values along with their respective reservoir temperature, oil composition and gas composition, the authors were able to develop a prediction MMP model based on SVM.

In [9], Urtubia, León and Vargas have explored the utilization of SVM as a tool for early detection of abnormal fermentation (stuck, sluggish and/or slow). The authors have studied two cases: SVM applied to group of chemicals present in the fermentation process, such as organic acids, and saturated and unsaturated organic acids; and applied to chemical variables used routinely in the process, such as density, nitrogen, Brix, and total acidity. They have concluded that aminoacids (group 1) was the best group for prediction of abnormal fermentation, while density and nitrogen were the best groups among the other set of variables (group 2).

Fault detection has huge application in chemical engineering process. In [10], the authors developed a SVM algorithm for fault detection of the Tennessee Eastman process, consisting of reactor, product condenser, vapor-liquid separator, compressor, and product stripper. 21 possible faulty condition have been considered, each one affecting somehow the process variables. The algorithm was able to correctly classify the faults with accuracy of at least 70%, with several faults being detected with accuracy of over 90%.

Kim et al. [11] considered 13 features for real-time chemical leak source tracking with random forest classifier: wind velocity, wind direction and concentration data of 11 sensors placed on the fence of the plant. Moreover, 40 leak regions (classes) has been considered. The authors concluded that random forest worked very well even with high variance in data and noise. The authors also highlighted the importance to eliminate unnecessary attributes to improve the learn rate and even its accuracy: a score of 87% has been obtained even when only 20% of the original features (8) were used.

Another example of application for both methods is in prediction of aqueous solubility. Palmer et al. [12] prepared a dataset containing 988 (of which 658 have been used for training and 330 for testing) structurally diverse organic compounds and their respective aqueous solubility at room temperature. 162 molecule descriptors ² (e.g. number of functional groups and water-force-field energy) have been generated

²These are mathematical representations of molecules' properties and their numerical values are used to quantitatively describe the physical and chemical information of the molecules.

for each molecule; those were the model features. The authors concluded that for this dataset, random forest worked better than SVM.

2.3 Unsupervised learning

In unsupervised learning, no supervision or ground-truth label are required [3]. This type of ML has many applications in feature learning, data clustering, dimensionality reduction and anomaly detection, for example [13].

2.3.1 Proper orthogonal decomposition and autoencoders

Proper orthogonal decomposition (POD) originated in fact from the field of turbulence as an attempt to decompose the fluid motion into deterministic functions that help capture a portion of the kinetic energy of the flow. In other words, POD helps to extract coherent structures from turbulent flow - and many other phenomena - that would be difficult to define and observe [14]. Therefore, POD aims to obtain low-dimensional representation of high-dimension data.

The key arguments for using this method are [7]:

- to compress initial data to speed up computational operations;
- to better visualize data by mapping it in a 2 or 3-dimensional space;
- to generate a smaller and useful - or even more effective - set of features.

POD can be formulated as a two-layer NN (autoencoder), or a more complex one (deep encoder) [3]. Both are presented in the figure below.

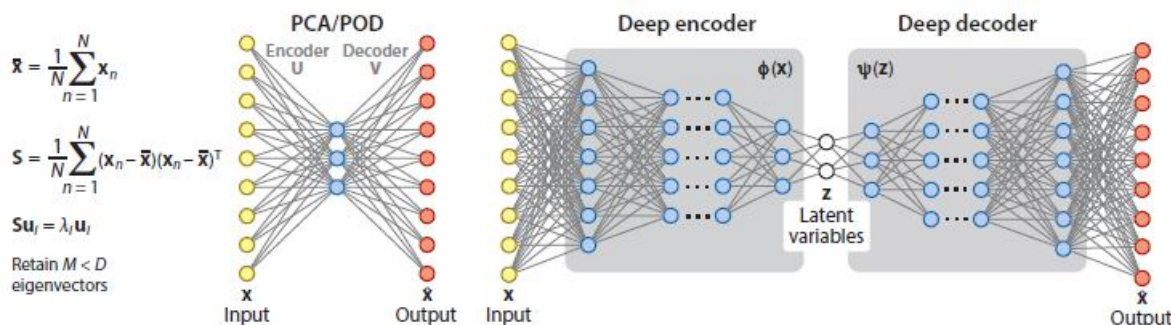


Figure 2.5: POD (left), autoencoder (middle) and deep autoencoder (right) structures [3].

As seen above, the goal of POD is to find a set of orthogonal axes aligned with the greatest variability directions of data [15]. The mathematical formulations in Figure 2.5 show that an empirical covariance matrix of mean-subtracted data (S) is decomposed in its eigenvalues and eigenvectors, where only the first (ordered by module) M eigenvectors are retained. One has, therefore, reduced the system from an initial D -dimension to a M -dimension one, still maintaining as much information as possible ($R^D \rightarrow R^M$, where $D \gg M$).

As mentioned, autoencoders - examples of NN - can also perform the task of POD. However, being a NN, it is (1) a non-linear transformation; and (2) axes in POD are ordered with respect to their representational power, while the same is not true in autoencoders [15].

In [16], Akkari et al. highlighted that in many applications, such as in optimal control problems, reduce the Navier-Stokes equations for different parametric values to then minimize a given function with respect to these values may be needed. Nevertheless, resolution cost for such optimization problems is too high from a computational storage capacity and time point of view. It can be solved by generating a reduced order model by POD.

In chemical process, a complexity of data is usually encountered. Teng et al. [17] demonstrated the application of order reduction via an autoencoder in a oil refinery plant. 15 input variables have been used as inputs for the network, optimized with 10,000 simulated datasets and tested with others 3,000. The authors were able to reconstructed the input data based on only 2 latent representations with mean absolute error of just 9.54%. The process representation alongside with the structure of the autoencoder can be seen in Figure 2.6

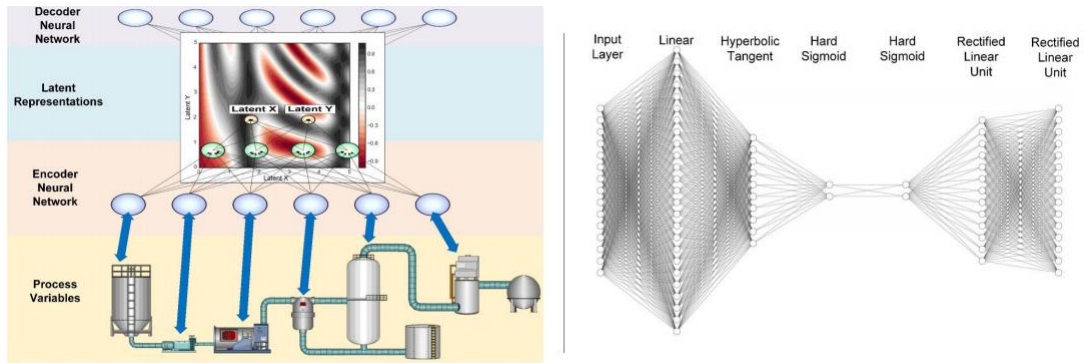


Figure 2.6: Process case study (left) and autoencoder structure (right). Source: adapted from [17]

Agostini [18] demonstrates the application of autoencoders in fluid dynamics by tackling the problem of a 2D unsteady flow around a cylinder. The flow was simulated and 12 stacked snapshots of the velocity field (a 256×88 mesh) were used as inputs for the network. The latent dimension reduction layer consisted of only 3 nodes, i.e., only $\frac{3}{12 \times 256 \times 88} \times 100 = 0.001\%$ of the original dimension information is used to reconstruct the inputs. Over 500 temporal sets of 12 snapshots are used for training, while 128 temporal sets are used for testing. A random chosen streamfield snapshot is show in the Figure 2.7, as well as its reconstructed field and reconstruction error.

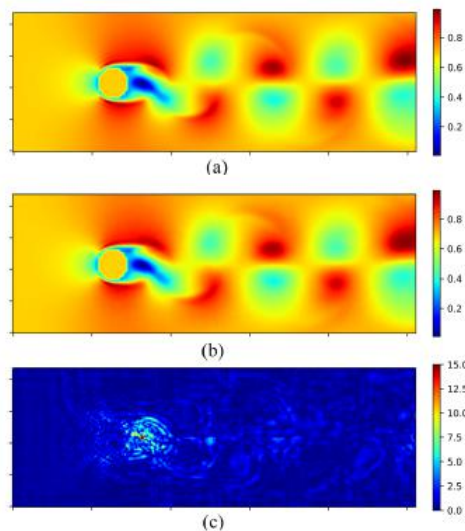


Figure 2.7: (a) Original velocity field, (b) reconstructed field from the 3-dimension latent variables and (c) reconstruction error. Source: adapted from [18].

Autoencoders have also demonstrated applicability as order-reduction prior to utilization of failure detection models. Scoralick et al. [19] obtained real data of a in-production gas-lift oil well; a total of 129,592 data groups (sampled at each 1 minute) of 16 process variables (discrete or continuous), such as pressure and temperature upstream and downstream choke valves, gas-lift valves and in the Christmas-tree, worked as input parameters for a stacked autoencoder. In stacked autoencoders the latent layer of one networks works as the input for the next autoencoder and so on; for this work, only two autoencoders have been used, reducing the 16 variable to first 9 and then to 5 (Figure 2.8).

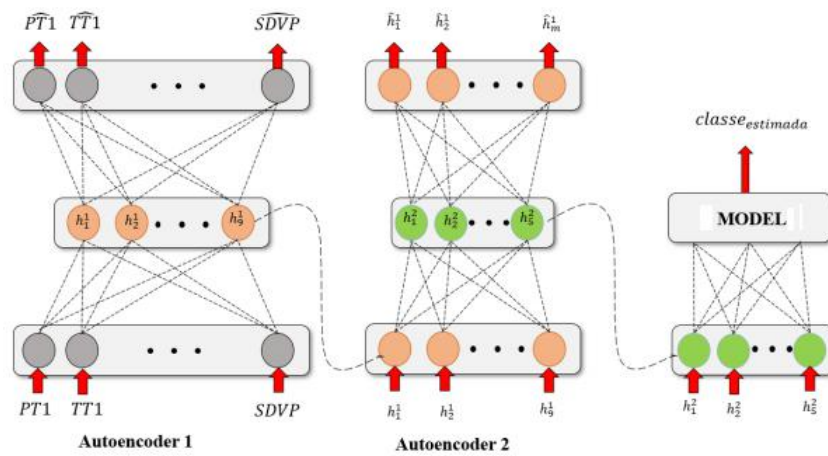


Figure 2.8: Stacked autoencoders for order reduction prior to model training. Source: adapted from [19].

Scoralick et al. have then used the reduced variables in supervised models, such as SVM and decision trees. It is important to highlight that for each group of 16 variables (and therefore for the 5 reduced ones) it was known the failure status - non-failure, soft-failure and hard-failure - all needed for training this final model. It is an example of hybrid machine learning technique, that uses unsupervised (autoencoders) for pre-processing of data and supervised (SVM and decision trees) learning techniques. The authors concluded that by reducing the order of the system has accelerated the failure training model maintaining its performance.

2.3.2 Clustering

A clustering problem can be understood as the attempt of finding homogeneous groups of data points in a data set [20]. k-means algorithm is the most common for this task [3] and is based in minimizing the distance of data in a cluster to its respective centroid.

The k-means clustering algorithm works as follow:

- a number k of cluster centers are initialized (usually randomly);
- each data point is assigned to the closest³ cluster center;
- each cluster center is re-centered based by meaning the data classified in each cluster;
- the algorithm repeats until data assigned to each cluster no longer changes.

A visual representation of this clustering algorithm is represented in figure 2.1.

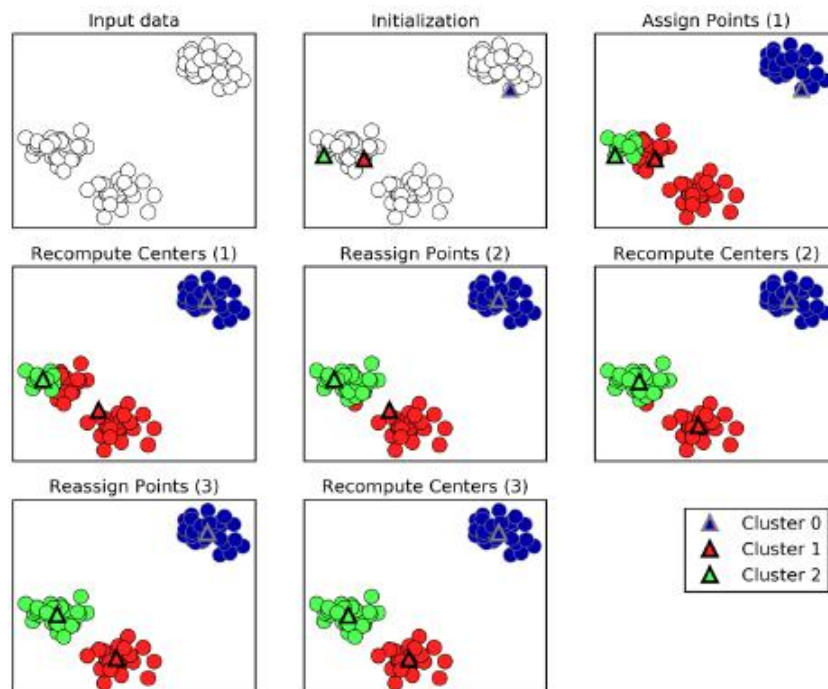


Figure 2.9: Representation of the k-means clustering algorithm for a 2D dataset and 3 clusters. Source: [5].

³The most common distance measurement is the Euclidean distance, but others are also possible, such as City block (Manhattan), Cosine and Correlation distances [21].

Shi and Zeng [22] applied clustering to environmental risk zoning of the Nanjing Chemical Industrial Park (642 km²) in China. Several risks indexes (e.g. chemical release quantity) have been listed, modeled and calculated for each grid (subarea) of 100 x 100 m within the studied area. Then, subareas have been clustered based on their different indexes (authors found that the best k value was equal to 5). Resulting zoning map is shown below. By doing so, authors proposed that application of the appropriate risk management policy would be facilitated (resume of the proposed actions also in table below).

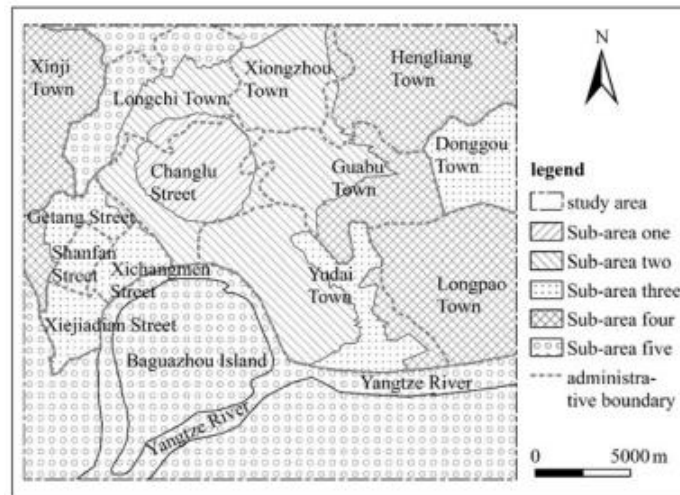


Figure 2.10: Risk zoning map of the Nanjing Industrial Park proposed by Shi and Zeng. Source: [22].

Table 2.1: Risk management measures and emergency responses table as proposed by Shi and Zeng. Source: [22].

sub-areas	sub-area denomination	risk management measures and emergency responses
sub-area one	risk sources gathered area	Enforce management and control of the risk sources; set up automatic monitoring, alarm, emergency cut-off, and accident disposal systems to improve fire prevention, explosion-proofing, poisoning prevention, etc.; quickly succor in case of accidents.
sub-area two	risk field intensification area	Ensure that succor passages are unblocked; rescue and transfer injured people.
sub-area three	target susceptibility area	Constitute a detailed evacuation plan to ensure a systematic evacuation and transfer of people.
sub-area four	target protection area	Establish refuges, protect people, and accept people evacuated from other sub-areas.
sub-area five	risk field attenuation area	Adopt measures such as interdiction and counteraction to lessen the intensity of the risk field; transfer possibly injured people.

2.4 Semisupervised learning

This category of algorithms require only partial supervision where either limited labeled training data is available or with corrective information from the environment. Usually, the precise definition of a semisupervised algorithm is not very clear in the literature; however, both [23], [24] and [25] agree that semisupervised learning is halfway between unsupervised and supervised ones: some labeled data is informed as well as unlabeled one and, therefore, dataset can be divided in two. This is a very shallow definition, but enough for this study. More specific details will be discussed below, where Generative adversarial networks (GAN) and reinforcement learning (RL), the main examples of semisupervised algorithms [3], are presented.

2.4.1 Generative adversarial networks

First introduced in 2014 [26] and, just like autoencoders mentioned before, GAN are based in NN (although other systems are also possible), where two of the networks are trained by competing with each other [27]. These networks are identified as:

- Generator: a network that aims to produce data (e.g. an image) that seems realistic for the discriminator;
- Discriminator: the second network and the only one to have access to both synthetic (from the generator) and real samples.

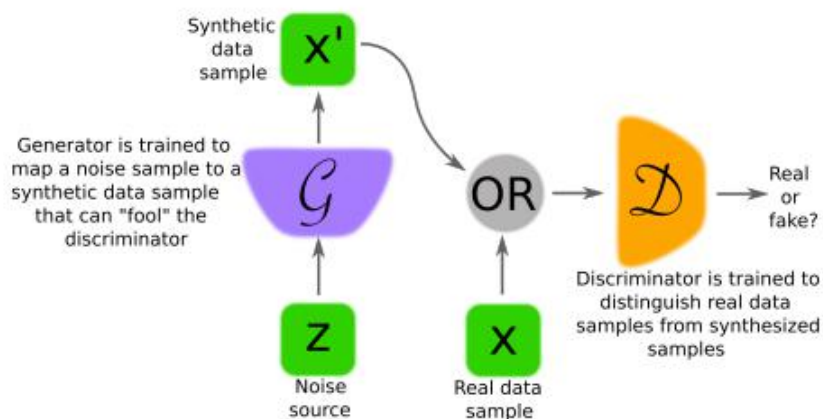


Figure 2.11: Representations of the generator and discriminator in a GAN. Source: [27].

The error in the discriminator training is provided knowing whether the image is real or from the generator. This error signal is also used to train the generator so it can improve its capacity to produce better quality data to “fool” the discriminator [27].

Generative adversarial networks have encountered application in design of new molecules and materials. Dan et al. [28] stated that given a large set of training samples, these networks are “capable of learning complicated hidden rules that generate the training data, and then applies these learned rules to create new samples with target properties”. In their study, a large database of materials (OQMD) was used: each material was converted in a sparse 0-or-1 $d \times s$ matrix, where d ($= 8$) represents the number of atoms of s ($= 85$) in the molecule. The network was trained in order to generate new samples (generator) and to better discriminate the generate data from the database one (discriminator). Then, 2 million samples were generated after training of the generative adversarial network; the authors highlighted the importance to filter such generated data in order to select the plausible ones, i.e., the ones that obey rules such as charge neutrality and electronegativity balance (that is a subject that will be evoked in chapter 4). To test the potential of discovery of new materials, the authors have cross-validated the predicted new materials with other database to check how many have already been confirmed as potential materials: more than 13,000 ones have been validated.

Another example of inverse problem using GAN can be found in [29]. Zheng et al. were interested in estimating liquid pipeline leakage parameters. Through simulation, the authors generated 100,000 data containing different process conditions (upstream pressure head and flowrate) and leakage parameters (leakage location, coefficient and time); only the leak parameters are used for training. Once trained, two real process data has been used as inputs of the GAN, and the 3 outputs (leakage parameters) where confronted with the real values. Good results for all parameters were obtained as can be seen in Table 2.2.

Table 2.2: Comparison of real leakage parameters and the ones generated by GAN. Source: [29].

		Leakage location (m)	Leakage coefficient	Leakage time (s)
Real	Example 1	2082.4	0.00320	141.4
	Example 2	5934.6	0.00212	155.8
The GANs framework	Example 1	2100.0	0.00314	142.5
	Example 2	6000.0	0.00216	156.6

2.4.2 Reinforcement learning

First of all, in RL, two definitions must be introduced: agent and reward. The agent is the entity that makes an action and observes its effects on the environment; therefore, the agent can be, for example, a NN [30]. The reward can be seen as a function that translates mathematically how close the agent is to the final objective.

In RL, the learning agent interacts with the environment in order to maximize the reward signal (or minimize punishment). Since its not a supervised method, the agent is not told what to do because it has no labeled information about the correct actions to take; instead, it must discover which actions yield the most reward [31].

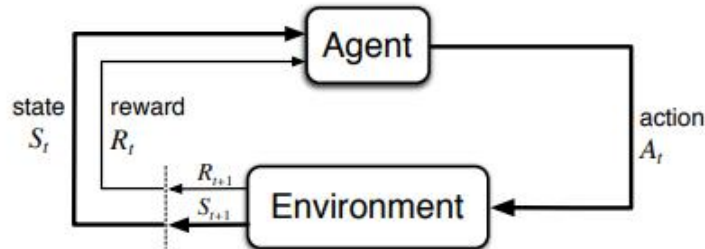


Figure 2.12: Block representation of RL agent-environment interaction at each interaction t. Source: [31].

In figure 2.12 it has been presented the diagram of the agent-environment interaction in RL. It is important to notice that in many cases the action affects not only the reward but also the next state and, therefore, all subsequent rewards so the learning process must account to optimize global reward [31].

Different approaches are currently being used for solving problems with reinforcement learning. To explain all of them is out of the scope of this work, nevertheless

to give a feeling of this important method, the deep learning approach is quickly introduced below because it is neural networks based and has huge applications in process control. The author highly suggests the lecture of [32] and [33].

Perhaps the most important application of Reinforcement learning in chemical engineering is for process control. In this context, a state (S) is defined as the the current state of the plant; an action (A) is the means through the agent (e.g. controller) interacts with the environment (e.g. of an action is the controller output); the reward (R) indicates how well the agent is doing at step t (e.g. how well the process output is reaching the setpoint value). The idea is to approximate two functions using deep (with a lot of layers) NN [32]:

- the policy function $\pi(S)$ that represents the actor and, given a state S , proposes the action A to take;
- the critic function $Q(S, A)$ evaluates the quality of the action A taken by the actor.

Rabault and Kuhnle [34] were interested in active flow control, that has many applications in drag reduction on vehicles and airplanes, and in optimization of combustion process in engines, for example. The environment consisted of a unsteady flow across a cylinder, pressure probes were placed nearby the solid as well as two small jets set on the sides of it able to inject fluid (normal direction) (Figure 2.15).

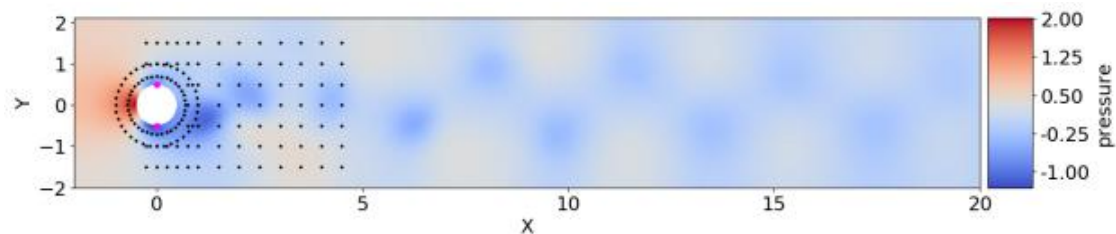


Figure 2.13: Environment showing pressure field without active control. Black dots represent the velocity probes while red ones represent control jets. Source: [34].

By applying reinforcement learning, the control law found was able to reduce up to 93% of the drag induced by shedding. Although good results were obtained,

24 hours of training were necessary (a problem that was further addressed in [35]). Reduction of the vortex due to active flow control is shown in the figure below.

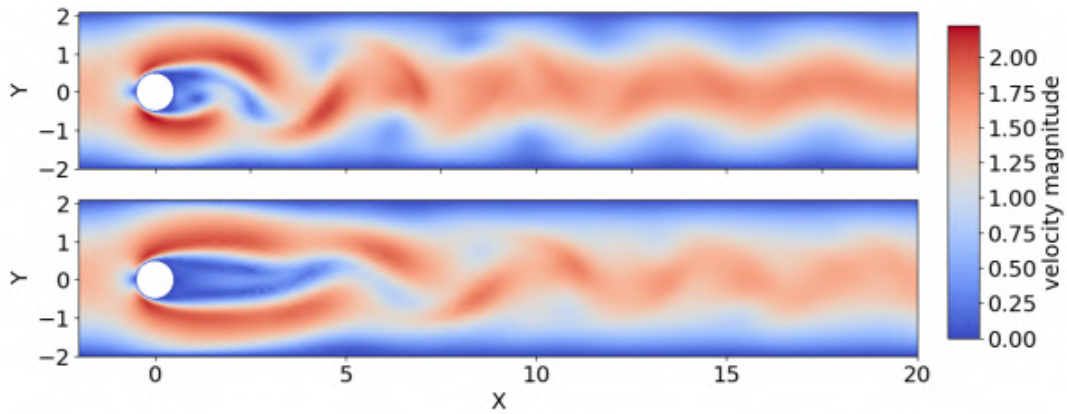


Figure 2.14: Velocity magnitude snapshot without actuation (top) and with active flow control (bottom). Source: [34].

Wang, Velswamy and Huang [36] studied an air heating system that used hot water. The system contains 3 variables influenced by controller (output air temperature, output hot water flowrate and hot water output temperature) and 3 others external variables (inlet air temperature, inlet air flowrate and inlet hot water temperature) not influenced by the controller. The networks were trained face different rollouts situations, given them enough time to achieve desired setpoint before the next rollout training set. Authors concluded that the proposed controlling technique worked much faster than the baseline controllers (e.g. PI).

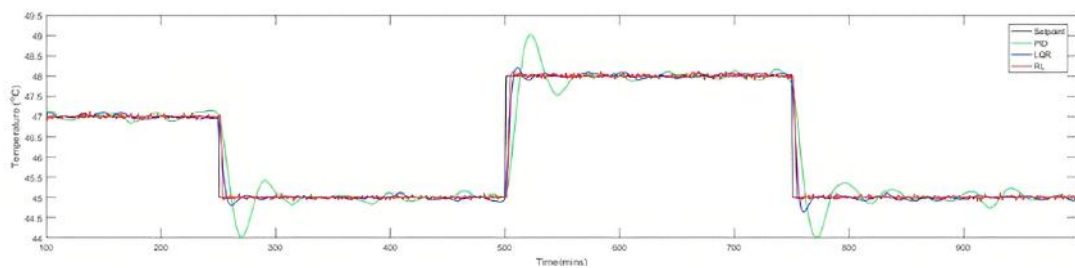


Figure 2.15: Outlet temperature setpoint (black) compared with outlet air temperature obtained with the reinforcement learning controller (red) and the baseline ones (green and blue). Source: [36].

2.5 Conclusion of the chapter

In this chapter, the principal machine learning techniques were described and classified as the type of supervision. Through the examples presented, one can have a good idea of the principal applications in chemical engineering: discover of new materials, process control, classification etc.

Neural networks have been described as a powerful tool for functions approximation. Moreover, they can also be applied as a tool for order reduction, new data generation and in reinforcement learning. It is a topic with great interest in the new 4.0 industry and deserves to be better discussed. Therefore, the next chapter will better describe them, discussing their applicability and their types/architectures.

Chapter 3

Neural networks

As the name suggests - and as it has already been introduced before - artificial neural networks (called neural networks here after) were inspired by the study of the biological system of neurons [37].

One of the first application of this concept happened in the 50s, when the Perceptron, an electronic device, was developed by Rosenblatt [38] inspired on the earlier work by McCulloch and Pitts [39]. The Perceptron was defined by Rosenblatt himself as a “nerve net’ consisting of logically simplified neural elements, which has been shown to be capable of learning to discriminate and to recognize perceptual patterns” [40], such as male/female recognition (Figure 3.1).



Figure 3.1: Rosenblatt and the Perceptron (left) and its training for image recognition (right). Notice that the training process required labeled data (“Right/wrong” and “Man/-woman”). Sources: Cornell University and YouTube’s “The Thinking Machine”.

This machine worked as follow:

- stimuli from training set impacted on a retina of the sensory units;
- the outcome value was weighted with initially-random values;

- this output was then “filtered” through a hard-limiter function: if it was greater than a threshold, then the final output was “true”, “false” if not;
- if the final output did not match the expected value, the weights were adjusted: decreased in the case of a false-true or increased if false-negative.

A representation of the Perceptron can be found in the figure below.

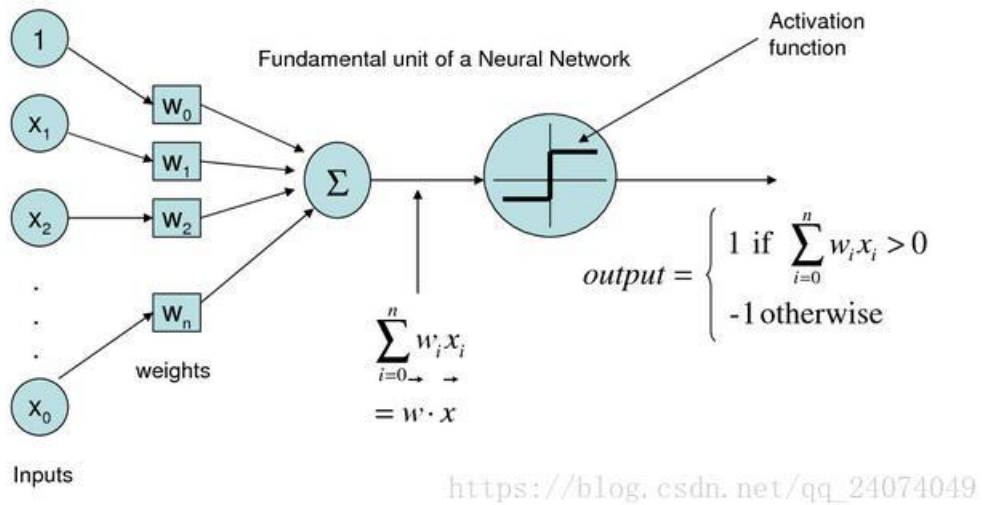


Figure 3.2: Diagram of the Perceptron. Source: [41].

However after years of new developments, the excitement concerning artificial intelligent in general has fade away due to disappointments in machine translation, obstacles evoked by multiple researchers, and the Lighthill report (1973) [42], that stated that the promises in this field were exaggerated.

It was much later that NN became again popular, specially with the development of deep neural networks. A boom in available data, and advances in computer capacity have largely contributed to that.

In the following sub-sections, a more rigorous description of neural networks is proposed, as well as its universal approximation characteristics. Finally, the deep learning context will also be discussed.

3.1 Nodes, activation functions and networks

The node (neuron) is the basic brick of a NN, that receives an input, transmitting an output (to another neuron or not) after processing it. The Perceptron model shown before is the simplest neural network consisting of d input nodes and one output node. These inputs are weighted and summed; after an activation function computes the output. Following paragraphs are based on [43], [37] and [44] if not informed otherwise.

Considering d input features represented by $\bar{X} = [x_1, \dots, x_d]$ and d weights (usually represented by the edges coming out of each node) represented by $\bar{W} = [w_1 \dots w_d]$, then the linear function $\bar{W} \cdot \bar{X} = \sum_{i=1}^d w_i x_i$ is computed at the output node. Then an activation function $\sigma(\cdot)$ is applied. Therefore, the prediction \hat{y} is computed as:

$$\hat{y} = \sigma(\bar{W} \cdot \bar{X}) \quad (3.1)$$

In most cases, a bias needs to be incorporated to consider invariant parts of the prediction (that do not depend on the input). It can be achieved by adding a value b to the inner product of the previous equation. However, in order to reduce notation, one can use a bias neuron: the inputs are now defined as $\bar{X} = [x_1, x_2, \dots, x_d, 1]$ and the weights as $\bar{W} = [w_1, w_2, \dots, w_d, b]$. Therefore, no modification in equation 3.1 is required.

Weights and bias are computed so to minimize a loss function $L(\cdot)$. Maybe the most basic yet effective loss function can be defined as:

$$L = \sum_{(\bar{X}, y) \in D} |y - \hat{y}| \quad (3.2)$$

Where D represents the space of input data (\bar{X}) and its respective label y .

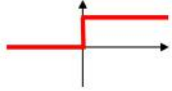
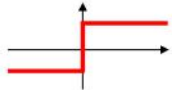



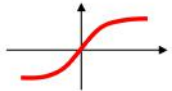
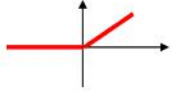
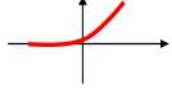
Many activation functions are in use nowadays. One of the most popular is the sigmoid or logistic function, which has interesting mathematical properties such as monotonicity, continuity and differentiability.

$$\sigma_{sigmoid}(z) = \frac{1}{1 + e^{-z}} \quad (3.3)$$

Other activation functions are presented in the Table 3.1. It is important to highlight that due to ease in training multilayers neural networks (a concept that will be introduced soon), the ReLU have largely replaced the sigmoid function [44].

Table 3.1: Activation functions (the notation $\phi(z)$ is the equivalent of $\sigma(z)$ in this work).

Source: Simplilearn.com website.

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = \max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

Copyright © Sebastian Raschka 2016
(<http://sebastianraschka.com>)

In 1969, Misky and Papert [40] published a book where limitations of the single layer Perceptron were pointed out. To overcome such limitations, a multi-layer network had to be used (although only with the back propagation algorithm development one could set the weights of the structure). This multi-layer Perceptron will be hereafter called a multi-layer neural network, or simply Neural networks (NN).

NN are composed of layers, a set of neurons. The first layer is the input one, where no computation is performed: it only transfers the input data to the subsequent layers. The following intermediate layers are called hidden layers, where the computation is performed. Finally, an output layer is used before generate the output(s) of the network. It is important to highlight that all the layers may vary concerning the number of neurons. Moreover, the hidden layers may also vary in the number of layers (the number of total layers is also called depth).

In a multi-layer NN, all nodes should be interconnected and successive layers feed one another in the forward direction (although it is not obligatory). These networks are called feed-foward NN. A complex NN can be seen in Figure 3.3.

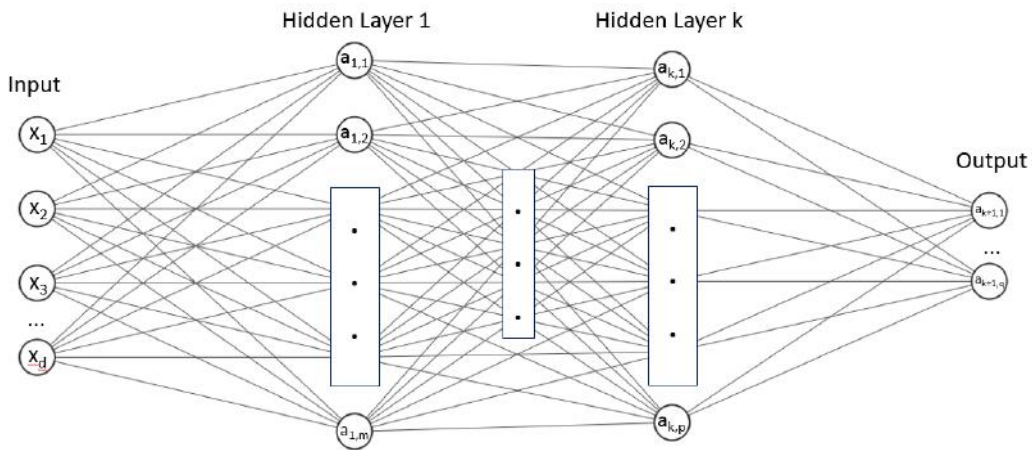


Figure 3.3: A multi-layer neural network with d inputs, q outputs and k hidden layers (containing each an undefined number of nodes). Source: adapted from Quora.com website.

As illustrated above, each layer value $a_{i,j}$ can be obtained through equation 3.1, i.e.:

$$a_{i,j} = \sigma(\bar{W}_{i-1,j} \cdot \bar{X}_{i-1}) \quad (3.4)$$

where $\bar{W}_{i-1,j}$ represents the weights of layer $i - 1$ that are connected to the node $a_{i,j}$, and \bar{X}_{i-1} represents the input values (if $i = 1$, then $\bar{X} = [x_1 \dots x_d \ 1]$).

As one can see, a NN is nothing but a composition of functions where:

$$\hat{y} = \bar{W}_{k+1,j} \cdot \sigma(\dots \bar{W}_{2,j} \cdot \sigma(\bar{W}_{1,j} \cdot \sigma(\bar{W}_{0,j} \cdot \bar{X}_0))) \quad (3.5)$$

In their famous paper published in 1989 [6], Hornikm Stinchcombe and White have have proved that:

“Standard multilayer feedforward networks are capable of approximating any measurable function to any desired degree of accuracy, in a very specific and satisfying sense. We have thus established that such ‘mapping’ networks are universal approximators. This implies that any lack of success in applications must arise from inadequate learning, insufficient numbers of hidden units or the lack of a deterministic relationship between input and target”.

The back propagation algorithm exploits the composition nature of NN in order to determine, based in the mathematical chain-rule, the weights that minimize the error function. Modern stochastic gradient descent and back propagation algorithms are able to accomplish this task [45].

3.2 Some considerations on NN optimization

Firstly, it is important to highlight that the function $\sigma(\cdot)$ must be differentiable - or piecewise differentiable - in order to be used in the optimization gradient descent algorithms [45]. Such examples of functions are: linear, binary step, logistic, tanh and ReLU.

It is not the objective of this thesis to explain in detail how does a backpropagation algorithm works from a mathematical point of view. However, its main idea can be understood as follows:

- Once the the NN is specified, initial weights are set at random values;
- The training data in run through the network and the error is computed;
- The derivatives with respect to each weight are computed;
- For a giving learning rate δ , the weights are updated;

- The algorithm returns to item two and continues to iterate until convergence (of error or weights) is achieved or the maximum number of iterations reached.

While backpropagation allows an efficient computation of the objective function's gradient, optimization of minimization of the loss is conducted by the Stochastic gradient descent algorithm, for example (other examples are the Adam and RMSProp algorithms).

It is also important to highlight that data usually need to be pre-processed prior to be used in a neural network or any other machine learning algorithm. Not only missing data or labels must be properly handled, but in many examples in chemical engineering data normalization should be conducted [46]. By doing so, one avoids that the networks concentrates in the big values (since they contribute to higher errors), neglecting information from small valued variables.

3.3 Deep learning

Deep learning refers to complex neuron networks with a high number of layers and neuron in each layer. In such context, three main types of networks arise: feedforward, convolutional and recurrent. Since just feedforward networks have been presented so far, just examples in chemical engineering will be presented in the subsection below, while it is worth to explain the others two in the following subsections.

3.3.1 Feedforward networks: some examples

The applications in chemical engineering of feedforward networks are numerous: thermodynamics, transport phenomena, catalysis, and process analysis and optimization are just a few examples. One example is demonstrated in section 3.4, where its capacity to approximate functions is demonstrated. A good review on the applications is proposed in [47] and [48].

When used to substitute flash algorithms, NN leads to faster calculations [49] (up to 35 times). Data generated from 101 different compositions of water-methanol, 500 temperatures and 500 pressures has been used for training the network, which demonstrated high accuracy for predicting phase classifications.

Alves, Quina and Nascimento [50] have developed a NN to determine whether binary mixtures exhibit azeotropy behavior. Only pure components properties has been used as input variables and good prediction was obtained (“the model failed in only a relatively small number of situations in which structurally homologous molecules are known to exhibit quite distinct azeotropic behavior”).

In [51], authors have modeled the heat transfer coefficient between fluidized bed and tube bundles immersed in it. Values were confronted with experimental ones and those obtained via correlations with errors as small as 0.6%. Authors also concluded that the feedforward NN represented system behavior more accurately than conventional models.

In kinetics, NN has been used for estimating reaction rate of methanol dehydration [52] and maltose hydrolysis [53], for example. Also, for optimization of catalysis, Omata and Yamada [54] related the physicochemical properties of elements (X) and the selectivity of the catalyst containing the element (nickel- X /active carbon); the authors concluded that selenium was a good additive, even if its properties has not been included in training data.

In process optimization, Assidjo et al. [55] have applied NN to predict the final moisture of coconut after passing through a dryer. The input dataset consisted of initial moisture, seven temperatures of each one of the dryer’s subcompartments, and the final temperature of the product. Errors as small as 0.35% were obtained even if the dynamics of the process of drying grated coconut are “poorly known”, accordingly to the authors

In [56], the authors designed a NN model (Figure 3.4) comprising 4 inputs and 3 hidden layers (with 10 neurons each) for prediction of total organic carbon removal and sludge production in reverse osmosis process. Authors concluded that

the technique was able to model the processes that is very difficult to describe with a parametric approach.

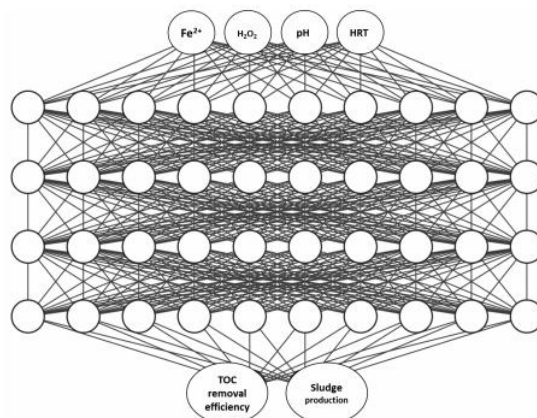


Figure 3.4: The feedforward model for total organic carbon removal and sludge production prediction (HRT stands for “hydraulic retention time”). Source: [56].

All the previous examples have in common the fact to have a quite simple NN, with little computational time needed to optimize them. Furthermore, a not so large amount of input inlets were needed to model systems whose mathematical model were, sometimes, unknown. Nevertheless, the amount of data to train such networks may be large and not necessarily available, so authors had to either collect data from experiments (expensive and time consuming), from simulations or even from published papers (time consuming). Despite that, feedforward networks are still an important tool in chemical engineering.

3.3.2 Convolutional networks

This specialized type of NN are specially suitable for processing data that has a grid-like topology, like pictures, volumetric data (e.g., computed tomography scan 3D images) and time-series data (e.g., audio, videos and computational fluid dynamics simulations).

Maybe the best example of how this type of NN works is to think on image recognition. It would be very naive to imagine that the networks presented so far would be able to take every single pixel of an image (input) and inform if the data contains a human face; human faces are prone to high diversity and, furthermore,

may be located in different positions in the image. Nevertheless, how can human-beings identify others as humans at the first time one sees another one? They simply look for features such as: faces, ears, noses, hair etc. CNN are able to do that through three main layers: convolutional layer, pooling layer and fully-connected layer [57].

A convolution operation is a generalization of averaging a function $x(t)$ through the weighted function $w(t)$. It is denoted as $s(t) = (x * w)(t)$ and is mathematically defined as [26]:

$$s(t) = \int x(\tau)w(t - \tau)d\tau. \quad (3.6)$$

In a more practical way, $x(t)$ can be seen as the input dataset (e.g., a 2D image), while $w(t)$ is a matrix, called kernel, that extracts features from data. The first hidden layers (the convolutional layer) are composed of filters, i.e. an ensemble of different kernels [57]. A matrix convolution product between two matrices (the input data and the kernel) is exhibited below.

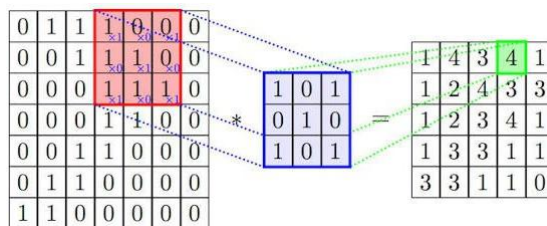


Figure 3.5: Convolution operation. Source: Quora.com website.

After convolution, the results must pass through an activation/detection stage [26]. Considering that the objective of the kernel is to identify edges in an image in the following way: returns a positive value if two adjacent pixels are of the same color, negative otherwise. The detection section simply means to pass the convolution result through a ReLu function: the output is 0 if no edge has been detected or a value indicating how much of an edge is the feature (Figure 3.7).

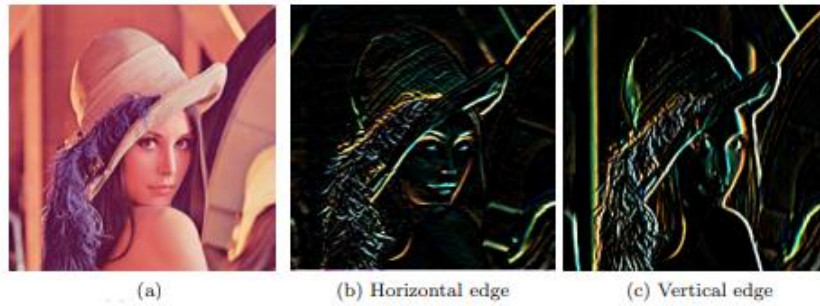


Figure 3.6: An example of different kernels after convolution and activation: (a) original image, (b) horizontal edge kernel and (c) vertical edge kernel. Source: [58].

Next there is the pooling layer where small rectangular blocks from the convolutional layer are subsampled, similar to an order reduction process. Usually, a max pooling is used (takes the highest value on the blocks), but other types are also possible, just like the average pooling [58]. Pooling is a way to create a lower resolution version of the input maintaining the important features, which is very important since feature in the original data may be prone to variation in the position and noise for example.

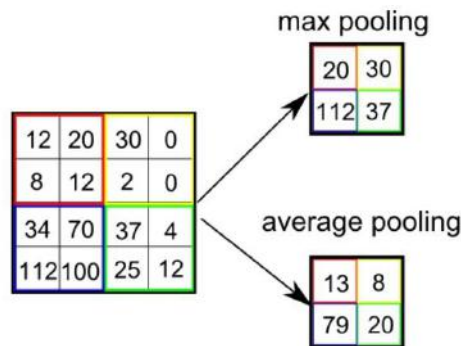


Figure 3.7: Two main examples of pooling: max and average. Source: Analyticsvidhya.com website.

This is a process that may occur repeatedly in the CNN architecture, i.e. the output of the pooling layer serves as the input of another convolution and so on. Finally, a fully-connected layer (the standard type of NN explained so far) is used to extract the information from the precedent transformations (e.g., “is there or not a human face in this image”). A simplified diagram of a CNN is provided below (Figure 3.8).

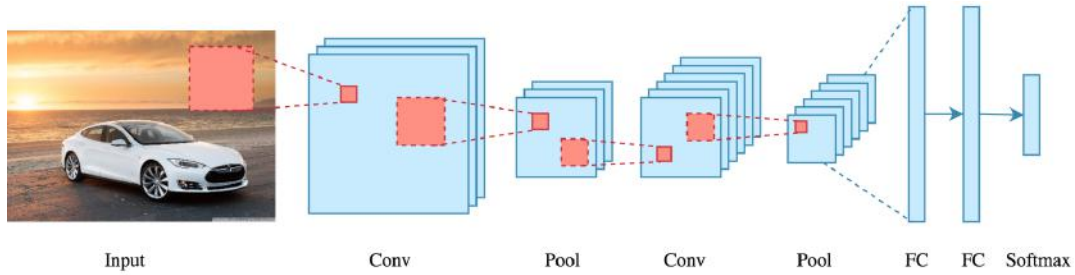


Figure 3.8: Simplified diagram of a CNN with two pairs of convolution-pooling layers and two feed-fowards ones. Source: Analyticsvidhya.com website.

This type of deep NN has been used to predict steady flow profiles around elementary (e.g., circles, triangles and squares) and real life (e.g., cars) 2D and 3D objects in [59]. In [60], a CNN is trained to predict velocity fields around 2D cylinders over various Reynolds numbers (from 60 to 1100); the authors used a time series dataset of pressure fluctuations on the object surface as input, while velocity field was the output. Pressure and velocity fields have also been predicted using CNN in [61], where the network was trained with random shapes (generated using Bézier curves) labeled with their respective fields, the network was then tested in unseen shapes, such as foils.

3.3.3 Recurrent networks

Recurrent Neural Network(s) (RNN) help process sequential data [26]. As the name suggests, it is designed with recurrent (cycle) connections within the networks; in other words, it is a structure that the output value from a neuron is directly, or indirectly, dependent on its early outputs, conferring to the system a dynamic character/ a temporal dimension.

This “memory” is done in the network through a Simple recurrent unit (Figure 3.9). In it, the historical information h_{t-1} (“h” for hidden) is combined with current input x_t (concatenation) to obtain the current output h_t . However, this unit lacks to handle long-term dependencies; therefore, a new unit, the Long short-term memory, has been developed (Figure 3.9), the nomenclature stays the same, with addition to c_t that represents the current cell memory.

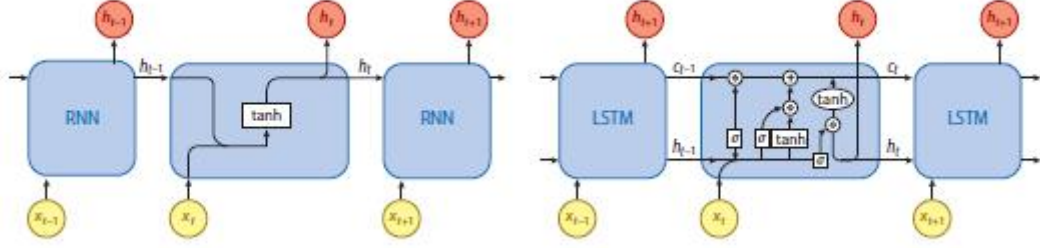


Figure 3.9: Simple recurrent units and Long short-term memory ones. Source: [3].

It is important to mention that, although not shown in the figure, in each of these units there are linear transformations occurring (weights and biases). Therefore, as expected, these networks also need to be optimized in order to reduce the prediction error.

This type of NN has been used to detect combustion instability [62], with huge applications in gas turbines, aviation and rocket engines. Authors have trained the model with data (video and audio) from a laboratory-scale combustion system. The proposed model was accurate in defining stability or instability from the testing data.

Other example in fluid mechanics is [63] where good turbulence statistics and dynamic behavior of the flow was obtained. In [64] they were used to predict water flow in a real power plant. A combination of convolution, autoencoder and recurrent network has been proposed [65] for modeling 3D turbulence at a low computational cost.

3.4 Practical example: Colebrook-White equation

To show its capacity to approximate functions, it is proposed to generate a feedforward neural network capable to predict the friction factor as given by the Colebrook-White equation [66]:

$$\frac{1}{\sqrt{f}} = -2 \log \left(\frac{\varepsilon}{3.7D} + \frac{2.51}{\text{Re} \sqrt{f}} \right) \quad (3.7)$$

Where f is the friction coefficient, ϵ is the pipe roughness, D is the diameter of the pipe and Re is the Reynolds number of the flow.

To be solved, such implicit equation requires iterations. For nowadays computational performance, such solvers are not hard to implement and does not take very long to converge; nevertheless, such physical equations are not always available. The main idea in this section is to pretend one does not know such equation, but has acces to experimental data.

The feed-forward NN proposed uses ReLU for the activation functions, takes 3 inputs (diameter, Reynolds number and pipe roughness) and consists of 3 hidden layers with 50, 100 and 10 neurons (no study concerning the best architecture has been conducted, values chosen aleatory). The only output of the network is the friction factor value. The objective is to optimize this network by determining the 6,330 ($50 \times 3 + 50 + 100 \times 50 + 100 + 10 \times 100 + 10 + 1 \times 10 + 10$) variables that minimize the squared error between expected and predicted friction values.

A dataset containing 10000 input-output pairs ($[Re, d, \epsilon]$, $[f]$) was used to train and validate the NN. Input values were randomly generated by uniform distributions with the following boundaries: from 1.0 to 10.0 (m) for the diameter; from $2.5 \cdot 10^3$ to 10^8 for the Reynolds number; and 10^{-5} to $5 \cdot 10^{-2}$ (m) for ϵ .

90% of this dataset was used for training the NN while the others 10% were used to check the accuracy of the model. The model loss is shown below.

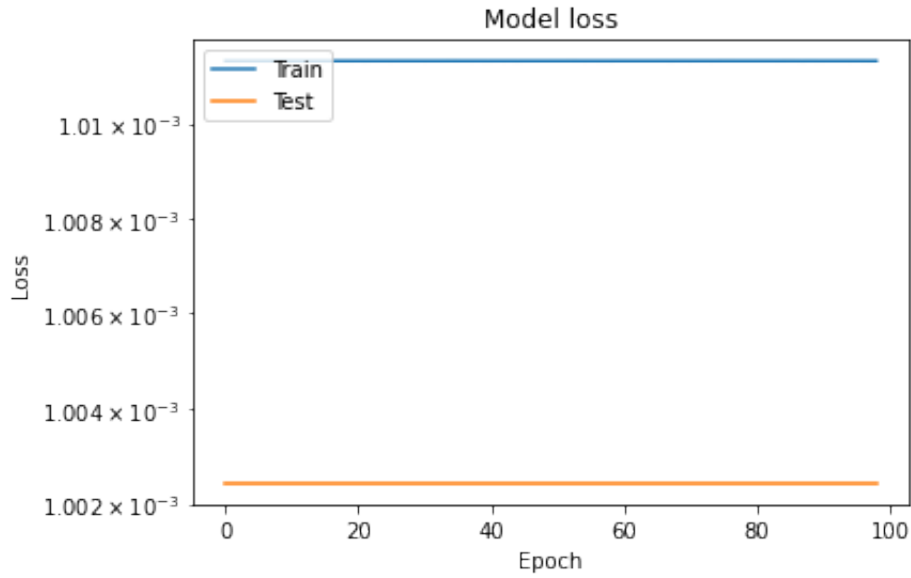


Figure 3.10: Model loss for the training dataset (blue) and the test dataset (orange). Loss got stuck in 0.001.

A fixed loss was quickly obtained. However, this value is not low enough and the graph indicated that it was probably a case of stuck in local-minima. It is suggested that it is due to the difference in magnitude between the Reynolds number and other inputs. To solve that, normalization of the input data [67] was conducted prior to NN optimization. By doing so, the model loss have decreased to a factor of order 10^{-6} .

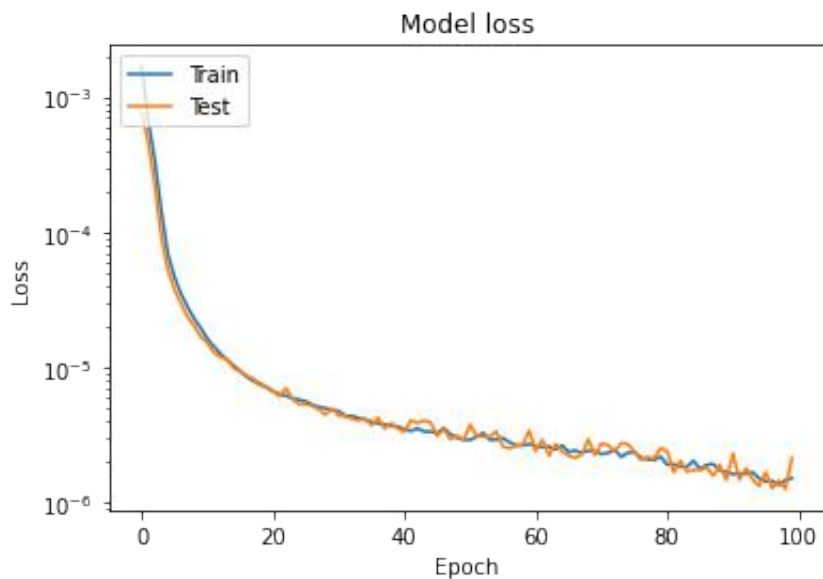


Figure 3.11: New model loss (in log scale) after normalization.

For better visualization, 400 friction values used for testing the NN were plotted alongside with their respective predicted value by the network:

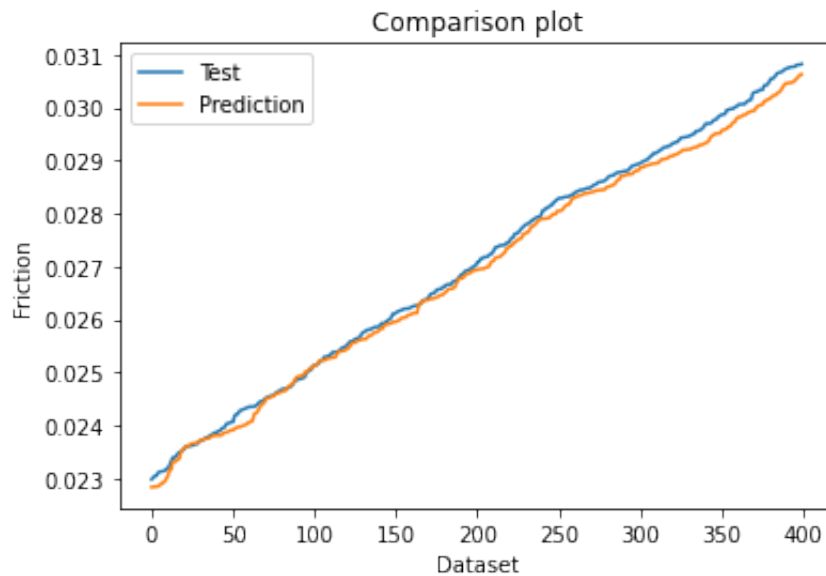


Figure 3.12: Comparison between friction values used for testing and their respective predicted values. Data ordered in crescent order for better visualization.

In author's opinion, it is a good example on how NN can be used for modeling data in different fields. In this case, a very useful fluid mechanics equation to determine the friction factor given few parameters of the flow and the pipe was approximated using a simple yet effective network.

It is worth to mention that such analysis may be extended to more complex fluids, such as non-Newtonian, and for more complex flows, such as multiphase flows. By easily adapting the procedure previously discussed with more adapted empirical or theoretical correlations (see [68] and [69], for instance) and with a suitable network design (e.g., that take also the compressibility factor and the flow composition), one can develop an useful tool with applications in flow assurance (oil and gas), for example.

Chapter 4

Physics-informed neural networks

One can categorize physical problems in three groups [70]:

- Where big data is available, but the governing physical law may not be known;
- On the other extreme, where little data is available (“small data regime”), but the describing physics is known;
- Finally, a third category arises: where the physics is partially known and several scattered measurements are available.

Although purely data-driven approaches have found success in several domains, they might lead to poor generalization performance due to predictions being physically inconsistent or implausible [70]. Moreover, as stated by [71], training a NN to identify a nonlinear map from some potentially very high-dimensional data seems at best “naive” from a computational point of view. Finally, it is obvious that the complete physics of many phenomena are not completely understood and/or does not have an analytical solution so far (e.g. Navier-Stokes equations).

With such limitations, the third scenario proposed - the one that merges mechanistic and NN models - emerges an important tool that is finding applications in many fields, such as fluid-mechanics (to mention [72], [73]), heat transfer (to mention [56]) and even in biophysics (to mention [74]).

But the question that arises is “How to integrate the known physics of a given problem in the process of optimizing the NN model?”. This chapter aims to answer

this, as well as proposes a state-of-the-art of the so-called Physics-informed neural network(s) (PINN).

4.1 Fundamentals of PINN: integrating physics to the model

Physics can be integrated in two principal ways [75]: (i) through the loss function and (ii) in the initialization process.

4.1.1 Loss function guided PINN

One of the most common - and maybe more powerful - technique to make NN consistent with physical knowledge is to incorporate the latter in the loss function of the model.

In [76] the authors where interested in training a NN capable of determining the lake temperature (T) for a given pair of depth values (d) at each timestep (t): $\hat{T}[d, t]$. It is known - and presented in the paper - the relationship between the lake fluid (water) density (ρ) and the temperature; therefore, it is possible to relate both predictions: $\hat{T}[d, t] \rightarrow \hat{\rho}[d, t]$. Furthermore, density increases with depth, so the NN should take into account that for consecutive depth values the density delta should be negative, otherwise it is a physical violation: $\Delta[i, t] = \hat{\rho}[d_i, t] - \hat{\rho}[d_{i+1}, t]$. Finally, the loss function is used alongside with the physics based loss function below:

$$\text{Loss}(\hat{T}) = \frac{1}{n_t(n_d - 1)} \sum_{t=1}^{n_t} \sum_{i=1}^{n_d-1} \text{ReLU}(\Delta[i, t]) \quad (4.1)$$

where n represents the points (depth or time) in the grid. It is important to highlight that the ReLU function was wisely used since it penalizes the network proportionally to delta if its big and greater than 0, while it does not penalizes it (value = 0) when delta is smaller than 0. An extension of this work - and therefore of this approach - has been conducted by [77] and [78].

However, the power of integrating physics knowledge into the loss function is better seen when the - dynamic - system is described by Partial Differential Equations

(PDE). A specific section will be dedicated to this topic.

4.1.2 Loss function guided PINN - solving PDE

This approach was first proposed by Raissi, Perdikaris and Karniadakis [71] and exploits both NN capability as function approximators and automatic differentiation technique. The authors considered differential equation with a general form:

$$u_t + \mathcal{N}[u; \lambda] = 0 \quad (4.2)$$

where $u(t, x)$ denotes the problem solution in time and space domain (which is approximated by a NN), u_k denotes its partial derivative with respect to k , and $\mathcal{N}[:, \lambda]$ is a nonlinear operator parametrized by λ .

Finally, a function f is defined as:

$$f(t, x) := u_t + \mathcal{N}[u; \lambda] \quad (4.3)$$

The parameters were learned by minimizing the mean squared error loss below, that includes not only minimization of $f(t, x)$ ($Error_f$), but also of any initial or boundary conditions ($Error_u$).

$$Error = Error_u + Error_f \quad (4.4)$$

where

$$Error_u = \frac{1}{N_u} \sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2 \quad (4.5)$$

and

$$Error_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2 \quad (4.6)$$

t_u^i , x_u^i and u^i denote the initial and boundary training data, while t_f^i and x_f^i specify the collocations points for $f(t, x)$.

It is evident that for computing $f(t, x)$, one must be capable to differentiate the NN u . This task is accomplished by automatic differentiation (a deep explanation of this algorithm can be found in [79]).

In [71], authors have shown that this framework can work in two types of problems: direct and inverse problems.

4.1.2.1 Direct problems

In direct or forward problems, all the physics of the problem is known. Therefore, NN can be trained using (randomly or not) generated variables without real data concerning the solution.

A schematic representation of a PINN for direct problems is seen in Figure 4.1.

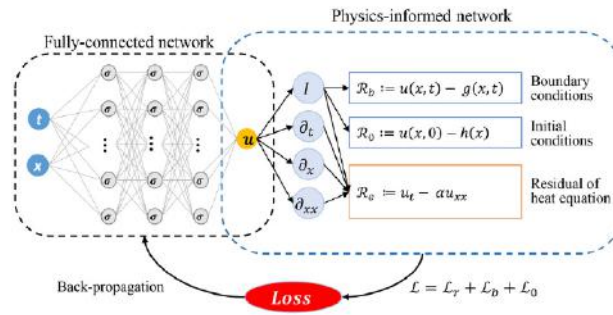


Figure 4.1: Schematic of a PINN framework in representing the equation, initial condition and boundary condition losses in a heat transfer problem. Source: [56].

This has been used in [71] to solve the Burguer's Equation, an equation that arises in fluid mechanics, acoustics, gas dynamics, and even traffic flow; this equations takes the following form: $u_t + \lambda_1 u u_x - \lambda_2 u_{xx} = 0$. Points have been chosen randomly and good solution was obtained. Notice in Figure 4.2 that no point inside the solution domain (represented in colors) was needed for training the network, i.e. no value for the variable $u(t, x)$ has been used other than initial and boundary conditions.

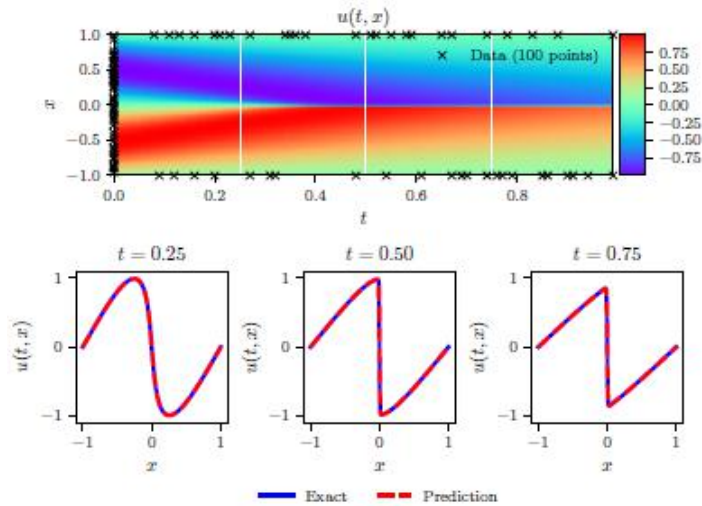


Figure 4.2: Training points (top) and predict and exact solutions for three different time values (bottom). Source: [71].

In [80] the authors have approximate Euler equations for high-speed aerodynamics flows in one and two-dimensions. The authors also highlighted the technique was better in inferring the solution where no discontinuity was presented (smooth). Still, even with discontinuities (due to shock wave, for instance) the PINN was able to good predict the solution when training points were chosen around the discontinuity instead of randomly. They have, therefore, shown how important it is to choose the training points, although this choice and/or the knowledge of the position (or its estimation) of the discontinuity are not always possible/available. Notice in Figure 4.3 how the discontinuity in density is better described when a portion of collocation points is placed in the vicinity of discontinuity when compared to random, even if fewer data points have been used for the former.

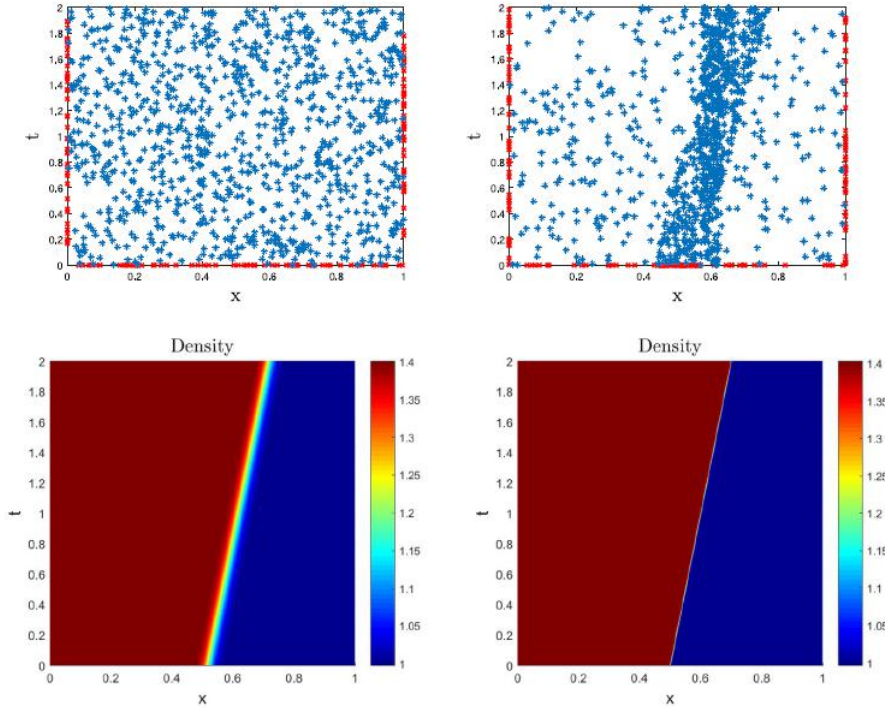


Figure 4.3: Distribution of training points randomly distributed (upper-left) and clustered (upper-right). Below the flow density prediction. Source: [80].

In [81] the authors have formulated a PINN for two different forms of the Navier-Stokes equations: velocity-pressure and velocity-vorticity. Not only they were capable of obtaining accurate values for the outputs velocity and pressure fields (for the first form) and for velocity and vorticity fields (second form) but also explored the potential of transfer learning, i.e. use the previously trained NN as a starting point for training a new one on a much higher Reynolds number. By doing so the solution speed, the computational efficiency and solution accuracy were improved.

Application in both heat transfer and kinetics has been done by Niaki et al. [82]. In their paper, they have simulated the thermochemical evolution of a composite material undergoing cure in an autoclave using a NN to solve two coupled PDE: exothermic heat transfer and resin reaction equations.

Xiang et al. [73] introduced weight variation in the loss function when modeling a PINN for the 3D incompressible Navier-Stokes equations. First they have studied different weights pairs and how they affected the errors in predicting both pressure and velocities. Then, a self-adaptive loss balanced method in order to learn these

parameters simultaneously in the process of optimizing the network. By doing so, higher accuracy was obtained.

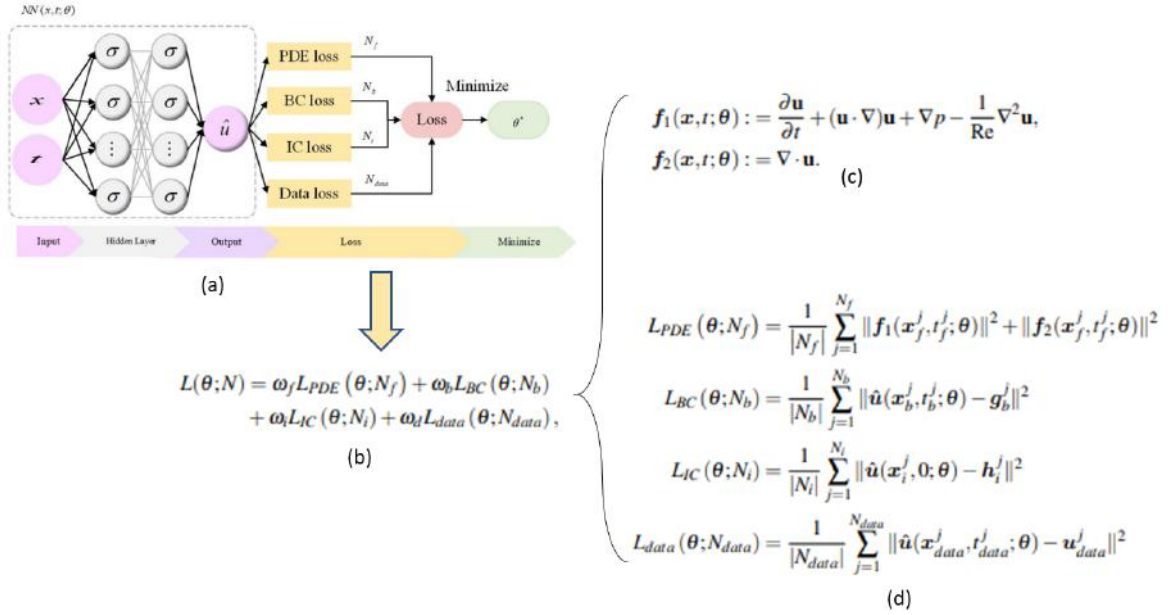


Figure 4.4: (a) Scheme of the PINN, (b) loss function with weights, (c) the PDE equations describing the flow and (d) the terms of the loss function. For better understanding, θ is the networks learning parameters (weights and biases). Source: adapted from [73].

4.1.2.2 Inverse problems

The use of measurements to infer information such as fluid velocity, pressure and stress fields is not a straightforward task [83]. Solving inverse problems using computational fluid dynamics is usually computationally prohibitive [84]. The use of PINN has been demonstrated to work well for solving such problems.

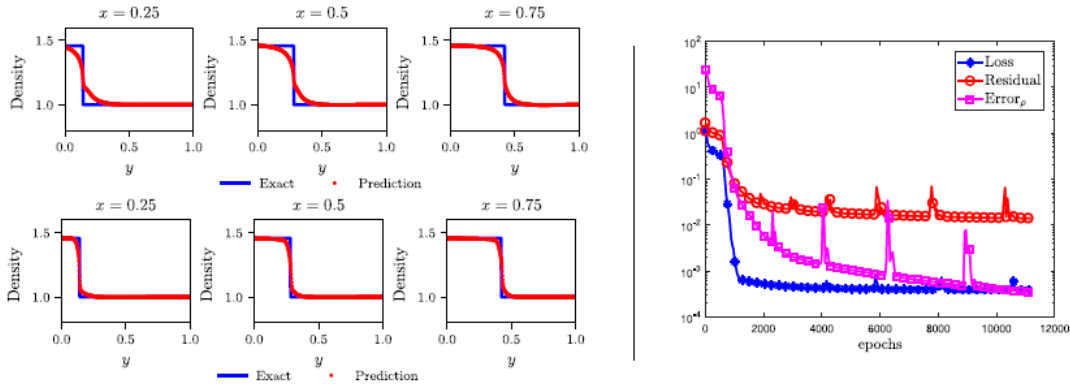


Figure 4.5: Comparison of density with exact solution at various x locations using randomly distributed training points (upper left) and clustered (lower left) alongside with the errors and loss for the clustered problem (right) (“epochs” are steps in the optimization algorithm). Source: [80].

In [81] an “ill-posed” problem was solved using PINN. A 2D flow (Kovasznay flow) without all the boundary conditions was studied in different ill-posed conditions: no upper and bottom boundaries, no lateral boundaries etc. Furthermore, one case where one of the boundaries conditions was noisy (up to 10%) was also studied. For all the cases, 1444 points were used for training and good results were obtained, except when the inlet boundary condition was not informed since it played an important role for the studied problem accordingly to the authors.

In [85], authors have used PINN with the same objective. They studied convective heat transfer around a cylinder, assuming just a few temperature measurements on the solid surface in addition to a few more measurements in the flow wake region; the entire thermal boundary condition on its surface was, therefore, unknown. The model was able to infer the temperature, velocity and pressure fields, and the unknown boundary conditions. Furthermore, the authors also have proposed a method to verify the best configuration for sensor placement.

Jin et al. [81] were also able to identify Reynolds number, which is usually passed as input in direct/forward problems, from 2,000 scattered velocity data. They concluded that the network obtained the entire flow fields with high accuracy as well as learned with success the unknown Reynolds number.

This has been used in [71] to determine the parameters of Burgers' Equation, an equation that arises in fluid mechanics, acoustics, gas dynamics, and even traffic flow; this equation takes the following form: $u_t + \lambda_1 u u_x - \lambda_2 u_{xx} = 0$. 2000 points have been generated using $(\lambda_1, \lambda_2) = (1.0, 0.01/\pi)$. These two parameters have been accurately identified (errors as little as 4%) even when noise levels up to 10% have been added to training data. Same approach has been used for the 2D Navier-Stokes equations, and also good results have been obtained.

Yin et al. [86] have employed PINN to infer properties of biological materials (permeability and viscoelastic modulus) from thrombus deformation data. Encoding both Cahn-Hilliard and Navier-Stokes equations into the loss function, the authors, were able to estimate these parameters over a wide range (from 10^{-4} to 10^4), with good matching with state-of-the-art simulation results. This approach is obviously useful in biochemistry as well. In [87], Yazdani et al. inferred unknown parameters for three biochemical models: yeast glycolysis, cell apoptosis and ultradian endocrine.

Mao, Jagtap and Karniadakis [80] evoked that for the equation of state used in the paper, the adiabatic index (γ) for polytropic gas was assumed known. Nevertheless, as they affirmed, its value varies depending on the type of gas and, therefore, it would be interesting to also learn the parameter γ along with the NN hyper-parameters (weights and biases). They have obtained good accuracy using clustered training points (they were dealing with discontinuity, and clustered means more training points around it) for clean data, and data with 1 or 2% noise.

When integrating experimental technique such as particle image velocimetry measurement of concentration field (using a passive scalar, e.g. smoke and dye, to study flow), inverse problems become even more interesting. From such data, very complex phenomena can be reconstructed.

In [83] used several snapshots of concentration field of a flow. By integrating this data and minimizing the loss error, they were able to obtain velocity and pressure field. A schema of this inverse problem based on image data is provided in Figure 4.6. Notice how the equation modeling dye advection by a given velocity field and

subject to molecular diffusion is also introduced as part of the loss function.

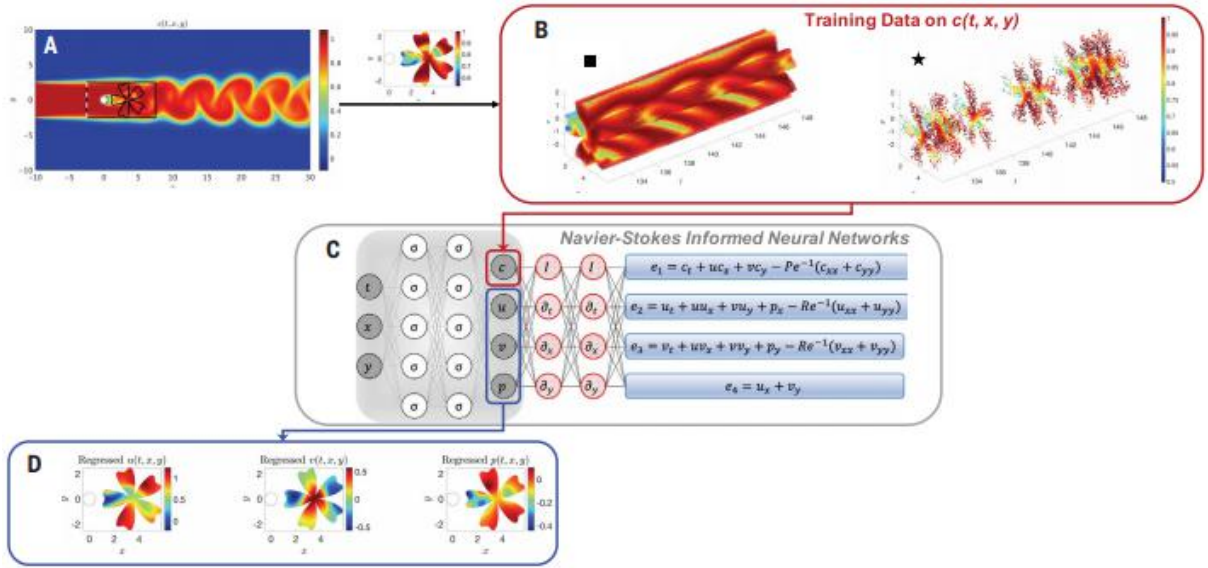


Figure 4.6: Schema of the PINN proposed. Source: [83].

Tomographic background oriented schlieren imaging measures the temperature or density fields in 3D using special cameras and can be used for instant flow visualization. In [88], Cai et al. were able to infer velocity and pressure fields over an espresso cup (Figure 4.7) by just providing the temperature field obtained by the tomographic image technique. The errors were computed using a set of approximate Navier-Stokes equations coupled with the heat transfer equation all in the 3D domain. Authors also have studied the effect of down-sampling the data in time and in space, i.e. reducing the temporal and spacial resolution (increasing sparsity in the training data). Finally, they concluded that the PINN was capable of inferring the fields without any information of the initial boundary conditions even if experimental data is sparse and limited.

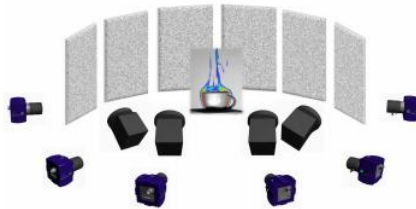


Figure 4.7: Imaging set to obtain temperature field around an espresso cup. Source: [88].

Chen et al. [89] used inverse problem solving using PINN in the field of nano-optics and metamaterials. Based on scattered data (e.g. from scanning near-field optical microscopy), authors could retrieve the permittivity distribution of materials. By doing so, a new era of designing novel functional photonic material structures arises.

To demonstrate the applications in hemodynamics, Raissi, Yazdani and Karniadakis [90] proposed a model for studying of intracranial aneurysm from scatter data (e.g. from angiography). The authors could reconstruct both velocity and pressure fields that could be further used to estimate other quantities such as shear stress.

4.1.3 Guided initialization PINN

Initializing a NN can play an important role. The main approach is to randomly initialize the weights, but it is known that poor initialization can cause models to stuck in local minima [75].

One technique to avoid that is Transfer learning, already introduced in the past section. From a base model (that can be much simpler than the real physics), one can generate training data used as input for the naive NN. Then, this network can be fine-tuned with real data.

In [77] and [91], once again in the context of lake temperature modeling, the authors have used such approach. It has been shown that, by doing so, the required training data for fine-tuning has been drastically reduced even with poor physics models, i.e. with incorrect set of parameters.

4.2 Additional thoughts

It is clear that combining physics into the NN model have huge potential for providing better prediction accuracy with smaller number of samples and better generalization, i.e. good performance with out-of-sample scenarios [75], solving differential set of equations and even discover unknown information concerning the

problem (e.g. parameters). However, after reading diverse papers in the topics, it was made evident that data generation is other important application. Another interesting application is in process control.

4.2.1 Data generation

Finally, integrating physics in the NN model can improve data generation quality. In [92], Cang et al. were interest in the generation of solid microstructures, a topic with very potential for prediction of material properties. Nevertheless, obtaining such material samples experimentally or computationally is costly. Therefore, authors have proposed a generative network modified so the error function for training was modified to enforce that the generate data would have the same morphology distribution as the authentic ones.

Statistics of training data has also been incorporated in the loss function for better data generation in [93]. Authors have quantified the difference between the covariance structures of training and generated data, then incorporating it into the original loss function as a penalty term. Up to 80% in training cost was achieved to reach solution with good quality.

2D and 3D fluid simulations have been synthetized for the first through a convolutional NN [94] from a set of reduced parameters. In their model, authors have inputted a divergence-free term into the loss function to ensure mass conservation. Their approach is up to 700 times faster than state-of-the-art solvers.

Chapter 5

Practical application of PINN

In this chapter, a real physics problem will be considered. For solving it, a PINN loss function guided (as discussed in section 4.1.2) will be modeled due to its great capacity of solving PDE. By doing so, this work aims to demonstrate the practical application of this technique specially in chemical engineering.

5.1 Addressed problem: 1D diffusion equation

In many problems - specially when one dimension is clearly more important than the other 2 - the 1D diffusion equation can describe the concentration ¹ of a given molecule in a stationary bulk (e.g. gas permeation trough a membrane). It has been chosen for being a very common and important mass transfer problem in chemical engineering. Assuming a source function $f(x, t)$ and constant diffusion coefficient D , it takes the following form [95]:

$$D.C_{xx} - C_t - f = 0 \quad (5.1)$$

$C(x,t)$ denotes the concentration - where $x \in [-1, 1]$ and $t \in [0, 1]$ - and D the diffusion coefficient.

Considering $D = 1.0$ (unities have been omitted), the Dirichlet boundary conditions $C(-1, t) = C(1, t) = 0$, the initial condition $C(x, 0) = \sin(\pi x)$, and $f(x, t) = e^{-t}(\sin(\pi x) - \pi^2 \sin(\pi x))$, the solution for $C(x,t)$ takes the following form:

¹The same equation can also describe other phenomena, such as heat diffusion.

$$C(x, t) = e^{-t} \sin(\pi x) \quad (5.2)$$

The solution was then obtained from a PINN using the DeepXDE library [96].

5.2 Solving the 1D heat equation with NN - forward problem

For the forward problem, a neural network containing 4 hidden layers with 25 neurons each, 2 input neurons (x and t) (architecture chosen at random) and one output neuron ($C(x, t)$). The hyperbolic tangent was used as activation function.

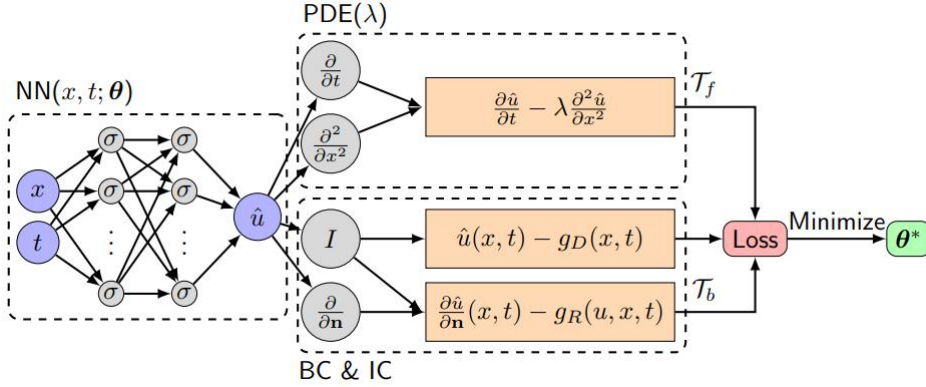


Figure 5.1: Schema of the PINN used for solving the forward 1D diffusion problem. Source: [96].

As a first approach, only boundary and initial condition data has been informed for training the network. A total of 20 boundary values (pairs $[-1.0$ or $1.0, t]$, with t randomly distributed) and 20 initial conditions (pairs $[x, 0]$, with x equally spaced from -1.0 to 1.0) were used. These points are shown in red in Figure 5.3.

The losses are plotted in Figure 5.2. Below, the solution surface for $C(x, t)$ predicted by the PINN alongside with its error (module of predicted - expected values).

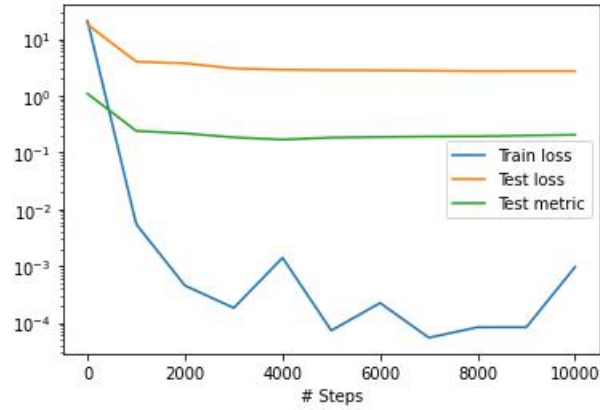


Figure 5.2: Losses for the model trained using only boundary and initial conditions of the 1D diffusion transfer problem.

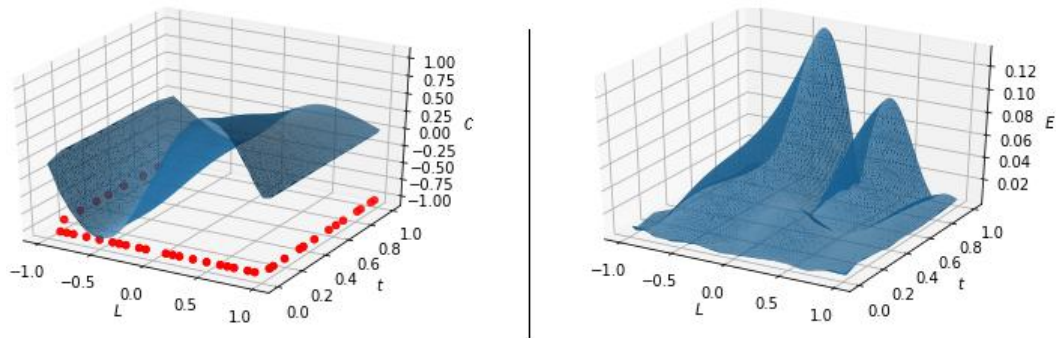


Figure 5.3: Predicted surface for $C(x, t)$ with training points in red (left) and the module error (right).

Notice that in Figure 5.3 the error is of the order 10^{-1} and is lower in the vicinity of the borders ($L = \pm 1.0$ or $t = 0.0$), exactly where the collocation points were placed. In order to improve this results, two modifications were made: (case i) additional 20 internal points to simulate experimental data were used and (case ii) increasing to 30 internal points, boundary conditions and initial conditions (90 points) and modification of the architecture to 6 hidden layers with 30 neurons each.

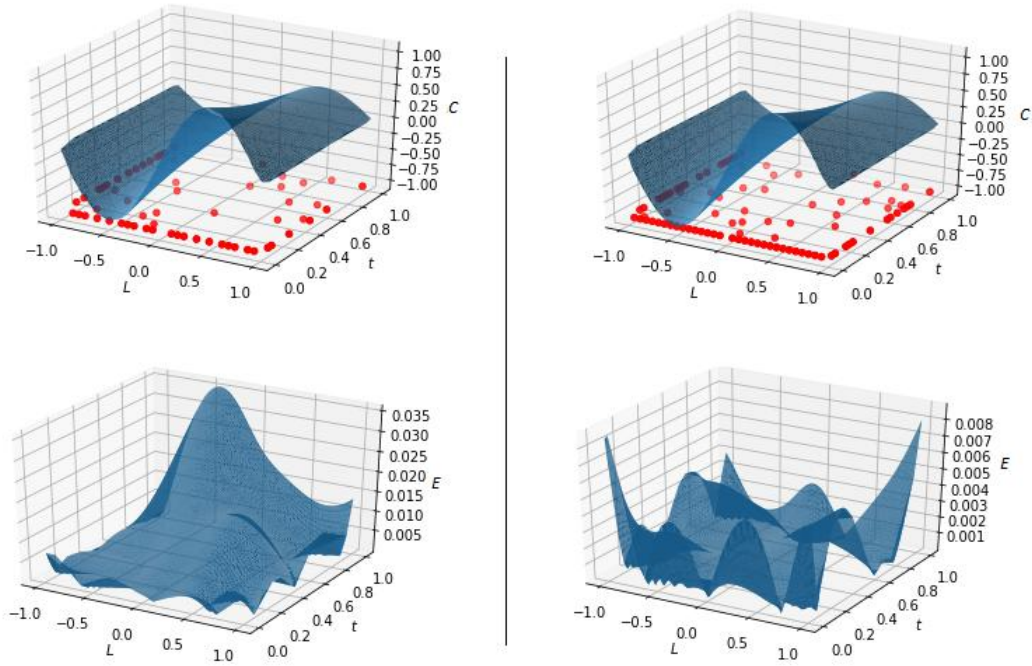


Figure 5.4: Predicted surface for $C(x, t)$ with training points in red (upper) and the module error surface (bottom) for case i (left) and case ii (right).

Interesting to notice that case ii, even if the error surface looks now less “smooth”, it was capable to reduce even more the module error.

5.3 Solving the 1D heat equation with NN - inverse problem

For the inverse problem, diffusion coefficient D (real value = 1.0) was assumed unknown and an inner neuron was dedicated to learn this parameter. An adaptation of the Figure 5.1 is proposed below.

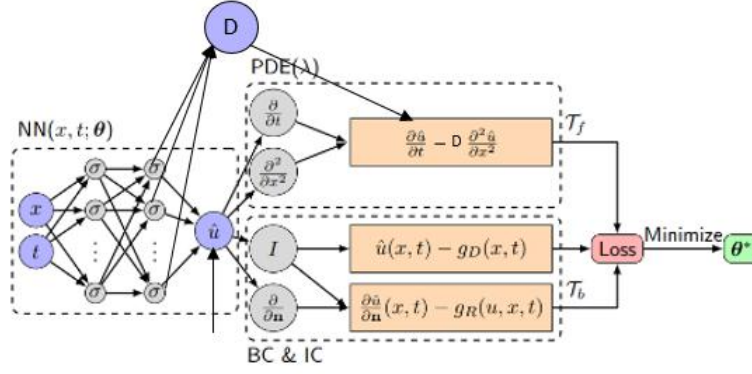


Figure 5.5: Schema of the PINN used for solving the reverse 1D diffusion problem. Source: adapted from [96].

Based on the good results obtained for the previous case ii of the direct problem, the networks for the inverse problem was designed with 6 hidden layers with 30 neurons each.

The diffusion coefficient was initialized with a value 5.0. Nevertheless, after about 95 seconds of training ², a 3.04 value was obtained. In order to improve this result, the number of data containing the real solution of the PDE was increased to 50; surprisingly it did not reduce the coefficient value obtained, but increase it a little. Therefore, a simple parametric study was conducted where number of domain (solution) data and the number of layers in the network were varied. Number of boundary and initial condition points where kept constant at 30 points each.

Table 5.1: Table with the number of domain data (real solutions), configuration of the hidden layer and the obtained D coefficient.

Number of domain data—	Network (Neurons x hidden layers)	Coefficient D obtained
30	30 x 6	3.04
50	30 x 6	3.27
75	30 x 6	2.74
30	30 x 8	3.19
30	30 x 3	1.03
50	30 x 3	1.00

²In an IntelCore2 Quad CPU Q8400 @ 2.66GHz 2.67GHz machine

The best coefficient result was obtained when 3 hidden layers and 50 training data inside the function domain were used. It has not yet been completely understood why, but the author believes that by reducing the number of layers (and therefore the number of neuron), the network had less parameters to train, so in a way the “extra” domain data informed was better used for training the intern parameter D .

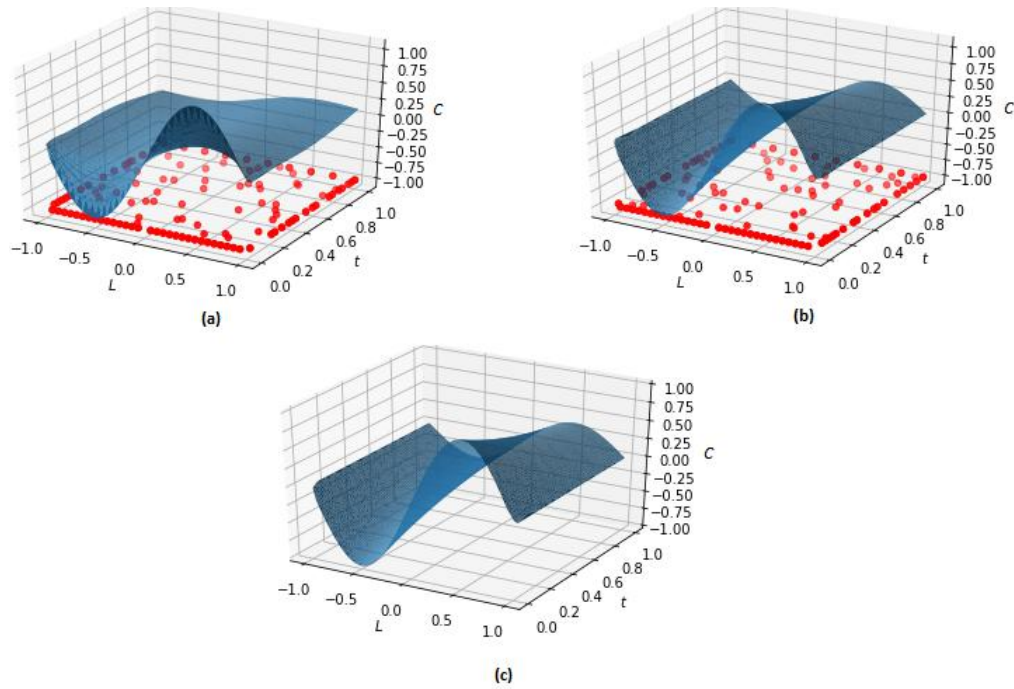


Figure 5.6: (a) Surface predicted solution for the case where a 3.19 D was obtained, (b) surface for predicted $D = 1.00$ and (c) real solution. In red the points used for training .

Chapter 6

Conclusion

6.1 Conclusion

This work has demonstrated several applications of machine learning techniques in engineering, more specifically in chemical engineering. The universal approximation capacity of neural networks, the most prominent example of machine learning, has been highlighted with a practical example. 10,000 friction values were obtained using the Colebrook-White equation and were applied as labels for training and testing the designed neural network, which received 3 inputs: Re , f and ϵ . Predicted results were in good accordance with true values.

The reader was sensitized to the fact that pure data-driven approaches could be prone to failure and/or not enough data is available for training the complex networks required for complex engineering problems. Recent works have demonstrated that to explore prior physical knowledge is a powerful way to improve neural networks' performance: that is the basis for the Physics-informed neural networks. Two approaches are possible: in the initialization process and by embedding the physics into the loss function. The latter is of particular interest for solving partial differential equations (forward problems), determining unknown parameters (inverse problem) and determining information from image-like data (also an inverse problem). Such techniques are of particular interest in fluid mechanics and heat transfer problems, but examples in areas such as biophysics are also arising.

The PINN framework used in this work was capable to solve the 1D heat diffusion equation coupled with a source term. Both forward and inverse problems were studied. In the inverse scenario, increasing the number of data inside the function domain without necessarily increasing the network complexity has demonstrated to be very efficient in determining the unknown diffusion coefficient.

Finally, this work may serve as a reference inside the School of Chemistry (EQ/UFRJ) for future bachelor thesis and for a necessary - from the author's point of view - modernization of Chemical Engineering's bachelor program by merging the machine learning concepts with the well established physical models.

6.2 Further researches and works

For further researches and works the author suggests solving other more complex examples in chemical engineering using neural networks (e.g., calculating the friction factor for a multiphase flow) and using real industrial data. Since real data comes usually with noise, it would be interesting to evaluate the effects on the final results and how to overcome this problematic.

Concerning PINN, a specific study concerning the uses of this technique for inverse imagery problems is suggested. Real flow images/videos (e.g., collected using hyperspectral cameras) could be used for real examples or application as well.

6.3 Suggestions

It is suggested some YouTube channels, specially for better visualization of the concepts introduced in this work:

- Steve Brunton: hot topics in machine learning are presented and discussed by Steve Brunton [↗](#)
- Raissi: channel with videos explaining the main concepts around NN [↗](#)
- Two Minute Papers: channel with interesting applications of AI [↗](#)

- PINN: presentations of different researchers around physics-informed neural networks new developments [↗](#)
- 3Blue1Brown: explanation of important concepts in mathematics, programming and physics through animations [↗](#)

Some Python libraries are also suggested:

- SciPy: offers modules for linear algebra, interpolation, special functions, Fourier transforms, image optimization, ODE solving etc [↗](#)
- Scikit-learn: library for supervised and unsupervised learning, as well as data preprocessing [↗](#)
- Tensorflow: Google's framework for machine learning models [↗](#)
- Keras: standard library for neural networks [↗](#)
- DeepXDE: library for scientific machine learning, including PINN [↗](#)

Appendix A

Modeling a NN to predict friction factor

```
#Data generator
import numpy
from scipy.optimize import root
from random import uniform

def f(f):
    return (-2*numpy.log10((2.51/(Re*numpy.sqrt(f))) + (e/(3.71*d))) -
            1.0/numpy.sqrt(f))

X, y = [], []
for i in range(10000):
    d = uniform(1., 10.)
    Re = uniform(2500., 10000000.)
    e = uniform(0.00001, 0.05)
    friction = root(f, 0.01).x
    X += [[d, Re, e]]
    y += [friction[0]]

X = numpy.array(X)
y = numpy.array(y)

#Divide danta into training and testing set
#Uncomment the code below to scale the data datased
#from sklearn.preprocessing import StandardScaler
```

```

#sc = StandardScaler()
#X = sc.fit_transform(X)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1)

#Designing a NN in Keras
import keras
from keras.models import Sequential
from keras.layers import Dense

# Neural network
model = Sequential()
model.add(Dense(50, input_shape=(3,), activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='relu'))
sgd = keras.optimizers.RMSprop(learning_rate=0.0001)
model.compile(loss=keras.losses.mean_squared_error,
              optimizer='adam')
epochs = 100
batch_size = 128

# Fit the model weights.
history = model.fit(X_train, y_train,
                   batch_size=batch_size,
                   epochs=epochs,
                   verbose=1,
                   validation_data=(X_test, y_test))

#Plotting Loss
from matplotlib import pyplot as plt

plt.semilogy(history.history['loss'])
plt.semilogy(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')

```

```
plt.legend(['Train', 'Test'], loc='upper_left')
plt.show()

#Visualize 50 testing data and their respective model prediction
y_model = model.predict(X_test)
plt.semilogy(y_test[:50])
plt.semilogy(y_model[:50])
plt.title('Comparison_plot')
plt.ylabel('Friction')
plt.xlabel('X_test')
plt.legend(['Test', 'Prediction'], loc='upper_left')
plt.show()
```

Appendix B

Solving the 1D heat equation using PINN in Python

```
import deepxde as dde
import numpy as np
from deepxde.backend import tf

def PDE_func(x, y):
    dy_t = dde.grad.jacobian(y, x, j=1)
    dy_xx = dde.grad.hessian(y, x, j=0)
    return (
        dy_t
        - dy_xx
        + tf.exp(-x[:, 1:])
        * (tf.sin(np.pi * x[:, 0:1]) - np.pi ** 2 * tf.sin(np.pi * x[:,
            0:1])))
    )

def PDE_sol(x):
    return np.sin(np.pi * x[:, 0:1]) * np.exp(-x[:, 1:])

geom = dde.geometry.Interval(-1, 1)
timedomain = dde.geometry.TimeDomain(0, 1)
geomtime = dde.geometry.GeometryXTime(geom, timedomain)
```



```

boundary_cond = dde.DirichletBC(geomtime, PDE_sol, lambda _,
    on_boundary: on_boundary)
initial_cond = dde.IC(geomtime, PDE_sol, lambda _, on_initial:
    on_initial)
data = dde.data.TimePDE(
    geomtime,
    PDE_func,
    [boundary_cond, initial_cond],
    num_domain=30, #number of data inside the domain for training
    num_boundary=30,
    num_initial=30,
    solution=PDE_sol,
    num_test=10000,
)

layer_size = [2] + [30] * 6 + [1]
activation = "tanh"
initializer = "Glorot_uniform"
net = dde.maps.FNN(layer_size, activation, initializer)

model = dde.Model(data, net)

model.compile("adam", lr=0.001, metrics=["l2_relative_error"])
losshistory, train_state = model.train(epochs=10000)

dde.saveplot(losshistory, train_state, issave=True, isplot=True)

###Plot the data (the "saveplot" attribute return a low quality plot)
###
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from IPython.core.display import Math
import random
import math

X_train, y_train, X_test, y_test, best_y, best_ystd = train_state.
    packed_data()
y_dim = best_y.shape[1]

```

```

X, t, Y = [], [], []
for i in best_y[:, 0]:
    Y.append(i)
for i in X_test[:, 0]:
    X.append(i)
for i in X_test[:, 1]:
    t.append(i)

x, y = [], []
for i in data.train_x[:,0]:
    x.append(i)
for i in data.train_x[:,1]:
    y.append(i)
z = [-1]*len(x)

plt.figure()
ax = plt.axes(projection=Axes3D.name)
ax.set_xticks([-1., -0.5, 0, 0.5, 1.])
ax.plot_trisurf(X, t, Y)
ax.scatter3D(x, y, z, c = 'red')
ax.set_xlabel("$L$")
ax.set_ylabel("$t$")
ax.set_zlabel("$C$")

y_r = []

for (i,j) in zip(X,t):
    y_r.append(math.sin(math.pi * i) * math.exp(-j))

diff = []
for (i,j) in zip(y_r, Y):
    diff.append(((i-j)**2)**0.5)

plt.figure()
ax = plt.axes(projection=Axes3D.name)
ax.set_xticks([-1., -0.5, 0, 0.5, 1.])
ax.plot_trisurf(X, t, diff)

```

```

ax.set_xlabel("$L$")
ax.set_ylabel("$t$")
ax.set_zlabel("$E$")

#####

###Inverse problem###

import deepxde as dde
import numpy as np
from deepxde.backend import tf

D = tf.Variable(5.0)

def PDE_func(x, y):
    dy_t = dde.grad.jacobian(y, x, i=0, j=1)
    dy_xx = dde.grad.hessian(y, x, i=0, j=0)
    return (
        dy_t
        - D * dy_xx
        + tf.exp(-x[:, 1:])
        * (tf.sin(np.pi * x[:, 0:1]) - np.pi ** 2 * tf.sin(np.pi * x[:,
            0:1]))
    )

def PDE_sol(x):
    return np.sin(np.pi * x[:, 0:1]) * np.exp(-x[:, 1:])

geom = dde.geometry.Interval(-1, 1)
timedomain = dde.geometry.TimeDomain(0, 1)
geomtime = dde.geometry.GeometryXTime(geom, timedomain)

bc = dde.DirichletBC(geomtime, PDE_sol, lambda _, on_boundary:
    on_boundary)

```

```

ic = dde.IC(geomtime, PDE_sol, lambda _, on_initial: on_initial)

observe_x = np.vstack((np.linspace(-1, 1, num=10), np.full((10), 1))).T
observe_y = dde.PointSetBC(observe_x, PDE_sol(observe_x), component=0)

data = dde.data.TimePDE(
    geomtime,
    PDE_func,
    [bc, ic, observe_y],
    num_domain=50,
    num_boundary=30,
    num_initial=30,
    anchors=observe_x,
    solution=PDE_sol,
    num_test=10000,
)

layer_size = [2] + [30] * 6 + [1]
activation = "tanh"
initializer = "Glorot_uniform"
net = dde.maps.FNN(layer_size, activation, initializer)

model = dde.Model(data, net)

model.compile(
    "adam", lr=0.001, metrics=["l2_relative_error"],
    external_trainable_variables=D)
variable = dde.callbacks.VariableValue(D, period=1000)
losshistory, train_state = model.train(epochs=50000, callbacks=[
    variable])

dde.saveplot(losshistory, train_state, issave=True, isplot=True)

from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from IPython.core.display import Math
import random
import math

```

```

X_train, y_train, X_test, y_test, best_y, best_ystd = train_state.
    packed_data()
y_dim = best_y.shape[1]

X, t, Y = [], [], []
for i in best_y[:, 0]:
    Y.append(i)
for i in X_test[:, 0]:
    X.append(i)
for i in X_test[:, 1]:
    t.append(i)

x, y = [], []
for i in data.train_x[:,0]:
    x.append(i)
for i in data.train_x[:,1]:
    y.append(i)
z = [-1]*len(x)

plt.figure()
ax = plt.axes(projection=Axes3D.name)
ax.set_xticks([-1., -0.5, 0, 0.5, 1.])
ax.plot_trisurf(X, t, Y)
ax.scatter3D(x, y, z, c = 'red')
ax.set_xlabel("$L$")
ax.set_ylabel("$t$")
ax.set_zlabel("$C$")

y_r = []
for (i,j) in zip(X,t):
    y_r.append(math.sin(math.pi * i) * math.exp(-j))

diff = []
for (i,j) in zip(y_r, Y):
    diff.append(((i-j)**2)**0.5)

plt.figure()
ax = plt.axes(projection=Axes3D.name)

```

```
ax.set_xticks([-1., -0.5, 0, 0.5, 1.])
ax.plot_trisurf(X, t, y_r)
ax.set_xlabel("$L$")
ax.set_ylabel("$t$")
ax.set_zlabel("$E$")
```

Bibliography

- [1] Osvaldo Simeone. A Very Brief Introduction to Machine Learning with Applications to Communication Systems. *IEEE Transactions on Cognitive Communications and Networking*, 4(4):648–664, 2018.
- [2] Guillaume Carlier Mqef. Mathematics for. *Quantitative literacy: Why numeracy matters for schools*, (c):533–540, 2009.
- [3] Steven L. Brunton, Bernd R. Noack, and Petros Koumoutsakos. Machine Learning for Fluid Mechanics. *Annual Review of Fluid Mechanics*, 52:477–508, 2020.
- [4] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*, volume 9781107057. 2013.
- [5] Andreas C. Müller and Sarah Guido. *Introduction to with Python Learning Machine*. 2017.
- [6] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.
- [7] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*, volume 0. The MIT Press, Cambridge, Massachusetts, 2012.
- [8] Hao Chen, Chao Zhang, Ninghong Jia, Ian Duncan, Shenglai Yang, and Yong Zhi Yang. A machine learning model for predicting the minimum miscibility pressure of CO₂ and crude oil system based on a support vector machine algorithm approach. *Fuel*, 290(September 2020):120048, 2021.
- [9] Alejandra Urtubia, Roberto León, and Matías Vargas. Identification of chemical markers to detect abnormal wine fermentation using support vector machines. *Computers and Chemical Engineering*, 145:107158, 2021.

- [10] Shen Yin, Xin Gao, Hamid Reza Karimi, and Xiangping Zhu. Study on support vector machine-based fault detection in Tennessee Eastman process. *Abstract and Applied Analysis*, 2014, 2014.
- [11] Hyunseung Kim, Addis Lulu Gebreselassie, Seungkyu Dan, and Dongil Shin. Random forest classifier for real-time chemical leak source tracking using fence-monitoring sensors. *Korean Journal of Chemical Engineering*, 35(6):1231–1239, 2018.
- [12] David S. Palmer, Noel M. O’Boyle, Robert C. Glen, and John B.O. Mitchell. Random forest models to predict aqueous solubility. *Journal of Chemical Information and Modeling*, 47(1):150–158, 2007.
- [13] Muhammad Usama, Junaid Qadir, Kok Lim, Alvin Yau, Amir Hussain, Aunn Raza, Hunain Arif, Yehia Elkhatab, and Ala Al-Fuqaha. Unsupervised Machine Learning for Networking: Techniques, Applications and Research Challenges. *IEEE Access*, 7:65579–65615, 2017.
- [14] Julien Weiss. A tutorial on the proper orthogonal decomposition. *AIAA Aviation 2019 Forum*, (June):1–21, 2019.
- [15] Saïd Ladjal, Alasdair Newson, and Chi-Hieu Pham. a Pca-Like Autoencoder a Preprint. 2019.
- [16] N. Akkari, A. Hamdouni, E. Liberge, and M. Jazar. A mathematical and numerical study of the sensitivity of a reduced order model by POD (ROM-POD), for a 2D incompressible fluid flow. *Journal of Computational and Applied Mathematics*, 270:522–530, 2014.
- [17] Sin Yong Teng, Vítězslav Máša, Petr Stehlík, and Hon Loong Lam. Deep learning approach for industrial process improvement. *Chemical Engineering Transactions*, 76(2016):487–492, 2019.
- [18] Lionel Agostini. Exploration and prediction of fluid dynamical systems using auto-encoder technology. *Physics of Fluids*, 32(6), 2020.

- [19] Rodrigo Scoralick Fontoura do Nascimento, Bruno Henrique Groenner, Ricardo Emanuel Vaz Vargas, and Ismael Humberto Ferreira dos Santos. Detecção de falhas com Stacked Autoencoders e técnicas de reconhecimento de padrões em poços de petróleo operados por gas lift. 2020.
- [20] Aristidis Likas, Nikos Vlassis, and Jakob J. Verbeek. The global k-means clustering algorithm. *Pattern Recognition*, 36(2):451–461, 2003.
- [21] Mr. Dibya Jyoti Bora and Dr. Anil Kumar Gupta. Effect of Different Distance Measures on the Performance of K-Means Algorithm: An Experimental Study in Matlab. 5(2):2501–2506, 2014.
- [22] Weifang Shi and Weihua Zeng. Application of k-means clustering to environmental risk zoning of the chemical industrial area. *Frontiers of Environmental Science and Engineering*, 8(1):117–127, 2014.
- [23] Jesper E. van Engelen and Holger H. Hoos. A survey on semi-supervised learning. *Machine Learning*, 109(2):373–440, 2020.
- [24] Xiangli Yang, Zixing Song, Irwin King, and Zenglin Xu. A Survey on Deep Semi-supervised Learning. pages 1–24, 2021.
- [25] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. *Semi-Supervised Learning*. MIT Press.
- [26] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.
- [27] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. Generative Adversarial Networks: An Overview. *IEEE Signal Processing Magazine*, 35(1):53–65, 2018.
- [28] Yabo Dan, Yong Zhao, Xiang Li, Shaobo Li, Ming Hu, and Jianjun Hu. Generative adversarial networks (GAN) based efficient sampling of chemical composition space for inverse design of inorganic materials. *npj Computational Materials*, 6(1):1–7, 2020.

- [29] Jianqin Zheng, Yongtu Liang, Ning Xu, Bohong Wang, Taicheng Zheng, Zhengbing Li, Qi Liao, and Haoran Zhang. Deeppipe: a customized generative model for estimations of liquid pipeline leakage parameters. *Computers and Chemical Engineering*, 149:107290, 2021.
- [30] Carlos Henrique Costa Ribeiro. A Tutorial on Reinforcement Learning Techniques. *Supervised Learning track tutorials of the 1999 International Joint Conference on Neural Networks*, pages 1–44, 1999.
- [31] Andrew G. Barto, Richard S. Sutton. Reinforcement Learning: An Introduction. *Decision Theory Models for Applications in Artificial Intelligence: Concepts and Solutions*, pages 63–80, 2011.
- [32] S. P.K. Spielberg, R. B. Gopaluni, and P. D. Loewen. Deep reinforcement learning approaches for process control. *2017 6th International Symposium on Advanced Control of Industrial Processes, AdCONIP 2017*, (1):201–206, 2017.
- [33] Steven Spielberg, Aditya Tulsyan, Nathan P. Lawrence, Philip D Loewen, and R. Bhushan Gopaluni. Deep Reinforcement Learning for Process Control: A Primer for Beginners. (Mc), 2020.
- [34] Jean Rabault, Miroslav Kuchta, Atle Jensen, Ulysse Réglade, and Nicolas Cerardi. Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control. *Journal of Fluid Mechanics*, 865:281–302, 2019.
- [35] Jean Rabault and Alexander Kuhnle. Accelerating deep reinforcement learning strategies of flow control through a multi-environment approach. *Physics of Fluids*, 31(9), 2019.
- [36] Yuan Wang, Kirubakaran Velswamy, and Biao Huang. A Novel Approach to Feedback Control with Deep Reinforcement Learning. *IFAC-PapersOnLine*, 51(18):31–36, 2018.
- [37] Kevin L. Priddy and Paul E. Keller. *Artificial Neural Networks: An Introduction*. 2009.

- [38] Frank Rosenblatt. Perceptron Simulation Experiments. *Proceedings of the IRE*, pages 301–309, 1960.
- [39] Warren S Mcculloch and Walter Pitts. A logical calculus nervous activity. *Bulletin of Mathematical Biology*, 52(1):99–115, 1943.
- [40] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1969.
- [41] Kritika Verma and Pradeep Kumar Singh. An Insight to Soft Computing based Defect Prediction Techniques in Software. *International Journal of Modern Education and Computer Science*, 7(9):52–58, 2015.
- [42] James Lighthill. Part I Arti cial Intelligence A general survey by Sir James Lighthill FRS Lucasian Professor of Applied Mathematics. (July):1–22, 1972.
- [43] Michael Nielsen. *Neural Networks and Deep Learning*. 2021.
- [44] C. Aggarwal Charu. *Neural Networks and Deep Learning: a Textbook*. Springer International Publishing AG.
- [45] J. Nathan Brunton, Steven L., Kutz. *Data-driven science and engineering: machine learning, dynamical systems, and control*, volume 60. Cambridge University Press, 2019.
- [46] J. Sola and J. Sevilla. Importance of input data normalization for the application of neural networks to complex industrial problems. *IEEE Transactions on Nuclear Science*, 44(3 PART 3):1464–1468, 1997.
- [47] Fabio Machado Cavalcanti, Camila Emilia Kozonoe, Kelvin André Pacheco, and Rita Maria de Brito Alves. Application of Artificial Neural Networks to Chemical and Process Engineering. *Artificial Neural Networks and Deep Learning - Applications and Perspective [Working Title]*, pages 1–18, 2021.
- [48] Hao Li, Zhien Zhang, and Zhijian Liu. Application of artificial neural networks for catalysis: A review. *Catalysts*, 7(10), 2017.

- [49] Jonah P. Poort, Mahinder Ramdin, Jan van Kranendonk, and Thijs J.H. Vlugt. Solving vapor-liquid flash problems using artificial neural networks. *Fluid Phase Equilibria*, 490:39–47, 2019.
- [50] Rita Maria Brito Alves, Frank H. Quina, and Claudio Augusto Oller Nascimento. New approach for the prediction of azeotropy in binary systems. *Computers and Chemical Engineering*, 27(12):1755–1759, 2003.
- [51] L. V. Kamble, D. R. Pangavhane, and T. P. Singh. Artificial Neural Network Based Prediction of Heat Transfer from Horizontal Tube Bundles Immersed in Gas-Solid Fluidized Bed of Large Particles. *Journal of Heat Transfer*, 137(1):1–9, 2015.
- [52] Peyvand Valeh-E-Sheyda, Fereydoon Yaripour, Gholamreza Moradi, and Mohammad Saber. Application of artificial neural networks for estimation of the reaction rate in methanol dehydration. *Industrial and Engineering Chemistry Research*, 49(10):4620–4626, 2010.
- [53] Deniz Baş, Fahriye Ceyda Dudak, and Ismail Hakki Boyaci. Modeling and optimization III: Reaction rate estimation using artificial neural network (ANN) without a kinetic model. *Journal of Food Engineering*, 79(2):622–628, 2007.
- [54] Kohji Omata and Muneyoshi Yamada. Prediction of effective additives to a Ni/active carbon catalyst for vapor-phase carbonylation of methanol by an artificial neural network. *Industrial and Engineering Chemistry Research*, 43(20):6622–6625, 2004.
- [55] E. Assidjo, B. Yao, K. Kisselmina, and D. Amané. Modeling of an industrial drying process by artificial neural networks. *Brazilian Journal of Chemical Engineering*, 25(3):515–522, 2008.
- [56] Shengze Cai, Zhicheng Wang, Sifan Wang, Paris Perdikaris, and George Em Karniadakis. Physics-Informed Neural Networks for Heat Transfer Problems. *Journal of Heat Transfer*, 143(6):1–15, 2021.
- [57] Chen Wang and Yang Xi. Convolutional Neural Network for Image Classification. (1969):1–7, 1997.

- [58] Jianxin Wu. Introduction to Convolutional Neural Networks. *Introduction to Convolutional Neural Networks*, pages 1–31, 2017.
- [59] Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 13-17-Aug:481–490, 2016.
- [60] Xiaowei Jin, Peng Cheng, Wen Li Chen, and Hui Li. Prediction model of velocity field around circular cylinder over various Reynolds numbers by fusion convolutional neural networks based on pressure on the cylinder. *Physics of Fluids*, 30(4):1–16, 2018.
- [61] Junfeng Chen, Jonathan Viquerat, Elie Hachem, Junfeng Chen, Jonathan Viquerat, and Elie Hachem. U-net architectures for fast prediction in fluid mechanics. 2019.
- [62] Tryambak Gangopadhyay, Sin Yong Tan, Anthony LoCurto, James B. Michael, and Soumik Sarkar. Interpretable deep learning for monitoring combustion instability. *IFAC-PapersOnLine*, 53(2):832–837, 2020.
- [63] Luca Guastoni, Prem A. Srinivasan, Hossein Azizpour, Philipp Schlatter, and Ricardo Vinuesa. On the use of recurrent neural networks for predictions of turbulent flows. *11th International Symposium on Turbulence and Shear Flow Phenomena, TSFP 2019*, pages 1–6, 2019.
- [64] Arlindo Rodrigues Galvao Filho, DIogo Fernandes Costa Silva, Rafael Viana De Carvalho, Filipe De Souza Lima Ribeiro, and Clarimar Jose Coelho. Forecasting of Water Flow in a Hydroelectric Power Plant Using LSTM Recurrent Neural Network. *2nd International Conference on Electrical, Communication and Computer Engineering, ICECCE 2020*, (April):14–15, 2020.
- [65] Arvind Mohan, Don Daniel, Michael Chertkov, and Daniel Livescu. Compressed Convolutional LSTM: An Efficient Deep Learning framework to Model High Fidelity 3D Turbulence. (Dl):1–27, 2019.

- [66] C F Colebrook, T Blench, H Chatley, E H Essex, J R Finnicome, G Lacey, J Williamson, and G G Macdonald. Correspondence. Turbulent Flow in Pipes, With Particular Reference To the Transition Region Between the Smooth and Rough Pipe Laws. (Includes Plates). *Journal of the Institution of Civil Engineers*, 12(8):393–422, 1939.
- [67] Gökhan Aksu, Cem Oktay Güzeller, and Mehmet Taha Eser. The Effect of the Normalization Method Used in Different Sample Sizes on the Success of Artificial Neural Network Model. *International Journal of Assessment Tools in Education*, 6(2):170–192, 2019.
- [68] T. A. Pimenta and J. B.L.M. Campos. Friction losses of Newtonian and non-Newtonian fluids flowing in laminar regime in a helical coil. *Experimental Thermal and Fluid Science*, 36:194–204, 2012.
- [69] Henock Mateos Mekisso and Afshim J Ghajar. Comparison of frictional pressure drop correlations for isothermal two-phase horizontal flow. *Mechanical Engineering*, Master The:156, 2004.
- [70] George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, (May), 2021.
- [71] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics Informed Deep Learning (Parts I and II): Data-driven Discovery of Nonlinear Partial Differential Equations. 2017.
- [72] Shengze Cai, Zhiping Mao, Zhicheng Wang, Minglang Yin, and George Em Karniadakis. Physics-informed neural networks (PINNs) for fluid mechanics: A review. pages 1–12, 2021.
- [73] Zixue Xiang, Wei Peng, Xiaohu Zheng, Xiaoyu Zhao, and Wen Yao. Self-adaptive loss balanced Physics-informed neural networks for the incompressible Navier-Stokes equations. 37(109):47–52, 2021.

- [74] Stefano Buoso, Thomas Joyce, and Sebastian Kozerke. Personalising left-ventricular biophysical models of the heart using parametric physics-informed neural networks. *Medical Image Analysis*, 71:102066, 2021.
- [75] Jared Willard, Xiaowei Jia, Shaoming Xu, Michael Steinbach, and Vipin Kumar. Integrating Physics-Based Modeling with Machine Learning: A Survey. 1(1):1–34, 2020.
- [76] Anuj Karpatne, William Watkins, Jordan Read, and Vipin Kumar. Physics-guided Neural Networks (PGNN): An Application in Lake Temperature Modeling. 2017.
- [77] Xiaowei Jia, Jared Willard, Anuj Karpatne, Jordan Read, Jacob Zwart, Michael Steinbach, and Vipin Kumar. Physics guided RNNs for modeling dynamical systems: A case study in simulating lake temperature profiles. *SIAM International Conference on Data Mining, SDM 2019*, pages 558–566, 2019.
- [78] Jordan S. Read, Xiaowei Jia, Jared Willard, Alison P. Appling, Jacob A. Zwart, Samantha K. Oliver, Anuj Karpatne, Gretchen J.A. Hansen, Paul C. Hanson, William Watkins, Michael Steinbach, and Vipin Kumar. Process-Guided Deep Learning Predictions of Lake Water Temperature. *Water Resources Research*, 55(11):9173–9190, 2019.
- [79] Atılım Güneş Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: A survey. *Journal of Machine Learning Research*, 18:1–43, 2018.
- [80] Zhiping Mao, Ameya D. Jagtap, and George Em Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.
- [81] Xiaowei Jin, Shengze Cai, Hui Li, and George Em Karniadakis. NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations. *Journal of Computational Physics*, 426:109951, 2021.

- [82] Sina Amini Niaki, Ehsan Haghghat, Xinglong Li, Trevor Campbell, and Reza Vaziri. Physics-Informed Neural Network for Modelling the Thermochemical Curing Process of Composite-Tool Systems During Manufacture. (1):1–24, 2020.
- [83] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.
- [84] Maziar Raissi, Zhicheng Wang, Michael S. Triantafyllou, and George Em Karniadakis. Deep learning of vortex-induced vibrations. *Journal of Fluid Mechanics*, 861:119–137, 2019.
- [85] Shengze Cai, Zhicheng Wang, and George Em Karniadakis. Heat transfer prediction with unknown thermal boundary conditions using physics-informed neural networks. *ASME*, 2020.
- [86] Minglang Yin, Xiaoning Zheng, Jay D. Humphrey, and George Em Karniadakis. Non-invasive inference of thrombus material properties with physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 375:113603, 2021.
- [87] Alireza Yazdani, Lu Lu, Maziar Raissi, and George Em Karniadakis. Systems biology informed deep learning for inferring parameters and hidden dynamics. *PLoS Computational Biology*, 16(11):1–18, 2020.
- [88] Shengze Cai, Zhicheng Wang, Frederik Fuest, Young Jin Jeon, Callum Gray, and George Em Karniadakis. Flow over an espresso cup: Inferring 3-D velocity and pressure fields from tomographic background oriented Schlieren via physics-informed neural networks. *Journal of Fluid Mechanics*, 915:1–17, 2021.
- [89] Yuyao Chen, Lu Lu, George Em Karniadakis, and Luca Dal Negro. Physics-informed neural networks for inverse problems in nano-optics and metamaterials. *Optics Express*, 28(8):11618, 2020.

- [90] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden Fluid Mechanics: A Navier-Stokes Informed Deep Learning Framework for Assimilating Flow Visualization Data. 2018.
- [91] Xiaowei Jia, Jared Willard, Anuj Karpatne, Jordan S. Read, Jacob A. Zwart, Michael Steinbach, and Vipin Kumar. Physics-Guided Machine Learning for Scientific Discovery: An Application in Simulating Lake Temperature Profiles. *ACM/IMS Transactions on Data Science*, 2(3):1–26, 2021.
- [92] Ruijin Cang, Hechao Li, Hope Yao, Yang Jiao, and Yi Ren. Improving direct physical properties prediction of heterogeneous materials from imaging data via convolutional neural network and a morphology-aware generative model. *Computational Materials Science*, 150(December 2017):212–221, 2018.
- [93] Jin Long Wu, Karthik Kashinath, Adrian Albert, Dragos Chirila, Prabhat, and Heng Xiao. Enforcing statistical constraints in generative adversarial networks for modeling chaotic dynamical systems. *Journal of Computational Physics*, 406:1–26, 2020.
- [94] Byungsoo Kim, Vinicius C. Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler. Deep Fluids: A Generative Network for Parameterized Fluid Simulations. *Computer Graphics Forum*, 38(2):59–70, 2019.
- [95] Adrienne S. Incropera, Frank P., Dewitt, David P., Bergman, Theodore L., Lavine. *Fundamentals of Heat and Mass Transfer*, volume 112. John Wiley Sons, 2015.
- [96] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. Deep-XDE: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.