Integrating Prior Knowledge into Machine Learning Models with Applications in Physics

PHILIPP PILAR
Integrating Prior Knowledge into Machine Learning Models with Applications in Physics

*Philipp Pilar*
philipp.pilar@it.uu.se

September 2023

*Division of Systems and Control*  
*Department of Information Technology*  
*Uppsala University*  
*Box 337*  
*SE-751 05 Uppsala*  
*Sweden*

http://www.it.uu.se/

Dissertation for the degree of Licentiate of Philosophy in Machine Learning

© Philipp Pilar 2023  
ISSN 1404-5117  
Printed by the Department of Information Technology, Uppsala University, Sweden
Abstract

At the extremes, two antithetical approaches to describing natural processes exist. Theoretical models can be derived from first principles, allowing for clear interpretability; on the downside, this approach may be infeasible or inefficient for complex systems. Alternatively, methods from statistical machine learning can be employed to learn black box models from large amounts of data, while providing little or no understanding of their inner workings.

Both approaches have different desirable properties and weaknesses. It is natural to ask how they may be combined to create better models. This is the question that the field of physics-informed machine learning is concerned with, and which we will consider in this thesis. More precisely, we investigate ways of integrating additional prior knowledge into machine learning models.

In Paper I, we consider multitask Gaussian processes and devise a way to include so-called sum constraints into the model, where a nonlinear sum of the outputs is required to equal a known value. In Paper II, we consider the task of determining unknown parameters from data when solving partial differential equations (PDEs) with physics-informed neural networks. Given the prior knowledge that the measurement noise is homogeneous but otherwise unknown, we demonstrate that it is possible to learn the solution and parameters of the PDE jointly with the noise distribution. In Paper III, we consider generative adversarial networks, which may produce realistic-looking samples but fail to reproduce their true distribution. In our work, we mitigate this issue by matching the true and generated distributions of statistics extracted from the data.
Acknowledgments

This thesis is the result of multiple years of sustained efforts and it would not have been possible to do it alone. I would like to thank my supervisor, Niklas Wahlström, for enthusiastically mentoring me over the past three years, my co-supervisor, Thomas Schön, for providing valuable advice, and my co-author, Carl Jidling, for stimulating discussions.

I would like to thank my colleagues at SysCon for making the division a pleasant place to work at. My special thanks go to Daniel, Fredrik, Jennifer, and Ludvig, for proofreading parts of the thesis. Finally, I would like to thank my family and friends, both from Austria and Sweden, for their support and for making life outside of work enjoyable! Especially, misaotra Miora!

This research was financially supported by the Swedish Research Council (VR) via the project *Physics-informed machine learning* (registration number: 2021-04321) and by the *Kjell och Märta Beijer Foundation*. 
List of Papers

This thesis is based on the following papers:


# Contents

1 Introduction .......................... 1
   1.1 The capabilities of machine learning .......................... 2
       1.1.1 Breakthroughs of recent years .......................... 2
       1.1.2 Machine learning and science .......................... 3
   1.2 Possible applications of physics-informed machine learning .... 3
   1.3 The inner workings of machine learning models ................. 4
       1.3.1 Basic machine learning concepts ........................ 5
       1.3.2 A word on generative modeling .......................... 7
       1.3.3 Example: neural networks ............................... 7
       1.3.4 Different ways of integrating prior knowledge ............ 8
   1.4 Contribution ................................................. 9
   1.5 Outline of the thesis ........................................ 10
   1.6 Papers included in this thesis ............................... 10

2 Physics-informed neural networks .......................... 13
   2.1 Background .................................................... 13
   2.2 Using neural networks to solve differential equations .......... 14
       2.2.1 The forward problem .................................... 17
       2.2.2 The inverse problem .................................... 19
   2.3 Technical details ............................................. 21
       2.3.1 Activation functions .................................... 21
       2.3.2 Placing the collocation points .......................... 21
   2.4 Thesis contribution: the inverse problem with unknown noise .... 22

3 Generative adversarial networks .......................... 25
   3.1 Training generative adversarial networks .................... 26
       3.1.1 The adversarial game .................................... 26
       3.1.2 A better way of comparing the distributions .............. 28
       3.1.3 Penalizing instead of clipping .......................... 30
   3.2 Including prior knowledge .................................... 32
       3.2.1 General considerations .................................. 32
3.2.2 Existing work ........................................... 32
3.3 Thesis contribution: probabilistically constrained GANs ... 33

4 Gaussian processes 37
4.1 From linear regression to Gaussian processes ............... 37
  4.1.1 Linear regression ...................................... 37
  4.1.2 The kernel trick ....................................... 39
  4.1.3 Beyond the point estimate .............................. 40
4.2 Gaussian process regression .................................. 41
  4.2.1 Basic definitions ...................................... 41
  4.2.2 The predictive distribution ............................. 42
  4.2.3 Kernels ................................................. 42
  4.2.4 Hyperparameter tuning .................................. 44
4.3 Extensions of the Gaussian process framework ................ 45
  4.3.1 Multitask Gaussian processes ......................... 45
  4.3.2 Approximate inference .................................. 45
  4.3.3 Constrained Gaussian processes ....................... 47
4.4 Thesis contribution: incorporating sum constraints ........... 49

5 Outlook 55
  5.1 Conclusions .............................................. 55
  5.2 Future work .............................................. 55

Paper I - Incorporating sum constraints into multitask Gaussian processes 69

Paper II - Physics-informed neural networks with unknown measurement noise 115

Paper III - Probabilistic matching of real and generated data statistics in generative adversarial networks 141
Chapter 1

Introduction

The classical approach for finding models for natural processes, as pioneered by scientists like Galileo, Newton, and Einstein, lies in conducting select experiments and pondering their outcomes carefully. Laws governing the observed processes can then be deducted and eventually, a new law of physics may be discovered. This approach has been common for centuries and has led to great scientific breakthroughs such as the law of gravitation, the theory of electromagnetism, quantum theory, and general relativity.

The discovered laws of physics made accurate predictions of physical systems possible, which resulted in better technologies. These, in turn, enabled the construction of more sophisticated apparatuses, required for more elaborate experiments. Together with the increasing availability of computing power, this interplay eventually resulted in the modern technologies of our time.

Today, we have reached a point where great effort goes into conducting novel physics experiments, as multibillion-dollar projects such as the large hadron collider near Geneva or the James Webb Space Telescope demonstrate. In the engineering disciplines, ever more elaborate technologies are created, some of which require the solving of complicated equations during the design stage. Others are capable of gathering data from many sensors which may be used for improving future iterations of the product, or to control real-time applications.

Both researchers and engineers may drown in data or find themselves grappling with simulations that may take weeks to execute. This is where machine learning (ML) may come to the rescue. It constitutes an alternative to the classical approach, where models are learned directly from data instead of through the careful interpretation of observations, and without taking into account prior knowledge from physics. ML models are typically black box models, which can work very well, yet they do not allow for
substantial insight into how they make their predictions.

Since both approaches have different strengths, we would like to somehow merge them. The combination of prior knowledge from physics and machine learning, resulting in the field of physics-informed machine learning, is the broad topic of this thesis.

1.1 The capabilities of machine learning

The field of machine learning concerns itself with extracting information from large amounts of data. It may be applicable to situations that are too complex for theoretical models to accurately describe, or where data is so plentiful that classical methods of statistics are no longer practical.

1.1.1 Breakthroughs of recent years

While the perceptron was already described in [Rosenblatt, 1958] and while backpropagation was already described in [Rumelhart et al., 1986], it was not until the 2010s that the field of machine learning, or rather, deep learning, where many simple ML models are combined, experienced major breakthroughs. Early successes of this period include the Watson system defeating human opponents in Jeopardy! [Ferrucci et al., 2010, Markoff, 2011], and a convolutional neural network outperforming state of the art models on subsets of the ImageNet dataset [Krizhevsky et al., 2012].

More successes soon followed: Deep Mind’s AlphaGo [Silver et al., 2016, 2017] triumphed over the human world champion in the game of Go, a game which until then was considered out of reach for AI agents, due to its immense complexity which far surpasses that of chess. Reinforcement learning algorithms learned to play Atari games [Mnih et al., 2013, 2015] and later to beat human professionals in video games such as Dota 2 [OpenAI et al., 2019]. Generative models like Dall-E [Ramesh et al., 2021, 2022] became capable of producing realistic-looking images from text prompts.

More recently, breakthroughs in large language models have produced headlines. First, ChatGPT [Liu et al., 2023] was released, which can answer questions and even perform coding tasks surprisingly well. Microsoft released the Chatbot Bing Search for its search engine, which amusingly (or frighteningly?) started to argue with users on issues like the current date and threatened another one for revealing its guidelines [Willison, 2023].

This list of achievements is far from exhaustive and we can expect to see many more exciting applications of machine learning in the next decade. For a more thorough discussion of the historical development of the field, see e.g. Russell [2010].
1.1.2 Machine learning and science

It is apparent that machine learning can be employed for many different purposes, with impressive results. And yet, at first it may appear questionable to use black box machine learning models for science. Is it not the goal of science to produce understanding? In the following, we will give a few examples of ways in which machine learning has already been used for science in fruitful ways.

The biggest breakthrough of machine learning in the scientific domain probably lies in AlphaFold [Jumper et al., 2021], which can solve the protein folding problem with high accuracy. With reinforcement learning, improved algorithms for multiplying matrices were discovered [Fawzi et al., 2022], promising speedups in varied applications. Reinforcement learning algorithms have also been applied for more effective plasma containment in nuclear fusion reactors [Degrave et al., 2022]. Guimerà et al. [2020] developed a ‘machine scientist’, capable of extracting governing equations from data, given a database of potentially relevant mathematical expressions which it explores using Markov chain Monte Carlo techniques.

These examples clearly show that many creative ways of using machine learning for research in science exist, with potential uses ranging from the more applied problem of controlling plasma in nuclear fusion reactors to very fundamental, theoretical discoveries in the case of matrix multiplication.

1.2 Possible applications of physics-informed machine learning

In our work, we consider multiple real-world datasets to apply our methods to. Roughly, they can be separated into two categories: on the one hand, we may have measurements of physical systems following (partly) known governing equations, and we want to model their time evolution. On the other hand, we may have data resulting from computationally expensive numerical simulations, and we would like to train generative machine learning models to allow for cheaper data generation.

In Figure 1.1, two datasets corresponding to the first category are shown: on the left side, the fluid flow behind an obstacle is depicted. Situations like this are governed by the Navier-Stokes equations, which can be taken into account when training physics-informed neural networks [Raissi et al., 2019]. In the plot on the right, we consider the trajectory of a double pendulum [Asseman et al., 2018]; it is possible to perform regression on the pendulum coordinates, while taking energy conservation into account.

In Figure 1.2, datasets relevant for generative modeling are depicted.
Chapter 1. Introduction

Figure 1.1: It can be useful to incorporate known laws of physics into machine learning models. **Left:** Vortices behind an obstacle, as modeled via the Navier Stokes equations. Knowledge of these equations can be encoded in the neural network training to learn the solution. **Right:** The motion of a double pendulum. When modeling the trajectories, energy conservation can be taken into account.

Figure 1.2: Generative models may be able to act as substitutes for expensive numerical simulations. **Left:** Cosmological temperature maps from the CAMELS dataset. **Right:** Radio signals from the planned IceCube-Gen2 detector.

In the left part, two samples of cosmological temperature maps from the CAMELS dataset [Villaescusa-Navarro et al., 2021] are depicted. In the right half of the plot, two simulated radio signals from the planned IceCube-Gen2 detector [Aartsen et al., 2021] are depicted.

1.3 The inner workings of machine learning models

It is now time to turn our attention to what lies under the hood of ML models. For a more exhaustive introduction to the field, see e.g. Bishop and Nasrabadi [2006] or Lindholm et al. [2021]; in the following, we restrict ourselves to the bare minimum of concepts required for a rough understanding of the different aspects that go into training a machine learning model. We continue with a brief description of fully-connected neural networks and discuss different ways of integrating prior knowledge into ML models.
1.3. The inner workings of machine learning models

1.3.1 Basic machine learning concepts

In this section, we introduce the most basic concepts for training and evaluating ML models. We will focus on the case of supervised machine learning, where we have a dataset \( \mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N_d} \) of \( N_d \) pairs of inputs \( x_i \) and outputs \( y_i \) at our disposal.

There are two main types of supervised learning problems: regression and classification. In the case of regression, the aim is for the ML model to fit a continuous function, and the ML model may output arbitrary values. For example, the task of modeling the ambient temperature as a function of time would constitute a regression problem. For classification, on the other hand, the aim is to correctly classify samples, e.g. whether an image shows a cat that is dead or alive. Here, it is the norm for ML models to predict class probabilities.

The machine learning model

The first step in training an ML model typically consists in deciding which type of ML model to employ. Many different options exist, such as neural networks, Gaussian processes, support vector machines, linear regression models, and so forth. The best choice will depend on the problem at hand and may not be obvious. The concepts presented in the following sections, however, are widely applicable. Hence, we will for now skip this step.

We denote the predictions of the machine learning model as

\[
\hat{y}_i = f(x_i; \theta),
\]

where \( \theta \) denotes the parameters of the ML model which are to be determined.

The loss function

In order to train our ML model, we need a way of telling how well it is performing. For this purpose, we define a loss function \( \mathcal{L}(y, \hat{y}) \) which compares the model predictions \( \hat{y} = [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_{N_d}]^T \) to the true outputs \( y = [y_1, y_2, \ldots, y_{N_d}]^T \); in principle, the loss function can be chosen arbitrarily, with the main requirement being that it should lead to effective training of the ML model.

One widely used loss function for regression tasks is given by the squared error loss:

\[
\mathcal{L}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2.
\]
In the case of binary classification, the cross-entropy loss is a common choice:

$$
\mathcal{L}(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^{N} y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i). \tag{1.3}
$$

**Gradient descent**

The standard way of training many different kinds of ML models is via one of the variants of stochastic gradient descent (SGD). In particular, it is applicable to parametric models that are smooth in the parameters. The main idea is to evaluate the loss of the model on the data available and then adjust the model parameters in order to minimize it. The “stochastic” in SGD refers to the fact that we typically do not evaluate the loss function on the entire dataset at hand, but instead on randomly sampled minibatches. The advantages of using minibatches are twofold: firstly, it significantly reduces the computational effort required for a parameter update. Secondly, it introduces some randomness into the minimization procedure, which can be helpful to avoid getting stuck in a local minimum.

At each SGD iteration, the gradient of the loss function $\nabla_{\theta} \mathcal{L}$ is calculated with respect to the network parameters; the parameters are subsequently updated via

$$
\theta_{i+1} = \theta_i - \gamma \nabla_{\theta} \mathcal{L}(y, \hat{y}(\theta_i)), \tag{1.4}
$$

where $\gamma$ is the learning rate. Numerous modifications of this algorithm exist, which introduce additional features such as momentum or adaptive learning rates. A popular variant, which combines both of these extensions, is the Adam (adaptive moment estimation) optimizer [Kingma and Ba, 2015].

**Model evaluation**

In order to evaluate the performance of a machine learning model in the supervised learning context, it needs to be tested on data not used in training the model. It is therefore very common to split the available data into a training, a validation, and a test set. As the name tells, the training data is for training the model. The validation data is used for evaluating the model during training; it can be used for feature selection and hyperparameter tuning. While the validation set is not used directly as training data, it enters indirectly into the model by influencing design choices such as the hyperparameters. The test set is supposed to be used only after the training is complete, in order to test the performance on data that the model has never seen before.
1.3.2 A word on generative modeling

So far we have only considered the supervised machine learning framework, where discriminative models are trained; that is, to predict $y$ given $x$, the models need to learn the conditional distribution $p(y|x)$. Generative models, on the other hand, are more general in scope in that they aim to generate new samples from the data distribution. To this end, knowledge of the joint distribution is required, $p(x, y) = p(y|x)p(x)$, which makes this problem more challenging since this also (at least implicitly) includes $p(x)$. It is also possible to train generative models when no labels $y$ exist, in which case only the distribution $p(x)$ is modeled. In Chapter 3, we will discuss generative adversarial networks, which constitute one example of generative models.

1.3.3 Example: neural networks

Neural networks are probably the most widely used ML model of recent years. In one way or another, they were involved in most of the examples given in Section 1.1. In this section, we briefly review the very basics of neural networks.

**Architecture**

A standard fully-connected neural network, or multilayer perceptron (MLP), consists of an input layer, an arbitrary number of hidden layers, and an output layer; $L$ layers in total, not counting the input layer. The units $q_l$ in a given layer $l$ are connected to the units $q_{l+1}$ in the next layer $l+1$ via a linear transformation followed by a nonlinear activation function $h$; the units in the hidden layers are also called hidden units.

Mathematically, this relationship can be expressed as

$$q_{l+1} = h(W_l q_l + b_l), \quad (1.5)$$

where the weight matrices $W_l$ and the bias vectors $b_l$ contain weights that constitute the parameters of the model and which are to be determined via a suitable optimization procedure. For the first layer, we have $q_0 = x$, and for the final layer $q_L = \hat{y}$.

It is crucial for the activation function $h$ to be nonlinear; otherwise, the resulting sequence of linear transformations would yield again a linear model. The ability to flexibly learn nonlinear relationships between (potentially high-dimensional) inputs and outputs constitutes the major strength of neural networks. Common activation functions include the hyperbolic tangent (tanh) activation function, $h(x) = \tanh(x)$, or the rectified linear unit, $h(x) = \text{ReLU}(x) = \max(0, x)$. 
Types of neural networks

The fully-connected neural network described here constitutes the most basic form of neural network and can already be useful for many tasks. However, countless alternative architectures exist: convolutional neural networks for spatio-temporal data, recurrent neural networks for time series, residual neural networks that allow for more effective gradient propagation, the U-Net as a variation of the convolutional network to extract features on multiple scales, and many more [Alzubaidi et al., 2021].

1.3.4 Different ways of integrating prior knowledge

Now that the basic components that go into training an ML model have been established, we can discuss ways of including prior knowledge. In this section, we give a brief overview of possible ways in which physical constraints may be incorporated into ML models.

Additional loss terms

A common method is to include constraints as additional terms in the loss function, such as requirements on the total energy of particle showers [Khattak et al., 2019] or physical laws that the outputs of the neural networks need to obey [Raissi et al., 2019].

Formally, this is rather simple to achieve, by defining the new loss function

$$\mathcal{L}_c(y, \hat{y}) = \mathcal{L}(y, \hat{y}) + \lambda \mathcal{L}_{\text{phys}}(\hat{y}).$$  \hspace{1cm} (1.6)

The factor $\lambda$ determines the relative weighting between the two terms. It is important to tune this factor adequately, since the two loss terms may counteract each other. Usually, there is no way of deriving the correct factor $\lambda$ and it needs to be determined along with other hyperparameters.

This type of constraint is often referred to as a soft constraint, since in the loss, constraints are not strictly enforced but merely penalize the model [Karniadakis et al., 2021]. In our work, we make use of additional loss terms when training physics-informed neural networks with unknown measurement noise (see Chapter 2 and Paper II), and when matching statistics between real and generated data in generative adversarial networks (see Chapter 3 and Paper III).

Additional inputs

Alternatively, it may be possible to feed physical parameters as additional inputs into the ML model, to condition it on them. That is, the model then
The main contribution of this thesis lies in devising new ways of integrating prior knowledge into machine learning models.

Gaussian processes (GPs) constitute a machine learning method that can be used for both regression and classification. Given noisy measurements, they give a distribution of functions that may fit the data well. In Paper I, we extend the multi-class GP, where the outputs of a multivariate function are learned, to include prior knowledge in the form of sum constraints. That is, a nonlinear sum of the GP outputs is required to sum to a known value.

Compared to standard PDE solvers, they provide greater flexibility when including data in the solution process. In Paper II, we show how PINNs can be trained in the case of homogeneous but unknown noise: by employing an energy-based model, it is possible to learn the noise distribution jointly with the PDE solution. Taking the obtained noise distribution into account can also yield more accurate results for the PDE solution.

Generative adversarial networks (GANs) are a popular framework of generative modeling; they can produce realistic-looking images and other types of data. However, it can be hard to ensure that they will generate a sufficient diversity of samples, in addition to the samples looking realistic. In Paper III, we provide a way of fitting the distribution of freely eligible dataset statistics between real and generated data. Domain knowledge can help to identify statistics of the dataset that should be matched.

1.5 Outline of the thesis

The first part of this thesis provides an overview of the models that we consider in Papers I-III: in Chapter 2, we give an introduction to physics-informed neural networks. In Chapter 3, we introduce generative adversarial networks. In Chapter 4, we give an overview of GP regression. Finally, in Chapter 5, we provide our conclusions and ideas for future work.

The second part of the thesis contains the papers, which are presented in the next section.

1.6 Papers included in this thesis

Paper I: Incorporating Sum Constraints into Multitask Gaussian Processes


**Summary:** This paper proposes a method of incorporating nonlinear sum constraints into multitask Gaussian processes; that is, a specific sum of the GP outputs is constrained to equal a known value. This is achieved by first transforming the GP outputs in such a way that the constraints become linear in the transformed outputs; then, the GP can be conditioned on the resulting linear constraint, and the original outputs are obtained by backtransforming.
Contribution: Niklas Wahlström devised the general idea of the project. The sum constraint framework was developed mainly by me, with fruitful feedback from Niklas Wahlström, Carl Jidling, and Thomas Schön. The implementation of the experiments was done by me. While I did the main part of the writing, all of the authors made notable contributions to the content and structure of the paper.

Paper II: Physics-informed neural networks with unknown measurement noise


Summary: In this paper, it is demonstrated that the standard way of solving the inverse problem with PINNs does not give correct results in the case of non-zero mean noise. A method is proposed to resolve this issue and to train PINNs on data contaminated with unknown homogeneous noise. To this end, an energy-based model is trained jointly with the PINN to learn the correct noise distribution. Employing multiple examples, the superior performance of this approach in the case of non-Gaussian noise is demonstrated.

Contribution: Niklas Wahlström and I developed the idea for this project together. The implementation and the main part of the writing were done by me, with helpful feedback from Niklas Wahlström.

Paper III: Probabilistic matching of real and generated data statistics in generative adversarial networks


Summary: When training generative adversarial networks, it is important that generated samples both look realistic and are distributed according to the true data distribution. In this paper, we focus on the latter aspect and give a way of constraining the distributions of certain statistics of the generated data to those of the real data. This is done by adding the Kullback-Leibler divergences between true and generated distributions as additional loss terms to the generator loss. In order to enable the evaluation of the corresponding probability densities, an energy-based model and kernel density estimation are employed to model them, respectively.
Contribution: I came up with the main idea for this project and it was refined in discussion with Niklas Wahlström. The implementation and the main part of the writing were done by me, with helpful feedback from Niklas Wahlström.
Chapter 2

Physics-informed neural networks

The term *physics-informed neural network* is ambiguous and can refer to any kind of neural network that incorporates physical knowledge in some way; see Section 1.3.4 for some examples. In the following, we will use the term physics-informed neural network (PINN) in the sense of Raissi et al. [2019]. The main idea is to substitute neural networks as solvers for differential equations, instead of commonly used finite difference or finite element methods.

In this chapter, we introduce standard methodology for training PINNs. We consider two types of problems, the forward and the inverse problem, and we employ a concrete example to compare the PINN with classical methods. We proceed to discuss some more technical details that need to be taken into account when training PINNs. Finally, we outline our contribution to the field, where we show how noise from an unknown distribution can be taken into account during PINN training.

2.1 Background

The idea of using neural networks to solve PDEs was pioneered by Lagaris et al. [1998] and found more widespread recognition when Raissi et al. [2019] published their seminal work. Since then, much research effort has been aimed at evaluating and improving the PINN framework as well as applying it to different problems [Karniadakis et al., 2021, Cuomo et al., 2022, Hao et al., 2022, Cai et al., 2021a,b, Markidis, 2021, Lawal et al., 2022].

It may appear questionable to switch from well-established classical methods of solving differential equations, with rigorously proven mathematical properties, to the heuristics of neural network training. It is indeed
the case that, as of now, most problems can be solved more efficiently via classical methods than by utilizing PINNs [Markidis, 2021]. The field of PINNs, however, is still in its infancy whereas the classical methods have resulted from decades of research. In that light, it certainly makes sense to sustain research efforts on PINNs.

One of the advantages of PINNs lies in the fact that they are mesh-free solvers. As a result, the PDE solution can be evaluated at any point in the input space after training, and derivatives of the solution can be computed straightforwardly. Secondly, due to the inherent capability of neural networks to model nonlinear relationships [Hornik et al., 1989], PINNs may be especially well suited for solving nonlinear differential equations. Furthermore, PINNs constitute promising candidates for solving high-dimensional PDEs, where they may be able to avoid the curse of dimensionality that classical methods are struck by [Grohs et al., 2018, Hu et al., 2023]. Finally, their biggest advantage probably lies in their flexibility: where classical methods may require different solvers corresponding to differences in the problem formulation or the data available, PINNs can very easily integrate such variations into the training procedure. Changes in the information available about the PDE solution only result in minor modifications of the PINN loss function, without requiring more fundamental changes in the training routine (compare Section 2.2.2).

2.2 Using neural networks to solve differential equations

When working with PINNs, the general task is to train neural networks to learn solutions of partial differential equations of the form

\[ \mathcal{F}(z, \lambda)u(z) = 0, \]

where \( z \) denotes the input, e.g. \( z = (t, x) \), \( u \) the solution of the PDE, and \( \mathcal{F} \) the differential operator defining the PDE. Optionally, \( \lambda \) denotes parameters of \( \mathcal{F} \) that may be unknown. The PINN solution \( \hat{u} \) is parameterized by a neural network.

The main ingredient for training PINNs is automatic differentiation functionality, such as, e.g., provided by python packages like PyTorch [Paszke et al., 2017] or Tensorflow [Abadi et al., 2016], which allows for differentiating the outputs \( \hat{u} \) of the neural network with respect to the inputs \( z \). Thus, the PDE residuals \( \mathcal{F}\hat{u} \) can be determined exactly at arbitrarily chosen collocation points. By minimizing these residuals, the neural network output can be driven towards the PDE solution.
2.2. Using neural networks to solve differential equations

PINN

\[
\begin{array}{c}
\begin{array}{c}
\text{IC, BC data} \\
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
\text{F(λ)} \\
\text{∂t} \\
\text{∂x} \\
\vdots \\
\end{array}
\end{array}
\Rightarrow
\begin{array}{c}
\begin{array}{c}
\text{L_{pde} + L_{IC, BC} + L_{data} = L_{tot}} \\
\end{array}
\end{array}
\]

A schematic of the PINN framework is given in Figure 2.1, where it is illustrated how different data modalities as well as initial conditions (ICs) and boundary conditions (BCs) can be taken into account by combining multiple losses. Details on these loss terms will be given in the following sections.

When working with PINNs, two main types of problems are distinguished: the forward and the inverse problem [Raissi et al., 2019]. In the forward problem, ICs and BCs are given, whereas in the inverse problem, (noisy) measurements of the solution are given. While, in the former case, the PINN concerns itself solely with learning the PDE solution, in the latter case, parameters \( \lambda \) of the differential operator \( \mathcal{F} \) can be determined jointly with the PDE solution.

For a schematic of the two cases, see Figure 2.2. Here, we consider a rectangular domain where either the BCs (forward problem) or measurements of the solution (inverse problem) are given. For both problems, so-called collocation points are distributed in the domain, which serve to enforce that the solution obeys the differential equation. As we will see, in the PINN framework, both forward and inverse problems can be solved with only minimal changes in the training procedure.

While it can be helpful to distinguish between forward and inverse problems, the distinction is not sharp and hybrid models are possible, where measurements may be given, together with BCs on only a part of the boundary.
Figure 2.2: The forward problem vs the inverse problem. **Left:** In the forward problem, the differential operator $\mathcal{F}$ defining the PDE is known, together with ICs and BCs. The allocation of collocation points allows for the PINN to learn the PDE solution in the entire domain. **Right:** In the inverse problem, the functional form of the operator $\mathcal{F}$ is known, but some parameters $\lambda$ may be unknown. The network is trained on (noisy) measurements of the PDE solution, together with collocation points at which the PDE residuals are minimized. The PDE parameters $\lambda$ can be determined jointly with the PDE solution.
2.2. The forward problem

In the forward problem, the differential operator $F$ is known completely and the objective is to determine the solution of the differential equation in the region of interest given ICs and BCs. Standard BCs include Dirichlet and von Neumann BCs. In principle, there is no restriction, and arbitrary BCs are possible, given, e.g., as functions of the inputs. In the following, we will only consider BCs, since ICs can also be viewed as BCs in the time domain.

Formulating the loss

In order to incorporate BCs, the following loss term can be defined:

$$L_{BC} = \frac{1}{N_{BC}} \sum_{i=1}^{N_{BC}} (\hat{u}(z_{BC}^i) - f(z_{BC}^i))^2, \quad (2.2)$$

where the $N_{BC}$ points $z_{BC}$ are chosen on the boundary, with known values $u(z_{BC}) = f(z_{BC})$.

The PINN is supposed to produce a solution to the PDE (2.1), hence we need to introduce information about the PDE in the training procedure. This can be achieved by placing $N_c$ collocation points $z_c$ in the domain of interest, and imposing the loss

$$L_{PDE} = \frac{1}{N_c} \sum_{i=1}^{N_c} (F\hat{u}(z_c^i))^2. \quad (2.3)$$

That is, we penalize non-zero PDE residuals of the PINN prediction on the collocation points. The total PINN loss is then obtained as

$$L_{tot} = L_{BC} + \omega_c L_{PDE}, \quad (2.4)$$

where $\omega_c$ is a weighting factor, and the neural network $\hat{u}$ is trained by minimizing this loss.

Example

To illustrate this, we consider the simple initial value problem

$$u''(t) = -\sin(u(t)), \quad u(0) = 0, \quad u'(0) = 1. \quad (2.5)$$

We compare the PINN solution to solutions as obtained via standard numerical methods for solving initial value problems; in particular, we consider the explicit Euler method and the Runge-Kutta-Fehlberg method (RKF45),
Figure 2.3: Solving the forward problem. **Left:** The differential equation (2.5) is discretized and solved via the explicit Euler method. Smaller step sizes lead to more accurate results. The solutions for different step sizes are plotted in different colors. **Middle:** The Runge-Kutta-Fehlberg method obtains more accurate results and does not require extremely small step sizes. **Right:** The PINN approach is used to learn a solution to (2.5). In contrast to the classical methods, no grid is required. The accuracy of the solution increases with the number of iterations, where earlier iterations are plotted in lighter shades of blue than later iterations.

with various step sizes $h$. From theoretical results, it is known that the former yields prediction errors $\epsilon \propto O(h)$, whereas the latter gives results with $\epsilon \propto O(h^4)$. Explicit Euler approximates the solution on a predefined grid, which is obtained from the chosen step size $h$, whereas RKF45 allows for an adaptive step size. In turn, the solution is obtained exclusively on the resulting grid points, and evaluating the solution at other points would require either interpolating or rerunning the algorithm on a new grid.

In the left plot of Figure 2.3, results for explicit Euler are depicted. Explicit Euler is a very simple method, and it only considers the derivative at the latest grid point to take a step via $u_{i+1} = u_i + hu'_i$. We observe that very small step sizes are required to obtain good results.

RKF45, on the other hand, is a more elaborate method and considers a weighted average of derivatives at multiple points for each step. In the middle plot, we see that it gives accurate predictions also for larger step sizes. Both explicit Euler and RKF45 start at the initial point and then proceed step by step to construct the solution.

In the right plot, the solution as obtained via the PINN approach is illustrated. Here, the solution is not obtained by taking steps along the time axis, but instead, a gradient descent algorithm is employed to optimize the PINN parameters with loss (2.4). In the plot, solutions starting at early iterations (depicted in lighter shades of blue) and proceeding to later iterations (darker shades of blue) are displayed, and we see that the solution gradually moves toward the correct solution.
2.2. Using neural networks to solve differential equations

2.2.2 The inverse problem

When considering the inverse problem, the objective is to learn the PDE solution from data instead of BCs. We have at our disposal a dataset \( D_d = \{ z_d, u_d \} \) of \( N_d \) pairs of inputs and corresponding measurements of the PDE solution. The differential operator \( \mathcal{F}(\lambda) \) is of known functional form, but may contain unknown parameters \( \lambda \). Same as before, we choose a set of collocation points, arbitrarily placed in the domain of interest in order to incorporate the PDE in the loss.

Formulating the loss

To account for the measurements, a squared-error loss term is chosen,

\[
L_{\text{data}} = \frac{1}{N_d} \sum_{i=0}^{N_d} (u_d^i - \hat{u}(z_d^i))^2,
\]

and for the collocation points we obtain again (2.3). The total PINN loss is then obtained as

\[
L_{\text{tot}} = L_{\text{data}} + \omega_c L_{\text{PDE}}
\]

and the neural network \( \hat{u} \) is trained by minimizing this loss.

Example

We consider again the second order differential equation (2.5); this time, however, we introduce an unknown parameter \( a \) (which we choose as \( a = 1 \) when simulating the data), that we want to learn from data (i.e. \( \lambda = \{a\} \)):

\[
\begin{align*}
    u''(t) &= -a^2 \sin(au(t)), \\
    u(0) &= 0, \\
    u'(0) &= a.
\end{align*}
\]

With the PINN approach, this is straightforward and only requires minor changes to the training procedure that was used for the forward problem in Section 2.2.1; that is, the loss function (2.4) is replaced by (2.7).

With classical methods, the switch from the forward to the inverse problem requires more substantial changes in the solution method. For the problem at hand, it is possible, for example, to employ a so-called shooting method, which simulates multiple trajectories for different ICs (here with RKF45) and then selects the one resulting in the largest likelihood (or lowest negative log-likelihood (NLL)) of the data.

For this example, we have employed a so-called single-shooting approach, where a numerical ODE solver is used in conjunction with a least-squares problem [Baake et al., 1992]. While this approach works well here, more
Figure 2.4: Solving the inverse problem. **Left:** Solving via the shooting method. Multiple trajectories (‘shots’) are calculated with a numerical ODE solver (here RKF45) for different parameters. **Middle:** Then the negative log-likelihood is calculated for the data given the generated trajectories. The parameter $a$ with minimal NLL is chosen as the solution. The colored dots correspond to the curves in the left plot. **Right:** The PINN approach is used to learn both, the ODE solution $u$ as well as the parameter $a$, for which the value $\hat{a} = 0.98$ is learned. Only minor changes need to be made in the training procedure when compared to the forward problem (see e.g. the losses (2.4) and (2.7)).

elaborate approaches exist, which may for example divide the domain into multiple smaller intervals and in that way combat error propagation [Müller and Timmer, 2004]. Contrary to the PINN, selecting the right approach can be challenging with classical methods and may require solvers very different from those for the forward problem.

In Figure 2.4, a comparison between the classical and the PINN approach is given. The results for the PINN are shown in the right-hand plot of Figure 2.4. Again, the PINN solution gradually approaches the correct solution as the number of iterations increases. The identified parameter value is $\hat{a} = 0.98$.

The extrapolation capability of PINNs is also on display, to the right of the last data point around $t = 8$, a feature that standard neural networks typically lack. As long as collocation points are placed in the regions of interest, PINNs can give precise solutions far away from the data, limited only by the accuracy with which the PDE parameters have been determined.

Results for the shooting method are given in the two left plots of Figure 2.4. Examples of multiple ‘shots’ are depicted in the leftmost plot. In order to determine the likelihood, it is important that the grid is chosen such that predictions are available at the same points as the data. In the middle plot, the NLL under the assumption of Gaussian noise is plotted for a range of parameter values, with the minimum at $\hat{a} = 1.02$. 
Figure 2.5: Different strategies for distributing collocation points. **Left:** The collocation points are distributed on a regular grid. **Middle:** The collocation points are distributed on a Latin hypercube. When projected either on the $x$- or the $t$-axis, exactly one point will be in each of the $N_c$ equally-sized intervals of the respective axis. **Right:** The region in which collocation points are sampled expands as the training proceeds. Points close to ICs (at $t = 0$) are sampled earlier (darker blue) than those further away (lighter blue).

### 2.3 Technical details

In this section, we discuss certain technical details that are specific to training PINNs.

#### 2.3.1 Activation functions

As is standard in neural networks, design choices in the network architecture can have a huge impact on the result. Choosing a suitable activation function requires additional care when using PINNs, since higher-order derivatives will be taken when calculating the gradient of the loss, due to the derivatives involved in the calculation of the PDE residuals. This implies that the popular ReLU activation function is not suitable for PINNs, since second-order derivatives vanish. Instead, the hyperbolic tangent activation function, which has infinitely many non-zero derivatives, is a common choice for PINNs [Cuomo et al., 2022]. The sigmoid activation function could, in principle, also be used; it has, however, the disadvantage that higher-order derivatives quickly decay which can result in too small gradients.

#### 2.3.2 Placing the collocation points

We have seen that collocation points can be distributed arbitrarily in the entire domain. One straightforward choice is to simply allocate them on a grid; given a tight enough grid, the PDE should then be well-obeyed.
everywhere. In higher dimensions, however, this approach requires many points to obtain a suitable coverage of the input space.

Latin hypercube sampling (LHS) [Stein, 1987] constitutes an alternative to sampling on a grid. It distributes $N_c$ points quasi-randomly, by splitting each axis in the considered input space into $N_c$ intervals; that way, in $d$ dimensions, we obtain $N_c^d$ volume elements. Subsequently, points are placed at random in some of these volume elements. This is done such that exactly one point will be in each of the $N_c$ intervals of the resulting marginal distributions along any axis. Compared to completely random sampling, LHS has the advantage that the samples are guaranteed to cover the entire range of possible input values. Compared to sampling on a grid, LHS has the advantage that a higher diversity of input values are sampled, whereas the grid would sample the exact same values many times.

Both the grid and the Latin hypercube cover the entire input space from the beginning. This, however, may not be the most efficient way for training the PINN. When considering the forward problem, we would expect the PINN to first learn accurate solutions close to the boundaries, where the output values are known, and only later propagate towards further away regions. This motivates the use of a schedule as to when collocation points for certain regions are added to the training procedure. One approach has been investigated in Krishnapriyan et al. [2021], where the input domain is split into a number of intervals; collocation points located in the next interval are only added once the solution in the previous one has been learned with sufficient accuracy. A somewhat more elaborate scheme has been developed in [Wang et al., 2022], where a dynamic weighting scheme for the collocation points is employed. In case of the inverse problem, an option might be to expand the potential locations for collocation points as bubbles around the measurements [Münzer and Bard, 2022].

In Figure 2.5, various schemes for allocating grid points in a 2D domain are depicted.

### 2.4 Thesis contribution: the inverse problem with unknown noise

In our work on PINNs (Paper II), we consider the case where available data is contaminated by unknown, homogeneous noise. That is, we consider the inverse problem, but loss (2.6) may no longer be appropriate.

The data loss (2.6) can be interpreted as resulting from a maximum likelihood requirement on data contaminated by Gaussian noise, which can be seen as follows: the likelihood of a measurement $u_d^i$ would be given by $p(u_d^i) = \mathcal{N}(u_d^i | u_i, \sigma^2)$, where $u_i$ is the corresponding true solution, and hence
Our aim is to replace this loss with a loss suitable for the situation we are dealing with. If the noise distribution was known, we could straightforwardly substitute a loss term corresponding to the correct likelihood. However, since we consider the case of unknown noise, an alternative way is needed.

Hence, in order to allow for unknown noise, we employ a second neural network, in the form of an energy-based model (EBM) [LeCun et al., 2006], to learn the noise probability density function (PDF) jointly with the PDE solution. This, in turn, also allows us to compute the corresponding likelihood. The assumption of homogeneity in the noise is important since it means that for a given PINN estimate of the solution, we obtain \(N_d\) data points from the unknown distribution, which serve as training data for the EBM.

The approach, which we refer to as PINN-EBM, is illustrated in Figure 2.6. To illustrate its capabilities, in Figure 2.7 we consider the simple example of the exponential differential equation

\[
\dot{x}(t) = \lambda x(t),
\]

where the data has been contaminated by noise following a multimodal distribution. In the figure, the first two plots show exemplary noise distributions as learned by the EBM are depicted, as well as an explicit solution to the problem comparing PINN and PINN-EBM. In the remaining plots, different evaluation metrics are depicted.

From these plots, it is apparent that the PINN-EBM considerably outperforms the standard PINN, both in terms of the accuracy of \(\lambda\) and the root-mean-square error with regard to the validation data. Both of these quantities would be impossible to evaluate when applying the model to real data, rendering them impractical. The log-likelihood, on the other hand, does not require knowledge of the solution and would hence be available for
Figure 2.7: Solving the exponential differential equation (8). Top left: noise distributions as learned by the EBM. Top middle: the true solution together with the PINN and PINN-EBM solution. The blue dots represent the training data, and the red dots the validation data. Top right: learned values of the parameter $\lambda$. Bottom left: The root-mean-square error on the validation data. Bottom right: The log-likelihood of the validation data given the models.

assessing the model performance also in practice. In the figure, it is apparent that the log-likelihood of the validation data also clearly indicates that the PINN-EBM delivers better results than the standard PINN.
Chapter 3

Generative adversarial networks

Generative adversarial networks (GANs) were first introduced by Goodfellow et al. [2014] as a novel approach to generative modeling (see also Section 1.3.2). The main idea is to train two neural networks jointly, the generator and the discriminator, and to let them compete against each other. The generator is trained to produce more realistic samples and the discriminator learns to distinguish between real and generated data. This is illustrated in Figure 3.1, where the dataset consists of waveforms (compare Section 3.3). In this example, the discriminator is well-trained and assigns accurate probability estimates to the shown samples. In the case of a very good generator, the discriminator would know no better than to output probabilities of 0.5 for all samples.

Since their conception, GANs have found widespread use [Gui et al., 2021, Saxena and Cao, 2021]. Apart from their well-known use in image synthesis [Wu et al., 2017], they also show promise in the sciences. In physics, they may act as substitute models for large scientific simulations [Rodriguez et al., 2018, Villaescusa-Navarro et al., 2021, Alanazi et al., 2021, Paganini et al., 2018, Kansal et al., 2023]; once a GAN has been trained, detector signals could be generated in a matter of minutes as opposed to days or weeks when using Monte Carlo techniques.

We start this chapter by outlining the basic GAN framework and discuss standard ways of improving it, resulting in the WGAN and WGAN-GP. We then proceed to discuss ways of including prior knowledge into GANs. We conclude by discussing our own extension of the GAN framework, which lies in giving a way of matching the distributions of certain statistics of interest between the real and generated data.
Figure 3.1: Schematic of a GAN. Random noise is fed into the generator to generate artificial samples $x_{gen}$. Both real samples $x_{true}$ and generated samples $x_{gen}$ are then passed to the discriminator, which assigns corresponding probability estimates that the samples are true. Here, the discriminator performs very well and produces good estimates, and/or the generator is poor.

3.1 Training generative adversarial networks

While the idea of having the discriminator and generator compete against each other may appear straightforward, it is not obvious how best to train these two models in parallel. In the following sections, we will discuss the original GAN framework as well as common modifications of it.

3.1.1 The adversarial game

In the standard GAN, as introduced by [Goodfellow et al., 2014], both the discriminator $D(\cdot|\theta_D)$ and the generator $G(\cdot|\theta_G)$ are parameterized as neural networks, where $\theta_D$ and $\theta_G$ denote the network parameters. Random noise $z \sim p_z(z)$ is input into the generator network which outputs artificial data $x_{gen} = G(z)$; the distribution $p_z(z)$ is usually chosen as a Gaussian and we denote the resulting distribution of generated samples as $x_{gen} \sim p_{gen}(x_{gen})$. In turn, both real ($x_{true}$) and generated data are fed into the discriminator network, which, for a given sample $x$, outputs its estimate $D(x)$ of the probability that the sample is real. During training, both generator and discriminator find themselves in an adversarial setup, where the former aims to have its outputs classified as real and the latter aspires to correctly distinguish between real and generated data.
Formally, this situation can be written as a minimax game with objective function

\[
\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{true}}} [\log D(x)] + \mathbb{E}_{z \sim p_z} [\log(1 - D(G(z)))].
\] (3.1)

Both terms in (3.1) can be interpreted as cross-entropies between the true labels and the probabilities assigned by the discriminator: the first term results from the cross-entropy between \([1, 0]^T\) and \([D(x), 1 - D(x)]^T\), and the second one from \([0, 1]^T\) and \([D(G(z)), 1 - D(G(z))]^T\) [Gui et al., 2021] (where the vector elements adhere to the order \([\text{true}, \text{gen}]\)). It is apparent that both cross-entropies are maximal when the discriminator classifies the samples correctly.

Considering (3.1), the following losses for training the discriminator and generator can be determined; by minimizing their respective loss functions, they would win the minimax game:

\[
\mathcal{L}^0_D = -\frac{1}{N} \sum_{i=1}^{N} [\log D(x_i) + \log(1 - D(G(z_i)))], \quad (3.2a)
\]
\[
\mathcal{L}^0_G = \frac{1}{N} \sum_{i=1}^{N} \log(1 - D(G(z_i))), \quad (3.2b)
\]

where the sum is taken over minibatch samples. Note, that two different, unrelated minibatches enter into \(\mathcal{L}^0_D\): one minibatch of true samples \(x_i\) and one minibatch of generated samples \(G(z_i)\), that is \(2N\) samples in total.

Goodfellow et al. [2014] analyzed the game (3.1) and obtained a row of important theoretical results: firstly, they show that for a fixed generator \(G\), the optimal discriminator would assign the following probabilities:

\[
D_G^*(x) = \frac{p_{\text{true}}(x)}{p_{\text{true}}(x) + p_{\text{gen}}(x)}. \quad (3.3)
\]

Hence, given a perfect generator, the discriminator would assign a probability of 0.5 to each sample.

Furthermore, they show that, with \(G\) still fixed, (3.1) can be written in terms of the Kullback-Leibler (KL) divergence, and ultimately the Jensen-Shannon (JS) divergence:

\[
C(G) = \max_D V(G, D_G^*)
\]
\[
= -\log(4) + \text{KL} \left( p_{\text{true}} \left| \left| \frac{p_{\text{true}} + p_{\text{gen}}}{2} \right\right) + \text{KL} \left( p_{\text{gen}} \left| \left| \frac{p_{\text{true}} + p_{\text{gen}}}{2} \right\right)
\]
\[
= -\log(4) + 2 \text{JS} (p_{\text{true}} \| p_{\text{gen}}). \quad (3.4)
\]
Algorithm 1: Training the GAN

**Input:** Untrained $D(\cdot|\theta_D)$ and $G(\cdot|\theta_G)$; true data $\{x_{\text{true}}\}$

**Result:** Trained $D$ and $G$

while Training do

for $j \in \text{range}(N_d)$ do

Sample minibatch $x_{\text{true}}$;

Sample minibatch $x_{\text{gen}} = G(z)$ with $z \sim p_z$;

$L_D = -\text{mean} \left[ \log D(x_{\text{true}}) + \log(1 - D(x_{\text{gen}})) \right]$;

$\theta_D \leftarrow \theta_D - \gamma \nabla_{\theta_D} L_D$;

end

Sample minibatch $x_{\text{gen}} = G(z)$ with $z \sim p_z$;

$L_G = \text{mean} \left( \log(1 - D(x_{\text{gen}})) \right)$;

$\theta_G \leftarrow \theta_G - \gamma \nabla_{\theta_G} L_G$;

end

Since the JS divergence attains its minimum value only when both of its arguments are the same, this implies that $p_{\text{gen}} = p_{\text{true}}$ is the optimal solution when minimizing $C(G)$ with respect to $G$.

Finally, they show that training the generator and discriminator alternatingly, with losses (3.2), and allowing the discriminator (which is assumed to be expressive enough) to reach its optimum at each iteration, will indeed result in this optimal solution. The algorithm for training the GAN is given in Algorithm 1.

These results show that the GAN framework rests on sound theoretical foundations. However, in practice, training GANs can be very tricky and it is common that (3.2) saturates early in the training, when the discriminator can easily tell the still very fake looking generated examples from the real data. It is common, as a heuristic, to replace the terms in (3.2b) with $-\log(D(G(z_i)))$ instead, to provide larger gradients for the generator early on; this mitigates the problem to some degree but can lead to instabilities in the training [Gui et al., 2021]. Another common issue with GANs is that of mode collapse [Saxena and Cao, 2021]: they are prone to learning merely part of the true data distribution and in turn, generate only very similar-looking samples.

### 3.1.2 A better way of comparing the distributions

Due to the aforementioned difficulties, it is important to find ways of improving the training procedure. Arjovsky et al. [2017] investigated the use of measures of distance between probability distributions for the GAN losses, instead of the cross-entropies. They introduced the Wasserstein GAN
3.1. Training generative adversarial networks

The Earth-Mover (EM) distance, or Wasserstein-1 metric. Roughly speaking, it can be thought of as the probability mass times the distance it needs to be transported in order to transform one PDF into the other. When considering two distributions that are shifted by a distance $\Delta x$ along the x-axis, but otherwise equal, the EM distance is given by the actual distance between both distributions. **Bottom row:** Comparison of different metrics. The KL divergence gives infinity as soon as there is a small region where $p_{true} > 0$ and $p_{gen} = 0$ (or vice versa). The JS divergence will only change for different amounts of overlap between $p_{true}$ and $p_{gen}$. The EM distance is the only one that gives expressive results for all relative shifts $\Delta x$ of the distributions.

(WGAN), where they employ the Wasserstein metric $W$, also known as Earth-Mover (EM) distance, as the new GAN-loss.

Intuitively, the EM distance can be thought of as quantifying the amount of work required to transform one probability distribution into another; i.e. the volume of probability density weighted according to the distance that it is moved. The EM distance is visualized in Figure 3.2.

More formally, it is defined as

$$W(p_{true}, p_{gen}) = \inf_{\gamma \in \Pi(p_{true}, p_{gen})} \mathbb{E}_{(x,y) \sim \gamma}[||x - y||], \quad (3.5)$$

where $\Pi(p_{true}, p_{gen})$ is the set of all joint distributions $\gamma(x, y)$ whose marginals are $p_{true}$ and $p_{gen}$. By making use of the Kantorovich-Rubinstein duality, it can be rewritten as a maximization over all 1-Lipschitz functions:

$$W(p_{true}, p_{gen}) = \sup_{||f||_{L^1}} \mathbb{E}_{x \sim p_{true}}[f(x)] - \mathbb{E}_{x \sim p_{gen}}[f(x)]. \quad (3.6)$$
The EM distance has notable advantages over other commonly used metrics to compare probability distributions. When considering the definition of the KL divergence, \( \text{KL}(p_{true} || p_{gen}) = \int_{-\infty}^{\infty} p_{true}(x) \log \left( \frac{p_{true}(x)}{p_{gen}(x)} \right) dx \), for example, it is apparent that it would yield infinity for all the cases displayed in Figure 3.2. The JS divergence would yield only a constant value, as soon as there is no overlap between the two distributions. The EM distance, on the other hand, gives results that are proportional to the actual distance between the two distributions. In turn, useful gradients for training the GAN are obtained, regardless of the (dis)similarity of the distributions.

In the end, the expectations in (3.6) are approximated by estimating over minibatches; then, the losses for the discriminator and generator are given by

\[
\mathcal{L}_D = -\frac{1}{N} \sum_{i=1}^{N} D(x_i) - D(G(z_i)), \quad (3.7)
\]

\[
\mathcal{L}_G = -\frac{1}{N} \sum_{i=1}^{N} D(G(z_i)). \quad (3.8)
\]

The main advantage of the WGAN is that the discriminator \( D \) now acts as a critic that can assign arbitrary scores to generated samples, instead of being limited to probabilities in the range \([0, 1]\). This means that it can provide meaningful gradients for the generator also when it is very certain about its predictions. Hence, it is no longer crucial to maintain a balance between both \( D \) and \( G \) during training (unlike before, with generator loss (3.2b)). In fact, it can even be beneficial to train the critic for multiple steps in each iteration in order to provide better gradients for the generator.

### 3.1.3 Penalizing instead of clipping

When training WGANs, it is important to enforce the Lipschitz constraint in (3.6); although, as Arjovsky et al. [2017] show, it is sufficient to require K-Lipschitz continuity. Rather crudely, this can be enforced by clipping the weights of the discriminator to some maximum value.

While effective in stabilizing the training, the weight-clipping employed in the WGAN can severely reduce the expressive capacity of the discriminator network; in turn, this may also impact the ability of the discriminator to correctly identify real and generated samples. In order to improve upon this coarse procedure of ensuring Lipschitz continuity, Gulrajani et al. [2017] proposed to instead add a gradient penalty term to the discriminator loss:

\[
\mathcal{L}^{GP}_D = \mathcal{L}_D + \lambda \frac{1}{N} \sum_{i=1}^{N} (||\nabla_{\hat{x}_i} D(\hat{x}_i)||_2 - 1)^2, \quad (3.9)
\]
3.1. Training generative adversarial networks

Algorithm 2: Training the WGAN and WGAN-GP.

**Input:** Untrained $D(\cdot|\theta_D)$ and $G(\cdot|\theta_G)$; true data $\{x_{\text{true}}\}$

**Result:** Trained $D$ and $G$

while Training do

for $j \in \text{range}(N_d)$ do

Sample minibatch $x_{\text{true}}$;  
Sample minibatch $x_{\text{gen}} = G(z)$ with $z \sim p_z$;  
$L_D = \text{mean } (D(x_{\text{gen}}) - D(x_{\text{true}}))$;

if WGAN-GP then

$\epsilon \sim \text{Uniform}[0, 1]$;  
$\hat{x} = x_{\text{true}} + (1 - \epsilon)x_{\text{gen}}$;  
$L_{\text{GP}} = \text{mean } (||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2$;  
$L_D = L_D + \lambda L_{\text{GP}}$;

end

$\theta_D \leftarrow \theta_D - \gamma \nabla_{\theta_D} L_D$;

if WGAN then

clip weights of D;

end

end

Sample minibatch $x_{\text{gen}} = G(z)$ with $z \sim p_z$;  
$L_G = -\text{mean } (D(x_{\text{gen}}))$;

$\theta_G \leftarrow \theta_G - \gamma \nabla_{\theta_G} L_G$;

end

where $\lambda$ is a weighting factor. The inputs $\hat{x}_i$ are obtained as random points between true and generated samples (from the minibatches used to calculate $L_D$) via $\hat{x}_i = \epsilon x_{i_{\text{true}}} + (1 - \epsilon)x_{i_{\text{gen}}}$, where $\epsilon \sim \mathcal{U}[0, 1]$. This additional term acts as a soft constraint to compel the discriminator towards Lipschitz continuity.

This method, which is referred to as WGAN-GP, effectively combats the issue of capacity underuse in the discriminator network of the WGAN, where weights will often converge to their minimum/maximum allowed values if clipped [Gulrajani et al., 2017]. As a result, in WGAN-GP the critic has the potential to learn more complex structures in the data, yielding in turn more accurate feedback for the generator. One drawback is the increased training time of WGAN-GP due to the new loss term, which requires additional gradient evaluations.

In Algorithm 2, the training procedure for WGAN and WGAN-GP is summarized. The parameter $N_d$ gives the number of iterations the discriminator is trained before each generator update.
3.2 Including prior knowledge

In this section, we provide some general considerations on incorporating constraints into GANs, and then proceed to discuss related work and our own work on matching the distributions of dataset statistics (Paper III).

3.2.1 General considerations

When constraining GANs, the general techniques from Section 1.3.4 are applicable. One special consideration that needs to be made for GANs in particular, however, lies in the fact that we now have two models to choose from. Should the constraint be incorporated in the discriminator, the generator, or both?

When considering the method of feeding additional inputs into the model, it is common to feed constraint residuals, i.e. values quantifying how strongly a given sample violates the constraint, into the discriminator [Stinis et al., 2019, Yang et al., 2019]. That way, the discriminator possesses explicit information about the constraint and can distinguish between real and generated data accordingly.

When additional information is given not in terms of quantities to be extracted from the samples, but instead in the form of e.g. class labels or text, then it is common to input the real and generated labels into both the discriminator and generator, respectively. Including prior knowledge of this kind falls into the broad framework of conditional GANs [Mirza and Osindero, 2014].

While it will depend, as so often, on the particular case at hand, when incorporating constraints via additional terms in the loss function, it is most common to add the constraint to the generator [Khattak et al., 2018, 2019, Yang et al., 2021]. The rationale behind this is that the task of the discriminator is usually easier, and it is the generator that is lagging behind. Hence, the constraint can be thought of as additional help for the generator. On the other hand, adding additional terms to both losses can allow for more balanced gradients [Daw et al., 2021].

3.2.2 Existing work

In this section, we will focus on existing work that tries to match statistics (or, equivalently, features) of real and generated data. Typically, these features are extracted from the data and the aim is to improve convergence of the GAN training process, as well as to improve the overall quality and diversity of the generated samples.
3.3. Thesis contribution: probabilistically constrained GANs

Using features instead of ratings

In Salimans et al. [2016], the idea is to substitute a new loss aiming to directly match dataset features of real and generated data for the standard generator loss (3.2b), instead of using the outputs of the discriminator. Here, the features are not manually chosen, but instead, the activations $f(x)$ of an intermediate layer of the discriminator are chosen; the discriminator is trained as usual. The new generator loss then becomes

$$L'_G = \left\| \frac{1}{N} \sum_{i=1}^{N} f(x_i) - f(G(z_i)) \right\|,$$

that is the means of the real and generated feature distributions are being matched. In the paper, they claim that this loss results in more stable GAN training.

Matching the covariance structure

In Wu et al. [2016], the generator loss is augmented with an additional term comparing the second-order moments of real and generated data:

$$L^c_G = L_G + \lambda d(\Sigma(p_{true}), \Sigma(p_{gen})),$$

(3.11)

where the Frobenius norm was chosen to measure the distance of the two covariance structures:

$$d(\Sigma(p_{true}), \Sigma(p_{gen})) = \|\Sigma(p_{true}) - \Sigma(p_{gen})\|_F$$

(3.12)

The covariances of the generated data $\Sigma(p_{gen})$ are estimated from the minibatch at each iteration. This approach and the choice of matching covariances were specifically designed for use in generating solutions for PDEs. Note that also here, only point estimates of the covariances enter into the loss; hence it is only the second-order moments that are being matched and not the full distribution.

3.3 Thesis contribution: probabilistically constrained GANs

Neither of the approaches that we have seen so far considered the actual shape of the distributions under consideration. In our work (Paper III), we do exactly that, by adding the KL divergences between true and generated feature distributions to the generator loss:

$$L^c_G = L_G + \lambda \sum_{s=1}^{N_s} KL(p_{true}(z_s) || p_{gen}(z_s)),$$

(3.13)
Figure 3.3: **Left:** Two samples from the synthetic dataset. **Right:** The histograms give the distributions of the respective power spectrum components as obtained from the data. The PDF as obtained via the EBM acts as a representation of the true distribution, whereas KDE is employed to obtain estimates of the PDFs from minibatches.

where the $z_s$ denote the features under consideration. Since the constraint does not act on individual samples, but instead on the distribution of dataset statistics, we refer to this approach as probabilistically constrained GAN (pcGAN).

**Approximating the distributions**

In order to be able to calculate the KL divergence between the two distributions, we require a representation for the respective PDFs. The true distribution of the statistics under consideration can be modeled by a conditional energy-based model (EBM); since the true distribution does not change, it suffices to determine it once ahead of the training.

The generated distribution, on the other hand, will change as the generator evolves, and hence training an EBM would be inefficient. We choose to estimate the generated distribution at each iteration instead via kernel density estimation (KDE) from the current minibatch:

$$
\tilde{p}_{\text{gen}}(z_s) = \frac{1}{n_{\text{batch}}} \sum_{i=1}^{n_{\text{batch}}} \mathcal{N}(z_s; z_{si}, \sigma_s).
$$

(3.14)

**Example**

For the purpose of experimentation, we consider a superposition of two sine waves $x = \frac{1}{2} \sum_{i=1}^{2} \sin(\omega_i t)$, with angular frequencies sampled randomly from $\omega_i \sim |\mathcal{N}(1, 1)|$, and where $t \in \text{linspace}(0, 20, 200)$.

In the left part of Figure 3.3, two samples from this dataset are depicted. In the right part, a comparison of the different methods employed for obtaining representations for the PDFs of the distributions of power spectrum components is given. The EBM constitutes an excellent approximation of
Figure 3.4: The distributions of different power spectrum components (ps\([i]\) denotes the \(i\)-th component), as obtained from the different GANs. From left to right, the different columns show the results as obtained from the true data, the WGAN, the pcGAN, and the method of Wu et al. [2020]. The histograms depict the respective distributions of the components, and the orange lines show the PDFs of the true distributions as learned via the EBM. The pcGAN manages to match real and generated distribution very well, whereas the other methods exhibit clear mismatches between the distributions.

The real data distribution, which is depicted by the histogram. The estimates obtained via KDE (here with minibatches sampled from true data) are reasonable and become better as the batch size increases.

In Figure 3.4, we consider power spectrum components and compare the matches of the distributions for the different models. It is apparent that the pcGAN succeeds at its task of matching the distributions of the statistics under consideration (here the power spectrum components of the waveforms). It significantly outperforms the standard WGAN, which does not cover the whole range of possible values. While the method of Wu et al. [2020] performs better than the WGAN, it still does not manage to match the actual shapes of the distributions.
Chapter 4

Gaussian processes

Gaussian processes (GPs) are powerful statistical models that can be used for both regression and classification. A notable advantage of GPs over other regression models is that they do not only give a point estimate but instead yield a distribution of potential solutions. GPs are Bayesian models, and as such provide credible intervals. Thus it is straightforward to get a good idea of the trustworthiness of the GP prediction at a given point. Intuitively, the GP can be thought of as a distribution over functions.

The GP is a commonly used model in the area of geostatistics, where it is known as kriging, with pioneering work done around 1970 [Matheron, 1973]. Over the years, GPs have become a popular tool and they are widely used [Rasmussen and Williams, 2006].

We start the chapter by outlining the connection between standard linear regression and GPs. We then discuss the basics of GP regression and give an overview of different extensions of the GP framework, including approximate inference and constrained GPs. Finally, we present our method of incorporating sum constraints into the GP.

4.1 From linear regression to Gaussian processes

The objective of regression models is to learn a mapping from inputs $x$ to outputs $y$. While many different regression models exist, in this section we restrict ourselves to a sequence of extensions of standard linear regression, which will eventually culminate in the GP. For additional information on the models discussed in this section, see e.g. Lindholm et al. [2021].

4.1.1 Linear regression

Linear regression is probably the best-known regression model and it is very powerful despite its simplicity. In the following, we consider the one-
dimensional case, where we are given a dataset $D = \{ (x_i, y_i) \}_{i=1}^N$ of $N$ pairs of inputs $x_i$ and outputs $y_i$; both the inputs and the outputs are one-dimensional. As a first try, we can fit the data as a straight line via

$$y = \theta_0 + \theta_1 x + \epsilon,$$  \hspace{1cm} (4.1)

where we assume that the data has been contaminated by Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$. The parameters $\theta_0$ and $\theta_1$ give the offset and the slope of the line, respectively, and need to be fitted to the data. If we suspect that the function in question may be non-linear, we can also employ a higher-order polynomial of order $p$, by adding additional terms to (4.1); e.g. for the case $p = 3$, we have

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \epsilon.$$  \hspace{1cm} (4.2)

In principle, arbitrary transformations of $x$ can be considered, and we can define a vector $\phi(x)$ of transformed inputs with a corresponding vector of parameters $\theta$; for example, $\phi(x) = [1, x, x^2, x^3]^T$ and $\theta = [\theta_0, \theta_1, \theta_2, \theta_3]^T$ in (4.2). In terms of $\phi$ and $\theta$, the relationship between $x$ and $y$ can be written as

$$y = \theta^T \phi(x) + \epsilon.$$  \hspace{1cm} (4.3)

It remains to determine the values of the parameters $\theta$. In order to find a good fit to the data, it is common to use the maximum likelihood approach; that is, we determine $\theta$ such that the data $D$ is as likely as possible given our model. For data with Gaussian noise, where the likelihood $p(y_i|\theta) = \mathcal{N}(y_i|\theta^T \phi(x_i), \sigma^2)$, this results in the optimization problem

$$\hat{\theta} = \arg \min_{\theta} \frac{1}{N} \sum_{i=1}^N \left( \theta^T \phi(x_i) - y_i \right)^2 \approx \arg \min_{\theta} \frac{1}{N} ||\Phi \theta - y||_2^2,$$  \hspace{1cm} (4.4)

where we have introduced the matrix $\Phi = [\phi(x_1), \phi(x_2), \ldots, \phi(x_N)]^T$ and the vector $y = [y_1, y_2, \ldots, y_N]^T$. The following analytical solution can be derived:

$$\hat{\theta} = (\Phi^T \Phi)^{-1} \Phi^T y.$$  \hspace{1cm} (4.5)

When using higher-order polynomials to fit the data, linear regression is prone to overfitting since a polynomial of order $p$ can fit $p + 1$ points exactly (compare the left plot in Figure 4.1). In order to mitigate this issue, regularization can be employed by adding a penalty term to (4.4); if we add an L2-term, we receive

$$\hat{\theta} = \arg \min_{\theta} \frac{1}{N} ||\Phi \theta - y||_2^2 + \lambda ||\theta||_2^2.$$  \hspace{1cm} (4.6)
4.1. From linear regression to Gaussian processes

4.1.1 From linear regression to Ridge regression

Figure 4.1: Comparison of different regression models. **Left:** Fitting a polynomial of order $p$ via standard linear regression. **Middle:** Fitting a polynomial of order $p = 6$ via Ridge regression, with different values of the regularization parameter $\lambda$. **Right:** Employing kernel Ridge regression with different length-scales $l$ to fit the data.

This form of regularization is referred to as Ridge regression or Tikhonov regularization, and the solution to (4.6) is given by

$$\hat{\theta} = (\Phi^T \Phi + N\lambda I)^{-1} \Phi^T y. \tag{4.7}$$

The main effect of the regularization term is that large parameter values are discouraged, and the data needs to provide stronger evidence for the model to include them.

4.1.2 The kernel trick

In principle, there is no limit to the number of features we can include in our model. However, a high number of features will result in high computational demands, and for this practical reason, having an efficient way of taking them into account is important. Writing out the expression for predicting $y_*$ at a new point $x_*$ (by combining (4.3) and (4.7)) gives

$$y_* = y^T \Phi (\Phi^T \Phi + N\lambda I)^{-1} \phi(x_*) \tag{4.8a}$$

$$= y^T (\Phi \Phi^T + N\lambda I)^{-1} \Phi \phi(x_*), \tag{4.8b}$$

where the push-through matrix identity, $A(A^T A + I)^{-1} = (AA^T + I)^{-1} A$, has been used in the second line. From (4.8b), it is apparent that the features $\phi(x)$ never show up individually, but only in terms of inner products $\phi(x)^T \phi(x')$. This motivates the introduction of a kernel,

$$\kappa(x, x') := \phi(x)^T \phi(x'), \tag{4.9}$$

which allows to rewrite (4.8b) as

$$y_* = y^T (K(x, x) + N\lambda I)^{-1} \kappa(x, x_*); \tag{4.10}$$
the Gram matrix $K(x, x)$ is evaluated element-wise such that $K_{ij} = (\Phi\Phi^T)_{ij} = \kappa(x_i, x_j)$.

In this way, it is no longer required to pick individual features, but choosing a kernel suffices. In our choice of kernel, we are limited by the requirement that the inverse in (4.8b) needs to exist. This can be ensured by restricting ourselves to positive semidefinite kernels, for which the Gram matrix is always positive semidefinite [Lindholm et al., 2021].

Depending on the choice of kernel, employing the kernel may be equivalent to having an infinite number of features. For example, this is the case for the squared exponential kernel

$$\kappa(x, x') = \exp\left(-\frac{||x - x'||^2}{2l^2}\right),$$

where $l$ denotes the length-scale parameter, which determines how quickly the outputs can vary. This kernel corresponds to having infinitely many Gaussian-shaped features $\phi_c(x) = \exp\left(-\frac{(x-c)^2}{2l^2}\right)$, with $c \in [-\infty, \infty]$ (up to a constant factor in the length-scale) [Rasmussen and Williams, 2006].

Using linear regression together with $L_2$ regularization and a kernel is called kernel Ridge regression. Making use of kernels allows for significantly more flexible fits to the data than standard linear regression with only a few features. A comparison of the different linear regression models discussed in this section is given in Figure 4.1.

### 4.1.3 Beyond the point estimate

The linear regression approaches we considered so far have in common that they give a point estimate for the parameters. However, it is often desirable to quantify uncertainty in the predictions, and to take into account prior knowledge. This can be achieved with the Bayesian approach.

In the Bayesian approach, prior information and beliefs on the parameters $\theta$ can be encoded in the prior $p(\theta)$. The posterior, i.e. the parameter estimates after taking the available data into account, is then given by

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)},$$

which follows from Bayes’ theorem. The conditional distribution $p(y|\theta)$ is called the likelihood, and it gives the probability of measuring $y$ given our model with parameters $\theta$. The term $p(y) = \int p(y|\theta)p(\theta)d\theta$ in the denominator is called the marginal likelihood and it gives the likelihood of the data after taking into account all possible values of $\theta$, weighted according to the prior.
4.2 Gaussian process regression

In Lindholm et al. [2021], it is shown that applying the Bayesian approach to kernel Ridge regression, or, equivalently, the kernel trick to Bayesian linear regression, results in the GP.

4.2 Gaussian process regression

Having discussed relevant extensions of linear regression, we can now turn our attention to the GP framework. In this section, we discuss the different components of GP regression, from the basic definitions to hyperparameter tuning. In contrast to the preceding sections, we will now consider high-dimensional inputs $x$, instead of one-dimensional ones. For more in-depth discussions, see e.g. Rasmussen and Williams [2006] or Lindholm et al. [2021].

4.2.1 Basic definitions

According to Rasmussen and Williams [2006, p. 13], a Gaussian process (GP) is “a collection of random variables, any finite number of which have a joint Gaussian distribution” and it is “completely specified by its mean function and covariance function”. For a GP $f(x)$, the mean function $m(x)$ and the covariance function (or kernel) $k(x, x')$ can be defined in terms of expectations of the GP,

$$m(x) = \mathbb{E}[f(x)] \quad (4.13a)$$
$$k(x, x') = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))], \quad (4.13b)$$

and the GP can be written as

$$f(x) \sim \mathcal{GP} \left( m(x), k(x, x') \right). \quad (4.14)$$

Sampling functions from the GP, evaluated at $n_*$ test inputs $X_* = [x_{*1}, x_{*2}, \ldots, x_{*n_*}]^T$, can then be done by sampling from the multivariate normal distribution

$$f_* = \mathcal{N}(m_*, K_{**}), \quad (4.15)$$

where the mean $m_* = m(X_*)$ and the covariance matrix $K_{**} = k(X_*, X_*)$ are obtained by evaluating the mean and covariance function, respectively, on $X_*; \text{ more precisely, } m_{*i} = m(x_{*i})$ and $K_{**ij} = k(x_{*i}, x_{*j})$.

These samples correspond to the prior distribution since no conditioning on any measurements has been performed yet. Intuitively, sampling from a GP can be thought of as sampling random functions.
4.2.2 The predictive distribution

When sampling functions from the GP, we want to take into account data in the form of noisy measurements \( y = f + \epsilon \) at the training inputs \( X \), where \( \epsilon \sim \mathcal{N}(0, \sigma_n^2I) \). We start by considering the joint distribution of the function values \( f \) and the function values \( f_* \) corresponding to test inputs \( X_* \):

\[
\begin{bmatrix} f \\ f_* \end{bmatrix} \sim \mathcal{N} \left( \begin{bmatrix} m \\ m_* \end{bmatrix}, \begin{bmatrix} K & K_* \\ K_*^T & K_{**} \end{bmatrix} \right),
\]

(4.16)

where \( m = m(X) \), \( m_* = m(X_*) \), \( K = k(X, X) \), \( K_* = k(X, X_*) \) and \( K_{**} = k(X_*, X_*) \).

In order to obtain the distribution for the noisy measurements \( y \), we need to take the noise into account via an additional term in the covariance matrix:

\[
\begin{bmatrix} y \\ f_* \end{bmatrix} \sim \mathcal{N} \left( \begin{bmatrix} m \\ m_* \end{bmatrix}, \begin{bmatrix} K + \sigma_n^2I & K_* \\ K_*^T & K_{**} \end{bmatrix} \right).
\]

(4.17)

Since we are interested in the predictions \( f_* \) given the measurements \( y \), we need to construct the corresponding conditional distribution. This can be done via Bayes' theorem (compare Section 4.1.3):

\[
f_*|X, y, X_* \sim \mathcal{N} \left( \bar{f}_*, \text{cov}(f_*) \right), \quad \text{where}
\]

(4.18a)

\[
\bar{f}_* \triangleq \mathbb{E}[f_*|X, y, X_*] = m_* + K_*^T (K + \sigma_n^2I)^{-1} (y - m),
\]

(4.18b)

\[
\text{cov}(f_*) = K_{**} - K_* (K + \sigma_n^2I)^{-1} K_*.
\]

(4.18c)

In order to obtain (4.18), standard relations for the multivariate Gaussian distribution were employed (see e.g. Lindholm et al. [2021]). Note that the mean prediction (4.18b) corresponds to the predictive equation (4.10); the covariance (4.18c), however, is new in the GP approach.

4.2.3 Kernels

The kernel \( k(\cdot, \cdot) \) is an essential constituent of the GP, as it greatly influences the properties of the prior functions. A very common choice of kernel is the squared exponential (SE) kernel (4.11), also known as radial basis function (RBF) kernel. Functions sampled from a GP with this kernel are smooth, where the length-scale hyperparameter \( l \) roughly determines the distances between adjacent minima and maxima. In Figure 4.2, samples from both prior and posterior functions obtained with this kernel are depicted.
Figure 4.2: Example of GP regression with squared exponential (SE) kernel. **Left:** Samples from the prior distribution, together with mean and credible intervals. **Middle:** Samples from the posterior distribution, together with mean and credible intervals. The GP has been conditioned on the data (black dots), but no hyperparameter tuning has taken place ($l = 0.97, \sigma_n = 0.97$). **Right:** Samples from the posterior distribution, together with mean and credible intervals. The GP has been conditioned on the data (black dots), after hyperparameter tuning ($l = 1.70, \sigma_n = 5.36$).

Figure 4.3: Example of GP regression with Matérn kernel ($\nu = 0.5$). **Left:** Samples from the prior distribution, together with mean and credible intervals. **Middle:** Samples from the posterior distribution, together with mean and credible intervals. The GP has been conditioned on the data (black dots), but no hyperparameter tuning has taken place ($l = 0.97, \sigma_n = 0.97$). **Right:** Samples from the posterior distribution, together with mean and credible intervals. The GP has been conditioned on the data (black dots), after hyperparameter tuning ($l = 1.61, \sigma_n = 0.01$).
Another common kernel, which gives rise to rugged functions, is the Matérn kernel,

\[ k_{\text{Matérn}}(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left( \frac{\sqrt{2\nu r}}{l} \right)^\nu K_\nu \left( \frac{\sqrt{2\nu r}}{l} \right), \]

(4.19)

with \( r = |x - x'| \), where \( \nu \) and \( l \) are positive hyperparameters, where \( \Gamma \) denotes the gamma function, and where \( K_\nu \) denotes a modified Bessel function. The hyperparameter \( l \) gives the length-scale, and \( \nu \) determines the ruggedness of the functions. Sample functions are depicted in Fig. 4.3; this kernel may find use in applications where the assumption of smoothness in the outputs does not hold.

Many more kinds of kernels exist, such as the periodic kernel or the linear kernel. It is also possible to construct new kernels by combining existing kernels, e.g. by summing them or forming the product between them. In the former case, one of the kernels might have a large length-scale and capture large-scale features whereas the other one may have a small length-scale allowing to fit the more fine-grained characteristics of the data. The best choice of kernel strongly depends on the problem under consideration.

### 4.2.4 Hyperparameter tuning

We have seen that different types of kernels exist, suitable for different applications. They have in common that the resulting GP outputs depend on the kernel hyperparameters. Given prior knowledge of these parameters, they could be fixed to that value, but usually, this is not the case and we are in need of a systematic way of choosing them.

To this end, we consider the log-marginal likelihood,

\[ \log p(y|X) = \log \left( \int p(y|f, X)p(f|X)df \right) \]

(4.20)

\[ = -\frac{1}{2}(y - m)^T(K + \sigma_n^2 I)^{-1}(y - m) \]

\[ - \frac{1}{2} \log |K + \sigma_n^2 I| - \frac{n}{2} \log 2\pi, \]

(4.21)

and require that it be maximal, i.e. that the measurements \( y \) are as likely as possible given our GP. The ‘marginal’ in the name log-marginal likelihood stems from the fact that we are marginalizing over all possible realizations of the function \( f \) in (4.20), when determining the likelihood of \( y \). With this criterion, we can determine values for the hyperparameters.

In practice, an exhaustive search of the hyperparameter space will be infeasible most of the time and it is common to resort instead to methods such
as stochastic gradient descent in order to determine suitable hyperparameter values.

4.3 Extensions of the Gaussian process framework

In this section, we discuss different additions and modifications of the GP framework, which will be relevant later on. In particular, we show how to modify the GP framework in order to take into account multiple outputs, and how non-Gaussian likelihoods can be dealt with.

4.3.1 Multitask Gaussian processes

In the multitask setting, we consider GPs with \( N_f \) outputs \( f_i \) at each point \( x \) in the input space. The \( N_f \) different functions which are learned are also referred to as tasks. In this case, there can also be task correlations between the different outputs, in addition to spatial correlations between different inputs. Purely spatial correlations are captured by the data kernel \( k_d(\cdot, \cdot) \), and correlations between tasks (which may also be input-dependent) are included in the task kernel \( k_t(\cdot, \cdot) \).

The mean and covariance matrix of the multitask GP are then given by

\[
\mathbf{m}_f(X) = [m_d(x_1)\mathbf{m}_t(x_1)^T, \ldots, m_d(x_N)\mathbf{m}_t(x_N)^T]^T, \tag{4.22}
\]

\[
\mathbf{K}_{f,f'}(X, X) = \begin{bmatrix}
k_{d11}k_t(x_1, x_1) & k_{d12}k_t(x_1, x_2) & \cdots \\
k_{d21}k_t(x_2, x_1) & k_{d22}k_t(x_2, x_2) & \cdots \\
\vdots & \vdots & \ddots
\end{bmatrix}, \tag{4.23}
\]

where \( m_d(\cdot) \) is the data mean, \( \mathbf{m}_t(\cdot) \) the task mean, and where \( k_{dij} = k_d(x_i, x_j) \). In the case where the task components are input-independent, these expressions simplify to Kronecker products:

\[
\mathbf{m}_f(X) = m_d(X) \otimes \mathbf{m}_t, \tag{4.24a}
\]

\[
\mathbf{K}_{f,f'}(X, X') = k_d(X, X') \otimes k_t. \tag{4.24b}
\]

The multitask GP is then written as \( f \sim \mathcal{N}(\mathbf{m}_f(X), \mathbf{K}_{f,f'}(X, X')) \), where \( f = [f_1^T, f_2^T, \ldots, f_N^T]^T \) and \( f_k = f(x_k) \).

4.3.2 Approximate inference

In the case of data with non-Gaussian noise and hence non-Gaussian likelihood, the posterior \( p(f|y) \) will also be non-Gaussian; this means that a closed-form solution will in general no longer be feasible. In such cases, a variety of approximation methods exist to deal with this issue. In the following, we will consider two common methods of approximating the posterior.
distribution as a Gaussian, i.e. of determining \( q(f) = \mathcal{N}(f|m_q, \Sigma_q) \) such that

\[
p(f|y) = \frac{p(y|f)p(f)}{p(y)} \approx q(f). \tag{4.25}
\]

The Laplace approximation

When using the Laplace approximation, the aim is to fit \( q(f) \) to the highest peak of the posterior distribution. In order to achieve this, the maximum \( f_{\text{max}} \) of \( p(f|y) \) (or, equivalently, \( \log p(f|y) \)) needs to be determined, together with the Hessian of \( -\log p(f|y) \) at that point, which captures the curvature. Note, that knowledge of the marginal likelihood \( p(y) \) is not required in order to do this, since it does not affect the position of \( f_{\text{max}} \).

Following Rasmussen and Williams [2006], the following relationship can be derived for \( f_{\text{max}} \):

\[
0 = \nabla_f \log (p(y|f)p(f)) |_{f=f_{\text{max}}} \tag{4.26}
= (\nabla_f \log p(y|f) - \mathbf{K}^{-1}f) |_{f=f_{\text{max}}}, \tag{4.27}
\rightarrow f_{\text{max}} = \mathbf{K}(\nabla_f \log p(y|f))|_{f=f_{\text{max}}}. \tag{4.28}
\]

For the Hessian, we obtain

\[
\Sigma_{\text{max}}^{-1} = -\nabla_f \nabla_f \log p(y|f)|_{f=f_{\text{max}}} \tag{4.29}
= -\nabla_f \nabla_f \log p(y|f)|_{f=f_{\text{max}}} + \mathbf{K}^{-1}. \tag{4.30}
\]

Given these quantities, we obtain \( q(f) = \mathcal{N}(f|f_{\text{max}}, \Sigma_{\text{max}}) \) for the approximate posterior. This approach is depicted in the left plot of Figure 4.4.

The variational approach

The idea of the variational approach is to fit a parameterized distribution to the true distribution by minimizing the difference between them. In our case, this means determining the entries of the mean and covariance matrix of \( q(f) \) by minimizing the KL divergence between \( q(f) \) and the posterior \( p(f|y) \),

\[
\hat{\theta}_q = \arg \min_{\theta_q} D_{KL} (q(f) || p(f|y)), \tag{4.31}
\]
4.3. Extensions of the Gaussian process framework

4.3.1 Approximating non-Gaussian distributions

Figure 4.4: Comparison of ways to approximate a non-Gaussian distribution as a Gaussian. **Left:** The Laplace approximation matches a Gaussian to the maximum of the distribution. **Right:** When using the variational approach, the KL divergence (or evidence lower bound (ELBO), if the KL divergence is intractable) between variational and true distribution is minimized.

where $\hat{\theta}_q = \{m_q, \Sigma_q\}$. The KL divergence can be written as

$$D_{KL}(q(f)||p(f|y)) = \int q(f) \log \left( \frac{q(f)}{p(f|y)} \right) df$$

$$= \int q(f) \log \left( \frac{q(f)p(y)}{p(f,y)} \right) df$$

$$= \mathbb{E}_{q(f)}[\log q(f)] - \mathbb{E}_{q(f)}[\log p(f,y)] + \log p(y)$$

$$= -\text{ELBO}(q) + \log p(y).$$

Since the marginal likelihood (or evidence) $p(y)$ is typically intractable, the evidence lower bound (ELBO) is minimized instead of the full KL divergence in such cases.

Results for this approach are depicted in Figure 4.4. When compared to the Laplace approximation, it is apparent that this approach puts more emphasis on matching the overall probability mass of the distribution than on matching its mode.

4.3.3 Constrained Gaussian processes

Constrained GPs are an active area of research, and the constraints that have been considered include, amongst others, monotonicity constraints, boundary condition constraints, and differential equation constraints; Swiler et al. [2020] give a comprehensive overview of existing approaches.

Different strategies exist to include the above-mentioned constraints in the GP framework. In the following, we will focus on those strategies that
are related to our method of incorporating sum constraints (see Section 2.2), namely transformations of the outputs and the construction of constrained kernels.

Transforming the outputs

Transforming the outputs and training a GP on transformed data can be useful in different situations. That is, an invertible function \( h(\cdot) \) is employed such that

\[
\begin{align*}
    f' &= h(f), \\
y' &= h(y),
\end{align*}
\]

where \( f \) and \( y \) denote the function values and the measurements in the original space, respectively. We denote the corresponding transformed quantities as \( f' \) and \( y' \). When using this approach, \( f' \) is modeled as a GP, and \( f \) is recovered by backtransforming \( f = h^{-1}(f') \).

In Snelson et al. [2004], warped GPs are introduced, in order to deal with data \( y \) that may be contaminated by non-Gaussian noise; they learn the transformation \( h(\cdot) \) to obtain transformed data \( y' \) for which the noise is approximately Gaussian, and which is hence well-modeled by a GP.

In a similar vein, Jensen et al. [2013] consider bounded data which will result in a non-Gaussian likelihood. One of their proposed solutions is to transform bounded data \( y \in [a, b] \) into unbounded data \( y' \in [-\infty, \infty] \), which is then modeled as a GP. For example, this can be achieved via the inverse CDF of the standard normal distribution, i.e. the probit function, \( \Phi^{-1}(\cdot) \). Alternatively, they propose to employ non-Gaussian likelihoods for this data and to subsequently approximate the posterior as a Gaussian, via approximation methods such as the Laplace approximation (compare Section 4.3.2).

Constructing constrained kernels

Another effective way of constraining the GP predictions is to use kernels tailored to the problem at hand. In Jidling et al. [2017], this approach is employed to include linear constraints into GPs. Here, the output of a multitask GP \( f(x) \sim \mathcal{GP}(\mu(x), K(x, x')) \) is required to fulfill

\[
\mathcal{F}_X[f(x)] = 0,
\]

where \( \mathcal{F}_X \) is a linear operator defining the constraint. They assume that \( f(x) \) is related to a latent function \( g(x) \) via \( f(x) = \mathcal{G}_X[g(x)] \). Then, using the closedness of GPs under linear transformations [Papoulis and Pillai, 2001],
4.4 Thesis contribution: incorporating sum constraints

If $f$ will also be a GP if $g$ is modeled as a GP. The core of the method lies in choosing $G_X$ such that

$$F_X G_X = 0,$$  \hspace{1cm} (4.39)

and hence $F_X[G_X[g(x)]] = 0$. The constrained GP on $f$ is then obtained as

$$f(x) = G_X g(x) \sim GP \left( G_X \mu_g(x), G_X K_g(x, x') G_X^T \right). \hspace{1cm} (4.40)$$

In the paper, an algorithm to determine the operator $G_X$ is given.

With this method, well-known kernels such as the curl- or divergence-free kernel [Wahlström et al., 2013] can be recovered in a systematic manner. When considering the divergence-free kernel, for example, we have $F_X[f] = \frac{\partial f_1}{\partial x_1} + \frac{\partial f_2}{\partial x_2} = 0$ and it follows that $G_X = [-\frac{\partial}{\partial x_2}, \frac{\partial}{\partial x_1}]^T$. Then we find

$$K'_g(x, x') = G_X K_g(x, x') G'_X = \begin{bmatrix} \frac{\partial^2}{\partial x_2 \partial x'_2} & -\frac{\partial^2}{\partial x_1 \partial x'_1} \\ -\frac{\partial^2}{\partial x_1 \partial x'_2} & \frac{\partial^2}{\partial x_1 \partial x'_1} \end{bmatrix} K_g(x, x'). \hspace{1cm} (4.41)$$

Applications of this method include the modeling of electromagnetic fields Solin et al. [2018] or tomographic reconstruction of strain fields [Jidling et al., 2018, Hendriks et al., 2019b,a, 2020].

4.4 Thesis contribution: incorporating sum constraints

Conserved quantities or equilibrium conditions are prevalent in physics and they often take the form of a nonlinear sum of system variables. When modeling these system variables via a GP, it turns out that existing approaches do not allow for the inclusion of this kind of constraint. In our work (Paper I), we develop such an approach.

Definition

Formally, we define such constraints as

$$\mathcal{F}[f(x)] = \sum_i a_i(x) h_i(f_i(x)) = C(x), \hspace{1cm} (4.42)$$

and we call them sum constraints. Here, $h_i$ are nonlinear transformations that are applied to the outputs $f_i$ of the GP; $a_i(x)$ are the coefficients of each term in the sum and $C(x)$ gives the value which the sum has to add up to at point $x$. 
As a simple example, we can consider the harmonic oscillator, where the energy is conserved:

\[ E = \frac{mv^2}{2} + \frac{kz^2}{2}, \]  

(4.43)

where \( v \) is the velocity and \( z \) the displacement. Comparing with (4.42), we obtain \( h_1(v) = v^2, h_2(z) = z^2, a_1 = m/2, a_2 = k/2, \) and \( C = E \). In this case, all of the coefficients \( a_i \), as well as \( C \), are constant. Hence, this is an example of a constant sum constraint.

Transforming and backtransforming

The main ideas of our approach to incorporate sum constraints into the GP are the following: first, we transform the outputs \( f_i \) with the nonlinearities \( h_i \) to obtain \( f'_i = h_i(f_i) \). The advantage of doing this lies in the fact that the constraint becomes linear in terms of the transformed outputs,

\[ \mathcal{F}[f'(x)] = \sum_i a_i(x)f'_i(x) = C(x). \]  

(4.44)

Now we can train a GP for \( f' \) on transformed data \( y' \), where \( y'_i = h_i(y_i) \). Subsequently, we recover the original outputs via \( f_i = h^{-1}(f'_i) \).

This immediately raises the question as to how one should deal with non-invertible functions, such as e.g. the square function \( h(f) = f^2 \). The square function has two inverses, \( f = h^{-1}(f') = -\sqrt{f'} \) on the negative half-axis, and \( f = h^{-1}(f') = \sqrt{f'} \) on the positive half-axis (compare the left plot in Figure 4.5).

To resolve this issue, we introduce a scheme that enables us to pick the correct inverse: in addition to the GP for the transformed outputs \( f' \), we also learn auxiliary variables \( f_{\text{aux}} \) which disambiguate the backtransformation. For example, in case of the square function, we learn the auxiliary output \( f_{\text{aux}} = f \), which allows for the extraction of the sign of \( f \) via \( \text{sign}(f) = \text{sign}(f_{\text{aux}}) \).

Furthermore, we utilize the auxiliary outputs to extract virtual measurements from them in order to make the learned transformed outputs consistent with the points at which we switch from one inverse to another. This is illustrated in the right-hand side of Figure 4.5.

Another issue that we need to deal with is the following: when transforming the data \( y' = h(y) \) in order to obtain training data for the transformed GP, Gaussian noise will become non-Gaussian, which results in a non-Gaussian likelihood, making exact inference intractable in most cases. To resolve this issue, approximate inference methods can be applied, for example, those discussed in Section 4.3.2.
4.4. Thesis contribution: incorporating sum constraints

Figure 4.5: Illustration of the way auxiliary variables are used at the example of the harmonic oscillator. **Left:** Depending on the value of $z$, $z^2$ needs to be backtransformed via either the positive or the negative square root. **Bottom right:** In order to determine the sign, the auxiliary output $z_{\text{aux}}$ is employed, which has been trained on untransformed data, from which the sign can be extracted. The second auxiliary output, $v_{\text{aux}}$, which is required to enable the backtransformation of $v^2$, has been omitted from the figure for the sake of clarity. **Top right:** In addition, the auxiliary outputs are also used to create virtual measurements at points where we switch from one inverse to the other, in order to ensure consistency; this is illustrated by the black arrows. The GP on the transformed outputs $z^2$ and $v^2$ is forced to pass through these points exactly, in order to ensure consistency between the learned functions and the backtransformation.
Constraining the Gaussian process

When training the transformed GP, we introduce the constraint by conditioning the transformed outputs \( f' \) on the (now linear) sum constraint. In order to do this, we first collect the constraint coefficients and the constants into two matrices \( F \) and \( S \): for the harmonic oscillator, at any point in the input space, the constraint is defined by \( F = [m/2, k/2] \) and \( S = E \). The constraints at all \( N \) points under consideration are then collected in the matrix \( F_{\text{tot}} = I_N \otimes F \) and the column vector \( S_{\text{tot}} = \text{ones}(N) \otimes S \).

Then, the multivariate distribution can be constrained [Majumdar and Majumdar, 2019] via

\[
(f' | F_{\text{tot}} f' = S_{\text{tot}}) \sim \mathcal{N}(m', K'),
\]

(4.45)

where

\[
m' = Am + D^T K, \quad K' = A^T K A, \\
D = (F K F^T)^{-1} F K^T, \quad A = I_N - D^T F,
\]

(4.46)

and where \( m \) and \( K \) are the mean and covariance matrix of the still unconstrained transformed GP, respectively.

The general procedure of constraining the GP then looks as follows: we start with the unconstrained prior

\[
\mathcal{N} \left( \begin{bmatrix} m \\ m_* \end{bmatrix}, \begin{bmatrix} K + \sigma_n^2 I & K_* \\ K_*^T & K_{**} \end{bmatrix} \right) \quad \sim \quad \mathcal{N} \left( \begin{bmatrix} m \\ m_* \end{bmatrix}, \begin{bmatrix} K' & K_*' \\ K_*'^T & K_{**}' \end{bmatrix} \right).
\]

(4.47a)

and obtain the constrained prior

\[
\left[ \begin{array}{c} y' \\ f'_* \end{array} \right] \sim \mathcal{N} \left( \begin{bmatrix} m' \\ m_*' \end{bmatrix}, \begin{bmatrix} K' + \sigma_n^2 I & K_*' \\ K_*'^T & K_{**}' \end{bmatrix} \right) \quad \sim \quad \mathcal{N} \left( \begin{bmatrix} m' \\ m_*' \end{bmatrix}, \begin{bmatrix} K' & K_*' \\ K_*'^T & K_{**}' \end{bmatrix} \right).
\]

(4.47b)

In the first step, the noise has been omitted since the constraint only holds for the true outputs and not the noisy data; after including the constraints in the second step, the noise variance is added back.

Results

Let us return to the example of the harmonic oscillator. In Figure 4.6, the results for the unconstrained and the constrained GP are depicted. The constrained GP manages to match the correct functions more accurately. This is especially apparent in the left part of the \( v \)-curve, where some measurements are missing, as well as in the vicinity of minima and maxima. These results clearly demonstrate that taking into account additional knowledge from physics can improve the predictive capabilities of the GP.
Figure 4.6: The unconstrained vs the constrained GP at the example of the harmonic oscillator. **Left:** Results for the unconstrained GP. Deviations from the true functions (dotted lines) are clearly visible. **Right:** Results for the constrained GP. The constrained GP matches the true functions significantly better.
Chapter 5

Outlook

The main subject of this thesis is the integration of prior knowledge into machine learning models. Here, we present our conclusions and give a brief outlook on potential future research.

5.1 Conclusions

We have seen repeatedly that taking into account additional domain knowledge when training machine learning models can improve model performance substantially.

In the example of the sum-constrained GP in Paper I, it became apparent that incorporating known governing laws from physics into the GP can prevent it from making unphysical predictions and can improve the overall predictive capabilities of the GP.

In Paper II, we considered PINNs and demonstrated that learning the homogeneous measurement noise distribution jointly with the PINN prediction can improve predictions significantly, depending on how much the noise distribution differs from standard Gaussian noise.

When considering GANs in Paper III, we have seen that it is possible to explicitly match the distributions of various dataset statistics between true and generated data. This can help to resolve the problem of mode collapse, and it can allow for GANs to more accurately reproduce the true data distribution.

5.2 Future work

When it comes to future research, it appears that two main research tracks exist.
The first lies in doing projects similar to those presented in this thesis, where we aim to combine a specific type of constraint with a specific model. When considering the PINN-EBM, it would be interesting to explore the applicability of the approach to more complicated, higher-dimensional PDEs. It may also be worthwhile attempting to generalize the approach to special classes of non-homogeneous noise. For the pcGAN approach, it may be possible to extend the approach to also include correlations between the statistics under consideration. It would also be interesting to see if similar approaches are possible in different generative models, such as denoising diffusion probabilistic models. When considering the GP with sum constraint, it might be worthwhile to further investigate the applicability of the approach in pose estimation or to find nonlinear constraints similar to the sum constraint, for which it may be possible to include them in the GP with similar techniques. Naturally, there are many more potential combinations of ML models and physical constraints which may require the development of entirely new approaches.

The other research track would lie in using machine learning techniques for research projects in physics. It would be interesting to apply one of the models we already have developed to a concrete research task. At the moment, it seems that there is great potential for using generative models as substitutes for expensive numerical simulations, for example when considering the IceCube-Gen2 detector.

To conclude, we can state that the field of physics-informed machine learning promises to offer considerably more than what has been found so far. Existing approaches may be refined to result in reliable tools for research and technological innovation. Imaginative new ways of combining scientific methods with machine learning may result in unpredictable discoveries.
Bibliography


C. Jidling, J. Hendriks, N. Wahlström, A. Gregg, T.B. Schön, C. Wensrich, and A. Wills. Probabilistic modelling and reconstruction of strain.


