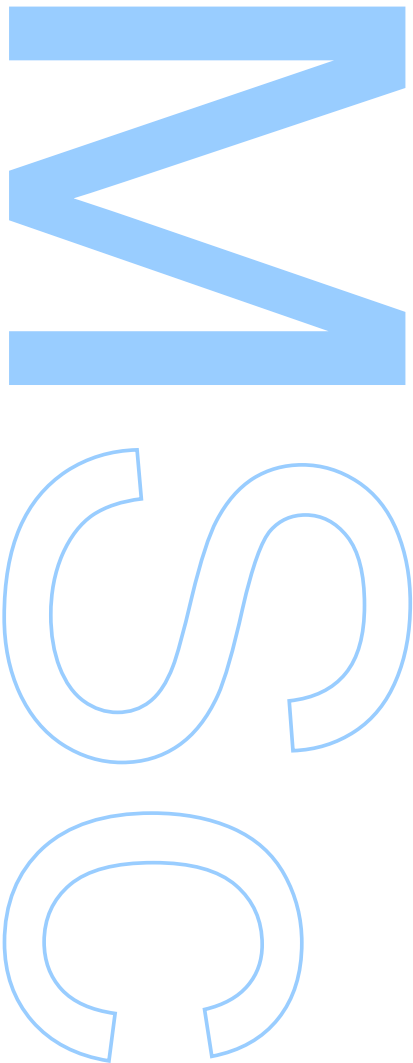
# Statistical Methods in Data Mining and Physics-Informed Neural Networks

Manuel Cabral
Master's degree in Mathematical Engineering
Department of Mathematics
2022

**Orientador**
Prof. Dr. Joaquim Pinto da Costa, Department of Mathematics, Faculty of Sciences of the University of Porto



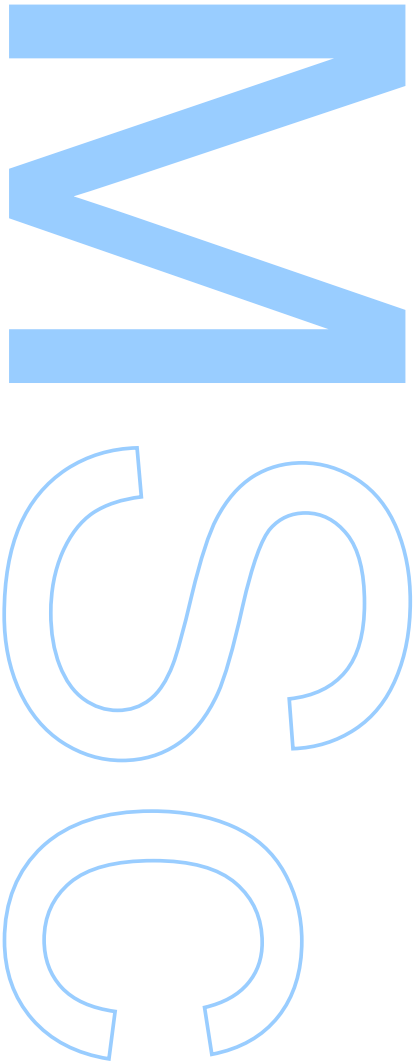
U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____



U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

MSc

2º
CICLO

FCUP
2022

U.PORTO

Statistical Methods in Data Mining and Physics-Informed Neural Networks

Manuel Cabral

FC

U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

# Statistical Methods in Data Mining and Physics-Informed Neural Networks

Manuel Cabral

Dissertação de Mestrado apresentada à
Faculdade de Ciências da Universidade do Porto em
Engenharia Matemática

2022

MSc

UNIVERSITY OF PORTO

MASTERS THESIS

# Statistical Methods in Data Mining and Physics-Informed Neural Networks

*Author:*
Manuel CABRAL

*Supervisor:*
Prof. Dr. Joaquim PINTO DA COSTA

*A thesis submitted in fulfilment of the requirements*
*for the degree of MSc. Mathematical Engineering*

*at the*

Faculty of Sciences of the University of Porto
Department of Mathematics

September 30, 2022

# Sworn Statement

I, Manuel Maria Pacheco do Valle Pereira Cabral, enrolled in the Master's Degree in Mathematical Engineering at the Faculty of Sciences of the University of Porto hereby declare, in accordance with the provisions of paragraph a) of Article 14 of the Code of Ethical Conduct of the University of Porto, that the content of this dissertation reflects perspectives, research work and my own interpretations at the time of its submission.

By submitting this dissertation, I also declare that it contains the results of my own research work and contributions that have not been previously submitted to this or any other institution.

I further declare that all references to other authors fully comply with the rules of attribution and are referenced in the text by citation and identified in the bibliographic references section. This dissertation does not include any content whose reproduction is protected by copyright laws.

I am aware that the practice of plagiarism and self-plagiarism constitute a form of academic offense.

Manuel Cabral,

September 30, 2022

# *Acknowledgements*

I would like to thank Prof. Dr. Joaquim Pinto da Costa for his guidance throughout this year, and for keeping me on track when I got lost in the *in-betweens*.

I would also like to thank my family, in particular my parents, for giving me the conditions to focus entirely on my studies, without any other concern, which is not always the case as an *older* student.

Finally, I thank my friends for their unconditional support; in particular, João António for the countless advice and endless discussions.

UNIVERSITY OF PORTO

# *Abstract*

Faculty of Sciences of the University of Porto
Department of Mathematics

MSc. Mathematical Engineering

**Statistical Methods in Data Mining and Physics-Informed Neural Networks**

by Manuel CABRAL

In a world with ever-growing amounts of data, Machine Learning, and, in particular, Data Mining, have become of paramount importance in order to extract hidden structure and make informed decisions from data in a wide range of areas of knowledge.

In the first chapter of this dissertation, a state-of-the-art of statistical methods in Data Mining is presented. The topics covered range from Statistics, to traditional methods in Machine Learning, to more recent developments in Deep Learning. Given the scope of the area, and how rapidly new advances are being published, a work of this nature is always bound to be, no matter how extensive, incomplete, and, no matter how recent, ephemeral. We do believe, however, that it constitutes an important tool for any researcher in the field.

The vast majority of the methods presented in the aforementioned chapter have the need for high volumes of data, in order to be able to find correlations and patterns within.

In the second chapter, we cover one of the few examples where Data Mining can be extended for the small-data regime: Physics-informed Neural Networks (PINNs). PINNs combine Machine Learning with Natural Sciences, making use of prior knowledge of the system, like known symmetries, conservation laws or governing equations to constrain the solution of the problem to *live* in this embedding (or close to it, in the case of *soft* constraints). Natural Sciences have been perhaps the last area where Machine Learning has failed to cause a major impact, due to the difficulty to obtain significant amounts of data. PINNs present a deceptively simple new framework, that can tackle any amount of data, and that has found promising applications in the physical sciences, since its introduction three years ago. This chapter covers the theory behind them, optimization techniques and the challenges for the future.

UNIVERSITY OF PORTO

# *Resumo*

Faculty of Sciences of the University of Porto
Department of Mathematics

Mestrado em Engenharia Matemática

**Métodos Estatísticos em Data Mining e Physics-Informed Neural Networks**

por Manuel CABRAL

Num mundo com sempre crescentes quantidades de dados, *Machine Learning*, e, em particular, *Data Mining*, têm-se tornado ferramentas de extrema importância para extrair padrões e tomar decisões informadas apartir dos dados, numa vasta gama de áreas do conhecimento.

No primeiro capítulo da presente dissertação, um estado da arte de métodos estatísticos em *Data Mining* é apresentado. Os tópicos abrangidos vão da Estatística, a métodos tradicionais em *Data Mining*, a desenvolvimentos mais recentes em *Deep Learning*. Dada a abrangência da área, e a rapidez com que novos avanços estão a ser publicados, um trabalho desta natureza está sempre destinado a ser, por mais extenso que seja, incompleto, e, por mais actualizado que esteja, efémero. Acreditamos, contudo, que constitui uma ferramenta importante para qualquer investigador neste campo.

A grande maioria dos métodos apresentados no capítulo mencionado tem a necissidade de um grande volume de dados, de forma a conseguir encontrar correlações e padrões nestes.

No segundo capítulo, apresentamos um dos poucos exemplos em que *Data Mining* pode ser alargado a casos com reduzido número de dados: *Physics-informed Neural Networks (PINNs)*. As *PINNs* combinam *Machine Learning* com as Ciências Naturais, fazendo uso do conhecimento prévio do sistema que possamos ter, como simetrias, leis de conservação ou equações governantes, restringindo a solução do problema a *viver* neste *embedding* (ou perto dele, no caso de restrições *soft*). As Ciências Naturais são talvez a última área onde *Machine Learning* ainda não causou um impacto significativo. As PINNs apresentam um novo *framework*, enganosamente simples, que pode lidar com qualquer quantidade de dados, e que encontrou já promissoras aplicações nas ciências físicas, desde a sua introdução

há três anos. Este capítulo cobre a teoria que lhes está subjacente, técnicas de optimização e os desafios para o futuro.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

**AD** Automatic Differentiation 38, 39, 41–43, 58

**BE** Burgers' Equation xiii, 49, 52, 54–56

**CFD** Computational Fluid Dynamics 59, 60

**DL** Deep Learning 31, 51

**DM** Data Mining 1, 2, 7, 10, 27, 30

**FE** Fractional Equation 32

**FEM** Finite Elements Method 31, 34, 35, 48, 50, 59

**IDE** Integral-Differential Equation 32

**ML** Machine Learning 31, 33, 42, 43, 60

**NN** Neural Network xiii, 19, 32–35, 37–42, 45–51, 54, 57, 58, 60

**NSE** Navier-Stokes Equation 36, 59, 60

**PDE** Partial Differential Equation 31, 32, 34–37, 45, 47, 49, 51, 53–55, 57, 58, 61

**PIML** Physics-Informed Machine Learning 32, 59, 61

**PINN** Physics-Informed Neural Network xiii, xv, 32–39, 41, 42, 45, 46, 48–51, 53–55, 57–61, 90

**WE** Wave Equation xiii, 54–56

# Chapter 1

# Statistical Methods in Data Mining

*Disclaimer: the entirety of this first chapter corresponds to the paper published on a special issue of the journal 'Mathematics' dedicated to Statistical Methods in Data Mining [1]. Small corrections and updates were made, but the content is, in its majority, faithful to the original. The work was done in equal parts by Prof. Dr. Joaquim Pinto da Costa and I.*

## 1.1 Introduction

Data Mining (DM) is the process of finding patterns and correlations within large data sets to predict outcomes. Through techniques that range from statistics, to machine learning or to artificial intelligence, DM has entered all areas of knowledge by allowing us to take informed decisions based on the the data itself.

In this review, we will provide the state of the art in statistical methods in DM, by going over the most cited papers in the most impactful journals over the period spanning mainly from 2020 to the present, 2022.

Given the vast range of topics that DM covers, it is not possible to have a strict procedure on which papers to choose. Statistical journals and computer science journals differ substantially in the way articles are shared and have their success and impact measured. Given this impossibility for an absolute procedure, each subsection has the most relevant recently published papers in the area.

In the special issue of Statistical Methods in Data Mining [*],

> "Statistics is, nowadays, more important than ever because of the availability of large amounts of data in many domains like science, finance, engineering,

[*] *https://www.mdpi.com/journal/mathematics/special_issues/Statistical_Methods_Data_Mining*

> medicine, etc. For a long time, statistics has developed as a sub-discipline of mathematics. Nevertheless, computing is also a very important tool for statistics. This is particularly true in statistical methods in DM, which is an interdisciplinary field involving the analysis of large existing databases in order to discover patterns and relationships in the data. It differs from traditional statistics on the size of the data set and on the fact that the data were not initially collected according to some experimental design but rather for other purposes. On the other hand, asymptotic analysis, which has for a long time been an important area of statistics approaching problems where the sample size (and more recently, also, the number of variables) tends to infinity, is obviously also appropriate in DM for dealing with huge amounts of data."

In the next sections, we will write about the problem of Statistical Significance (the famous $p$-value), which is very relevant in the presence of large number of tests; then, we discuss high-dimensional data, including LASSO, RIDGE, PCA and other topics. Other recent topics in Regression will follow and then other recent developments in supervised classification methods. Although all of these topics can be cast in the area of Statistical Learning, we consider next a new section devoted to other recent developments in Statistical Learning. We proceed with a more extensive section of Neural Networks and Deep Learning, including applications to Computer Vision, Time-Series and other sequence data, Dimensionality Reduction, Density Estimation, etc. The last three sections are devoted to Clustering, Software Applications and Closing Remarks.

## 1.2 Statistical Significance

We will start by briefly discussing recent views on the topic of statistical significance - the claim that a result from experiment is not likely to occur randomly or by chance but is instead likely to be attributable to a specific cause. Many areas of knowledge that depend on data analysis and research, such as physics, medicine or economy, need this metric as a way to evaluate the accuracy of the conclusions.

The most broadly accepted metric in the science community is the $p$-value; results that have a $p$-value less than the threshold (usually, 0.05) are said to be statistically significant, while the rest are branded inconclusive. The drawbacks of this classification are well known ([2]) and recently, there has been a pull against this binary classification, as

many studies are labeled contradictory and many research goes to waste [3]. A group of authors in [4] proposed tightening the $p$ threshold to 0.005. In [5], the authors go further by proposing to abandon completely the absolute screening role of statistical significance. They instead propose to treat the p-value as a continuous metric, that, helped by the other metrics - e.g. data quality, related prior evidence, plausibility of mechanism, study design, and other factors that vary by research domain - leads to a more complete decision on the study in question.

A related fundamental question in statistical significance is whether two means differ or not. For this, the most popular approach is the $t$-test, having a primary role in many of the empirical sciences. In recent years, several different Bayesian t-tests have been introduced, due to their theoretical and practical advantages. In [6], it is proposed a flexible t-prior for standardized effect size that gives the possibility to compute the Bayes factor by evaluating a single numerical integral. It generalizes prior work, having the previous objective and subjective t-test Bayes factors as special cases.

Another relevant question in the area is on how to combine individual $p$-values to aggregate multiple small effects. In order to overcome computational issues that performing this operations with the traditional methods in large and complex data sets provoke, in [7], the authors propose a new test that is defined as a weighted sum of Cauchy transformation of the individual p-values. They show that this new test is not only accurate, but also equally as simple to perform as the classic t-test or z-test.

When performing model selection, often different criteria used reach different conclusions, and justification of their use and why it was chosen is often lacking in research papers. In [8], the case of Information criteria (ICs) based on penalized likelihood is analysed and a different view on these criteria that can help in interpreting their practical implications in more complex situations in order to make informed decisions is presented.

## 1.3 High-Dimensional Data

In this section we discuss the case when the number of features $p$ is much larger than the number of observations $N$, $p >> N$. Data in this category suffers from the *curse of dimensionality*: computational burden, statistical inaccuracy, and algorithmic instability. An overview of methods in high-dimensional data can be found in [9]. In this section, we consider penalized regression and feature selection, feature screening for ultra-high

dimensional problems, estimation and inversion of covariance matrices and Linear Discriminant Analysis in high dimensions, statistical inference for longitudinal data with ultrahigh dimensional covariates and a new method of Principal Component Analysis.

In the case of regression, the most common approach consists is using regularization, that is, to apply penalties on the size of the regression coefficients, performing feature selection and model fitting simultaneously. In general, given input data $X \in \mathbb{R}^p$ and response data $Y \in \mathbb{R}$, with a Tikhonov regularization term given by $\frac{1}{2\gamma}||\beta||_2^2$, the problem becomes finding the parameters $\beta$ that,

$$\begin{aligned} \min_{\beta} \quad & \frac{1}{2}||Y - X\beta||_2^2 + \frac{1}{2\gamma}||\beta||_2^2 \\ s.t. \quad & ||\beta||_k \leq \lambda \end{aligned} \tag{1.1}$$

For LASSO, we drop the Tikhonov regularization term and use $k = 1$, for RIDGE regression we also drop the Tikhonov regularization term and use $k = 2$, and for Elastic Net, we keep the full formulation with $k = 1$. An equivalent and more common formulation for Elastic Net is:

$$\min_{\beta} \quad ||Y - X\beta||_2^2 + \lambda_1||\beta||_1 + \lambda_2||\beta||_2^2 \tag{1.2}$$

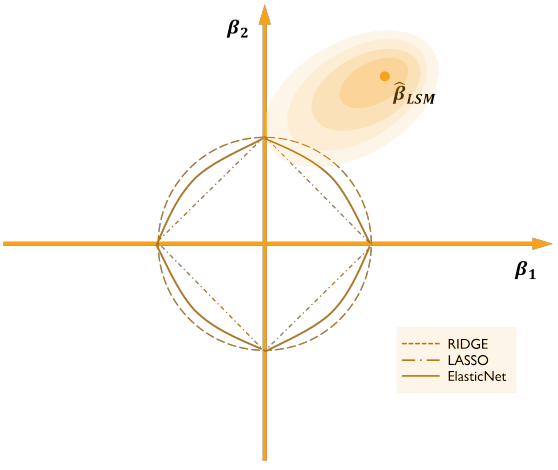



FIGURE 1.1: Parameter domains for RIDGE, LASSO and ElasticNet heuristics; $\hat{\beta}_{LSM}$ is the usual least squares estimator.

The LASSO regression has the advantage over RIDGE of, not only reducing the size of the coefficients, but also setting several to zero, thus reducing the dimensionality of the data. However, LASSO does poorly with highly correlated features, tending to choose one of them and ignore the others, and generally selects $n$ features before saturating. To overcome this limitation, *Elastic net* combines the two approaches. In Figure 1.1 one can see why LASSO usually sets some of the coefficients to zero, since the point inside the corresponding parameter space closest to

the least squares solution is in a corner; that is, has the first coordinate equal zero (in the case of the figure). In the case of a higher dimensional space, the number of corners and flat edges is considerably higher, increasing therefore the possibility of zero coefficients.

One traditional approach for selecting features consists in finding the best subset of the $p$ original features; unfortunately, it is not feasible in practice when $p$ is moderately large. This is also a particular case of formulation (1.1) if we drop the Tikhonov regularization term and use $k = 0$ and is known as the sparse regression problem [10] (actually, $||\beta||_0$ is not really a norm). This case has the advantage of *sparsifying* the regressor without unwanted shrinking, while using the $l_1$-norm leads to biased regression regressors, penalizing both large and small coefficients.

In [10], a new binary convex reformulation, which is equivalent to (1.1) with $k = 0$, and a novel cutting plane algorithm that solves to provable optimality exact sparse regression problems are presented which can deal with regressor dimensions in the 100,000's, a two orders of magnitude improvement over similar known methods, while being faster than the method discussed, in particular, faster than LASSO. Their results show that exact sparse regression can solve high-dimensional problems, refuting the idea that heuristic methods are necessary (see table 1 in [10]).

When performing a regression model on high dimensional data, if the data is *distributed* (meaning, large-scale data stored in $L$ different local machines), traditional methods discussed above as LASSO are not applicable due to storage and computation limitations. Furthermore, if the data has heavy-tailed noise, most theories, namely least squares and Huber loss based statistics, won't work as the assumption of a finite variance for the noise is not fulfilled. The common approach to such problem is the averaging divide-and-conquer approach, where one builds, on each machine, a local estimator $\hat{\beta}_k$ by solving

$$\hat{\beta}_k = \underset{\beta \in \mathbb{R}^{p+1}}{\operatorname{argmin}} \frac{1}{m} \sum_{i \in H_k} \rho_\tau(Y_i - X_i^T \beta) + \lambda_m |\beta|_1 \tag{1.3}$$

,where $H_k$ is the set of data indexes from the $k^{th}$ machine, $m = \frac{n}{L}$ and $\rho_\tau$ is the quantile regression loss function (see Roger Koenker. Quantile regression. Cambridge university press, 2005), and then averaging over all machines: $\hat{\beta}_{avg} = \frac{1}{L} \sum_{k=1}^{L} \hat{\beta}_k$. Similar constructed de-biased approaches also exist but they all suffer from several problems, namely being computationally costly and that the local estimator no longer being sparse. In [11], the authors propose a new distributed estimator for estimating high-dimensional linear model

with heavy-tailed noise that achieves the same convergence rate as the ideal case with pooled data, while establishing the support recovery guarantee.

For ultra-high-dimensional problems, the seminal work by Fan and Lv [12] introduces the notion of sure screening and proposes a new method called sure independence screening based on correlation learning in order to reduce dimensionality from high to a moderate scale below the sample size. In [13], the authors also use screening procedures that make some assumptions on the the data. Their approach, which eliminates redundant covariates in high-dimensional data, is a model free feature screening procedure. The method, called covariate information number-sure independence screening (CIS), can be used for data with continuous features, with any kind of response. The new method is based on a covariate information number (CIN), which captures marginal association of each feature with the response without assuming any specific underlying model, and can be interpreted in terms of the traditional Fisher information in Statistics. A first model-free forward screening based on the concept of cumulative divergence is introduced in [14]. Cumulative divergence is a new correlation metric that characterizes functional dependence; it is robust to the presence of outliers in the conditioning variable. Contrarily to marginal screening approaches, in this new model, the joint correlations among features are considered, while also being robust to model misspecification.

In statistical methods, we often need to use covariance matrices and its inversion; their calculation can be innacurate, numerically unstable and unfeasible in high-dimensional problems. The authors in [15] establish the first analytic formula for nonlinear shrinkage estimation of covariance matrices. It performs better than previous similar models, being about 1000 times faster, with similar accuracy, when compared with Quantized Eigenvalues Sampling Transform (QuEST), and is also able to deal with covariance matrices of dimension of the order up to 10,000. Methods like Linear Discriminant Analysis are heavily dependent on a good estimation the mean vectors and the population covariance matrix, while being based on the premise of an equal population covariance matrix among all classes. In [16], the authors introduce an improved LDA classifier based on the assumption that the covariance matrices follow a spiked covariance model (see [16] for further details).

In [17], the authors approach the problem of statistical inference for longitudinal data

with ultra-high-dimensional features, by introducing a novel quadratic decorrelated inference function approach. It simultaneously eliminates the effect of unwanted parameters and accounts for the correlation to improve the efficiency of the estimation process. Simulation studies conducted by the authors show that the newly proposed method can control Type I error for testing a low dimensional parameter of interest and the false discovery rate in the multiple testing problem.

Linear combinations of covariates is a traditional way of reducing dimension like for instance Principal Component Analysis (PCA), although the components retained have some loss of information which, sometimes, is crucial for other tasks. In [18], the authors introduce a new paradigm which replace high dimensional covariates with a small number of linear combinations in the context of regression, which does not cause loss of information. In [19], a new method of PCA, called Tensor Robust Principal Component Analysis (TRPCA), is introduced. It aims to exactly recover the low-rank and sparse components. This same low-rank tensor recovery has applications in many areas given the amount of high-dimensional data available nowadays, often un-labelled. In [20], the authors analyze the case of visual data, that can be easily corrupted and noisy, and introduce a unified presentation of the surrogate-based formulations that include simultaneously the features of rectification and alignment, and establish error bounds for the recovered tensor.

## 1.4 Regression

Regression is one of the oldest areas of DM that this paper covers, but it is still one of the most active fields of research given its important and wide uses across science. In this section we will cover some of these recents developments.

When dealing with big data sets, dividing the data among several machines is a common approach to work around hardware limitations. Recently, this *divide and conquer* technique has been adapted in the field of statistics to include inferential procedures, based on the mean and other measures of central tendency. These approaches have some shortcomings, namely assuming homoskedastic errors or sub-Gaussian tails. In order to tackle these issues, the work in [21] proposes to use quantile regression in order to extract features of the conditional distribution of the response in a two-step model that doesn't sacrifice accuracy.

The *RIDGE* formulation we have seen in the High-dimensional Data section (eq. (1.2) with $\lambda_1 = 0$) attempts to find the best balance between variance and bias, by fine-tuning the parameter $\lambda_2$. However, in practice, the best solution found is often setting $\lambda_2 = 0$ and finding the minimum-norm solution among those that interpolate the training data. In [22], the authors isolate *what appears to be* a new phenomenon of implicit regularization for interpolated minimum-norm solutions in Kernel "Ridgeless" Regression.

The Maximum likelihood estimator (MLE), on which methods like logistic regression heavily rely on, has been shown to not always exist. In particular, the MLE doesn't exist if and only if there is no overlap of data points, i.e., there is a hyper plane separating the data classes in logistic regression. Working on this, the authors in [23] establish the existence of a phase transition in the logistic model with Gaussian features, and computes the phase transition boundary explicitly (see [24] for an *R* package, **mlt**, that implements maximum likelihood estimation in the class of conditional transformation models).

In [25], the authors propose a new framework that allows to approximate the eigendecomposition using only the eigendecomposition of the Laplacian and the spectral density of the covariance function. The method allows for theoretical analysis of the error caused by the truncation of the series and the boundary effects. This has vast applications, in diverse areas, like in medical imaging.

When dealing with strong prior information and weak data, it may happen that the fitted variance is higher than total variance; in this case, the proportion of variance explained, $R^2$, can be greater than one, which is a problem. In [26], the authors propose a generalization that has a Bayesian interpretation as a variance decomposition, by suggesting the alternative definition of the variance of the predicted values divided by the variance of predicted values plus the expected variance of the errors.

In [27], the authors reformulate the modal regression problem from a statistical learning viewpoint, which renders the problem dimension-independent.

Although modern massive and high-dimensional data promises the discovery of subtle patterns that might not be possible with a small dataset, it brings also great challenges both computational and statistical. Large progress has been made in this century for obtaining useful information from large datasets with high-dimensional covariates and sub-Gaussian tails. However, sub-Gaussian tails are not a valid assumption in many practical applications; in fact, heavy-tailed distributions and outliers are a common presence for high-dimensional data. In 1973, Peter Huber introduced the concept of robust regression,

which is not very much affected by some violations of the linear model assumptions as the traditional least squares model is; for instance in the presence of outliers and heavy-tailed distributions. However, the robustification parameter is set fixed in the works about robust regression, and this brings problems of estimation when the sample distribution is not symmetric. In [28] an adaptive Huber regression for robust estimation and inference is proposed where the robustification parameter adapts to the sample size, dimension and moments for optimal balance between bias and robustness. Their methodology, which is extended to allow both heavy-tailed predictors and observation noise, shows to be more robust and predictive in a real life application.

In [29] a new method, SuSiE, is introduced for variable selection in linear regression. It focus on quantifying uncertainty in which variables should be selected. For this, the sparse vector of regression coefficients are written as a sum of 'single-effect' vectors, each with one non-zero element. Also, a Bayesian analogue of stepwise selection (IBSS ) is introduced which, instead of selecting a single variable at each step, computes a distribution on variables to capture uncertainty in which variable to select. Their method is very appropriate in the presence of higly correlated and sparse variables such as in genetic fine mapping applications, which they illustrate. In [29] it is also discussed the possibility of applying these methods to generic variable-selection problems.

## 1.5 Other Recent Developments in Supervised Classification Methods

We will now describe some recents developments in other supervised classification methods, like Decision Trees, Support Vector Machines, K-NN, etc.

The oldest method of supervised classification is Linear Discriminant Analysis (LDA) of Fisher (1936) for two classes and Rao (1948) for more than two. Somewhat similar to PCA, LDA projects the data into a space of lower dimension $(K-1)$; however, unlike PCA, which finds the space where data has maximum dispersion, LDA finds the space where the classes are well separated and then, in the prediction phase, for a new query, it chooses the class whose projected mean is closest. In [30] by using the connection that naturally exists between LDA and Linear Regression, a penalized LDA is introduced, called the moderately clipped LASSO (MCL), which can be applied when the number of

variables is larger than the sample size. In numerical studies they find that it has better finite sample performance than LASSO.

On the topic of linear boundaries, when the data can be perfectly separated by a hyperplane, the Support Vector Machine (SVM) finds the hyperplane that maximizes the margin between the classes, which is different from the LDA hiperplane. Statistical Learning Theory suggests also that this hiperplane has good generalization properties. It is of course rare that the classes are linearly separable and for that reason, SVMs use the Kernel trick, which consists in projecting the data into a space of larger dimension, eventually infinite, hoping that data are linearly separable there. This is done without actually needing to define the mapping. Since all we need is an inner product in the final space, a Kernel function is used that corresponds to an inner product. If $w$ is a vector defining the hiperplane that maximizes the margin ($2/||w||$ ), our problem consists in $min\ ||w||^2\ s.t.\ Y_i(w^T z_i + b) \geq 1,\ \forall i = 1, 2, ..., n$, where $(x_i, Y_i)$ are the initial data, $\mathbf{x} \in \mathbb{R}^K, \mathbf{Y} \in \{-1, 1\}$ and $Z_i$ corresponds to the projected values of $X_i$. Of course, even in the larger space data might not be linearly separable and for that reason, SVMs use a relaxation, which consists in using slack variables $\xi_i \geq 0$ and the problem becomes,

$$min\ \frac{1}{2}||w||^2 + C\sum_{i=1}^{n}\xi_i \quad s.t.\ Y_i(w^T z_i + b) \geq 1 - \xi_i,\ \ \forall i = 1, 2, ..., n \tag{1.4}$$

SVMs have been implemented in many research fields, ranging from text classification, face recognition, financial application, etc. Although SVMs have good properties, like a small number of parameters to estimate, possibility of finding the global optimum (unlike neural nets), fast in predicting (since only the support vectors are used), they take long time to train, of the order of $n^2$. For that reason they are not very popular with very large data sets and there is not much work about it. Recently, in [31], the authors explore support vector machines in DM classification algorithms and summarize the research status of various improved methods of SVM. They find a solution to speed up the SVM algorithm and conclude that it can be widely used in the context of big data. The performance of SVMs depends highly upon the choice of kernel functions and its parameters and also on the distance used. Many improvements have been made in the last decade to enhance the accuracy of SVM (see [32]), like twin SVM (TWSVM), which has a computational cost of approximately one-fourth of the SVM. It requires to solve two small sized quadratic programming problems in lieu of solving a single large size one (SVM) in order to find two nonparallel hyperplanes. In [33] a comprehensive review on twin support

vector machines (TWSVM) and twin support vector regression (TSVR) is given with applications in Classification, Regression, Semi-Supervised Learning, Clustering and with applications like Alzheimer's disease prediction, speaker recognition, text categorization, image denoising, etc.

Many improvements of TWSVM have been proposed by researchers due to its favorable performance especially in-case of handling large datasets.

Decision Trees (DT) are a non-parametric supervised learning method used for classification and regression. The objective is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piece-wise constant approximation. It approximates the data via a series of *if-then-else* decision rules. It facilitates feature importance and data relations analysis, being easy to interpret.

Despite being a simple model to understand, DTs, by themselves, have some drawbacks. Namely, they are unstable to changes in data, often quite inaccurate and calculations can get quite complex if a right pruning algorithm is not applied jointly.

In [34], it is proposed generalized random forests, a method for nonparametric statistical estimation based on random forests (Breiman,2001). Forests estimate $\mu(x) = E(Y|X = x)$ and theoretical results about the consistency and confidence intervals exist for such estimates (see [34]). This paper extends Breiman's random forests into a flexible method for estimating any quantity $\theta(x)$, not just $\mu(x)$, and use their approach to create new methods for non-parametric quantile regression, conditional average partial effect estimation, and heterogeneous treatment effect estimation via instrumental variables. A software implementation is also provided.

Bayesian additive regression trees (BART) provides an alternative to the parametric assumptions of the linear regression model. In [35], extensions of the original BART to new models are introduced for several different data types and changes to the BART prior to deal with higher dimensions and smooth regression functions. It is also presented and discussed recent theoretical results about BART, as well as the application of methods based on BART to causal inference problems. The paper describes also available software and challenges it may face in the future.

Another non-parametric method of supervised classification is the well known KNN, which, although coming from the non-parametric estimation of the densities $f_k$, $k = 1, ..., K$, it results in classifying any new query to the most dominant class amongst its $K$

neighbors. In [36], authors seek estimators for the entropy of a density distribution, which represents the average information content of an observation, and can be seen as a measure of unpredictability of the system. This has applications in many statistical moethods, namely to test goodness-of-fit and independent component analysis. They use weighted averages of existing estimators based on the k-nearest neighbour distances of a sample of n independent and identically distributed random vectors. They obtain efficient estimators for larger dimensions than the original estimators and also confidence intervals for the entropy asymptotically valid.

In practice, when applying regression or classification models, we can have underfitting (large error in both training and testing) or overfitting (low training error and large expected test error). By using regularization techniques, we can address this gap between the training error and the test error in overfitting. The work [37] introduces a new regularization method for semi-supervised learning that identifies the direction in which the classifier's behavior is most sensitive. Naturally occurring systems and several studies suggest that a predictor robust against random and local perturbations is effective in semi-supervised learning. For instance, with neural networks, it is possible to improve the general performance by applying random perturbations to each input in order to create artificial input points and encouraging the model to assign similar outputs to the set of artificial inputs derived from the same point (see [37]). It has been found, however, that random noise and random data augmentation often causes the predictor to be very vulnerable to a small perturbation in a specific direction, called the adversarial direction; for instance when using $L_1$ and $L_2$ regularization. Adversarial training (Goodfellow et al. 2015) is an attempt to solve this problem that succeeded in improving generalization performance and made the model robust against adversarial perturbation. Unlike adversarial training, the work [37] proposes a method that defines the adversarial directions (only "virtually" adversarial, in fact) without label information and can, for this reason, be applied in semi-supervised learning. The application to supervised and semi-supervised learning tasks on many benchmark datasets demonstrates very competitive performance when compared to state-of-the-art methods.

## 1.6 Statistical Learning

The concept of Statistical Learning refers to a set of tools for modeling and understanding complex data sets, and it combines the areas of statistics and machine learning. It covers

a wide range of concepts. In this section it will be discussed those not covered on other sections, normally filed under the topic of statistical learning themselves.

Zero-shot learning aims to recognize objects whose instances may not have been seen during training (classifying images where there is a lack of labeled training data). In the past few years, there has been a rapid increase in the number of new zero-shot learning methods that have been proposed. These are due to the many cases where AI needs to adapt to previously unseen data, from self-driving cars to new diseases' diagnosis - like COVID-19. In [38], the authors present a comprehensive review on the state-of-the-art of the area, comparing all methods and presenting a new data set to perform the testing in order to have common metrics between models.

When the data in study is corrupted or heavy-tailed, recently introduced median-of-means (MOM) based procedures have been shown to outperform classical least-squares estimators (e.g., LASSO). MOM estimators partition the data set into $k$ blocks, $(Z_i)_{i\in B_k}$, $k = 1, ..., K$ of the same cardinality, and then calculate the median of the $K$ empirical means of each block:

$$MOM_k(Z) = median\left\{ \frac{1}{|B_k|} \sum_{i\in B_k} Z_i, k = 1, ..., K \right\} \tag{1.5}$$

In [39], the authors introduce a min-max MOM estimators and demonstrate that, both in small and high-dimensional statistics, they obtain similar sub-Gaussian deviation bounds as the alternatives models, while being efficient under moments assumptions on data that may have been corrupted by a few outliers.

In many areas of knowledge, non-parametric estimation of a probability density function is an essential tool in the analysis. Using local polynomial techniques, in [40], the authors introduce a novel non-parametric estimator of a density function. The distinctive feature of this estimator is that it adapts to the boundaries of the support of the density automatically; it doesn't require particular data modification or choosing additional tuning parameters, something almost all state-of the art methods are incapable of performing without compromising the finite and large-sample properties of the estimator.

Many of the new methods cited in this paper - namely, neural nets, deep learning, boosting, support vector machines, random forests - have became widely popular and are widely used in all kinds of data sets due to their well known advantages. In [41], the author compares these new 'trends' with the classic approaches, like ordinary least

squares and logistic regression, centering on the differences between prediction and estimation or prediction and attribution (significance testing). The author concludes that there is not one best global approach, but rather that each problem might require a different solution. Furthermore, modern pure prediction algorithms, like black-box models, even if more accurate in general, suffer from lack of interpretability. The paper discusses how and when to combine both approaches and its main results are summarized in table 1.1 (table 5 in the paper).

| **Traditional regressions methods** | **Pure prediction algorithms** |
|---|---|
| Surface plus noise models<br>(continuous, smooth) | Direct prediction<br>(possibly discrete, jagged) |
| Scientific truth Empirical<br>(long-term) | prediction accuracy<br>(possibly short-term) |
| Parametric modeling<br>(causality) | Nonparametric<br>(black box) |
| Parsimonious modeling<br>(researchers choose covariates) | Anti-parsimony<br>(algorithm chooses predictors) |
| $\mathbf{x}p \times n$: with $p \ll n$<br>(homogenous data) | $p \gg n$, both possibly enormous<br>(mixed data) |
| Theory of optimal inference<br>(mle, Neyman–Pearson) | Training/test paradigm<br>(Common Task Framework) |

TABLE 1.1: A comparison between traditional regression methods and pure prediction algorithms, as shown in table 5 of [41].

Given the increasing size and complex structure of data sets in the most varied areas, black-box supervised learning models like neural networks, boosted trees, random forests, k-nearest neighbors or support vector regression, commonly replace more transparent linear and logistic regression models in order to capture nonlinear phenomena. They are, however, difficult to interpret in terms of fully understanding the effects of the features on the response. In many cases, this relationship is vital information. Generally, the solution is using Partial dependence (PD) plots. However, they can produce bad results if the covariates are strongly correlated, because they require extrapolation of the response at predictor values that are significantly outside the multivariate envelope of the training data. To solve this, in [42], the authors present a new visualization approach

called accumulated local effects (ALE) plots, which, besides being computationally less expensive, does not require this unreliable extrapolation with correlated features.

In [43], the authors review and introduce a general framework on the area of meta-research. Meta-research is the area of research that studies research itself, in order to investigate quality, bias, and efficiency. Its work is important in maintaining credibility of the scientific method.

A concept we have all familiarize ourselves with (even if unknowingly) is prediction-based decision-making. It has made its way to government policy decisions, environmental discussions or industry standards. Decisions based on predictions of an outcome have become the norm. This newly acquired attention has focused on how consequential predictive models may be biased, in aspects such as race, gender, or class. Studying and correcting such biases has motivated a field of research called algorithmic fairness. The work [44] provides a framework for this scattered topic, setting terminology, notation and definitions, offering a concise reference frame for thinking through the choices, assumptions, and fairness considerations of prediction-based decision-making.

In social sciences, structural equation modeling (SEM) - which explain the relationships between measured variables and latent variables, and relationships between latent variables themselves - has become the standard. Recently however, partial least squares path modeling (PLS-PM), a composite-based approach to SEM, has been gaining traction in a wide range of uses, as it allows researchers to estimate complex models with many constructs and indicator variables, even at low sample sizes. In [45], the authors explore whether, and when, the in-sample measures such as the model selection criteria can substitute for out-of-sample criteria that require a holdout sample.

MCMC methods are a fundamental tool for Bayesian Inference and use the Metropolis–Hastings (MH) algorithm ([46, 47]), which is used to produce samples from distributions that may otherwise be difficult to sample from and is generally used for sampling from multi-dimensional distributions, especially when the number of dimensions is high. Yet, when faced with big data, these methods do not work well. In [48], a new family of Monte Carlo methods is introduced which is based upon a multidimensional version of the Zig-Zag process of [49]. This new method has often more favourable convergence properties. A sub-sampling version of the Zig-Zag process is introduced which seems to work well with big data.

Low-rank matrix estimation, which is used in mathematical modelling and data compression consists in finding an approximating matrix to a given data matrix and is related to PCA, factor analysis, etc. Many of these estimators are NP-hard but fortunately, some computationally efficient algorithms using leading eigenvectors exist. The authors in [50] investigate the behavior of eigenvectors for a large class of random matrices whose expectations are low-rank.

In many large scale regression problems, matrix products like $\mathbf{X^T W X}$ are needed and in [51] a new and considerably more efficient way to compute $\mathbf{X^T W X}$ is presented. Starting from discretized covariates,these new algorithms manage to be more efficient than previous algorithms, thereby substantially reducing the computational burden for large data sets.

In [52] the authors introduce an alternative procedure to the Minimum Covariance Determinant (MCD) approach. MCD estimates the location and scatter matrix using the subset of given size with lowest sample covariance determinant; this is useful for instance, to avoid outliers. However, the MCD approach cannot be applied when the dimension of the data, $p$, exceeds the subset size, $h$. In [52], the auhtors introduce the minimum regularized covariance determinant (MRCD) approach. It can be applied when $p > h$ and it differs from the MCD in that the scatter matrix is a convex combination of a target matrix and the sample covariance matrix of the subset. The aim is to substitute the subset-based covariance with a regularized covariance estimate, defined as a weighted average of the sample covariance of the h-subset and a predetermined positive definite target matrix. This estimated covariance matrix is guaranteed to be invertible, performs in higher dimensions, it is suitable for computing robust distances, and for linear discriminant analysis and graphical modeling.

In machine learning and statistics, the most common is to have data which consists of vectors of features. However, in the era of big data, other types of data exist, like graphs recommender systems (users and products with transactions and rating relations), ontologies (concepts with relations), computational biology (protein-protein interactions), computational finance (web of companies with competitor, customer, subsidiary relations, supply chain graph, graph of customer-merchant transactions), etc (see [53]). It is of course possible to ignore relations and treat these data as vectors of features; however, these relations have additional valuable information. Most of the research on graphs

has been done on static graphs (fixed nodes and edges). Many applications, however, involve dynamic graphs, that change over time, and in [53] a survey is presented of the recent advances in representation learning for dynamic graphs with several prominent applications and widely used datasets.

KNN Classification also has relevant applications in many different areas. In particular, in [54], the authors discuss geo-spatial applications, like road networks, by introducing a sparse reconstruction approach to select the optimized neighbors for each object, that outperforms existing methods, while being robust to noise.

## 1.7 Neural Networks and Deep Learning

### 1.7.1 Introduction

The function obtained by a neural network with input vector $X = (X_1, X_2, ..., X_p)$, one hidden layer $T = (T_1, T_2..., T_l)$ and output $Y = (Y_1, Y_2, ..., Y_K)$ is such that,

$$Y_j = f_o(b + \sum_{i=1}^{l} w_{ij}^h f_h(a + \sum_{m=1}^{p} w_{mi}^I X_m)),$$

where the parameters $b$, $a$, and matrices $W_I$ and $W_h$ are to be estimated from data and the activation functions $f_o$, $f_h$ are usually the hiperbolic tangent, sigmoid, RELU, etc. Deep networks have many more hidden layers and so, the final function becomes quite difficult to differentiate in order to estimate the parameters, which has to be done using some form of gradient descent (GD). Back-propagation was a breakthrough that allowed the training of multi-layer networks of neurons and new architectures (SOM, RBF, Hopfield,...) and applications appeared. In the 1990s the use of neural networks was generalized and new developments emerged.

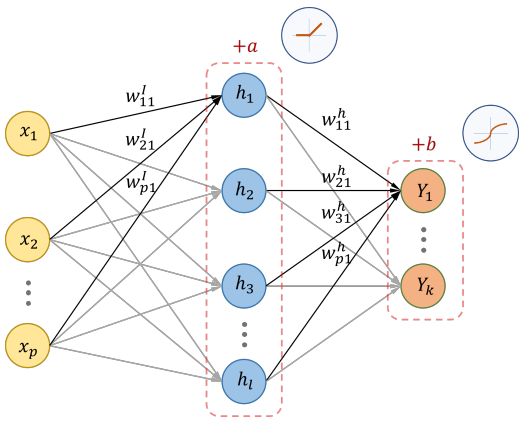



FIGURE 1.2: Generic neural network.

Deep Learning or deep neural networks (see chapter 14 of [55] and [56] for an overview) has achieved tremendous success in the last decade in many areas like artificial intelligence, statistics, applied mathematics, clinical research, etc. Deep Learning uses many compositions of linear transformations followed by nonlinear ones in order to approximate high-dimensional functions and has greatly improved the performance on complex datasets such as images, texts and voices in applications such as computer vision, natural language processing, machine translation, etc. The tremendous success of Deep Learning is due in part to their response to the bias-variance tradeoff, since by using huge datasets, with millions of samples, the variance is reduced and at the same time, due to the current availability of enormous computing power, one can train large neural networks, which reduces biases. On the other hand, given the huge number of parameters to be estimated, sometimes more parameteres than data observations, the training of a deep neural network requires the use of some form of regularization in order to prevent the parameters to 'explode' and the overfitting of the model. The regularization consists in adding a penalty to the loss to be optimized and it results in simpler models. For instance, *L2* regularization or weight decay is very popular in training neural networks.

On the other hand, *L1* regularization introduces sparsity in the weights by forcing some of the weights to be zero which in particular makes some of the features to be discarded. Other forms of regularization exist like Dropout (which has some resemblance to bagging introduced by Breiman in 1996 [57] where during training some number of connections are randomly ignored, with probability $p$, although during test all of the connections are taken into account.

Popular Deep Learning softwares use computational graphs, which allow an efficient way of representing function composition and computing gradients by back-propagation. In addition, in order to minimize the total "error" or loss, that is, the average of the errors between the output value given by the network and the real value, for each data observation, another important aspect is the use of stochastic gradient descent (SGD). Thus, in each pass, instead of computing the gradient for the entire dataset, which is very computationally costly, the gradient is only computed for a small random sample. By the law of large numbers this stochastic gradient should be close to the full sample one, converges faster than GD and is widely used in ML. Other important aspects that contribute to the success of Deep Learning will emerge as we introduce some popular models below.

Despite the huge success of Deep Learning, one must remember that according to

the universal approximation theorem, a neural network with a single hidden layer and a linear output layer may approximate any Borel measurable functions arbitrarily accurately. In practice, one hidden-layer neural network with a large number of nodes can still achieve high prediction performance. Using this fact, recently the authors in [58] introduced a training algorithm, called Local Linear Approximation (LLA), which starts with a one-hidden-layer NN that, by using the first order Taylor expansion to locally approximate activation functions of each neuron, like RELU, can be used in both regression and classification problems. They also generalize it for deep NN.

Note that, while any state-of-the-art review always runs the risk of quickly becoming outdated, this section in particular moves fast, and any top method at the time of publishing will likely be surpassed on the benchmark tests by next month. For this reason, the selection of the papers cited in this topic will follow the methods that caused a paradigm shift in the approaches used, as most of the others are combinations and variations on these. This is the case for instance of computer vision.

A historical survey on deep learning in neural networks can be found in [59].

### 1.7.2 A Statistical View of Deep Learning

In [60], the authors present a survey of recent progress in Statistical Learning Theory, that are useful in Deep Learning.

Acording to them,

> "Broadly interpreted, deep learning can be viewed as a family of highly nonlinear statistical models that are able to encode highly nontrivial representations of data. A prototypical example is a *feed-forward neural network with L layers*, which is a parameterized family of functions $x \mapsto f(x;\theta)$ defined on $\mathbb{R}^d$ by
>
> $$f(x;\theta) := \sigma_L(W_L\sigma_{L-1}(W_{L-1}\dots\sigma_1(W_1x)\dots)) \tag{1.6}$$
>
> where the parameters are $\theta = (W_1, ..., W_L)$ with $W_l \in \mathbb{R}^{d_l \times d_{l-1}}$ and $d_0 = d$, and $\sigma_l : \mathbb{R}^{d_l} \to \mathbb{R}^{d_l}$ are fixed non-linearities, called *activation functions*."

These authors conclude that, surprisingly, deep learning models find solutions that give a near-perfect fit to noisy training data, and at the same time, lead to excellent prediction performance on test data; and this is achieved with no explicit effort to control model complexity.

The classical approach in statistical learning involves a rich, high-dimensional model, combined with some kind of regularization, to encourage simple models but allowing more complexity if that is warranted by the data. The deep learning models, on the other hand, are built on two surprising empirical discoveries: 1) As deep learning uses models with many parameters, the fit to the training data simplifies and simple, local optimization approaches, variants of stochastic gradient methods, are extraordinarily successful in finding near-optimal fits to training data. The idea of over-parametrization seems contradictory from the point of view of classical learning theory, which states that these models should not generalize well. 2) Deep learning models are, in fact, outside the realms of classical learning theory, are trained with no explicit regularization and, although they typically over-fit the training data, exhibiting a near-perfect fit, with empirical risk close to zero, they produce nonetheless excellent test prediction performance in a number of settings. They conclude that deep learning practice has demonstrated that poor predictive accuracy is not an inevitable consequence of this benign overfitting.

See the work in [60] for further theoretical details for understanding this behavior of deep learning models and see why, although there is no explicit regularization, implicitly they impose regularization.

### 1.7.3 Examples of Deep Neural Networks

#### 1.7.3.1 Convolutional Neural Network (CNN) & Computer Vision

Methods in computer vision have vastly improved in recent years. *Object detection* - where the goal is to classify individual objects and localize each using a bounding box and *semantic detection/image segmentation* - where the task consists in clustering parts of an image together which belong to the same object class are the two main problems. A performance comparison between all these methods in different benchmark test sets with accompanying papers can be found here *, while a review in object detection using deep learning can be found in [61] and a review in image segmentation can be found in [62].

*https://paperswithcode.com/area/computer-vision

Currently, the vast majority of computer vision algorithms use convolutional neural networks (CNNs). CNNs are a specialized kind of neural network for processing data that has a known grid-like topology. Their uses are diverse, but they are more commonly used on input of the form of an image data (2-D).

The idea of using this mathematical concept in computer vision was first introduced by LeCun *et al* [63] in 1998, to help solve the problem of recognition of hand-written digits. CNNs are simply neural networks that use convolution in place of matrix multiplication in at least one of their initial layers. This, however, speeds the learning period immensely as matrix multiplication is computationally costly, and, in these applications, usually returns a sparse matrix which a CNN can obtain with less redundant information (for more details, see chapter 9 of [64]).

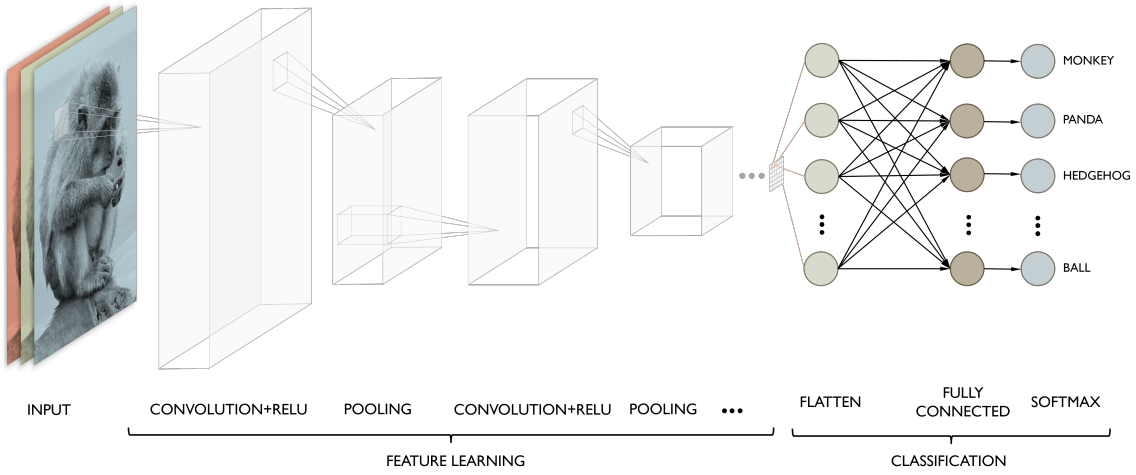



FIGURE 1.3: Configuration of a typical convolutional network.

In fig. 1.3 we can see the usual configuration of a CNN: in the first stage, the layer performs several convolutions in parallel to produce a set of linear activations; in the second stage, each linear activation is run through a nonlinear activation function, such as the rectified linear activation function (RELU); in the third stage, we use a pooling function to modify the output in order to make it invariant to small translations of the imput while, typically, reducing the size of the data - the most common pooling function is max pooling which returns the maximum output within a rectangular neighborhood. In the end, the output is flattened into a 1-D array so it can be ran throught a neural network for classification. Most popular image classification algorithms, including LeNet [63] and AlexNet [65], follow this structure, varying the number of convolutional layers and changing the hyperparameters like the pooling width and stride.

While traditional object detectors would usually consist of three stages - frame candidate regions on a specified image and locate the object; extract features of these candidate regions; use the trained classifier for classification -, current state-of-the-art object detectors either have two stages or only one stage. A two-stage detector is a proposal-driven mechanism, where it initially uses a Region Proposal Network to generate Regions of Interest (RoIs), and, on the second stage, sends the region proposals down the pipeline for object classification and bounding-box regression. They are the most precise framework, but also slower when compared to their counter parts. One of the most successful two-stage detector is Region-based Convolutional Neural Network (R-CNN) [66], which obtains a manageable number of candidate regions and evaluates convolutional networks independently on each region of interest, by classifying each candidate location as one of the foreground classes or as background. On the other hand, there have also been promising results on one-stage detectors, namely YOLO [67] and SSD [68], which lose 10%-40% to the best two-stage methods in terms of accuracy, but are considerably faster. In [69], the authors present a simple two-stage dense detector named *RetinaNet* which match the speed of previous one-stage detectors while surpassing the accuracy of all existing state-of-the-art two-stage detectors, including the ones mentioned in this section.

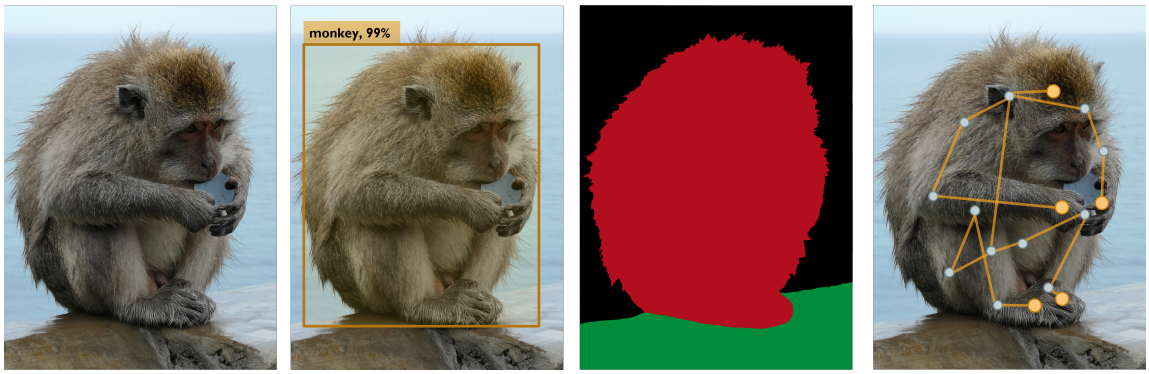



FIGURE 1.4: Different methods on image analysis. From left to right, original image, image recognition, image segmentation, and pose estimation.

In [70], the authors present a new general framework called Mask R-CNN for *instance segmentation*, which combines elements from the two classic computer vision tasks mentioned before in a simple and efficient way, claiming to surpass all previous methods. Most of the advances in computer vision have been driven by powerful baseline systems like Fast/Faster R-CNN [71, 72] and Fully Convolutional Network (FCN) [73] frameworks. Faster R-CNN is built on the R-CNN framework by learning the attention mechanism with a Region Proposal Network (RPN). Building on Faster R-CNN, Mask

R-CNN adds a branch for predicting segmentation masks on each region of interest in parallel with the existing branch for bounding box regression. That is, to the previous outputs - class label and bounding box offset - for each candidate object, it adds a third branch that outputs the mask that bounds the object. It improves on previous instance segmentation algorithms, while also excelling at object detection; it is also easily generalized to perform different vision tasks, like human pose estimation as the paper shows. Mask R-CNN is simple to train and adds only a small overhead to Faster R-CNN, running at 5 fps.

At the time of publishing, the best performing model in both object detection and instance segmentation on the COCO test-dev data set is *SwinV2-G (HTC++)* [74], with 63.1% object AP (average precision) on the former, and 54.4% mask AP on the latter.

| Model | Backbone Architecture | Date | Box AP | Mask AP |
|---|---|---|---|---|
| Mask R-CNN | ResNetXt-101-FPN | Jan/2018 | 39.8% | 37.1% |
| | SpineNet-190, 1536×1536 | Jun/2020 | - | 46.1% |
| RetinaNet | ResNetXt-101-FPN | Feb/2018 | 40.8% | - |
| SwinV2-G | HTC++ | Nov/2021 | 63.1% | 54.4% |

TABLE 1.2: Comparison table between the forementioned methods on the COCO dataset benchmark.

An important challenge in vision tasks is the diversity of the scale of the features we are trying to identify. Convolutional neural networks (CNNs) naturally learn coarse-to-fine

multi-scale features through a stack of convolutional operators. Such inherent multi-scale feature extraction ability of CNNs leads to effective representations for solving numerous vision tasks. How to design a more efficient network architecture is the key to further improving the performance of CNNs. In [75], a new multi-scale backbone archictecture is introduced, *Res2Net*, which adds a new dimension, scale, to the previous existing dimensions of depth, width and cardinality.

By exploiting the multi-scale potential at a granular level, orthogonal to existing methods that utilize layer-wise operations, it can be easily added to previous CNN models, like *ResNet* or *DLA*, further improving them.

A common practice used in convolutional neural networks is flattening a layer followed by a fully connected layer. Despite empirical success, this method discards possible multi-linear structures within the data. A solution is proposed in [76] of using tensor algebraic operations, thus preserving this multilinear structure at every layer. The authors propose *Tensor Contraction Layers* (TCLs) - that reduce the dimensionality of their input while preserving their multi-linear structure using tensor contraction - and *Tensor Regression Layers* (TRLs) - that express outputs through a low-rank multi-linear mapping from a high-order activation tensor to an output tensor of arbitrary order - as end-to-end trainable components of neural networks. By replacing fully connected layers with tensor contractions, this method aggregates long-range spatial information while preserving multi-modal structure, while by enforcing low rank, it reduces the number of parameters needed significantly with minimal impact on accuracy.

Extracting low-rank tensors from data, where unknown deformations and sparse but arbitrary errors exist, is an important problem with lot of applications across fields. Noise pollution, missing observations, partial occlusion, misalignments and other degradation factors are common occurrence in visual data, from mobile phones, to cameras, to surveillance cameras, to medical imaging equipments. In [77], a general framework that incorporates the features of rectification and alignment simultaneously, and establish worst-case error bounds of the recovered tensor is presented. Previous state-of-the-art methods, like *RASL* or *TILT*, are paricular cases of this formulation.

#### 1.7.3.2 Recurrent Neural Network (RNN) for Time Series Data and other Sequential Data

Recurrent Neural Networks (RNNs) are a class of neural networks that allow previous outputs to be used as inputs while having hidden states. They are naturally suited for processing time-series data and other sequential data, allowing inputs of arbitrary length. They consists of several successive recurrent layers, and these layers are sequentially modeled in order to map the sequence with other sequences. RNN have a strong capability of capturing the contextual data from a sequence. They are mostly used for natural language processing and speech recognition. RNNs can solve many different kind of problems and can be categorized into the following: **One to One** - standard mode for classification without RNN (e.g., image classification); **Many to One** - sequence of inputs and a single output (e.g., sentiment analysis); **One to Many** - takes an input and produces

a sequence of outputs (e.g., image captioning); **Many to Many** - sequences of inputs and outputs (e.g., text translation) or sequence to sequence learning (e.g., video classification).

RNNs, alone or together with other neural network types, have had applications in the most diverse areas in the past few years. From enhancing the resolution of images [78], to predicting the price of the stock market [79], to predicting the behavior of chaotic systems [80] or detecting process fault [81].

A review on RNNs and Long Short-Term Memory (LSTM) networks can be found in [82].

#### 1.7.3.3 Autoencoders for Dimensionality Reduction and Applications

An autoencoder is a deep neural network that is trained to attempt to copy its input to its output. It is an unsupervised artificial network that efficiently compresses the data from the reduced encoded representation to a representation that is as close to the original input as possible. In doing so, it learns a lower-dimensional representation (encoding) of the higher-dimensional data, capturing the most important parts of the input. Its uses are diverse, from reduction of dimensionality (actually, Principal Component Analysis is a special case of autoencoders), to anomaly detection, to image denoising.

In particular, in recent years, it has been shown to work in character recognition [83], unsupervised image clustering [84], denoising RNA sequences [85], in cybersecurity [86] (see survey of deep learning methods in cybersecurity in [87]), in 3D MRI brain segmentation [88], fault diagnosis [89] and unsupervised anomaly detection in high energy physics [90].

#### 1.7.3.4 Generative Adversarial Networks (GAN) for Density Estimation and Applications

In classical density estimators, the density function is defined in relatively low dimensions. Generative Adversarial Network ([91]), on the other hand, is an implicit density estimator in much higher dimensions, such as data from images, natural scenarios or handwrittings. GANs put more emphasis on sampling from the distribution $\mathbb{P}_X$ rather than estimation and define the density implicitly through a source distribution $\mathbb{P}_Z$ and a generator function $g(.)$ which is usually a deep neural network [55].

GANs use an unsupervised algorithm that discovers and learns regularities and patterns in input data in such a way that it can be used to generate new examples that plausibly could have been drawn from the original dataset. It combines two models: a model that generates candidate outputs and a classifier that evaluates if the data created by the first model is either real or computer generated. The algorithm 'fools' the classifier when half of the input is misclassified. GANs are one of the most cited topic in DL. Its fame is mostly due to the ability to create photorealist objects and people or to change settings in a scene [92], but GAN's uses are not only limited to image processing. Speech and audio processing, medical information processing and many other areas have active research in this field.

Recently, due to the outbreak of Covid-19, GANs have been used to improve positive cases detection and treatment ( [93] and [94]). Other uses include fault diagnosis ( [95] and [96]), or de-rain images (i.e., take out weather effects such as rain or snow from images; [97]). A literature review on the topic can be found in [98], a comparative analysis of Deep Convolutional GAN and Conditional GAN can be found in [99] and a review on progress and research issues in [100].

#### 1.7.3.5 Other Applications of Deep Learning

Graph Neural Networks (GNNs) are a class of deep learning methods designed to perform inference on data described by graphs. A comprehensive survey on GNNs can be found in [101].

Reinforcement learning (RL) is about an agent interacting with the environment, learning an optimal policy, by trial and error, for sequential decision making problems. It began with Google DeepMind, becoming popular when it was used to beat the the best chess player and go player in the world. An overview in deep reinforcement learning can be found in [102] and in [103].

Most machine learning methods are built on a common premise: the training and test data are drawn from the same feature space and the same distribution. If this distribution changes, the majority of the models described in this papar have to be rebuilt from scratch using newly collected training data. This can be expensive or even impossible to recollect the needed training data and rebuild the models in many practical examples. In such cases, knowledge transfer or transfer learning between task domains would be desirable. A network is trained with a big amount of data, with the model learning the weights

and bias during during training. These weights can be transferred to other networks for testing or retraining a similar new model, so that the network can start with pre-trained weights instead of training from zero. An overview on the topic can be found in [104] and an application to natural language processing can be found in [105].

Drug design aims to identify (new) molecules with a set of specified properties. Given that it still is a lengthy, expensive, difficult, and inefficient process with a low rate of new therapeutic discovery, there is a great interest in developing automated techniques to discover sizeable numbers of plausible, diverse, and novel candidate molecules with desirable properties. In [106] it is proposed a novel variational autoencoder for molecular graphs, whose encoder and decoder are specially designed to account for the unique characteristics of molecular graphs. Experiments reveal that their variational autoencoder can discover plausible, diverse and novel molecules more effectively than several state of the art models.

A new area that has gather momentum in deep learning is Physics-Informed Neural Networks (PINNs) - neural networks that are trained to solve a problem while obeying physical laws (either explicitly or using system's symmetries and conservation laws), or to find the physical laws from existing data. Its framework was introduced in [107], and a current state of the art can be found in [108]. This concept of data-driven discover of governing equations has also had significant advances recently on the topic of the discovery of coordinates and universal embedding of non-linear dynamic systems [109], [110] (see [111] for a review in Koopman's theory for non-linear dynamics).

Speech recognition ([112], [113], [114]), machine translation ([115] and [116] (Google)) are other applications of deep learning.

## 1.8 Clustering

In DM, we are often faced with high dimensional data where observations have from a few dozen to thousands of features or dimensions. In order to cope with the difficulties associated, like visualization of the data and the curse of dimensionality (enumeration of all subspaces is prohibitive), a dimensionality reduction technique or feature selection is often employed beforehand. However, the results obtained depend on the dimensionality reduction technique used and many dimensions can be irrelevant to clustering and identifying them is not easy.

Subspace Clustering tries to circumvent these problems, by clustering, simultaneously, features and observations. This in turn might result in overlapping clusters in both the space of features and observations. This means that, for instance, it might be possible to identify a cluster of the observations in a subspace with 10% of the dimensions that was not possible to identify in the entire space. There are: 1) bottom up approaches which start by finding clusters in low dimensional spaces and iteratively merge them to find clusters in higher dimensional spaces; 2) Top down approaches, wich find clusters in the full set of dimensions and then, for each cluster, find the corresponding subspace. In addition, subspace clustering approaches can be categorized into iterative methods, algebraic approaches, statistical methods and spectral clustering-based methods (see [117] ).

Subspace clustering has been successfully applied to many applications. Recently novel subspace clustering models have been proposed which uses multiple views of the data, not just one, in order to decrease the influence the original features have when the observations are insufficient. These methods produce $v$ subspace representations $\mathbf{Z}^{(v)}$ and feature matrix $\mathbf{X}^{(v)}$, corresponding to the optimization problem,

$$\min_{\{\mathbf{Z}^{(v)}\}_{v=1}^{V}} \mathcal{L}\Big(\{(\mathbf{X}^{(v)}, \mathbf{X}^{(v)}\mathbf{Z}^{(v)})\}_{v=1}^{V}\Big) + \lambda\Omega\Big(\{\mathbf{Z}^{(v)}\}_{v=1}^{V}\Big), \tag{2}$$

where $\mathcal{L}$ is a loss function and $\Omega$ a regularization term.

As a way to explore the relationships between data points and effectively deal with noise, the work [117] proposes using a latent representation for multiple views (Latent Multi-view Subspace Clustering). Their method learns a latent representation to encode complementary information from multi-view features and produces a common subspace representation for all views rather than that of each individual view. Furthermore, they generalize their model for non-linear correlation, and propose generalized Latent Multi-view Subspace Clustering (gLMSC). They show by experimenting on both synthetic and benchmark datasets significant advantages of the learned latent representation for multi-view subspace clustering when compared to the other moder multi-view clustering approaches.

As another illustration/application of the use of multiple views in Clustering, in [118] a new method is proposed for detecting coherent groups in crowd scenes.

Other types of data are also subject to clustering algorithms nowadays; for instance, longitudinal data, which differs from time-series data because series, although being

shorter, are in much larger numbers. For that reason, is is of very practical importance to cluster them. Recently, in [119] the authors introduce a new confinement index and a new way of comparing countries, by using clustering of three dimensional longitudinal trajectories, in the context of COVID-19. Also in [120], new methods of clustering longitudinal data are presented.

## 1.9 Software Applications

With the amount and diversity of algorithms and applications, one of the current challenges in software is to explain its outputs to a broader audience. **AI Explainability 360** presented in [121], is another step in that direction by providing an open-source Python toolkit that provides explainability, interpretability and transparency to the algorithms.

In [122] the package **Tslearn**, a Python machine learning library, is introduced to handle **Time Series Data**. It includes pre-processing routines, feature extractors, and machine learning models for classification, regression and clustering and can treat both univariate as well as multivariate time series data. It treats also series with variable length. In [123], a similar package for time-series classification in Python is presented, called **pyts**. It provides implementations of several algorithms published in the literature, preprocessing tools, and data set loading utilities.

For time series using deep learning, the **Gluon Time Series Toolkit (GluonTS)** introduced in [124] provides the necessary components and tools for quick model development, as well as efficient experimentation and evaluation. It is based on the *Gluon API*[2] of the **MXNet** deep learning framework. Based in this same structure, in [125] the libraries **GluonCV** and **GluonNLP** are introduced . They are deep learning toolkits for computer vision and natural language processing, which provide state-of-the-art pre-trained models, in order to facilitate rapid prototyping and promote reproducible research. They are flexible, being applicable to different programming languages.

Given the rapid growth in diverse areas of graph-structured data, the authors of [126] introduce **GraKeL**, a library that unifies several graph kernels into a common framework, and can be easily combined with other modules of the scikit-learn interface.

Graphical models are very popular in order to identify patterns amongst a set of observed variables in many disciplines. Often, there are a mix of variable types, like binary, categorical, ordinal, counts, continuous, skewed, etc. In addition, if measurements are taken across time, we can be interested in studying the relations between variables not

only at one time point (Mixed Graphical Models (MGMs)) but also across time (mixed Autoregressive (mVAR) Models). In [127], the **mgm** package is introduced for estimating time-varying mixed graphical models in high-dimensional data which extend graphical models for only one variable type, since data sets consisting of mixed types of variables are very common in applications. It is written in R and uses the glmnet package (Friedman, Hastie, and Tibshirani 2010) for Generalized Linear Models (GLMs). The **mgm** package is used to estimate Mixed Graphical Models (MGMs) and mixed Autoregressive (mVAR) Models, both as stationary models (mgm() and mvar()) and time-varying models (tvmgm() and tvmvar()). In [127], the authors provide the background implemented methods and also examples that illustrate how to use the package.

## 1.10 Closing Remarks

Any state-of-the-art paper in DM runs the risk of quickly being outdated. Any article-driven review in such specialized sub-topics runs the risk of appearing disjointed in some parts. Any review paper runs certainly the risk of leaving out important developments in the area. Despite trying to be as objective as possible in the criteria used to choose the papers here presented, many other choices of topics and papers would be equally valid.

# Chapter 2

# Physics-Informed Neural Networks *

## 2.1 Introduction

Despite the huge strides Machine Learning (ML), and, in particular, Deep Learning (DL), have achieved over the past decade, in a wide range of areas - from computer vision to natural language processing, and everything in between -, understanding and forecasting physical systems remains a problem too complex to solve.

Physical and biological systems tend to cover a wide range of scales, both spatial and temporal, and the varied level of accuracy that the data obtained from these systems usually have, say Earth's ecosystem or fluid dynamics, makes simulating these environments computationally costly and the multiple source of uncertainties makes prediction of chaotic systems like these irrelevant past short time scales. The universe is mostly governed by Partial Differential Equations (PDEs), and solving them accurately and efficiently is essential for a more complete understanding and to able to harness its potential in our favour. While traditional methods, like Finite Elements Methods (FEMs), finite differences or spectral methods do very well at solving PDEs, they struggle in this multi-physics, multi-scaled, multi-fidelity case described. Furthermore, inverse problems - infer models or parameters from the set of observations - and dealing with noisy or *gappy* real data is often simply not possible. DL, on the other hand, can deal with imperfect data and extract meaningful features and hidden structures from observations, but it requires a large amount of data in order to learn its many parameters, which is not available in many complex physical/biological systems. Under this limitation, most algorithms fail

*An early version of this chapter - *Deep Learning for Physics: new tools for model, coordinates, and dynamics discovery* - was presented at Young Researchers Conference in Porto (*IJUP*), in May of 2022, where it received the distinction of 'Best Oral Presentation in Mathematics'.

to provide meaningful conclusions about the data, fitting the training data well, but failing to generalize, often giving solutions that are not physically consistent.

Physics-Informed Machine Learning (PIML) attempts to solve this issue by enforcing the system' solution to obey physical governing laws and any other priors about the domain we might have. This chapter will focus on a particular subset of this recent area, PINNs.

Given the limited-data regime most physical systems are in, PINNs use prior knowledge about the system - like known symmetries, conservation laws (symmetries in time) or known dynamics -, to constrain the search space for a solution from the entire set of functions to the the set of functions that are consistent with these laws. For example, in the case of incompressible fluid flows, these constraints come in the form conservation of mass, $\nabla \cdot \mathbf{v} = 0$, and conservation of momentum, $\rho\left(\partial_t \mathbf{v} + (\mathbf{v} \cdot \nabla)\mathbf{v}\right) = -\nabla \mathbf{p} + \mu \nabla^2 \mathbf{v} + \sum \mathbf{F}$. Due to the way these constraints are encoded into their NN architecture, PINNs have the particularity of being able to deal with any amount of data, including no data. This means that PINNs can be used to solve PDEs [128], high-dimensional PDEs [129], stochastic PDEs [130, 131], Fractional Equations (FEs) [132] or Integral-Differential Equations (IDEs) [133, 134].

Applications of PINNs can be found diverse areas such as fluid mechanics [135–138], parameter estimation in biology [139, 140], edge plasma control in magnetic confinement in fusion [141], epidemiology [142, 143] or quantum chemistry [144].

This chapter will cover the theory behind this new and exciting area, as well as present recent variations on the original concept and current avenues of research.

## 2.2 Foundations

Despite recent results that show theoretically the consistency of PINNs [145], it remains mostly an *experimental* area, with most advances being empirical. Many aspects of the PINN algorithm, such as choice of optimizer and activation function, have been *borrowed* from other areas of knowledge where deep-NN are used, as computer vision or language processing. Naturally, these *out-of-the-box* methods are not always the best choice, and, recently, new methods tailored for solving PDEs have been put forward. Finding ways to optimize PINNs is an ongoing effort.

This section introduces the main architecture and the important elements of PINNs, that will then serve to study variants and ways of optimizing PINNs.

### 2.2.1 Motivation

The explosive rate of the amount of data available has allowed for deeper NN, that are able to learn better hidden structure in data. The top part of figure 2.1 is often reproduced in literature, exemplifying how in the small data regime traditional methods in ML (including shallow NNs) tend to perform better, but, as the size of the data grows, the price of having a large number of parameters can be met, and the flexibility of deep NNs allows them to learn deep hidden structures, surpassing significantly other methods in the big data regime.

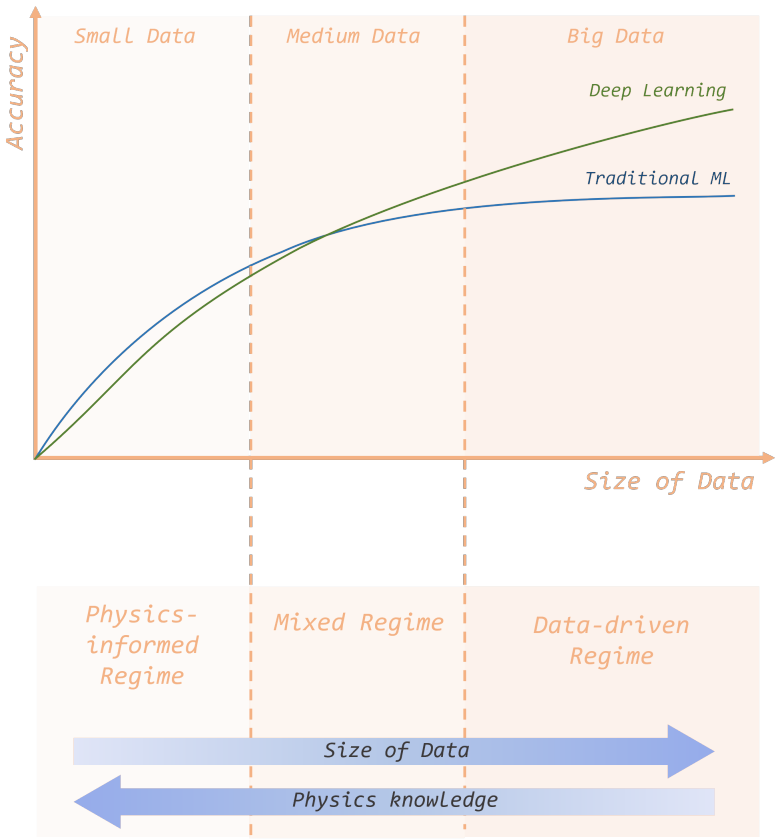



FIGURE 2.1: Data and Physics dependency.

When dealing with physical systems, we are often in the small data regime, as obtaining data can be too costly or even impossible. However, these systems also often follow known laws that constraint their behaviour. The bottom part of figure 2.1 represents that inverse relationship often found, and that can be used to simulate these environments.

PINNs are a way to incorporate this physical knowledge into the network, by encoding this information into the loss function, which is a way of imposing constraints on the parameters of the model. They can deal with any amount of data, working in the three

regimes mentioned in figure 2.1: from big data (closer to a generic NN), to small or no data (solving a generic PDE).

Despite the explosive rate of advances in the area over the past few years, the idea behind solving a physical problem using an artificial NN is not new. It was first presented in 1990, by Lee *et al* [146], further developed by Lagaris *et al* [147, 148] and, later, by Rudd [149], until recently, making use of recent computational advances, namely automatic differentiation, the framework of PINNs that all posterior work builds upon was formulated [150, 151].

PINNs impose the system's priors as soft constraints, by adding a penalty to the loss function depending on the networks performance (soft boundary conditions). There is also a related concept, called Physics-Constrained Neural Networks [152–154], where the priors are entirely enforced, *i.e.*, via specially designed network architectures with strong inductive bias, the model is forced to obey to the initial and boundary conditions (hard boundary conditions); this comes at the cost of implementation complexity, and, eventually, run-time. This variation has some interesting applications [155, 156], but this work will focus on the first kind as it allows for more flexibility.

While in the mixed data-physics regime, the utility of PINNs is clear, it might not be obvious why use them in a no-data regime, to solve PDEs. In fact, for *well posed*, *forward* problems (with *well behaved* solutions), PINNs offer no improvement over previous methods. In fact, the time it takes to train them is significantly longer than the time it takes to run FEMs, often by orders of magnitude. Moreover, PINNs give no guarantee of unique solution, as they solve non-convex optimization problems, which might have different solutions. The choice of hyper parameters and initial values might affect the obtained solution (to avoid being stuck at a local minima, it is useful to run the program several times and choose the run with better accuracy; newer variations of PINNs - covered in section 2.2.8 - can avoid these pathologies and assure convergence in most cases; besides the network's architecture, the sampling of the residual points can also be improved to avoid these failure modes [157] - covered in section 2.2.3). There are several other instances, however, where PINNs excel, where traditional, grid-based methods do not. In particular, the following reasons make PINNs not only very promising, but already the state-of-the-art method in some applications:

- **solving not only forward problems, but also inverse problems [150], with minimal changes to the code:** unknown parameters can be treated as general parameters

to be optimized by the network, so the same code used for forward problems (*solve* PDEs) can be used to infer properties of the system from the data (*e.g.*, finding a fluid's properties from sensor data);

- **the code is general:** while FEM methods generally require the algorithm to be tailored to the problem they are tackling, the code used in PINNs stays essential the same for different problems;

- **dealing with high-dimensionality:** deep NN can, under some assumptions, break the curse of dimensionality [158] (hence their use in areas like image or language processing); this allows them to tackle high-dimensional PDEs, where traditional methods fail (*e.g.*, high-dimensional Black–Scholes, Hamilton–Jacobi–Bellman and Allen–Cahn equations [159]);

- **dealing with noisy and *gappy* data:** as PINNs solutions are smoth and differentiable, they can deal with imperfect or incomplete data. For this, it is useful to also incorporate the Bayesian approach for uncertainty quantification [159], the so called B-PINNs [160].

- **mesh-free:** PINNs do not rely on fixed grids (any point in the domain can be taken as input), which generation is computationally expensive; this allows them to tackle not only high-dimensional domains, but also complex-geometry domains [161]. Another advantage is that a trained PINN can be used to predict values on grids of different resolutions without re-training, and thus, the simulation time does not scale with the number of grid point like in traditional methods [162].

- **running code in parallel:** the mesh-less property of PINNs allows to divide the domain in sections that can be run in different GPUs, in parallel, with different NN in each, in order to speed up the training; this is particularly useful when tackling multi-scale problems.

### 2.2.2 Theory

The idea behind PINNs is to combine the optimization problem of minimizing the error of prediction over a data-set, with the physical knowledge of the system, given by a PDE. In the case of fluid dynamics, for example, one intends to minimize the error of prediction of eventual velocity data points, while obeying the Navier-Stokes Equations (NSEs):

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{N_{data}} \sum_{i=1}^{N_{data}} \left\| \hat{\mathbf{v}}_i(\theta) - \mathbf{v}_i \right\|^2 \tag{2.1a}$$

$$\text{s.t.} \quad \rho \left( \partial_t \mathbf{v} + (\mathbf{v} \cdot \nabla)\mathbf{v} \right) + \nabla \mathbf{p} - \mu \nabla^2 \mathbf{v} - \sum \mathbf{F} = 0 \tag{2.1b}$$

$$\nabla \cdot \mathbf{v} = 0 \qquad \text{(for incompressible flows)} \tag{2.1c}$$

$$\nabla \times \mathbf{v} = 0 \qquad \text{(for irrotational flows)} \tag{2.1d}$$

In general, for PDE given by $f$, we have the following constrained optimization problem:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{N_{data}} \sum_{i=1}^{N_{data}} \left\| \hat{\mathbf{u}}_\theta(z_i) - \mathbf{u}_i \right\|^2 \tag{2.2a}$$

$$\text{s.t.} \quad f\left( t, \mathbf{x}; \frac{\partial \mathbf{u}_\theta}{\partial x_1}, ..., \frac{\partial \mathbf{u}_\theta}{\partial x_d}; \frac{\partial^2 \mathbf{u}_\theta}{\partial x_1 \partial x_1}, ..., \frac{\partial^2 \mathbf{u}_\theta}{\partial x_1 \partial x_d}; ...; \mu \right) = 0, \quad (t, \mathbf{x}) \in [0, T) \times \Omega \tag{2.2b}$$

where $\mathbf{x} = (x_1, ..., x_d)$, $\Omega$ is the spatial domain, $f$ is the PDE in question, rearranged such that it takes the value 0, $\mathbf{u}_\theta$ is the solution of $f$ (parameterized by $\theta$) and $\mu$ are parameters of the PDE, that can also be learned as an inverse problem using PINNs. Note that we are using the word *data* in a broad sense, for points on the domain for which the value is known; it can be simply points along the boundary, $\partial\Omega$.

To solve this problem, we start by transforming the constraint $f$ into a multi-dimensional integral over the domain, and applying a Lagrange multiplier, $\lambda$, as shown in eq. 2.3. Note that $f$ is now squared so that minimizing the second term of eq. 2.3 is the same as obeying the constraint in eq. 2.2b.

$$\begin{aligned} \theta^* = &\underset{\theta}{\operatorname{argmin}} \frac{1}{N_{data}} \sum_{i=1}^{N_{data}} \left\| \hat{\mathbf{u}}_\theta(z_i) - \mathbf{u}_i \right\|^2 \\ &+ \lambda \int \cdots \int_{[0,T) \times \Omega} f\left( t, \mathbf{x}; \frac{\partial \mathbf{u}_\theta}{\partial x_1}, ..., \frac{\partial \mathbf{u}_\theta}{\partial x_d}; \frac{\partial^2 \mathbf{u}_\theta}{\partial x_1 \partial x_1}, ..., \frac{\partial^2 \mathbf{u}_\theta}{\partial x_1 \partial x_d}; ...; \mu \right)^2 dt dx_1 \dots dx_d \end{aligned} \tag{2.3}$$

Other than trivial cases, the integral in the second term is not solvable. A common approach is to numerically integrate this term via *Monte Carlo* methods, by sampling $N_{phys}$ points across the domain and discretize the integral into a sum over those points:

$$\begin{aligned}\theta^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{N_{data}} \sum_{i=1}^{N_{data}} \left\| \hat{\mathbf{u}}_\theta(z_i) - \mathbf{u}_i \right\|^2 \\ + \lambda \frac{1}{N_{phys}} \sum_{i=1}^{N_{phys}} f\left(t, \mathbf{x}; \frac{\partial \mathbf{u}_\theta}{\partial x_1}, ..., \frac{\partial \mathbf{u}_\theta}{\partial x_d}; \frac{\partial^2 \mathbf{u}_\theta}{\partial x_1 \partial x_1}, ..., \frac{\partial^2 \mathbf{u}_\theta}{\partial x_1 \partial x_d}; ...; \mu\right)^2\end{aligned} \tag{2.4}$$

It is this optimization problem, over the augmented data set of $N_{phys} + N_{bound} + N_{data}$, that PINNs are designed to solve.

Note that, unlike NNs, PINNs are not bounded by the necessity of having data as input: they can deal with any amount of data, including no data. In this *no-data* regime, usually when solving PDEs, the input can simply be points uniformly distributed across the domain (this is not, however, the most efficient distribution, and optimizing collocation points' sampling is an active area of research - see Section 2.2.3).

During this chapter, we will use the more common notation for the terms of eq. 2.4 of *loss*, $\mathcal{L}$:

$$\begin{cases} \mathcal{L}_{data} = \frac{1}{N_d} \sum_{i=1}^{N_d} \left\| \hat{\mathbf{u}}_\theta(z_i) - \mathbf{u}_i \right\|^2 \\ \mathcal{L}_{res} = \frac{1}{N_c} \sum_{i=1}^{N_c} \left\| \mathbf{r}_\theta(z_i) \right\|^2 \\ \mathcal{L}_{bound} = \frac{1}{N_b} \sum_{i=1}^{N_b} \mathcal{B}(\hat{\mathbf{u}}_\theta(z_i))^2 \end{cases} \tag{2.5}$$

Some literature divides the boundary term in an initial conditions term and a boundary (spatial) conditions term. In order to keep this formulation as general as possible, we opted to treat time in a similar fashion to space dimensions. When dealing with particular examples, however, it is often useful to make that distinction, so that they are given different weights and the network can better infer from their relevance to the learning procedure.

The problem is then to find the parameters $\theta$ that minimize $\mathcal{L}(\Theta)$:

**PINNs Optimization Problem**

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}(\theta) \tag{2.6}$$

$$\mathcal{L}(\theta) = \omega_{data}\mathcal{L}_{data} + \omega_{res}\mathcal{L}_{res} + \omega_{bound}\mathcal{L}_{bound} \tag{2.7}$$

, where $\omega_i$ are the weights given to each loss component.

PINNs are constituted of four main building blocks: **NN**, **automatic differentiation**, **loss function** and **feedback mechanism**.

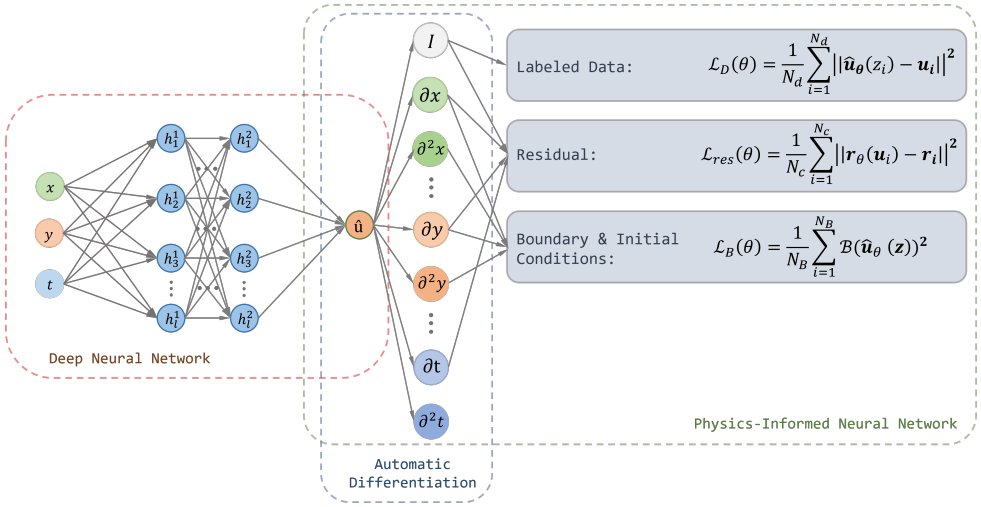



FIGURE 2.2: Generic PINN.

Figure 2.2, represents schematically the process of PINNs: propagate the data in a feed-forward-NN, keep a tape of all necessary derivatives using Automatic Differentiation (AD), calculate the loss functions, adjust weights on the NN in order to minimize the total loss, and repeat the process until the error is smaller than a pre-set value $\epsilon$.

**Algorithm: Training PINNs**

1. Sample $N$ collocation points $\{t_j, x_j\}_{j=1}^{N}$ in $\Omega$ and $N_{bound}$ boundary points $\{x_j\}_{j=1}^{N}$ in $\partial\Omega$, following a distribution $\mathcal{P}$.

2. Set NN architecture – width, depth, activation function, weights initialization distribution -, learning parameters – learning rate $l_r$, batch size $m$, number of epochs -, and choose the loss function' weights $\omega_j$.

3. Initialize model parameters $\theta$ (the weights and biases of the NN).

4. for in range(epochs):

   i. perform forward pass of $m$ collocation points;

   ii. during forward pass, use AD to keep a *tape* of all partial derivatives needed;

   iii. calculate loss function from residuals;

   iv. backward propagate, adjusting parameters $\theta$ using (stochastic) gradient descent.

5. Return $\hat{u}(x, t)$

In order to understand the elements that make PINNs unique, it is useful to review the concepts of NNs' architecture, so we can remark the differences between the two.

#### 2.2.2.1 Neural Networks

NNs have been used to solve a wide variety of problems in different areas. In particular, their well known capabilities as universal approximators of any (Borel measurable) function makes them incredibly versatile [163, 164] *:

**Universal Approximation Theorem for Neural Networks**

A Neural Network with one hidden layer with a finite number of neurons can approximate arbitrarily well any continuous function $f : [0,1]^n \to [0,1]$.

The dual version of this theorem - a NN of fixed width (number of neurons per hidden layer) and arbitrary depth (number of layers) - has been recently proved [165], providing evidence that the number of hidden layers may be more important than the number of neurons per hidden layer for expressiveness in networks which use $ReLU = \max(0, x)$ as a non-linear activation function.

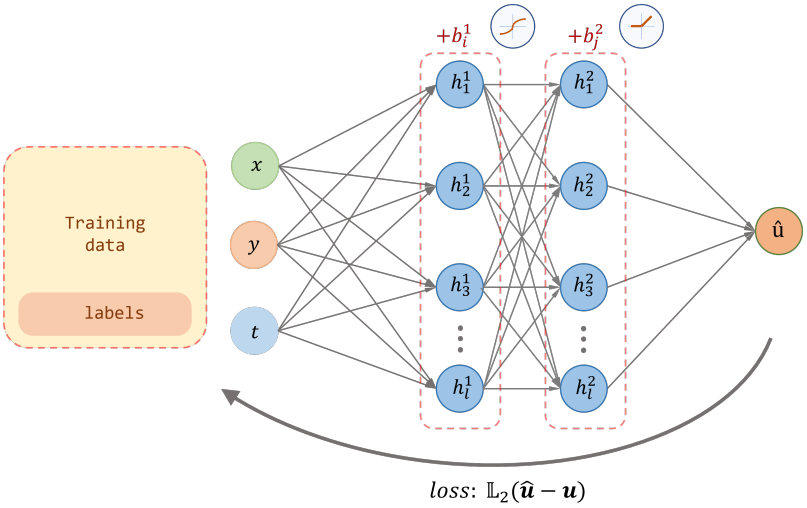



FIGURE 2.3: Generic feed-forwardNN.

A feed-forward-NN (schematically represented in figure 2.3) takes a batch of the training data and propagates it forward, by multiplying the inputs by the weights of the connections, $w_{ij}$, adding a bias, $b_j$, and then applying a non-linear activation function, $\sigma$ (pictured, the two most common: *sigmoid* and *ReLU*).

$$h_j^k = \sigma\left(\sum_k w_{jk}^l h_k^{l-1} + b_j^l\right) \tag{2.8}$$

*in fact, another important and similar result that goes mostly unnoticed is that **a neural network with a single layer can approximate accurately any non-linear continuous functional or non-linear operator**. This theorem will be discussed in section 2.2.8.7, when presenting *DeepONet*.

The network then *learns* via backpropagation: the weights are adjusted proportionally to the gradient of the error function between the output and the labeled result from the input data. The magnitude of this adjustment is controlled by the learning rate parameter, $\alpha$.

$$w_{ij}^{l} \leftarrow w_{ij}^{l} - \alpha \frac{\partial E(\mathbf{X}, \Theta)}{\partial w_{ij}^{l}} \tag{2.9}$$

$$\frac{\partial E(\mathbf{X}, \Theta)}{\partial w_{ij}^{l}} = \frac{1}{N} \sum_{d=1}^{N} \frac{\partial}{\partial w_{ij}^{l}} \left( \frac{1}{2} \left( \hat{\mathbf{u}}_d - \mathbf{u}_d \right)^2 \right) \tag{2.10}$$

This process is repeated over several *epochs*, and, given enough training data, the adjustment of the network' weights will allow it to approximate the true hidden function and provide meaningful predictions of the labels. Its accuracy is then measured in a *test set*, unknown to the NN during the training phase. Significant discrepancies between the accuracy in both sets are usually dealt with using regularization techniques (*e.g.*, dropout).

**In PINNs, the training data set becomes the collocation points; the only labeled data is the (external) data points - that are optional; and the loss function becomes composite of several loss functions and governs the learning phase. All other remaining components here described remain true.**

#### 2.2.2.2 Automatic differentiation

The advent of PINNs in the past few years was only made possible, in practice, due to the unreasonable efficiency of AD [166]. Traditionally methods for computing derivatives, like numerical differentiation, are computationally costly, prone to round-off and truncation errors and scale poorly, struggling with high-dimensional data. Symbolic differentiation, on the other hand, while fast and exact, can result in complex and redundant formulas and has limited applications due to the necessity of closed-form expressions. AD decomposes the problem in a graph (or network) of interconnected nodes, where each connection represents an elementary mathematical operation - addition, subtraction, multiplication, division, exponentiation, or transcendental functions like trigonometric or logarithmic functions -, for which the derivative is already known. The accumulation of the values of all the nodes gives the final value of the derivative; AD does not give an expression for the derivative. The graph nature of AD allows for loops, recursion and

branching. Furthermore, it calculates the value of the derivative at machine precision, and it only adds a small constant factor of processing to the original program [166].

AD is the backbone of ML frameworks like *Tensorflow* [167] (used in this paper) or *PyTorch* [168], being responsible for the incredible advances in the area, over the past decade. The backpropagation algorithm [169] mentioned in section 2.2.2.1, in the context of NNs, is a particular case of AD * (a special case of the reverse mode, where the objective function is the NN's error function, and the derivative is with respect to the network's weights).

Given the importance of this algorithm for PINNs usefulness, it is relevant to show how it works with an example. It is a common critique in literature that AD is often presented via examples, lacking the formal treatment that makes it so powerful and with so many other applications; however, for the purpose and depth of this thesis, we believe an example is the right choice. A more complete view on AD can be found in [170].

Let's then evaluate the partial derivatives of the function $f(x_1, x_2) = x_1 \exp(x_2) - \sin\left(\frac{x_1}{x_2}\right)$, at the point $(x_1, x_2) = (2, 3)$.

We start by decomposing the expression into its elementary components, and saving the intermediate values in new variables $v_i$:

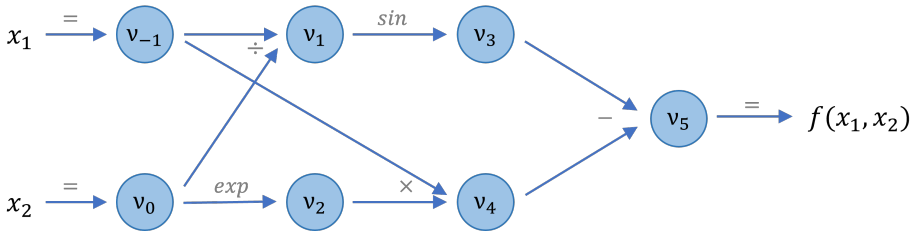



FIGURE 2.4: Computational graph of the function $f(x_1, x_2) = x_1 \exp(x_2) - \sin\left(\frac{x_1}{x_2}\right)$.

Note that, in this example, symbolic differentiation can quickly give the exact derivative expression. However, as the complexity of the function grows, so does the size of the expression, while the computational graph presented in figure 2.4 remains more manageable.

The first step is to fill in the intermediate values $v_i$, and record the inter-variable dependencies in the computational graph, by doing a forward pass of the graph, starting at the input variables $v_{-1}$ and $v_0$, and finishing at the output variable $v_5 = f(x_1, x_2)$. This is the *Forward Primal Trace*, and can be seen in tables 2.1a and 2.2a.

*in ML, the names *backpropagation* and *autodiff* are often used interchangeably.

AD has two main modes, depending on the direction the graph is travelled: **forward accumulation mode** (or tangent linear mode) and **reverse accumulation mode** (or adjoint/cotangent mode).

Each pass in the forward mode computes one column of the Jacobian matrix - $\left(\frac{\partial f_1}{\partial x_i}, ..., \frac{\partial f_m}{\partial x_i}\right)$ -, while each pass in the reverse mode computes one line of the Jacobian - $\left(\frac{\partial f_i}{\partial x_1}, ..., \frac{\partial f_i}{\partial x_n}\right)$. In general, this means that, for functions $f : \mathbb{R}^n \to \mathbb{R}^m$, the forward mode is more suitable when $n < m$ and the reverse mode otherwise. In our example, as the number of inputs is greater than the number of outputs, the reverse mode is more efficient. In table 2.2, we can see that it only takes one reverse pass to evaluate both $\frac{\partial f}{\partial x_1}$ and $\frac{\partial f}{\partial x_2}$, while, in table 2.1, we need one pass to evaluate $\frac{\partial f}{\partial x_1}$ (table 2.1b) and one for $\frac{\partial f}{\partial x_2}$ (table 2.1c).

In ML, often the function being studied is of the type $f : \mathbb{R}^n \to \mathbb{R}$. As the reverse mode can evaluate $\nabla f = \left(\frac{\partial f}{\partial x_1}, ..., \frac{\partial f}{\partial x_n}\right)$ in a single pass, it is the main mode used in ML (often in the form of backpropagation).

Forward mode, table 2.1, works in an intuitively way. To evaluate $\left(\frac{\partial f_1}{\partial x_i}, ..., \frac{\partial f_m}{\partial x_i}\right)$, we:

1. associate the intermediate variables $v_i$ to their derivative $\dot{v}_j = \frac{\partial v_j}{\partial x_i}$;
2. set the derivatives of the initial variables, $\dot{\mathbf{v}}_-$, to $\dot{\mathbf{x}} = \mathbf{e_i}$, as $\mathbf{e_i}$ is the direction along which we want to differentiate our function;
3. evaluate $\dot{v}_i$ in succession, following the computational graph in figure 2.4;
4. obtain the final intended result, $\frac{\partial \mathbf{f}}{\partial x_i} = \dot{\mathbf{v}}_\mathbf{k}$, at the point $(x_1, ..., x_n)$.

Note that, during step 2, if instead of setting $\dot{\mathbf{x}} = \mathbf{e_i}$, we set $\dot{\mathbf{x}} = \mathbf{r}$, for some vector $\mathbf{r}$, we obtain the directional derivative of $f$ along the vector $\mathbf{r}$, $\nabla f \cdot \mathbf{r}$.

Reverse mode, table 2.2, propagates derivatives backwards, from the output $f$ to the inputs $\mathbf{x}$. To evaluate $\left(\frac{\partial f_i}{\partial x_1}, ..., \frac{\partial f_i}{\partial x_n}\right)$, we:

1. associate the intermediate variables $v_i$ with an adjoint variable $\bar{v}_j = \frac{\partial f_i}{\partial v_j}$ - this variable represents the sensitivity of $f$ to changes in $v_j$;
2. set the adjoint of output, $\bar{\mathbf{v}}_\mathbf{k} = \bar{\mathbf{f}}$, equal to 1;
3. following the computational graph, use the chain rule to write $\bar{v}_j$ as a function of the adjoint variables connected to $j$, *e.g.*, in figure 2.4, $v_0$ connects to $v_1$ and $v_2$, so:

$$\bar{v}_0 = \frac{\partial f}{\partial v_0} = \frac{\partial f}{\partial v_1}\frac{\partial v_1}{\partial v_0} + \frac{\partial f}{\partial v_2}\frac{\partial v_2}{\partial v_0} = \bar{v}_1\frac{\partial v_1}{\partial v_0} + \bar{v}_2\frac{\partial v_2}{\partial v_0}$$

Forward propagation

| | |
|---|---|
| $v_{-1} = x_1$ | $= 2$ |
| $v_0 = x_2$ | $= 3$ |
| $v_1 = \frac{v_{-1}}{v_0}$ | $= 2/3$ |
| $v_2 = \exp(v_0)$ | $= \exp(3)$ |
| $v_3 = \sin(v_1)$ | $= 0.618$ |
| $v_4 = v_{-1} \times v_2$ | $= 40.171$ |
| $v_5 = v_4 - v_3$ | $= 39.553$ |
| $y = v_5$ | $= 39.553$ |

(A) Forward Primal Trace.

Forward propagation

| | |
|---|---|
| $\dot{v}_{-1} = \dot{x}_1$ | $= 1$ |
| $\dot{v}_0 = \dot{x}_2$ | $= 0$ |
| $\dot{v}_1 = \frac{\dot{v}_{-1}v_0 - v_{-1}\dot{v}_0}{v_0^2}$ | $= 1/3$ |
| $\dot{v}_2 = \dot{v}_0 \times v_2$ | $= 0$ |
| $\dot{v}_3 = \dot{v}_1 \cos(v_1)$ | $= 0.262$ |
| $\dot{v}_4 = \dot{v}_{-1} \times v_2 + v_{-1} \times \dot{v}_2$ | $= 20.086$ |
| $\dot{v}_5 = \dot{v}_4 - \dot{v}_3$ | $= 19.824$ |
| $\frac{\partial y}{\partial x_1} = \dot{v}_5$ | $= 19.824$ |

(B) Forward mode, with $\dot{\mathbf{x}} = \mathbf{e_1}$ for $\frac{\partial f}{\partial x_1}$.

Forward propagation

| | |
|---|---|
| $\dot{v}_{-1} = \dot{x}_1$ | $= 0$ |
| $\dot{v}_0 = \dot{x}_2$ | $= 1$ |
| $\dot{v}_1 = \frac{\dot{v}_{-1}v_0 - v_{-1}\dot{v}_0}{v_0^2}$ | $= -2/9$ |
| $\dot{v}_2 = \dot{v}_0 \times v_2$ | $= \exp(3)$ |
| $\dot{v}_3 = \dot{v}_1 \cos(v_1)$ | $= -0.175$ |
| $\dot{v}_4 = \dot{v}_{-1} \times v_2 + v_{-1} \times \dot{v}_2$ | $= 40.171$ |
| $\dot{v}_5 = \dot{v}_4 - \dot{v}_3$ | $= 40.346$ |
| $\frac{\partial y}{\partial x_2} = \dot{v}_5$ | $= 40.346$ |

(C) Forward mode, with $\dot{\mathbf{x}} = \mathbf{e_2}$ for $\frac{\partial f}{\partial x_2}$.

TABLE 2.1: Forward Accumulation Mode

For simplicity, we will represent this step incrementally, as in [166],

$$\bar{v}_0 = \bar{v}_2 \frac{\partial v_2}{\partial v_0}$$

, and

$$\bar{v}_0 = \bar{v}_0 + \bar{v}_1 \frac{\partial v_1}{\partial v_0}$$

4. evaluate $\bar{v}_i$ in succession, following the computational graph in figure 2.4;

5. obtain the final intended result, $\left(\frac{\partial f_i}{\partial x_1}, ..., \frac{\partial f_i}{\partial x_n}\right) = \bar{\mathbf{v}}_-$, at the point $(x_1, ..., x_n)$.

Forward pass

| | |
|---|---|
| $v_{-1} = x_1$ | $= 2$ |
| $v_0 = x_2$ | $= 3$ |
| $v_1 = \frac{v_{-1}}{v_0}$ | $= 2/3$ |
| $v_2 = \exp(v_0)$ | $= \exp(3)$ |
| $v_3 = \sin(v_1)$ | $= 0.618$ |
| $v_4 = v_{-1} \times v_2$ | $= 40.171$ |
| $v_5 = v_4 - v_3$ | $= 39.553$ |
| $y = v_5$ | $= 39.553$ |

(A) Forward Primal Trace.

Reverse pass

| | | |
|---|---|---|
| $\bar{x}_1 = \bar{v}_{-1}$ | | $= 19.824$ |
| $\bar{x}_2 = \bar{v}_0$ | | $= 40.346$ |
| $\bar{v}_{-1} = \bar{v}_{-1} + \bar{v}_1 \frac{\partial v_1}{\partial v_{-1}}$ | $= \bar{v}_{-1} + \bar{v}_1 \times \frac{1}{v_0}$ | $= 19.824$ |
| $\bar{v}_0 = \bar{v}_0 + \bar{v}_1 \frac{\partial v_1}{\partial v_0}$ | $= \bar{v}_0 + \bar{v}_1 \times \left(-\frac{v_{-1}}{v_0^2}\right)$ | $= 40.346$ |
| $\bar{v}_{-1} = \bar{v}_4 \frac{\partial v_4}{\partial v_{-1}}$ | $= \bar{v}_4 \times v_2$ | $= 20.086$ |
| $\bar{v}_0 = \bar{v}_2 \frac{\partial v_2}{\partial v_0}$ | $= \bar{v}_2 \times \exp(v_0)$ | $= 40.171$ |
| $\bar{v}_1 = \bar{v}_3 \frac{\partial v_3}{\partial v_1}$ | $= \bar{v}_3 \times \cos(v_1)$ | $= -0.786$ |
| $\bar{v}_2 = \bar{v}_4 \frac{\partial v_4}{\partial v_2}$ | $= \bar{v}_4 \times v_{-1}$ | $= 2$ |
| $\bar{v}_3 = \bar{v}_5 \frac{\partial v_5}{\partial v_3}$ | $= \bar{v}_5 \times (-1)$ | $= -1$ |
| $\bar{v}_4 = \bar{v}_5 \frac{\partial v_5}{\partial v_4}$ | $= \bar{v}_5 \times 1$ | $= 1$ |
| $\bar{v}_5 = \bar{y}$ | $= 1$ | |

(B) Reverse mode.

TABLE 2.2: Reverse Accumulation Mode.

#### 2.2.2.3 Loss Function and Feedback Mechanism

In PINNs, the loss function gains a special importance as it is where the physics of the system is encoded. In general, the loss function is a combination of the loss from the residual of the interior collocation points, $\mathcal{L}_{res}$, with the residual of the boundary points, $\mathcal{L}_{bound}$, with the *classical* $L^2$ loss from known training data (as in NNs), $\mathcal{L}_{data}$.

$$\mathcal{L}(\theta) = \omega_{res}\mathcal{L}_{res} + \omega_{bound}\mathcal{L}_{bound} + \omega_{data}\mathcal{L}_{data} \tag{2.11}$$

, where

$$\begin{cases} \mathcal{L}_{data} = \frac{1}{N_d}\sum_{i=1}^{N_d} ||\hat{\mathbf{u}}_\theta(z_i) - \mathbf{u}_i||^2 \\ \mathcal{L}_{res} = \frac{1}{N_c}\sum_{i=1}^{N_c} ||\mathbf{r}_\theta(z_i)||^2 \\ \mathcal{L}_{bound} = \frac{1}{N_b}\sum_{i=1}^{N_b} \mathcal{B}(\hat{\mathbf{u}}_\theta(z))^2 \end{cases} \tag{2.12}$$

, and $\omega_i$ are the weights that measure how important each of the terms is.

Note that we defined the loss functions $\mathcal{L}_i$ using the standard $L^2$ loss. This is the choice made in the original paper [150], and in virtually all papers since, following the common practice of using it when dealing with NN, given their smoothness and differentiability capabilities.

However, recently, some work has suggested that some applications require other metrics for the loss function. In particular, in ref[171], the authors show that, in the case of Hamilton-Jacobi-Bellman equation, a high-dimensional non-linear PDE, the $L^2$ is not suitable so solve them. In fact, stability is only guaranteed for $L^p$ distances where $p$ is sufficiently large. They showed empirically that, in this case, $L^\infty$ is a better choice.

The introduction of the weights $\omega_i$ in equation 2.11 intends to give some flexibility to the system, as different sources of loss contribute in different ways for the total error (*e.g.*, prioritize minimizing the loss on the boundary points may lead to a faster and better convergence than on the collocation points). However, these hyper-parameters are chosen at the start and the choice of values is not always clear. Furthermore, in many cases, regardless of the weights, the PINN is still too 'stiff' and it will not converge [172, 173].

In order to give the network more flexibility to improve convergence, several methods that rely on treating the weights $\omega_i$ as learning parameters have been presented [174]. All vastly improve the NN's performance. The most relevant ones are **Learning Rate Annealing** [172], **Neural Tangent Kernel Weighting** [173], **Minimax Weighting** [154] and

**Soft Attention Weighting (SA-PINNs)** [174]. The latter is the most recent method and, despite some conceptual differences, can be seen as a more general case of the others. For this reason, and given their importance in order to optimize PINNs, we will present this method in this section.

In SA-PINNs the weights in the loss function are updated via gradient descent in parallel with the NN's weights and biases. Then, equation 2.11 becomes:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}_{res}, \boldsymbol{\lambda}_{bound}, \boldsymbol{\lambda}_{data}) = \mathcal{L}_{res}(\boldsymbol{\theta}, \boldsymbol{\lambda}_{res}) + \mathcal{L}_{bound}(\boldsymbol{\theta}, \boldsymbol{\lambda}_{bound}) + \mathcal{L}_{data}(\boldsymbol{\theta}) \tag{2.13}$$

where $\boldsymbol{\lambda}_{res} = (\lambda^1_{res}, ..., \lambda^{N_c}_{res})$ and $\boldsymbol{\lambda}_{bound} = (\lambda^1_{bound}, ..., \lambda^{N_{res}}_{b})$ are trainable, non negative parameters. Here lies the main difference with the other mentioned methods: in SA-PINNs each weight corresponds to a point individually.

Equation 2.12 then becomes:

$$\begin{cases} \mathcal{L}_{data}(\boldsymbol{\theta}) = \frac{1}{N_d} \sum_{i=1}^{N_d} ||\hat{\mathbf{u}}_\theta(z_i)||^2 \\ \mathcal{L}_{res}(\boldsymbol{\theta}, \boldsymbol{\lambda}_{res}) = \sum_{i=1}^{N_c} m(\lambda^i_{res}) ||\mathbf{r}_\theta(z_i) - \mathbf{u}_i||^2 \\ \mathcal{L}_{bound}(\boldsymbol{\theta}, \boldsymbol{\lambda}_{bound}) = \sum_{i=1}^{N_b} m(\lambda^i_{bound}) \mathcal{B}(\hat{\mathbf{u}}_\theta(z))^2 \end{cases} \tag{2.14}$$

where $m(\lambda)$ is the *self-adaptation mask function*, a non-negative, differentiable, strictly increasing function of $\lambda$. Common choices for $m(\lambda)$ are polynomial functions, $m(\lambda) = c\lambda^q$, for $c, q > 0$, or sigmoidal functions. Large values of $q$ should be avoided to avoid numerical overflow, and too sharp sigmoidal functions should be avoided to avoid vanishing gradients issues.

The new objective of this PINNs variant is then:

**SA-PINNs Optimization Problem**

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \max_{\boldsymbol{\lambda}_{res}, \boldsymbol{\lambda}_{bound}} \mathcal{L}(\theta, \boldsymbol{\lambda}_{res}, \boldsymbol{\lambda}_{bound}) \tag{2.15}$$

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}_{res}, \boldsymbol{\lambda}_{bound}, \boldsymbol{\lambda}_{data}) = \mathcal{L}_{res}(\boldsymbol{\theta}, \boldsymbol{\lambda}_{res}) + \mathcal{L}_{bound}(\boldsymbol{\theta}, \boldsymbol{\lambda}_{bound}) + \mathcal{L}_{data}(\boldsymbol{\theta}) \tag{2.16}$$

The learning procedure for these new variables is done via gradient descent and further details can be found in [174].

Note that the parameters $\lambda$ have to be initiated at the beginning of training, as with other network parameters. This can be done in an uniform way (*e.g.*, setting them equal

to 1) or in a random fashion, as with NN's weights. The authors of [174] suggest a more PDE informed approach: if a specific set of data is likely to be more important for the convergence, it is useful to inform the network, by setting the corresponding $\lambda$ to be higher, and thus avoiding the gradient being *stuck* in a local minima during the learning phase.

### 2.2.3 Collocation Points Sampling

One aspect often used to exemplify the simplicity of PINNs' implementation is how elementary point distributions, like uniform distribution or random sampling, are sufficient to assure convergence in most cases. In earlier works, in fact, they have been used in similar fashion to mesh-grids in FEMs. More recently, however, the distribution of the collocation points has gained importance, as it was understood that, not only do some specific distributions provide faster convergence, but, by changing these points throughout the learning phase, proportionally to the residual function, loss can be significantly minimized.

This section will cover these two possibilities: static distributions - which remain constant throughout every epoch of learning, and adaptive residual-based distributions - which change the point distributions, either by adding new points, either by changing the location of original points, so that the NN is forced to learn *better* where the residual is greater.

A comprehensive review on sampling methods for PINNs can be found in [175].

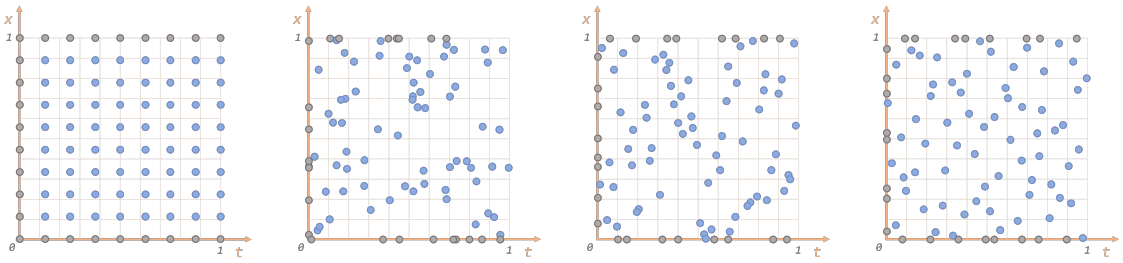
#### 2.2.3.1 Static Distributions

Of the static distributions, **uniform sampling** (placing points equispaced in the spatio-temporal domain, figure 2.5a) and **random sampling** (placing points following an uniform probability distribution across the domain, figure 2.5b) are the most simple and can be used in simpler cases. In PINNs, interior domain points and boundary points are treated alike, so they are sampled from the same distribution [*] .

It is known that, when discritizing the domain in order to evaluate numerically integrals, slightly more structured but still random distributions can perform better. Given the theoretical motivation given in section 2.2.2 of PINNs being a Monte Carlo approach to the contrained optimization problem posed, it is natural that such distributions perform well in this context and that is exactly what simulations show. For this, we chose Latin Hypercube sampling (LHS) - figure 2.5c -, and Sobol sequences - figure 2.5d.

LHS [176] is a Monte Carlo sampling method that partition the intended region into non-overlapping intervals of equal probability, following a normal distribution.

[*] in the random sampling case, typically, instead of sampling $N$ points for $x = 0$ and other $N$ points for $x = 1$, one samples $2N$ points in the intended domain, and then uses a Bernoulli distribution to randomly attribute the points to each boundary; this allows for more flexibility, at the cost of possibly having an unbalance set, as is the case of figure 2.5b, where some pieces of the boundary will not be represented well.



(A) Uniform sampling. (B) Random sampling. (C) Latin Hypercube sampling. (D) Sobol sequences.

FIGURE 2.5: The four most common static collocation points' distributions. In blue, interior domain collocation points - 64 points in the domain $(0,1] \times (0,1)$-, and, in gray, boundary collocation points (the case represented is when spatial boundaries, $x = 0$ and $x = 1$, are known and the initial boundary, at $t = 0$, is also known).

Sobol sequences [177] are a quasi-random low-discrepancy *, base-2 †, digital sequences, that fill the domain in a more structured way.

Figure 2.6, represents the average of 10 runs with the 4 sampling methods mentioned. It was used the geometric mean, as it is less sensitive to outliers (bad runs) than the arithmetic mean. It was run over 30k epochs, with $2^{11}$ main collocation points, $2^6$ points for $t = 0$ and $2^6$ for each of the two spatial boundaries. The PDE in question is BE (see section 2.2.7). The NN used had 8 hidden layers, with 20 neurons each; no regularization method was used.

Note that there is a high variance between runs: if the bottom error band was represented, there would be an overlap between all results, which does not allow to make conclusions with statistical significance (more runs would be needed). We can conclude, however, that this simulation supports other studies, that conclude that uniform sampling is the worst option, followed by random sampling, and that LHC sampling and Sobol sequences converge better than the other two [132, 150, 175]. Most literature comes to the conclusion that Sobol sequences are slightly better, while this simulation suggests the opposite. The data is too scarce, however, to make a definite conclusion on this.

A variation on the methods here described is the algorithm that is most used in PINNs' papers (that could also be included in the next section, on adaptive distributions): using the same distribution throughout learning, but re-sample all the points on each iteration (or every $N$ iteractions). This ensures a much greater robustness of NN, as more domain

*the discrepancy parameter measures how uniform or how random the sequence is.
†for this reason, Sobol sequences require that the number of points sampled is a power of 2.

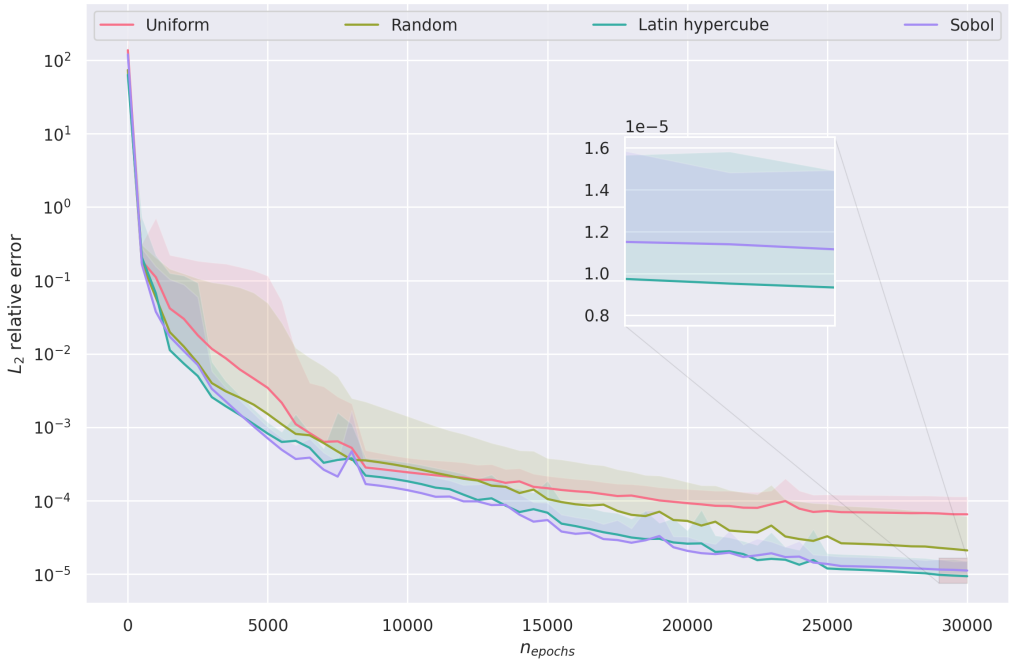



FIGURE 2.6: $L_2$ relative error for the 4 static distributions mentioned. In shade, it is represented one standard deviations over the 10 runs; given the logarithmic nature of the y-axis, the lower band of the error takes the entirety of the lower region of the graph; for clarity, we opted for only representing the top band.

will be represented in it, and provides a greater generalization quality and confers multi-scale properties to the network that truly differentiate them from traditional methods [133]. In particular, re-sampling with the random distribution as base is called *Random-R*.

#### 2.2.3.2 Adaptive Residual-based Distributions

Like in FEM, in PINNs, the natural progress from the static distributions is to use dynamic distributions, that adapt with the need to better learn some regions of the domain over others. Recent studies have supported this idea, by showing a significant improvement in performance when using non-uniform adaptive sampling [133, 178].

At the moment of writing, the main methods of adaptive residual-based sampling are **Residual-based adaptive refinement with greed (RAR-G)**, **Residual-based adaptive distribution (RAD)** and **Residual-based adaptive refinement with distribution (RAR-D)**. All three methods are based on the same idea: repeatedly adding points to the training set after one (or $k$) iterations, placing them where the residual is greater so that the NN minimize the error on that area, where the error is greater. These methods differ on how *greedy* their approach is; RAR-G focus on the points at locations with greater residuals, RAD samples the points following a certain distribution and controls the balance between

points with high residuals and points with low residuals via the use of hyper-parameters, and RAR-D is a hybrid of the other two methods, and thus is the more general approach.

An in depth discussion on this topic can be found in [175].

### 2.2.4 Activation Function

In DL, the most common activation functions used in NNs are *ReLU*, $f(x) = \max(0, x)$, and the *sigmoid function*, $\sigma(x) = \frac{1}{1+\exp(-x)}$. Other important functions are *tanh* $= \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$ or *swish* $= x \cdot \sigma(\beta x)$, where $\sigma$ is the sigmoid function, and $\beta$ can be either a constant or a trainable parameter [179].

The choice of the activation function is central for the consistency of PINNs. Several works have shown that PINNs require smooth activation functions [162, 180] (hence, *ReLU*, *ELU*, *LReLU* should not be used). A common choice for the activation function is the hyperbolic tangent, *tanh*, given that it is infinitely differentiable (and the activation function should be, at least, as many times differentiable as the order of the PDE). Note that the range of the domain is important as most of these functions are linear away from zero (normalization to the $[0, 1]$ or $[-1, 1]$ is the solution often employed) [181].

In [182], the authors show that the *swish* activation function, with $\beta$ as a trainable parameter, outperforms all other activation functions mentioned.

### 2.2.5 Optimization Method

In PINN, usually the *loss function is optimized via minibatch sampling using ADAM and/or L-BFGS (limited-memory Broyden-Fletcher-Goldfarb-Shanno)* algorithm. Tipically, intially the networks learns with *ADAM*, and then *L-BFGS* is used [183]. Different studies show that, in many cases, *L-BFGS* obtains faster convergence rate and reducing computing cost than other algorithms [181, 184].

### 2.2.6 Number of Points Sampled

While the time it takes to perform a simulation on a PINN does not scale with the number of points considered, as traditional methods do, the time it takes to train them does scale with the number of collocation and boundary points. In fact, it scales linearly, as with any other NN, $\mathcal{O}(\mathcal{N})$. It is relevant, then, to study the tradeoff between longer training with more points and the accuracy of prediction we get from the same.

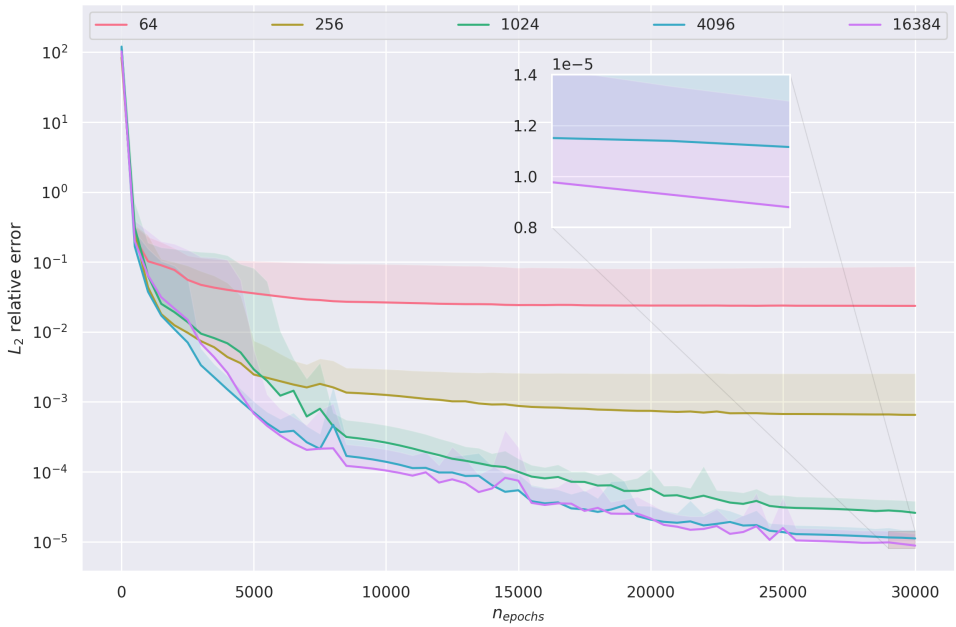



FIGURE 2.7: $L_2$ relative error for varying $N_{res}$ ($\lambda$ constant). For clarity, only represented top error shaded band.

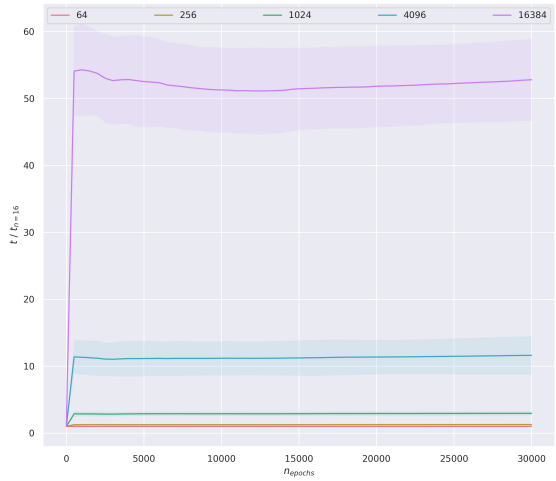
Figure 2.7 represents the difference in performance while varying $N_{res}$, averaged over than runs (geometric mean). We used $N_{res} \in \{2^6, 2^8, 2^{10}, 2^{12}, 2^{14}\}$. The number of collocation and boundary points also changed with the $N_{res}$, in such a that that $\lambda = \log_2 \frac{N_{res}}{N_{bound} \cdot N_{t=0}}$ remained constant (and equal to 1, as in an uniform grid). It was run over 30k epochs, using Sobol sequences, and on BE.

As expected, as we feed more collocation points to the network, the error decreases, but by a smaller amount each time. Figure 2.8 shows how long each of these simulations took to converge, so we can analyze the trade-off.

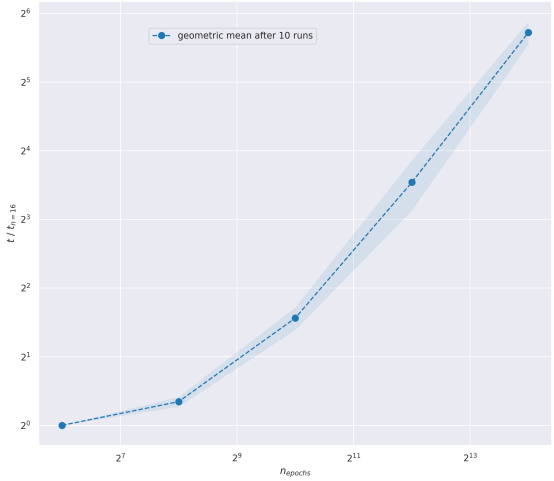
Once again, as expected, the running time grows approximately linearly with the number of collocation points. The reason this relationship is not exact - apart from random error - is that we prioritized keeping the ratio $\lambda$ constant, and thus, the total number of points does not grow linearly ($N_{res} = N_{t=0} \cdot N_{bound} = C^2$, so $N_{total} = 3C + C^2$; as $C^2 \to k \cdot C^2$, $N_{total} = k \cdot N_{total} - 3C\sqrt{k}(\sqrt{k} - 1)$, or, in this case, $N_{total} = k \cdot N_{total} - 6C$).

Table 2.3 resumes figures 2.7 and 2.8, in particular the inverse relationship aforementioned. There is no particular reason to choose one over the other, the choice should be made based on the problem in question, the accuracy intended and the availability of computing power.

Figure 2.9 shows how the $L_2$ error changes with the number of points on the boundaries, keeping the number of interior collocation points fixed, *i.e.*, varying $\lambda$. We used




(A) $t/t_{n=16}$, showing that all proportion between training per epoch for PINNs with different $N_{res}$ is approximately constant.




(B) $t/t_{n=16}$, for epoch $= 30k$, in logarithmic scale.

FIGURE 2.8: Computing time for PINNs with different $N_{res}$.

| $N_{res}$ | $L_2$ error | $\sigma$ | $\frac{L_2(n-1)}{L_2(n)}$ | $\frac{t_n}{t_{n-1}}$ |
|---|---|---|---|---|
| 64 | $2.36 \times 10^{-2}$ | $6.25 \times 10^{-2}$ | — | — |
| 256 | $6.51 \times 10^{-4}$ | $1.89 \times 10^{-3}$ | 36.22 | 1.27 |
| 1024 | $2.58 \times 10^{-5}$ | $1.21 \times 10^{-5}$ | 25.19 | 2.32 |
| 4096 | $1.12 \times 10^{-5}$ | $3.76 \times 10^{-6}$ | 2.32 | 3.94 |
| 16384 | $8.79 \times 10^{-6}$ | $4.16 \times 10^{-6}$ | 1.27 | 4.55 |

TABLE 2.3: Comparison between the performance for PINNs with different $N_{res}$.

$N_{res} = 2^{13} = 8192$, so that $\lambda \in \{-5, -3, -1, 1, 3, 5, 7, 9\}$.

Here, the total running time is less relevant than before because, despite growing with $N_{bound}$, for small values of $\lambda$, the dominating factor is $N_{res}$.

We can see in figure 2.9 that the data suggest that, besides extremely small values (negative $\lambda$), the number of points on the boundaries only slightly affects the performance, in some cases not in a linear fashion (and well within the error bars). **For this reason, we suggest choosing a small, non-negative value of $\lambda$, $\{0, 1, 2\}$ (the magnitude depending on the importance of the boundaries for the PDE in question).**

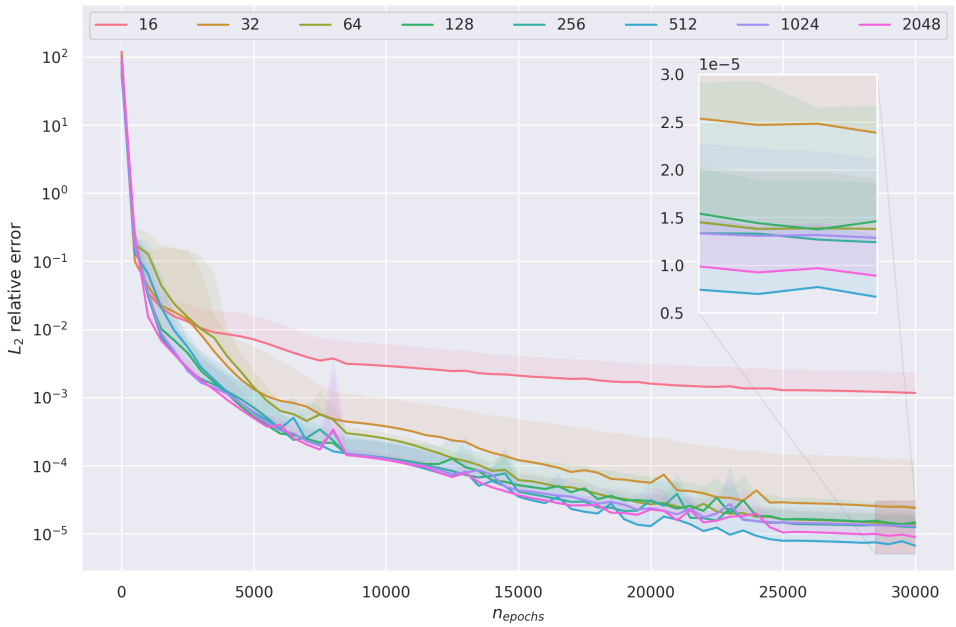



FIGURE 2.9: $L_2$ relative error for varying $\lambda$.

### 2.2.7 Solving PDEs with PINNs

As mentioned before, one of the main applications of PINNs is to solve PDEs.

Consider the following general PDE, parametrized by the parameters $\lambda$ defined on a domain $\Omega$:

$$f\left(\mathbf{x}; \frac{\partial u}{\partial x_1}, ..., \frac{\partial u}{\partial x_d}; \frac{\partial^2 u}{\partial x_1 \partial x_1}, ..., \frac{\partial^2 u}{\partial x_1 \partial x_d}; ...; \lambda\right) = 0, \quad \mathbf{x} = (x_1, ..., x_d) \in \Omega \tag{2.17}$$

, and with boundary conditions defined on a domain $\partial\Omega$:

$$\mathcal{B}(u, \mathbf{x}) = 0 \quad on \quad \partial\Omega \tag{2.18}$$

Initial conditions and boundary conditions are treated the similarly. Our goal is to find a surrogate solution $\hat{\boldsymbol{u}}(\boldsymbol{x}, t)$ that approximates the true solution for $\boldsymbol{u}(\boldsymbol{x}, t)$

The choice of PDE in PINN literature is very diverse, normally depending on the strengths of the method being studied.For low-dimensional PDEs, we suggest using BE and the WE, with the initial conditions shown in this section, as they have an analytical solution and they force the NN to adapt to *sharp* solutions and multi-scaled solutions, respectively.

The BE is the most common PDE seen in PINNs' literature. It corresponds to a one-dimensional time-depent case of the more general Navier-Stokes equations.

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}, \qquad x \in [-1,1], t \in [0,1] \tag{2.19}$$

In PINNs literature, BE is almost always solved with the following Dirichlet boundary conditions:

$$\begin{aligned} u(x,0) &= -\sin(\pi x), \qquad x \in [-1,1] \\ u(-1,t) &= u(1,t) = 0, \qquad t \in [0,1] \end{aligned} \tag{2.20}$$

This is due to the solution's sharp edge at $x = 0$ and $t >> 0$, which makes it useful to test PINNs accuracy when dealing with *shocks*. However, with this boundary conditions, the PDEs has no analytical solution, and thus, we have to obtain a numerical solution via finite elements to compare. This can be done to arbitrary precision, but, to avoid the risk of scale choosing and error propagation, we chose to solve the BE with the following (also Dirichlet) boundary conditions:

$$\begin{aligned} u(x,0) &= \frac{2\pi\nu\sin(\pi x)}{1+\epsilon+\cos(\pi x)}, \qquad x \in [0,1] \\ u(0,t) &= u(1,t) = 0, \qquad t \in [0,1] \end{aligned} \tag{2.21}$$

which has the exact solution

$$u(x,t) = \frac{2\pi\nu\sin(\pi x)\exp(-\pi^2\nu t)}{1+\epsilon+\cos(\pi x)\exp(-\pi^2\nu t)}, \qquad \epsilon > 0 \tag{2.22}$$

The parameter $\epsilon$ measures how *sharp* the solution's edge, near $x = 1$ and $t = 0$, is. To mimic the same challenge to the network as the BE with eq. 2.20, we chose a small $\epsilon$: $\epsilon = \nu = 0.01/\pi$. The plot of the solution can be seen in figure 2.10a.

The WE PDE takes the form:

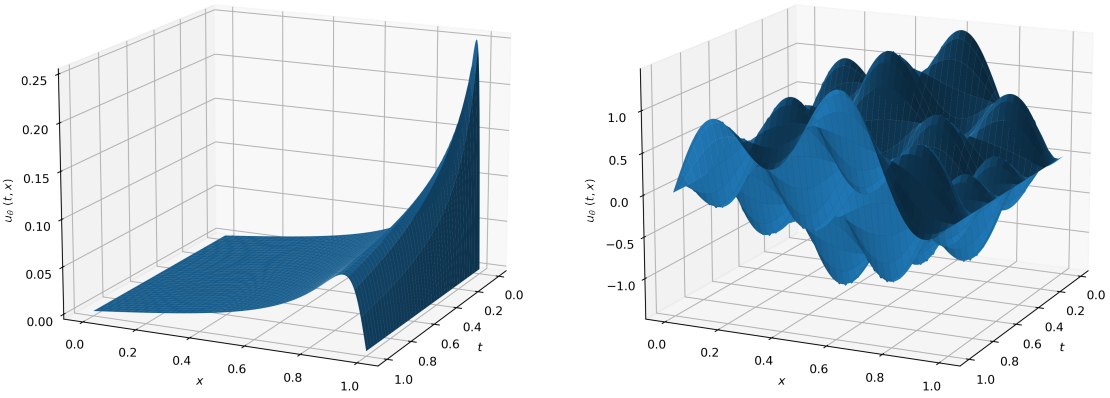
$$\frac{\partial^2 u}{\partial t^2} - 4\frac{\partial^2 u}{\partial x^2} = 0, \qquad x \in [0,1], t \in [0,1] \tag{2.23}$$

Here, we will also choose a very particular set of mixed boundary conditions, in order to test PINNs ability to find multi-scale solutions.

$$\begin{aligned} u(x,0) &= \sin(\pi x) + \frac{1}{2}\sin(4\pi x), & x \in [0,1] \\ u(0,t) &= u(1,t) = 0, & t \in [0,1] \\ \frac{\partial u}{\partial t}(x,0) &= 0, & x \in [0,1] \end{aligned} \tag{2.24}$$

This has the exact solution, represented in figure 2.10b,

$$u(x,t) = \sin(\pi x)\cos(2\pi t) + \frac{1}{2}\sin(4\pi x)\cos(8\pi t), \qquad x \in [0,1], t \in [0,1] \tag{2.25}$$




(A) Exact solution to BE (eq. 2.22). (B) Exact solution to WE (eq. 2.25).

FIGURE 2.10: Exact solutions to BE and WE.

### 2.2.8 Exotic PINNs

The original, *vanilla* version of PINN is a very general approach, more than an algorithmic method. So, it is not surprising that over the past few months, a whole *zoo* of variations of the standard PINN have been published. Some are simply improvements on the original paper, while others use the idea as a starting point to explore different applications. In this section, we present some of the most relevant PINN-derivative acronyms.

#### 2.2.8.1 Symmetry-preserving Physics-Informed Neural Networks (S-PINNs)

*S-PINNs*, introduced in 2021 in [185], extends the original concept of PINNs to equivariant NNs, namely with respect to parity and time-reversal. The authors found this formulation to be superior to the *simple* version by one or two orders of magnitude. Incorporate all different kind of symmetries and conservation laws into the network is the most relevant open problem relating to PINNs [156], and it will be a common theme in the variants mentioned in this section.

#### 2.2.8.2 Gradient-enhanced Physics-Informed Neural Networks (gPINNs)

*gPINNs*, introduced in 2021 in [186], add a new penalty term to the loss function that forces not only the PDE residual to be close to zero, but also its derivative (or more generally, its gradient):

$$\mathcal{L}(\theta) = \omega_{data}\mathcal{L}_{data} + \omega_{res}\mathcal{L}_{res} + \omega_{bound}\mathcal{L}_{bound} + \sum_{i=1}^{d} \omega_{g_i}\mathcal{L}_{g_i}(\boldsymbol{\theta}) \tag{2.26}$$

,where the new loss is

$$\mathcal{L}_{g_i}(\boldsymbol{\theta}) = \frac{1}{|\mathcal{T}_{g_i}|} \sum_{x \in \mathcal{T}_{g_i}} \left| \frac{\partial f}{\partial x_i} \right|^2 \tag{2.27}$$

The authors show that, for the same number of collocation points, gPINNs have a smaller $L_2$ error than PINNs. However, they also take longer to train due to the extra term of the loss function. When adding points to the *classical* PINN, so that the training time matches, both methods had similar performance. But, given the simplicity of the concept, we believe that it is a promising concept that should be researched further, and hence its inclusion here.

#### 2.2.8.3 Conservative Physics-Informed Neural Networks (cPINNs)

*cPINNs*, introduced in 2020 in [187], are an extension of PINNs that allow for domain decomposition into non-overlapping sub-domains, which transforms the global PDE problem to a series of local problems, each one with their own, separate NN. By enforcing interface conditions - from continuity and smoothness to the underlying conservation laws of the system -, one can guarantee to obtain a global solution consistent with the symmetries of the problem, and less prone to error pathologies, as local issues influence less the solution across the entire domain. The authors show that this formulation 'drastically increase the convergence rate'.

#### 2.2.8.4 Extended Physics-Informed Neural Networks (xPINNs)

*xPINNs* were introduced in 2020 in [188], and they further generalize the *cPINNs* previously mentioned, by being able to deal with any kind of PDE and with more complex domains, namely high-dimensional domains.

#### 2.2.8.5 Fractional Physics-Informed Neural Networks (fPINNs)

*fPINNs* were introduced in 2018 in [132], and are an extension of the original paper to cover fractional-order PDEs, in particular space-time fractional advection-diffusion equations. The approach is general, and can also be used to solve integro-differential equations. The key to the method is its hybrid approach of both AD for the integer-order operators and numerical discretization for the fractional-order ones.

#### 2.2.8.6 Bayesian Physics-Informed Neural Networks (B-PINNs)

*B-PINNs*, introduced in 2021 in [160], extend the original concept of PINNs to be able to deal with noisy data. They achieve this by making use of the Bayesian framework. As a result, they are able to quantify the *aleatoric* uncertainty of the predictions, as well as being robust to outliers.

#### 2.2.8.7 Deep Operator Network (DeepONet)

As mentioned when presenting NNs, they are not only great approximators of functions, they also approximate with remarkable success operators. *DeepONet* [189, 190] is a PINN adapted to learn non-linear operators. It constructs two sub-networks that encode the

input functions (branch net) and location variables separately (trunk part), that then are used to identify the operators. Results show that it performs remarkably well even in small data sets, and it significantly reduces the generalization error when compared with PINNs.

Other methods recently presented include Physics-Informed Adversarial Training (PIAT) [191], Weak Physics-Informed neural networks (wPINNs) [192] and Variational Energy based PINNs (VE-PINNs) [193].

## 2.3 Fluid Mechanics

One of the main applications of PINNs is Computational Fluid Dynamics (CFD). The multi-scale, chaotic, turbulent behaviour of fluid dynamics makes them a prime target of PIML, given its unique capability to deal with high-dimensional systems, noisy data, and inverse flow problems (like unknown boundary conditions) [136, 194].

Despite the immense progress made over the past decades in solving numerically compressible and incompressible flow fields given by the NSEs, traditional methods, such as FEMs or spectral methods, rely either on precise global measures of the fluid's velocity and pressure or on defined domain geometry and boundary conditions. In many cases, this is not feasible: often the problem is ill-defined, and solving all relevant dissipation scales in a turbulent flow is simply impossible, even with the most powerful computers. Studies have shown that PINNs can successufully tackle these cases, both for incompressible flows [195] and compressible flows [196]. Note that PINNs and traditional methods excel in mostly disjoint sets of problems, and, thus, progress in CFD is mostly likely dependent on the cooperation of both approaches for a full understanding of these complex phenomena. The usefulness and limitations of PINNs when compared with traditional methods is best presented in [136]:

> "PINNs are not meant to be a replacement of the existing CFD codes, and in fact the current generation of PINNs is not as accurate or as efficient as high-order CFD codes [197] for solving the standard forward problems. This limitation is associated with the minimization of the loss function, which is a high-dimensional non-convex function, a limitation which is a grand challenge of all neural networks for even commercial machine learning. However, PINNs

> perform much more accurately and more efficiently than any CFD solver if any scattered partial spatio-temporal data are available for the flow problem under consideration. Moreover, the forward and inverse PINN formulations are identical so there is no need for expensive data assimilation schemes that have stalled progress especially for optimization and design applications of flow problems in the past."

Applications of PINNs to CFD are many and varied. From high-resolution reconstruction of flow-field data from low-resolution and noisy measurements [198], to solving Reynolds-averaged NSEs for incompressible turbulent flows [199], to using non-linear manifold reduced order models to better approximate high-fidelity model solutions with a smaller latent space dimension [200].

Although training a NN is typically quite slow, approaches like the ones mentioned also allow for the use of techniques like transfer learning. For example, one can use a NN that has been trained for a small Reynolds number to make predictions for a flow with a higher Reynolds number. Furthermore, important efforts have been made to make it possible to infer velocity and pressure fields solely from the knowledge of the time-evolution of a passive scaler [135].

One of the most surprising applications of PINNs relates to the most fundamental problem of mathematical fluid mechanics: whether initially smooth solutions of the 3D incompressible Euler equations can form a singularity in finite time (*blow up*) [201]. The concept of using numerical computational simulations to solve exactly a theoretical problem is not new; in fact, a simpler version of this same problem - the one dimensional version - was solved via *computer-assisted proof* [202]. In a similar approach, and building on previous work that hinted that a particular case of flow solutions - periodic cylinder with solid boundaries - might *blow up* in finite time [203], the authors of [204] used PINNs to find smooth self-similar solution that could be the basis for the so sought after proof of the problem.

For a more complete reviews on ML methods for CFD (not exclusively PINNs) we suggest reading [136, 194, 205].

## 2.4 Closing Remarks

As a very recent area, new papers are being published every week that expand the ideas of the original PINN paper and further our understanding of this still not very well understood area. Due to this lack of a complete foundational theory supporting it, the results being published and presented in this chapter are mostly empirical. For this reason, it is the author's understanding that it is important to define concrete benchmarks across PIML, as a first step for a framework of PINNs. It does not suffice to compare convergence rates in well-behaved PDEs (say, Burger's equation), as some struggle, for example, with discontinuous solutions of PDEs such as nonlinear hyperbolic equations. With so many different methods with different strengths and weaknesses, how to choose the right one for each task and how to choose the properties of each is a difficult task at the moment.

We hope that this work serve as a state-of-the-art to help progress this discussion, and as a tool to direct researchers working with this algorithm to the right resources.

# Bibliography


[1] J. F. Pinto da Costa and M. Cabral, "Statistical methods with applications in data mining: A review of the most recent works," *Mathematics*, vol. 10, no. 6, 2022. [Online]. Available: https://www.mdpi.com/2227-7390/10/6/993 [Cited on page 1.]

[2] W. Krämer, "Stephen t. ziliak and deirdre n. mccloskey, the cult of statistical significance: how the standard error costs us jobs. justice and lives," *Statistical Papers*, vol. 53, pp. 243–244, 01 2009. [Cited on page 2.]

[3] R. Wasserstein, A. Schirm, and N. Lazar, "Moving to a world beyond $p < 0.05$," *American Statistician*, vol. 73, pp. 1–19, 2019. [Cited on page 3.]

[4] D. Benjamin, J. Berger, M. Johannesson, B. Nosek, E.-J. Wagenmakers, R. Berk, K. Bollen, B. Brembs, L. Brown, C. Camerer, D. Cesarini, C. Chambers, M. Clyde, T. Cook, P. De Boeck, Z. Dienes, A. Dreber, K. Easwaran, C. Efferson, and V. Johnson, "Redefine statistical significance," *Nature Human Behaviour*, vol. 2, 09 2017. [Cited on page 3.]

[5] B. McShane, D. Gal, A. Gelman, C. Robert, and J. Tackett, "Abandon statistical significance," *American Statistician*, vol. 73, pp. 235–245, 2019. [Cited on page 3.]

[6] Q. F. Gronau, H. Singmann, and E.-J. Wagenmakers, "bridgesampling : An r package for estimating normalizing constants," *Journal of Statistical Software*, vol. 92, 2020, 20 citations. [Online]. Available: http://www.jstatsoft.org/v92/i10/ [Cited on page 3.]

[7] Y. Liu and J. Xie, "Cauchy combination test: A powerful test with analytic p-value calculation under arbitrary dependency structures," *Journal of the American Statistical Association*, vol. 115, pp. 393–402, 2020. [Cited on page 3.]

[8] J. J. Dziak, D. L. Coffman, S. T. Lanza, R. Li, and L. S. Jermiin, "Sensitivity and specificity of information criteria," *Briefings in Bioinformatics*, vol. 21, no. 2, pp. 553–565, 03 2019. [Online]. Available: https://doi.org/10.1093/bib/bbz016 [Cited on page 3.]

[9] J. Liu, W. Zhong, and R. Li, "A selective overview of feature screening for ultrahigh-dimensional data," *Science China Mathematics*, vol. 58, no. 10, 2015, cited By 35. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84942372194&doi=10.1007%2fs11425-015-5062-9&partnerID=40&md5=d1740291ecbf6498ccb134f020c4bbec [Cited on page 3.]

[10] D. Bertsimas and B. van Parys, "Sparse high-dimensional regression: Exact scalable algorithms and phase transitions," *Annals of Statistics*, vol. 48, pp. 300–323, 2020. [Cited on page 5.]

[11] X. Chen, W. Liu, X. Mao, and Z. Yang, "Distributed high-dimensional regression under a quantile loss function," *Journal of Machine Learning Research*, vol. 21, 2020. [Cited on page 5.]

[12] J. Fan and J. Lv, "Sure independence screening for ultrahigh dimensional feature space," *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, vol. 70, no. 5, pp. 849–911, 2008, cited By 1291. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-53849086824&doi=10.1111%2fj.1467-9868.2008.00674.x&partnerID=40&md5=9e192deaf6073173c9b810792b7e3783 [Cited on page 6.]

[13] D. Nandy, F. Chiaromonte, and R. Li, "Covariate information number for feature screening in ultrahigh-dimensional supervised problems," *Journal of the American Statistical Association*, 2021, cited By 0. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85101058900&doi=10.1080%2f01621459.2020.1864380&partnerID=40&md5=c2021d754a70c88f71e06f9a70a54fc6 [Cited on page 6.]

[14] T. Zhou, L. Zhu, C. Xu, and R. Li, "Model-free forward screening via cumulative divergence," *Journal of the American Statistical Association*, vol. 115, no. 531, pp. 1393–1405, 2020, cited By 7. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=

2-s2.0-85090101443&doi=10.1080%2f01621459.2019.1632078&partnerID=40&md5=8484c99d17cd3c0e25dc63fed7c1c516 [Cited on page 6.]

[15] O. Ledoit and M. Wolf, "Analytical nonlinear shrinkage of large-dimensional covariance matrices," *Annals of Statistics*, vol. 48, pp. 3043–3065, 2020. [Cited on page 6.]

[16] H. Sifaou, A. Kammoun, and M.-S. Alouini, "High-dimensional linear discriminant analysis classifier for spiked covariance model ," *Journal of Machine Learning Research*, vol. 21, 2020. [Cited on page 6.]

[17] E. Fang, Y. Ning, and R. Li, "Test of significance for high-dimensional longitudinal data," *Annals of Statistics*, vol. 48, no. 5, pp. 2622–2645, 2020, cited By 3. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85092320223&doi=10.1214%2f19-AOS1900&partnerID=40&md5=aa517bd4193800d09d58dfbd475f3b7e [Cited on page 6.]

[18] Z. Cai, R. Li, and L. Zhu, "Online sufficient dimension reduction through sliced inverse regression," *Journal of Machine Learning Research*, vol. 21, 2020, cited By 7. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85086798987&partnerID=40&md5=f1e5240efc5b486ca05c9736c9ef538b [Cited on page 7.]

[19] C. Lu, J. Feng, Y. Chen, W. Liu, Z. Lin, and S. Yan, "Tensor robust principal component analysis with a new tensor nuclear norm," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, pp. 925–938, 2020. [Cited on page 7.]

[20] X. Zhang, D. Wang, Z. Zhou, and Y. Ma, "Robust low-rank tensor recovery with rectification and alignment," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, pp. 238–255, 2021. [Cited on page 7.]

[21] S. Volgushev, S.-K. Chao, and G. Cheng, "Distributed inference for quantile regression processes," *Annals of Statistics*, vol. 47, pp. 1634–1662, 2019. [Cited on page 7.]

[22] T. Liang and A. Rakhlin, "Just interpolate: Kernel ridgeless regression can generalize," *Annals of Statistics*, vol. 48, pp. 1329–1347, 2020. [Cited on page 8.]

[23] E. Candès and P. Sur, "The phase transition for the existence of the maximum likelihood estimate in high-dimensional logistic regression," *Annals of Statistics*, vol. 48, pp. 27–42, 2020. [Cited on page 8.]

[24] T. Hothorn, "Most likely transformations: The mlt package," *Journal of Statistical Software*, vol. 92, 2020. [Cited on page 8.]

[25] A. Solin and S. Särkkä, "Hilbert space methods for reduced-rank gaussian process regression," *Statistics and Computing*, vol. 30, pp. 419–446, 2020. [Cited on page 8.]

[26] A. Gelman, B. Goodrich, J. Gabry, and A. Vehtari, "R-squared for bayesian regression models," *American Statistician*, vol. 73, pp. 307–309, 2019. [Cited on page 8.]

[27] Y. Feng, J. Fan, and J. Suykens, "A statistical learning approach to modal regression," *Journal of Machine Learning Research*, vol. 21, 2020. [Cited on page 8.]

[28] Q. Sun, W.-X. Zhou, and J. Fan, "Adaptive huber regression," *Journal of the American Statistical Association*, vol. 115, pp. 254–265, 2020. [Cited on page 9.]

[29] G. Wang, A. Sarkar, P. Carbonetto, and M. Stephens, "A simple new approach to variable selection in regression, with application to genetic fine mapping," *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, vol. 82, pp. 1273–1300, 2020. [Cited on page 9.]

[30] S. Kwon, S. Lee, and Y. Kim, "Moderately clipped lasso," *Computational Statistics Data Analysis*, vol. 92, pp. 53–67, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167947315001589 [Cited on page 9.]

[31] B. Gaye, D. Zhang, and A. Wulamu, "Improvement of support vector machine algorithm in big data background," *Mathematical Problems in Engineering*, vol. 2021, p. 5594899, Jun 2021. [Online]. Available: https://doi.org/10.1155/2021/5594899 [Cited on page 10.]

[32] J. Chang, H. Moon, and S. Kwon, "High-dimensional linear discriminant analysis with moderately clipped lasso," *Communications for Statistical Applications and Methods*, vol. 28, no. 1, pp. 21–37, Jan 2021. [Online]. Available: http://http://www.csam.or.kr/journal/view.html?doi=10.29220/CSAM.2021.28.1.021 [Cited on page 10.]

[33] M. Tanveer, T. Rajani, R. Rastogi, and Y. Shao, "Comprehensive review on twin support vector machines," 05 2021. [Cited on page 10.]

[34] S. Athey, J. Tibshirani, and S. Wager, "Generalized random forests," *Annals of Statistics*, vol. 47, pp. 1179–1203, 2019. [Cited on page 11.]

[35] J. Hill, A. Linero, and J. Murray, "Bayesian additive regression trees: A review and look forward," *Annual Review of Statistics and Its Application*, vol. 7, pp. 251–278, 2020. [Cited on page 11.]

[36] T. Berrett, R. Samworth, and M. Yuan, "Efficient multivariate entropy estimation via k-nearest neighbour distances," *Annals of Statistics*, vol. 47, pp. 288–318, 2019. [Cited on page 12.]

[37] T. Miyato, S.-I. Maeda, M. Koyama, and S. Ishii, "Virtual adversarial training: A regularization method for supervised and semi-supervised learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, pp. 1979–1993, 2019. [Cited on page 12.]

[38] Y. Xian, C. Lampert, B. Schiele, and Z. Akata, "Zero-shot learning-a comprehensive evaluation of the good, the bad and the ugly," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, pp. 2251–2265, 2019. [Cited on page 13.]

[39] G. Lecué and M. Lerasle, "Robust machine learning by median-of-means: Theory and practice," *Annals of Statistics*, vol. 48, pp. 906–931, 2020. [Cited on page 13.]

[40] M. Cattaneo, M. Jansson, and X. Ma, "Simple local polynomial density estimators," *Journal of the American Statistical Association*, vol. 115, pp. 1449–1455, 2020. [Cited on page 13.]

[41] B. Efron, "Prediction, estimation, and attribution," *Journal of the American Statistical Association*, vol. 115, pp. 636–655, 2020. [Cited on pages xv, 13, and 14.]

[42] D. Apley and J. Zhu, "Visualizing the effects of predictor variables in black box supervised learning models," *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, vol. 82, pp. 1059–1086, 2020. [Cited on page 14.]

[43] T. Hardwicke, S. Serghiou, P. Janiaud, V. Danchev, S. Crüwell, S. Goodman, and J. Ioannidis, "Calibrating the scientific ecosystem through meta-research," *Annual Review of Statistics and Its Application*, vol. 7, pp. 11–37, 2020. [Cited on page 15.]

[44] S. Mitchell, E. Potash, S. Barocas, A. D'Amour, and K. Lum, "Algorithmic fairness: Choices, assumptions, and definitions," *Annual Review of Statistics and Its Application*, vol. 8, pp. 141–163, 2021. [Cited on page 15.]

[45] P. Sharma, G. Shmueli, M. Sarstedt, N. Danks, and S. Ray, "Prediction-oriented model selection in partial least squares path modeling," *Decision Sciences*, vol. 52, pp. 567–607, 2021. [Cited on page 15.]

[46] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953. [Online]. Available: https://doi.org/10.1063/1.1699114 [Cited on page 15.]

[47] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, 04 1970. [Online]. Available: https://doi.org/10.1093/biomet/57.1.97 [Cited on page 15.]

[48] J. Bierkens, P. Fearnhead, and G. Roberts, "The zig-zag process and super-efficient sampling for bayesian analysis of big data ¡sup¿1¡/sup¿," *Annals of Statistics*, vol. 47, pp. 1288–1320, 2019. [Cited on page 15.]

[49] J. Bierkens and G. Roberts, "A piecewise deterministic scaling limit of lifted Metropolis–Hastings in the Curie–Weiss model," *The Annals of Applied Probability*, vol. 27, no. 2, pp. 846 – 882, 2017. [Online]. Available: https://doi.org/10.1214/16-AAP1217 [Cited on page 15.]

[50] E. Abbe, J. Fan, K. Wang, and Y. Zhong, "Entrywise eigenvector analysis of random matrices with low expected rank," *Annals of Statistics*, vol. 48, pp. 1452–1474, 2020. [Cited on page 16.]

[51] Z. Li and S. Wood, "Faster model matrix crossproducts for large generalized linear models with discretized covariates," *Statistics and Computing*, vol. 30, pp. 19–25, 2020. [Cited on page 16.]

[52] K. Boudt, P. Rousseeuw, S. Vanduffel, and T. Verdonck, "The minimum regularized covariance determinant estimator," *Statistics and Computing*, vol. 30, pp. 113–128, 2020. [Cited on page 16.]

[53] S. Kazemi, R. Goel, K. Jain, I. Kobyzev, A. Sethi, P. Forsyth, and P. Poupart, "Representation learning for dynamic graphs: A survey," *Journal of Machine Learning Research*, vol. 21, 2020. [Cited on pages 16 and 17.]

[54] W. Yu, Y. Zhang, Z. Chen, and T. Ai, "Sparse reconstruction with spatial structures to automatically determine neighbors," *International Journal of Geographical Information Science*, vol. 36, no. 2, pp. 338–359, 2022. [Online]. Available: https://doi.org/10.1080/13658816.2021.1885675 [Cited on page 17.]

[55] J. Fan, R. Li, C.-H. Zhang, and H. Zou, *Statistical Foundations of Data Science (1st ed.)*. Chapman and Hall/CRC, 2020, https://doi.org/10.1201/9780429096280. [Cited on pages 18 and 25.]

[56] J. Fan, C. Ma, and Y. Zhong, "A selective overview of deep learning," 2019. [Cited on page 18.]

[57] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996. [Cited on page 18.]

[58] M. Zeng, Y. Liao, R. Li, and A. Sudjianto, "Local linear approximation algorithm for neural network," *Mathematics*, vol. 10, no. 3, 2022. [Online]. Available: https://www.mdpi.com/2227-7390/10/3/494 [Cited on page 19.]

[59] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608014002135 [Cited on page 19.]

[60] P. L. Bartlett, A. Montanari, and A. Rakhlin, "Deep learning: a statistical viewpoint," *Acta Numerica*, vol. 30, p. 87–201, 2021. [Cited on pages 19 and 20.]

[61] Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, pp. 3212–3232, 2019. [Cited on page 20.]

[62] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image segmentation using deep learning: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021, cited By 113. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=

2-s2.0-85100948197&doi=10.1109%2fTPAMI.2021.3059968&partnerID=40&md5=115a97279a77fb75cf00d20ca1578417 [Cited on page 20.]

[63] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. [Cited on page 21.]

[64] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org. [Cited on page 21.]

[65] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf [Cited on page 21.]

[66] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013. [Online]. Available: http://arxiv.org/abs/1311.2524 [Cited on page 22.]

[67] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. [Online]. Available: http://arxiv.org/abs/1506.02640 [Cited on page 22.]

[68] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015. [Online]. Available: http://arxiv.org/abs/1512.02325 [Cited on page 22.]

[69] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal loss for dense object detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, pp. 318–327, 2020. [Cited on page 22.]

[70] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, pp. 386–397, 2020. [Cited on page 22.]

[71] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015. [Online]. Available: http://arxiv.org/abs/1504.08083 [Cited on page 22.]

[72] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015. [Online]. Available: http://arxiv.org/abs/1506.01497 [Cited on page 22.]

[73] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *CoRR*, vol. abs/1411.4038, 2014. [Online]. Available: http://arxiv.org/abs/1411.4038 [Cited on page 22.]

[74] Z. Liu, H. Hu, Y. Lin, Z. Yao, Z. Xie, Y. Wei, J. Ning, Y. Cao, Z. Zhang, L. Dong, F. Wei, and B. Guo, "Swin transformer V2: scaling up capacity and resolution," *CoRR*, vol. abs/2111.09883, 2021. [Online]. Available: https://arxiv.org/abs/2111.09883 [Cited on page 23.]

[75] S.-H. Gao, M.-M. Cheng, K. Zhao, X.-Y. Zhang, M.-H. Yang, and P. Torr, "Res2net: A new multi-scale backbone architecture," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, pp. 652–662, 2021. [Cited on page 23.]

[76] J. Kossaifi, Z. Lipton, A. Kolbeinsson, A. Khanna, T. Furlanello, and A. Anandkumar, "Tensor regression networks," *Journal of Machine Learning Research*, vol. 21, 2020. [Cited on page 24.]

[77] X. Zhang, M. Fan, D. Wang, P. Zhou, and D. Tao, "Top-k feature selection framework using robust 0-1 integer programming," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, pp. 3005–3019, 2021. [Cited on page 24.]

[78] Z. Li, J. Yang, Z. Liu, X. Yang, G. Jeon, and W. Wu, "Feedback network for image super-resolution," vol. 2019-June, 2019, pp. 3862–3871, cited By 271. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85074492188&doi=10.1109%2fCVPR.2019.00399&partnerID=40&md5=69cc956c19c7c0ab44b42e7df3d282ae [Cited on page 25.]

[79] Z. Berradi, M. Lazaar, H. Omara, and O. Mahboub, "Effect of architecture in recurrent neural network applied on the prediction of stock price," *IAENG International Journal of Computer Science*, vol. 47, no. 3, pp. 436–441, 2020, cited By 4. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85089796244&partnerID=40&md5=87bd2abbaf6380d018273d6e5bb94318 [Cited on page 25.]

[80] J. Serrano-Pérez, G. Fernández-Anaya, S. Carrillo-Moreno, and W. Yu, "New results for prediction of chaotic systems using deep recurrent neural networks," *Neural Processing Letters*, vol. 53, no. 2, pp. 1579–1596, 2021, cited By 2. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85102234476&doi=10.1007%2fs11063-021-10466-1&partnerID=40&md5=726c0eb4003f9129a686e63050b2f059 [Cited on page 25.]

[81] G. Chadha, A. Panambilly, A. Schwung, and S. Ding, "Bidirectional deep recurrent neural networks for process fault classification," *ISA Transactions*, vol. 106, pp. 330–342, 2020, cited By 11. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85087942640&doi=10.1016%2fj.isatra.2020.07.011&partnerID=40&md5=9f3b5ec281f9d6470ec02034c1165e57 [Cited on page 25.]

[82] A. Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network," *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167278919305974 [Cited on page 25.]

[83] A. Gogna and A. Majumdar, "Discriminative autoencoder for feature extraction: Application to character recognition," *Neural Processing Letters*, vol. 49, no. 3, pp. 1723–1735, 2019, cited By 18. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85050795257&doi=10.1007%2fs11063-018-9894-5&partnerID=40&md5=d539da6a743314a39c35aeb41fe89dd0 [Cited on page 25.]

[84] P.-Y. Chen and J.-J. Huang, "A hybrid autoencoder network for unsupervised image clustering," *Algorithms*, vol. 12, no. 6, 2019, cited By 6. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85077464645&doi=10.3390%2fa12060122&partnerID=40&md5=218f7c3f98909d7cbbb04737db854d01 [Cited on page 25.]

[85] G. Eraslan, L. Simon, M. Mircea, N. Mueller, and F. Theis, "Single-cell rna-seq denoising using a deep count autoencoder," *Nature Communications*, vol. 10, no. 1, 2019, cited By 196. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=

2-s2.0-85060401600&doi=10.1038%2fs41467-018-07931-2&partnerID=40&md5=21b9d7b9feebb8f5e8cbf3f7972e05a1 [Cited on page 25.]

[86] M. Ferrag, L. Maglaras, S. Moschoyiannis, and H. Janicke, "Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study," *Journal of Information Security and Applications*, vol. 50, 2020, cited By 169. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85076848850&doi=10.1016%2fj.jisa.2019.102419&partnerID=40&md5=906855f749daaa4b583980474be8fdc5 [Cited on page 25.]

[87] D. Berman, A. Buczak, J. Chavis, and C. Corbett, "A survey of deep learning methods for cyber security," *Information (Switzerland)*, vol. 10, no. 4, 2019, cited By 130. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85065867536&doi=10.3390%2finfo10040122&partnerID=40&md5=fac83292d3a127514bb7e9f824aafe64 [Cited on page 25.]

[88] A. Myronenko, "3d mri brain tumor segmentation using autoencoder regularization," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11384 LNCS, pp. 311–320, 2019, cited By 167. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85063482275&doi=10.1007%2f978-3-030-11726-9_28&partnerID=40&md5=72bffdf30a9256c75be69f43ecafd89b [Cited on page 25.]

[89] D.-T. Hoang and H.-J. Kang, "A survey on deep learning based bearing fault diagnosis," *Neurocomputing*, vol. 335, pp. 327–335, 2019, cited By 180. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85056325395&doi=10.1016%2fj.neucom.2018.06.078&partnerID=40&md5=03b33357c05cd62c38691663942959f1 [Cited on page 25.]

[90] T. Finke, M. Krämer, A. Morandini, A. Mück, and I. Oleksiyuk, "Autoencoders for unsupervised anomaly detection in high energy physics," *Journal of High Energy Physics*, vol. 2021, no. 6, Jun 2021. [Online]. Available: http://dx.doi.org/10.1007/JHEP06(2021)161 [Cited on page 25.]

[91] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014. [Cited on page 25.]

[92] M. Mehralian and B. Karasfi, "Rdcgan: Unsupervised representation learning with regularized deep convolutional generative adversarial networks," in *2018 9th Conference on Artificial Intelligence and Robotics and 2nd Asia-Pacific International Symposium*, 2018, pp. 31–38. [Cited on page 26.]

[93] A. Waheed, M. Goyal, D. Gupta, A. Khanna, F. Al-Turjman, and P. Pinheiro, "Covidgan: Data augmentation using auxiliary classifier gan for improved covid-19 detection," *IEEE Access*, vol. 8, pp. 91 916–91 923, 2020, cited By 185. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85085557685&doi=10.1109%2fACCESS.2020.2994762&partnerID=40&md5=6b60dfb2a4684573e8bc8abe403e00ae [Cited on page 26.]

[94] M. Jamshidi, A. Lalbakhsh, J. Talla, Z. Peroutka, F. Hadjilooei, P. Lalbakhsh, M. Jamshidi, L. Spada, M. Mirmozafari, M. Dehghani, A. Sabet, S. Roshani, S. Roshani, N. Bayat-Makou, B. Mohamadzade, Z. Malek, A. Jamshidi, S. Kiani, H. Hashemi-Dezaki, and W. Mohyuddin, "Artificial intelligence and covid-19: Deep learning approaches for diagnosis and treatment," *IEEE Access*, vol. 8, pp. 109 581–109 595, 2020, cited By 126. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85087668610&doi=10.1109%2fACCESS.2020.3001973&partnerID=40&md5=a1b1053892365ba335791cdc2ff13cc4 [Cited on page 26.]

[95] J. Wang, B. Han, H. Bao, M. Wang, Z. Chu, and Y. Shen, "Data augment method for machine fault diagnosis using conditional generative adversarial networks," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, vol. 234, no. 12, pp. 2719–2727, 2020, cited By 11. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85086039710&doi=10.1177%2f0954407020923258&partnerID=40&md5=19c40606c555042aeee98ca2206d5ffd [Cited on page 26.]

[96] F. Zhou, S. Yang, H. Fujita, D. Chen, and C. Wen, "Deep learning fault diagnosis method based on global optimization gan for

unbalanced data," *Knowledge-Based Systems*, vol. 187, 2020, cited By 122. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85068868626&doi=10.1016%2fj.knosys.2019.07.008&partnerID=40&md5=9effe0c57705b8feacacf837341d6e30 [Cited on page 26.]

[97] H. Zhang, V. Sindagi, and V. Patel, "Image de-raining using a conditional generative adversarial network," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 11, pp. 3943–3956, 2020, cited By 129. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85095684743&doi=10.1109%2fTCSVT.2019.2920407&partnerID=40&md5=dd6483eddc8b36e3165b0e4722ecbba0 [Cited on page 26.]

[98] J. Cheng, Y. Yang, X. Tang, N. Xiong, Y. Zhang, and F. Lei, "Generative adversarial networks: A literature review," *KSII Transactions on Internet and Information Systems*, vol. 14, no. 12, pp. 4625–4647, 2020, cited By 4. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85099381593&doi=10.3837%2ftiis.2020.12.001&partnerID=40&md5=8faca2e2946b38e13849ca660544d721 [Cited on page 26.]

[99] Prabhat, Nishant, and D. Vishwakarma, "Comparative analysis of deep convolutional generative adversarial network and conditional generative adversarial network using hand written digits," 2020, pp. 1072–1075, cited By 5. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85087443365&doi=10.1109%2fICICCS48265.2020.9121178&partnerID=40&md5=19779e422a44b00c933fd1f3c57da81e [Cited on page 26.]

[100] T. Mukhiddin, W. Lee, S. Lee, and T. Rashid, "Research issues on generative adversarial networks and applications," 2020, pp. 487–488, cited By 3. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85084368269&doi=10.1109%2fBigComp48618.2020.00-19&partnerID=40&md5=33eeaf89e9b8d6cd60f38ba05dd0498e [Cited on page 26.]

[101] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, pp. 4–24, 2021. [Cited on page 26.]

[102] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017. [Cited on page 26.]

[103] Y. Li, "Deep reinforcement learning: An overview," *CoRR*, vol. abs/1701.07274, 2017. [Online]. Available: http://arxiv.org/abs/1701.07274 [Cited on page 26.]

[104] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010. [Cited on page 27.]

[105] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, 2020. [Cited on page 27.]

[106] B. Samanta, A. De, G. Jana, V. Gomez, P. Chattaraj, N. Ganguly, and M. Gomez-Rodriguez, "Nevae: A deep generative model for molecular graphs," *Journal of Machine Learning Research*, vol. 21, 2020. [Cited on page 27.]

[107] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019. [Cited on page 27.]

[108] S. Cuomo, V. S. D. Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, "Scientific machine learning through physics-informed neural networks: Where we are and what's next," *CoRR*, vol. abs/2201.05624, 2022. [Online]. Available: https://arxiv.org/abs/2201.05624 [Cited on page 27.]

[109] K. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton, "Data-driven discovery of coordinates and governing equations," *Proceedings of the National Academy of Sciences*, vol. 116, no. 45, pp. 22 445–22 451, 2019. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.1906995116 [Cited on page 27.]

[110] B. Lusch, J. N. Kutz, and S. L. Brunton, "Deep learning for universal linear embeddings of nonlinear dynamics," *Nature Communications*, vol. 9, no. 1, p. 4950, Nov 2018. [Online]. Available: https://doi.org/10.1038/s41467-018-07210-0 [Cited on page 27.]

[111] S. Brunton, M. Budišić, E. Kaiser, and J. Kutz, "Modern koopman theory for dynamical systems," 02 2021. [Cited on page 27.]

[112] J. Hou, S. Wang, Y. Lai, J. Lin, Y. Tsao, H. Chang, and H. Wang, "Audio-visual speech enhancement based on multimodal deep convolutional neural network," *CoRR*, vol. abs/1703.10893, 2017. [Online]. Available: http://arxiv.org/abs/1703.10893 [Cited on page 27.]

[113] Y. Xu, Q. Kong, Q. Huang, W. Wang, and M. D. Plumbley, "Convolutional gated recurrent neural network incorporating spatial features for audio tagging," *CoRR*, vol. abs/1702.07787, 2017. [Online]. Available: http://arxiv.org/abs/1702.07787 [Cited on page 27.]

[114] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, and R. Pang, "Conformer: Convolution-augmented transformer for speech recognition," vol. 2020-October, 2020, pp. 5036–5040, cited By 141. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85097919414&doi=10.21437%2fInterspeech.2020-3015&partnerID=40&md5=518f1dea3acf6d5e0b441df0301b810a [Cited on page 27.]

[115] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, Sep. 2015, pp. 1412–1421. [Online]. Available: https://aclanthology.org/D15-1166 [Cited on page 27.]

[116] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," *CoRR*, vol. abs/1609.08144, 2016. [Online]. Available: http://arxiv.org/abs/1609.08144 [Cited on page 27.]

[117] C. Zhang, H. Fu, Q. Hu, X. Cao, Y. Xie, D. Tao, and D. Xu, "Generalized latent multi-view subspace clustering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, pp. 86–99, 2020. [Cited on page 28.]

[118] Q. Wang, M. Chen, F. Nie, and X. Li, "Detecting coherent groups in crowd scenes by multiview clustering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, pp. 46–58, 2020. [Cited on page 28.]

[119] J. da Costa and A. Garcia, "New confinement index and new perspective for comparing countries- COVID-19," *COMPUTER METHODS AND PROGRAMS IN BIOMEDICINE*, vol. 210, p. 106346, 2021, citations: crossref, scopus, unpaywall, wos. [Cited on page 29.]

[120] J. F. P. da Costa, F. Ferreira, M. Mascarello, and R. Gaio, "Clustering of Longitudinal Trajectories Using Correlation-Based Distances," *SN Computer Science*, vol. 2, no. 6, 2021, citations: crossref, unpaywall. [Cited on page 29.]

[121] V. Arya, R. Bellamy, P.-Y. Chen, A. Dhurandhar, M. Hind, S. Hoffman, S. Houde, Q. Liao, R. Luss, A. Mojsilović, D. Wei, and Y. Zhang, "Ai explainability 360: An extensible toolkit for understanding data and machine learning models," *Journal of Machine Learning Research*, vol. 21, 2020. [Cited on page 29.]

[122] R. Tavenard, J. Faouzi, G. Vandewiele, F. Divo, G. Androz, C. Holtz, M. Payne, R. Yurchak, M. Rußwurm, K. Kolar, K. Kolar, and E. Woods, "Tslearn, a machine learning toolkit for time series data," *Journal of Machine Learning Research*, vol. 21, 2020. [Cited on page 29.]

[123] J. Faouzi and H. Janati, "Pyts: A python package for time series classification," *Journal of Machine Learning Research*, vol. 21, 2020. [Cited on page 29.]

[124] A. Alexandrov, K. Benidis, M. Bohlke-Schneider, V. Flunkert, J. Gasthaus, T. Januschowski, D. Maddix, S. Rangapuram, D. Salinas, J. Schulz, A. Türkmen, and Y. Wang, "Gluonts: Probabilistic and neural time series modeling in python," *Journal of Machine Learning Research*, vol. 21, 2020. [Cited on page 29.]

[125] J. Guo, H. He, T. He, L. Lausen, M. Li, H. Lin, X. Shi, C. Wang, J. Xie, S. Zha, S. Zheng, and Y. Zhu, "Gluoncv and gluon nlp: Deep learning in computer vision and natural language processing," *Journal of Machine Learning Research*, vol. 21, 2020. [Cited on page 29.]

[126] G. Siglidis, G. Nikolentzos, S. Limnios, C. Giatsidis, K. Skianis, and M. Vazirgiannis, "Grakel: A graph kernel library in python," *Journal of Machine Learning Research*, vol. 21, 2020. [Cited on page 29.]

[127] J. M. B. Haslbeck and L. J. Waldorp, "Mgm : Estimating time-varying mixed graphical models in high-dimensional data," *Journal of Statistical Software*, vol. 93, 2020, 43 citations. [Online]. Available: http://www.jstatsoft.org/v93/i08/ [Cited on page 30.]

[128] J. Blechschmidt and O. G. Ernst, "Three ways to solve partial differential equations with neural networks — a review," *GAMM-Mitteilungen*, vol. 44, no. 2, p. e202100006, 2021. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/gamm.202100006 [Cited on page 32.]

[129] J. Han, A. Jentzen, and W. E, "Solving high-dimensional partial differential equations using deep learning," *Proceedings of the National Academy of Sciences*, vol. 115, no. 34, pp. 8505–8510, 2018. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.1718942115 [Cited on page 32.]

[130] D. Zhang, L. Lu, L. Guo, and G. E. Karniadakis, "Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems," *Journal of Computational Physics*, vol. 397, p. 108850, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999119305340 [Cited on page 32.]

[131] D. Zhang, L. Guo, and G. Karniadakis, "Learning in modal space: Solving time-dependent stochastic pdes using physics-informed neural networks," *SIAM Journal on Scientific Computing*, vol. 42, pp. A639–A665, 01 2020. [Cited on page 32.]

[132] G. Pang, L. Lu, and G. Karniadakis, "fpinns: Fractional physics-informed neural networks," *SIAM Journal on Scientific Computing*, vol. 41, pp. A2603–A2626, 01 2019. [Cited on pages 32, 49, and 58.]

[133] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, "Deepxde: A deep learning library for solving differential equations," *SIAM Review*, vol. 63, no. 1, pp. 208–228, 2021. [Online]. Available: https://doi.org/10.1137/19M1274067 [Cited on pages 32 and 50.]

[134] N. Kumar, E. Philip, and V. E. Elfving, "Integral transforms in a physics-informed (quantum) neural network setting: Applications & use-cases," 2022. [Online]. Available: https://arxiv.org/abs/2206.14184 [Cited on page 32.]

[135] M. Raissi, A. Yazdani, and G. E. Karniadakis, "Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations," *Science*, vol. 367, no. 6481, pp. 1026–1030, 2020. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.aaw4741 [Cited on pages 32 and 60.]

[136] S. Cai, Z. Mao, Z. Wang, M. Yin, and G. E. Karniadakis, "Physics-informed neural networks (pinns) for fluid mechanics: a review," *Acta Mechanica Sinica*, vol. 37, no. 12, pp. 1727–1738, Dec 2021. [Online]. Available: https://doi.org/10.1007/s10409-021-01148-1 [Cited on pages 59 and 60.]

[137] M. Raissi, H. Babaee, and P. Givi, "Deep learning of turbulent scalar mixing," *Phys. Rev. Fluids*, vol. 4, p. 124501, Dec 2019. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevFluids.4.124501

[138] X. Jin, S. Cai, H. Li, and G. E. Karniadakis, "NSFnets (navier-stokes flow nets): Physics-informed neural networks for the incompressible navier-stokes equations," *Journal of Computational Physics*, vol. 426, p. 109951, feb 2021. [Online]. Available: https://doi.org/10.1016%2Fj.jcp.2020.109951 [Cited on page 32.]

[139] A. Yazdani, L. Lu, M. Raissi, and G. E. Karniadakis, "Systems biology informed deep learning for inferring parameters and hidden dynamics," *PLOS Computational Biology*, vol. 16, pp. 1–19, 11 2020. [Online]. Available: https://doi.org/10.1371/journal.pcbi.1007575 [Cited on page 32.]

[140] M. Daneker, Z. Zhang, G. E. Karniadakis, and L. Lu, "Systems biology: Identifiability analysis and parameter identification via systems-biology informed neural networks," 2022. [Online]. Available: https://arxiv.org/abs/2202.01723 [Cited on page 32.]

[141] A. Mathews, M. Francisquez, J. W. Hughes, D. R. Hatch, B. Zhu, and B. N. Rogers, "Uncovering turbulent plasma dynamics via deep learning from partial observations," *Physical Review E*, vol. 104, no. 2, aug 2021. [Online]. Available: https://doi.org/10.1103%2Fphysreve.104.025205 [Cited on page 32.]

[142] E. Kharazmi, M. Cai, X. Zheng, Z. Zhang, G. Lin, and G. E. Karniadakis, "Identifiability and predictability of integer- and fractional-order epidemiological models using physics-informed neural networks," *Nature Computational Science*,

vol. 1, no. 11, pp. 744–753, Nov 2021. [Online]. Available: https://doi.org/10.1038/s43588-021-00158-0 [Cited on page 32.]

[143] A. Rodríguez, J. Cui, N. Ramakrishnan, B. Adhikari, and B. A. Prakash, "Einns: Epidemiologically-informed neural networks," 2022. [Online]. Available: https://arxiv.org/abs/2202.10446 [Cited on page 32.]

[144] D. Pfau, J. S. Spencer, A. G. D. G. Matthews, and W. M. C. Foulkes, "Ab initio solution of the many-electron schrodinger equation with deep neural networks," *Phys. Rev. Research*, vol. 2, p. 033429, Sep 2020. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevResearch.2.033429 [Cited on page 32.]

[145] Y. Shin, "On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs," *Communications in Computational Physics*, vol. 28, no. 5, pp. 2042–2074, jun 2020. [Online]. Available: https://doi.org/10.4208%2Fcicp.oa-2020-0193 [Cited on page 32.]

[146] H. Lee and I. S. Kang, "Neural algorithm for solving differential equations," *Journal of Computational Physics*, vol. 91, no. 1, pp. 110–131, 1990. [Online]. Available: https://www.sciencedirect.com/science/article/pii/002199919090007N [Cited on page 34.]

[147] I. Lagaris, A. Likas, and D. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 987–1000, 1998. [Cited on page 34.]

[148] I. Lagaris, A. Likas, and D. Papageorgiou, "Neural-network methods for boundary value problems with irregular boundaries," *IEEE Transactions on Neural Networks*, vol. 11, no. 5, pp. 1041–1049, 2000. [Cited on page 34.]

[149] K. Rudd, "Solving partial differential equations using artificial neural networks." *PhD Thesis, Duke University*, 2013. [Online]. Available: https://hdl.handle.net/10161/8197 [Cited on page 34.]

[150] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019. [Cited on pages 34, 45, 49, and 90.]

[151] J. Sirignano and K. Spiliopoulos, "Dgm: A deep learning algorithm for solving partial differential equations," *Journal of Computational Physics*, vol. 375, pp. 1339–1364, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999118305527 [Cited on page 34.]

[152] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, and P. Perdikaris, "Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data," *Journal of Computational Physics*, vol. 394, pp. 56–81, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999119303559 [Cited on page 34.]

[153] L. Sun, H. Gao, S. Pan, and J.-X. Wang, "Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data," *Computer Methods in Applied Mechanics and Engineering*, vol. 361, p. 112732, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S004578251930622X

[154] D. Liu and Y. Wang, "A dual-dimer method for training physics-constrained neural networks with minimax architecture," *Neural Networks*, vol. 136, pp. 112–125, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608020304536 [Cited on pages 34 and 45.]

[155] R. Kondor, H. Son, H. Pan, B. Anderson, and S. Trivedi, "Covariant compositional networks for learning graphs," 01 2018. [Cited on page 34.]

[156] A. Bogatskiy, S. Ganguly, T. Kipf, R. Kondor, D. Miller, D. Murnane, J. Offermann, M. Pettee, P. Shanahan, C. Shimmin, and S. Thais, "Symmetry group equivariant architectures for physics," 03 2022. [Cited on pages 34 and 57.]

[157] A. Daw, J. Bu, S. Wang, P. Perdikaris, and A. Karpatne, "Rethinking the importance of sampling in physics-informed neural networks," 07 2022. [Cited on page 34.]

[158] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao, "Why and when can deep-but not shallow-networks avoid the curse of dimensionality: A review," *International Journal of Automation and Computing*, vol. 14, no. 5, pp. 503–519, Oct 2017. [Online]. Available: https://doi.org/10.1007/s11633-017-1054-2 [Cited on page 35.]

[159] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, "Physics-informed machine learning," *Nature Reviews Physics*, vol. 3, no. 6, pp. 422–440, Jun 2021. [Online]. Available: https://doi.org/10.1038/s42254-021-00314-5 [Cited on page 35.]

[160] L. Yang, X. Meng, and G. E. Karniadakis, "B-pinns: Bayesian physics-informed neural networks for forward and inverse pde problems with noisy data," *Journal of Computational Physics*, vol. 425, p. 109913, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999120306872 [Cited on pages 35 and 58.]

[161] F. S. Costabal, S. Pezzuto, and P. Perdikaris, "∇-pinns: physics-informed neural networks on complex geometries," 2022. [Online]. Available: https://arxiv.org/abs/2209.03984 [Cited on page 35.]

[162] S. Markidis, "The old and the new: Can physics-informed deep-learning replace traditional linear solvers?" 2021. [Online]. Available: https://arxiv.org/abs/2103.09655 [Cited on pages 35 and 51.]

[163] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608089900208 [Cited on page 40.]

[164] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, Dec 1989. [Online]. Available: https://doi.org/10.1007/BF02551274 [Cited on page 40.]

[165] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, "The expressive power of neural networks: A view from the width," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/32cbf687880eb1674a07bf717761dd3a-Paper.pdf [Cited on page 40.]

[166] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: A survey," *J. Mach. Learn. Res.*, vol. 18, no. 1, p. 5595–5637, jan 2017. [Cited on pages 41, 42, and 44.]

[167] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf [Cited on page 42.]

[168] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf [Cited on page 42.]

[169] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct 1986. [Online]. Available: https://doi.org/10.1038/323533a0 [Cited on page 42.]

[170] C. Elliott, "The simple essence of automatic differentiation," *Proceedings of the ACM on Programming Languages*, vol. 2, pp. 1–29, 07 2018. [Cited on page 42.]

[171] C. Wang, S. Li, D. He, and L. Wang, "Is $l^2$ physics-informed loss always suitable for training physics-informed neural network?" 06 2022. [Cited on page 45.]

[172] S. Wang, Y. Teng, and P. Perdikaris, "Understanding and mitigating gradient pathologies in physics-informed neural networks," 01 2020. [Cited on page 45.]

[173] S. Wang, X. Yu, and P. Perdikaris, "When and why pinns fail to train: A neural tangent kernel perspective," *Journal of Computational Physics*, vol. 449, p. 110768, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002199912100663X [Cited on page 45.]

[174] L. McClenny and U. Braga-Neto, "Self-adaptive physics-informed neural networks using a soft attention mechanism," 2020. [Online]. Available: https://arxiv.org/abs/2009.04544 [Cited on pages 45, 46, and 47.]

[175] C. Wu, M. Zhu, Q. Tan, Y. Kartha, and L. Lu, "A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks," 2022. [Online]. Available: https://arxiv.org/abs/2207.10289 [Cited on pages 48, 49, and 51.]

[176] M. Stein, "Large sample properties of simulations using latin hypercube sampling," *Technometrics*, vol. 29, no. 2, pp. 143–151, 1987. [Online]. Available: http://www.jstor.org/stable/1269769 [Cited on page 48.]

[177] I. Sobol', "On the distribution of points in a cube and the approximate evaluation of integrals," *USSR Computational Mathematics and Mathematical Physics*, vol. 7, no. 4, pp. 86–112, 1967. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0041555367901449 [Cited on page 49.]

[178] M. A. Nabian, R. J. Gladstone, and H. Meidani, "Efficient training of physics-informed neural networks via importance sampling," *Computer-Aided Civil and Infrastructure Engineering*, vol. 36, no. 8, pp. 962–977, 2021. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/mice.12685 [Cited on page 50.]

[179] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," 2017. [Online]. Available: https://arxiv.org/abs/1710.05941 [Cited on page 51.]

[180] S. Mishra and R. Molinaro, "Estimates on the generalization error of physics informed neural networks (pinns) for approximating pdes," 2020. [Online]. Available: https://arxiv.org/abs/2006.16144 [Cited on page 51.]

[181] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, "Scientific machine learning through physics–informed neural networks: Where we are and what's next," *Journal of Scientific Computing*, vol. 92, no. 3, p. 88, Jul 2022. [Online]. Available: https://doi.org/10.1007/s10915-022-01939-z [Cited on page 51.]

[182] C. Cheng and G.-T. Zhang, "Deep learning method based on physics informed neural network with resnet block for solving fluid flow problems," *Water*, vol. 13, no. 4, 2021. [Online]. Available: https://www.mdpi.com/2073-4441/13/4/423 [Cited on page 51.]

[183] Q. He, D. Barajas-Solano, G. Tartakovsky, and A. M. Tartakovsky, "Physics-informed neural networks for multiphysics data assimilation with application to subsurface transport," *Advances in Water Resources*, vol. 141, p. 103610, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0309170819311649 [Cited on page 51.]

[184] A. Mathews, M. Francisquez, J. W. Hughes, D. R. Hatch, B. Zhu, and B. N. Rogers, "Uncovering turbulent plasma dynamics via deep learning from partial observations," *Phys. Rev. E*, vol. 104, p. 025205, Aug 2021. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.104.025205 [Cited on page 51.]

[185] W. Zhu, W. Khademi, E. G. Charalampidis, and P. G. Kevrekidis, "Neural networks enforcing physical symmetries in nonlinear dynamical lattices: The case example of the ablowitz–ladik model," *Physica D: Nonlinear Phenomena*, vol. 434, p. 133264, jun 2022. [Online]. Available: https://doi.org/10.1016%2Fj.physd.2022.133264 [Cited on page 57.]

[186] J. Yu, L. Lu, X. Meng, and G. E. Karniadakis, "Gradient-enhanced physics-informed neural networks for forward and inverse PDE problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 393, p. 114823, apr 2022. [Online]. Available: https://doi.org/10.1016%2Fj.cma.2022.114823 [Cited on page 57.]

[187] A. Jagtap, E. Kharazmi, and G. Karniadakis, "Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 365, p. 113028, 06 2020. [Cited on page 58.]

[188] A. D. J. Karniadakis and George Em, "Extended Physics-Informed Neural Networks (XPINNs): A Generalized Space-Time Domain Decomposition Based Deep Learning Framework for Nonlinear Partial Differential Equations," *Communications in Computational Physics*, vol. 28, no. 5, pp. 2002–2041, Jun. 2020. [Cited on page 58.]

[189] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators," *Nature Machine Intelligence*, vol. 3, no. 3, pp. 218–229, mar 2021. [Online]. Available: https://doi.org/10.1038%2Fs42256-021-00302-5 [Cited on page 58.]

[190] S. Goswami, A. Bora, Y. Yu, and G. E. Karniadakis, "Physics-informed deep neural operator networks," 2022. [Online]. Available: https://arxiv.org/abs/2207.05748 [Cited on page 58.]

[191] S. Shekarpaz, M. Azizmalayeri, and M. H. Rohban, "Piat: Physics informed adversarial training for solving partial differential equations," 2022. [Online]. Available: https://arxiv.org/abs/2207.06647 [Cited on page 59.]

[192] T. De Ryck, S. Mishra, and R. Molinaro, "wpinns: Weak physics informed neural networks for approximating entropy solutions of hyperbolic conservation laws," 2022. [Online]. Available: https://arxiv.org/abs/2207.08483 [Cited on page 59.]

[193] S. Goswami, C. Anitescu, and T. Rabczuk, "Adaptive fourth-order phase field analysis using deep energy minimization," *Theoretical and Applied Fracture Mechanics*, vol. 107, p. 102527, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167844219306858 [Cited on page 59.]

[194] S. L. Brunton, B. R. Noack, and P. Koumoutsakos, "Machine learning for fluid mechanics," *Annual Review of Fluid Mechanics*, vol. 52, no. 1, pp. 477–508, 2020. [Online]. Available: https://doi.org/10.1146/annurev-fluid-010719-060214 [Cited on pages 59 and 60.]

[195] X. Jin, S. Cai, H. Li, and G. E. Karniadakis, "NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations," *Journal of Computational Physics*, vol. 426, p. 109951, Feb. 2021. [Cited on page 59.]

[196] Z. Mao, A. D. Jagtap, and G. E. Karniadakis, "Physics-informed neural networks for high-speed flows," *Computer Methods in Applied Mechanics and Engineering*, vol. 360, p. 112789, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045782519306814 [Cited on page 59.]

[197] G. Karniadakis and S. Sherwin, *Spectral/hp Element Methods for Computational Fluid Dynamics*. Oxford University Press, 06 2005. [Online]. Available: https://doi.org/10.1093/acprof:oso/9780198528692.001.0001 [Cited on page 59.]

[198] H. Eivazi, M. Tahani, P. Schlatter, and R. Vinuesa, "Physics-informed neural networks for solving reynolds-averaged navier–stokes equations," *Physics of*

*Fluids*, vol. 34, no. 7, p. 075117, jul 2022. [Online]. Available: https://doi.org/10.1063%2F5.0095270 [Cited on page 60.]

[199] H. Eivazi and R. Vinuesa, "Physics-informed deep-learning applications to experimental fluid mechanics," 2022. [Online]. Available: https://arxiv.org/abs/2203.15402 [Cited on page 60.]

[200] Y. Kim, Y. Choi, D. Widemann, and T. Zohdi, "A fast and accurate physics-informed neural network reduced order model with shallow masked autoencoder," 2020. [Online]. Available: https://arxiv.org/abs/2009.11990 [Cited on page 60.]

[201] V. Yudovich, "Eleven great problems of mathematical hydrodynamics," *Moscow Mathematical Journal*, vol. 3, 01 2003. [Cited on page 60.]

[202] J. Chen, T. Y. Hou, and D. Huang, "On the finite time blowup of the de gregorio model for the 3d euler equations," *Communications on Pure and Applied Mathematics*, vol. 74, no. 6, pp. 1282–1350, apr 2021. [Online]. Available: https://doi.org/10.1002%2Fcpa.21991 [Cited on page 60.]

[203] G. Luo and T. Y. Hou, "Potentially singular solutions of the 3d axisymmetric euler equations," *Proceedings of the National Academy of Sciences*, vol. 111, no. 36, pp. 12 968–12 973, 2014. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.1405238111 [Cited on page 60.]

[204] Y. Wang, C.-Y. Lai, J. Gómez-Serrano, and T. Buckmaster, "Asymptotic self-similar blow up profile for 3-d euler via physics-informed neural networks," 2022. [Online]. Available: https://arxiv.org/abs/2201.06780 [Cited on page 60.]

[205] R. Vinuesa and S. L. Brunton, "Enhancing computational fluid dynamics with machine learning," *Nature Computational Science*, vol. 2, no. 6, pp. 358–366, Jun 2022. [Online]. Available: https://doi.org/10.1038/s43588-022-00264-7 [Cited on page 60.]

# Appendix A

# *Python* Code

Code to generate the four collocation points sampling, and exemplified in figure 2.5 - uniform, random , latin hypercube and *Sobol*.

```
import numpy as np

def uniform_dist(l_bound: list, u_bound: list, n: tuple,
                 lb_x=True,ub_x=True,lb_t=True,ub_t=False):
    x,t = np.meshgrid(
            np.linspace(l_bound[0],u_bound[0],n[0]+lb_x+ub_x)[lb_x:n[0]+lb_x],
            np.linspace(l_bound[1],u_bound[1],n[1]+lb_t+ub_t)[lb_t:n[1]+lb_t])
    return x.ravel(),t.ravel()

def random_dist(l_bound: list, u_bound: list, n: int):
    x,t = (u_bound[0]-l_bound[0])*np.random.rand(n)+l_bound[0],
            (u_bound[1]-l_bound[1])*np.random.rand(n)+l_bound[1]
    return x,t

def random_dist_bound(l_bound: int, u_bound: int, n: int):
    x,t = np.zeros((2,n))
    bernoulli = np.random.binomial(size=n, n=1, p=0.5)
    for i,r in enumerate(np.random.rand(n)):
        t[i] = r
        if bernoulli[i]: x[i] = u_bound
        else: x[i] = l_bound
    return x,t

```

```
from scipy.stats import qmc

def latinhypercube_dist(l_bound: list, u_bound: list, n: int):
    sampler = qmc.LatinHypercube(d=2)
    sample = sampler.random(n=64)
    scaled = qmc.scale(sample, l_bound, u_bound)
    x,t = scaled[:,0], scaled[:,1]
    return x,t

def sobol_dist(l_bound: list, u_bound: list, n: int):
    sampler = qmc.Sobol(d=2)
    sample = sampler.random_base2(m=int(np.log2(n)))
    scaled = qmc.scale(sample, l_bound, u_bound)
    x,t = scaled[:,0], scaled[:,1]
    return x,t
```

Next, we have the main library for PINNs. Most of the code shown here is the one used in the seminal papers in the area [150]; small changes and new functions were created for the purpose of this paper:

```
# import relevant packages
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize
import pickle
import time
from tensorflow.keras.layers import Dense, Lambda

# set data type
DTYPE='float32'
tf.keras.backend.set_floatx(DTYPE)

# # set random seed for reproducible results
tf.random.set_seed(101)

```

```
# set constants (for Burgers Equation)
pi = tf.constant(np.pi, dtype=DTYPE)
viscosity = .01/pi



import sys
sys.path.insert(0, PATH)

with open('exact_solution.pkl', 'rb') as f:
    exact_ = pickle.load(f)
exact = exact_[2].reshape(201*201,1)
T_exact, X_exact, U_exact = exact_

# Define model architecture
class PINN_NeuralNet(tf.keras.Model):
    """ Set basic architecture of the PINN model."""

    def __init__(self, lb, ub,
                 output_dim=1,
                 num_hidden_layers=8,
                 num_neurons_per_layer=20,
                 activation='tanh',
                 kernel_initializer='glorot_normal',
                 **kwargs):
        super().__init__(**kwargs)

        self.num_hidden_layers = num_hidden_layers
        self.output_dim = output_dim
        self.lb = lb
        self.ub = ub

        # Define NN architecture
        self.scale = Lambda(lambda x: 2.0*(x - lb)/(ub - lb) - 1.0)
        self.hidden = [Dense(num_neurons_per_layer,
                             activation=tf.keras.activations.get(activation),
                             kernel_initializer=kernel_initializer)
                       for _ in range(self.num_hidden_layers)]
        self.out = Dense(output_dim)
```

```
58      def call(self, X):
            """Forward-pass through neural network."""
60          Z = self.scale(X)
            for i in range(self.num_hidden_layers):
62              Z = self.hidden[i](Z)
            return self.out(Z)
64
    class PINNSolver():
66      def __init__(self, model, X_r):
            self.model = model
68
            # Store collocation points
70          self.t = X_r[:,0:1]
            self.x = X_r[:,1:2]
72
            # Initialize history of losses and global iteration counter
74          self.hist = []
            self.iter = 0
76
            self.l2error = []
78          self.time = []

80      def get_r(self):

82          with tf.GradientTape(persistent=True) as tape:
                # Watch variables representing t and x during this GradientTape
84              tape.watch(self.t)
                tape.watch(self.x)
86
                # Compute current values u(t,x)
88              u = self.model(tf.stack([self.t[:,0], self.x[:,0]], axis=1))

90              u_x = tape.gradient(u, self.x)

92          u_t = tape.gradient(u, self.t)
            u_xx = tape.gradient(u_x, self.x)
94
```

```
        del tape

        return self.fun_r(self.t, self.x, u, u_t, u_x, u_xx)

    def loss_fn(self, X, u):

        # Compute phi_r
        r = self.get_r()
        phi_r = tf.reduce_mean(tf.square(r))

        # Initialize loss
        loss = phi_r

        # Add phi_0 and phi_b to the loss
        for i in range(len(X)):
            u_pred = self.model(X[i])
            loss += tf.reduce_mean(tf.square(u[i] - u_pred))

        return loss

    def get_grad(self, X, u):
        with tf.GradientTape(persistent=True) as tape:
            # This tape is for derivatives with
            # respect to trainable variables
            tape.watch(self.model.trainable_variables)
            loss = self.loss_fn(X, u)

        g = tape.gradient(loss, self.model.trainable_variables)
        del tape

        return loss, g

    def fun_r(self, t, x, u, u_t, u_x, u_xx):
        """Residual of the PDE"""
        return u_t + u * u_x - viscosity * u_xx

    def solve_with_TFoptimizer(self, optimizer, X, u, N=1000):
        """This method performs a gradient descent type optimization."""
```

```
134         @tf.function
            def train_step():
136             loss, grad_theta = self.get_grad(X, u)

138             # Perform gradient descent step
                optimizer.apply_gradients(zip(grad_theta, self.model.
    trainable_variables))
140             return loss

142         for i in range(N+1):
                loss = train_step()
144

                self.current_loss = loss.numpy()
146             self.callback()

148     def solve_with_ScipyOptimizer(self, X, u, method='L-BFGS-B', **kwargs):

150         def get_weight_tensor():
                """Function to return current variables of the model
152             as 1d tensor as well as corresponding shapes as lists."""

154             weight_list = []
                shape_list = []
156

                # Loop over all variables, i.e. weight matrices, bias vectors
158             and unknown parameters
                for v in self.model.variables:
160                 shape_list.append(v.shape)
                    weight_list.extend(v.numpy().flatten())
162

                weight_list = tf.convert_to_tensor(weight_list)
164             return weight_list, shape_list

166         x0, shape_list = get_weight_tensor()

168         def set_weight_tensor(weight_list):
                """Function which sets list of weights
```

```
170             to variables in the model."""
                idx = 0
172             for v in self.model.variables:
                    vs = v.shape
174
                    # Weight matrices
176                 if len(vs) == 2:
                        sw = vs[0]*vs[1]
178                     new_val = tf.reshape(weight_list[idx:idx+sw],(vs[0],vs[1]))
                        idx += sw
180
                    # Bias vectors
182                 elif len(vs) == 1:
                        new_val = weight_list[idx:idx+vs[0]]
184                     idx += vs[0]

186                 # Variables (in case of parameter identification setting)
                    elif len(vs) == 0:
188                     new_val = weight_list[idx]
                        idx += 1
190
                    v.assign(tf.cast(new_val, DTYPE))
192
        def get_loss_and_grad(w):
194         """Function that provides current loss and gradient
            w.r.t the trainable variables as vector. This is mandatory
196         for the LBFGS minimizer from scipy."""

198         # Update weights in model
            set_weight_tensor(w)
200         # Determine value of \phi and gradient w.r.t. \theta at w
            loss, grad = self.get_grad(X, u)
202
            # Store current loss for callback function
204         loss = loss.numpy().astype(np.float64)
            self.current_loss = loss
206
            # Flatten gradient
```

```
            grad_flat = []
            for g in grad:
                grad_flat.extend(g.numpy().flatten())

            # Gradient list to array
            grad_flat = np.array(grad_flat,dtype=np.float64)

            # Return value and gradient of \phi as tuple
            return loss, grad_flat


        return scipy.optimize.minimize(fun=get_loss_and_grad,
                                       x0=x0,
                                       jac=True,
                                       method=method,
                                       callback=self.callback,
                                       **kwargs)

    def callback(self, xr=None):
        if self.iter % 500 == 0:
            N = 200
            tspace = np.linspace(self.model.lb[0], self.model.ub[0], N+1)
            xspace = np.linspace(self.model.lb[1], self.model.ub[1], N+1)
            T, X = np.meshgrid(tspace, xspace)
            Xgrid = np.vstack([T.flatten(),X.flatten()]).T
            upred = self.model(tf.cast(Xgrid,DTYPE))
            U = upred.numpy().reshape(N+1,N+1)
            erro = np.sum((U_exact-U)**2)/np.sum(U_exact**2)
            print(f'epoch {self.iter}: loss = {self.current_loss:10.8e},
            L2 relative error = {erro:10.8e}')
            self.l2error.append(erro)
            self.time.append(time.time())
        self.hist.append(self.current_loss)
        self.iter+=1

    def plot_solution(self, **kwargs):
        N = 200
        tspace = np.linspace(self.model.lb[0], self.model.ub[0], N+1)
```

```
        xspace = np.linspace(self.model.lb[1], self.model.ub[1], N+1)
        T, X = np.meshgrid(tspace, xspace)
        Xgrid = np.vstack([T.flatten(),X.flatten()]).T
        upred = self.model(tf.cast(Xgrid,DTYPE))
        U = upred.numpy().reshape(N+1,N+1)
        fig = plt.figure(figsize=(9,6))
        ax = fig.add_subplot(111, projection='3d')
        ax.plot_surface(T, X, U, cmap='viridis', **kwargs)
        ax.set_xlabel('$t$')
        ax.set_ylabel('$x$')
        ax.set_zlabel('$u_\\theta(t,x)$')
        ax.view_init(35,35)
        return ax

    def plot_l2error(self, filename, ax=None, save=True):
        if not ax:
            fig = plt.figure(figsize=(7,5))
            ax = fig.add_subplot(111)
        ax.semilogy([500*i for i in range(len(self.l2error))], self.l2error,'k-')
        ax.set_xlabel('No. of Epochs')
        ax.set_ylabel('$L_2$ relative error')
        if save:
            f = f'{filename}_L2error.pkl'
            with open(f, 'wb') as f:
                pickle.dump(self.l2error, f)
        return ax

    def plot_time(self, filename, ax=None, save=True):
        if not ax:
            fig = plt.figure(figsize=(7,5))
            ax = fig.add_subplot(111)
        ax.semilogy([500*i for i in range(len(self.time))], self.time,'k-')
        ax.set_xlabel('No. of Epochs')
        ax.set_ylabel('Time')
        if save:
            f = f'{filename}_time.pkl'
            with open(f, 'wb') as f:
                pickle.dump(self.time, f)
```

```
284         return ax


286     def plot_loss_history(self, filename, ax=None, save=True):
            if not ax:
288             fig = plt.figure(figsize=(7,5))
                ax = fig.add_subplot(111)
290         ax.semilogy(range(len(self.hist)), self.hist,'k-')
            ax.set_xlabel('$n_{epoch}$')
292         ax.set_ylabel('$\\phi^{n_{epoch}}$')
            if save:
294             f = f'{filename}_hist.pkl'
                with open(f, 'wb') as f:
296                 pickle.dump(self.hist, f)
            return ax
298

    def plot_residual(self, filename, save=True, **kwargs):
300         den=200
            N = den
302         tspace = np.linspace(self.model.lb[0], self.model.ub[0], N+1)
            xspace = np.linspace(self.model.lb[1], self.model.ub[1], N+1)
304         T, X = np.meshgrid(tspace, xspace)
            Xgrid = np.vstack([T.flatten(),X.flatten()]).T
306         upred = self.model(tf.cast(Xgrid,DTYPE))
            U = upred.numpy().reshape(den+1,den+1)
308         # Surface plot of solution u(t,x)
            fig = plt.figure(figsize=(12,8))
310         ax = fig.add_subplot(111, projection='3d')
            ax.plot_surface(T_exact, X_exact, -abs(U_exact-U), cmap='viridis');
312         ax.view_init(35,35)
            ax.set_xlabel('$t$')
314         ax.set_ylabel('$x$')
            ax.set_zlabel('$u_\\theta(t,x)$')
316         ax.set_title("Residual of Burgers equation' solution")
            if save:
318             f = f'{filename}_residual.pkl'
                with open(filename, 'wb') as f:
320                 pickle.dump(-abs(U_exact-U), f)
            return ax
```

```
322
    def save_solution(self, filename):
324     N = 200
        tspace = np.linspace(self.model.lb[0], self.model.ub[0], N+1)
326     xspace = np.linspace(self.model.lb[1], self.model.ub[1], N+1)
        T, X = np.meshgrid(tspace, xspace)
328     Xgrid = np.vstack([T.flatten(),X.flatten()]).T
        upred = self.model(tf.cast(Xgrid,DTYPE))
330     U = upred.numpy().reshape(N+1,N+1)
        f = f'{filename}_solution.pkl'
332     with open(filename, 'wb') as f:
            pickle.dump(U, f)
334
    def save_time(self, filename):
336     f = f'{filename}_time.pkl'
        with open(f, 'wb') as f:
338         pickle.dump(self.time, f)

340 def save_l2error(self, filename):
        f = f'{filename}_L2error.pkl'
342     with open(f, 'wb') as f:
            pickle.dump(self.l2error, f)
```