Integrating Prior Knowledge into Machine Learning Models with Applications in Physics

PHILIPP PILAR

UPPSALA UNIVERSITY
Department of Information Technology
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 Schönh, 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.
1.3.1 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 $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ which compares the model predictions $\hat{\mathbf{y}} = [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_{N_d}]^T$ to the true outputs $\mathbf{y} = [y_1, y_2, \ldots, y_{N_d}]^T$; 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:

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2.$$  

(1.2)
In the case of binary classification, the cross-entropy loss is a common choice:

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

(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

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

(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

$$q_{l+1} = h(W_lq_l + b_l),$$  \hspace{1cm} (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

$$L_c(y, \hat{y}) = L(y, \hat{y}) + \lambda L_{\text{phys}}(\hat{y}).$$

(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
takes multiple inputs

\[ \hat{y} = f(x, p), \]

where \( p \) denotes additional information that may be available about the input \( x \). This method is popular for constraining generative adversarial networks (Chapter 3), as it allows to pass constraint residuals as additional information to the discriminator to help it distinguish between true and generated samples [Stinis et al., 2019].

Model architectures

It can be possible to encode prior knowledge directly into the model architecture. Constraints of this type are also referred to as inductive biases [Karniadakis et al., 2021]. For example, when dealing with differential equations, this can be done for boundary conditions. When considering a simple 1D example, this can be achieved by defining

\[ \hat{y} = g(x_0) + f(x)(x - x_0), \]  

(1.7)

where \( g(x_0) \) gives the known value at the boundary \( x_0 \).

Periodic boundary conditions can be taken into account by learning instead Fourier components of the solution [Lu et al., 2021]. Further alternatives are discussed in Hao et al. [2022].

Depending on how broadly this idea is interpreted, even convolutional neural networks can be considered an example, as they incorporate translational invariance. The same holds for graph neural networks, as they encode permutation invariance.

When working with Gaussian processes (Chapter 4), certain constraints can be incorporated into the kernel. We utilize this in Paper I, when incorporating sum constraints into Gaussian processes.

1.4 Contribution

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.
1.6. Papers included in this thesis

**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

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

where \( z \) denotes the input, e.g. \( z = (t, x) \), \( u \) the solution of the PDE, and \( F \) the differential operator defining the PDE. Optionally, \( \lambda \) denotes parameters of \( F \) 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 \( F\hat{u} \) 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

\[ \hat{u}(x, t) = e^{\int_{t_0}^{t} \cdots} \]

\[ \mathcal{F}(\lambda) \]

\[ \mathcal{L}_{\text{pde}} + \mathcal{L}_{\text{IC, BC}} + \mathcal{L}_{\text{data}} = \mathcal{L}_{\text{tot}} \]

Figure 2.1: Schematic of the PINN framework. The inputs \( x \) and \( t \) are fed through the neural network to obtain predictions \( \hat{u}(x, t) \). Subsequently, the derivatives of these predictions with respect to the inputs are calculated via automatic differentiation, in order to determine the PDE residuals \( \mathcal{F}(\lambda) \hat{u} \), which then enter as a term in the loss function. In addition, loss terms measuring the differences between PINN predictions and ICs/BCs or measurements, respectively, are added as the second and third loss term. The parameters \( \lambda \) of the differential operator \( \mathcal{F} \) may be unknown and can be learned together with the PDE solution, given enough data.

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. Using neural networks to solve differential equations

2.2.1 The forward problem

In the forward problem, the differential operator $\mathcal{F}$ 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:

$$
\mathcal{L}_{BC} = \frac{1}{N_{BC}} \sum_{i=1}^{N_{BC}} (\hat{u}(z_{BC}^i) - f(z_{BC}^i))^2,
$$

(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

$$
\mathcal{L}_{PDE} = \frac{1}{N_c} \sum_{i=1}^{N_c} (\mathcal{F}\hat{u}(z_c^i))^2.
$$

(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

$$
\mathcal{L}_{tot} = \mathcal{L}_{BC} + \omega_c \mathcal{L}_{PDE},
$$

(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

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

(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 \( F(\lambda) \) 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,

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

(2.6)

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

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

(2.7)

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. \( \lambda = \{ a \} \)):

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

(2.8)

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
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 $\hat{a} = 1.02$.
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
2.4. Thesis contribution: the inverse problem with unknown noise 23

Figure 2.6: The PINN-EBM approach. The distribution of unknown, homogeneous noise is learned jointly with the PINN solution. This allows for the PINN to obtain correct results also in the case of non-zero mean noise. More information can be found in Paper II.

\[- \log p(u^i_d) = \frac{(u^i_d - u^i_i)^2}{2\sigma^2} + \text{const.}; \]

The constant does not affect training and the factor $2\sigma^2$ can be absorbed into the learning rate, leaving us with $(u^i_d - u^i_i)^2$.

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

$$\dot{x}(t) = \lambda x(t),$$  \hspace{1cm} (2.9)

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.
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.
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.
3.1. Training generative adversarial networks

Formally, this situation can be written as a minimax game with objective function

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

(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 [true, 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:

$$L^0_D = -\frac{1}{N} \sum_{i=1}^{N} \left[\log D(x_i) + \log(1 - D(G(z_i)))\right],$$

(3.2a)

$$L^0_G = \frac{1}{N} \sum_{i=1}^{N} \log(1 - D(G(z_i))),$$

(3.2b)

where the sum is taken over minibatch samples. Note, that two different, unrelated minibatches enter into $L^0_D$: 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:

$$D^*_G(x) = \frac{p_{\text{true}}(x)}{p_{\text{true}}(x) + p_{\text{gen}}(x)}.$$  

(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:

$$C(G) = \max_D V(G, D^*_G)$$

$$= -\log(4) + \text{KL} \left( p_{\text{true}} \left\| \frac{p_{\text{true}} + p_{\text{gen}}}{2} \right\| \right) + \text{KL} \left( p_{\text{gen}} \left\| \frac{p_{\text{true}} + p_{\text{gen}}}{2} \right\| \right)$$

$$= -\log(4) + 2 \text{JS} (p_{\text{true}} \| p_{\text{gen}}).$$

(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 $z \sim p_z$;
    $\mathcal{L}_D = -\text{mean} \left[ \log D(x_{\text{true}}) + \log(1 - D(x_{\text{gen}})) \right]$;
    $\theta_D \leftarrow \theta_D - \gamma \nabla_{\theta_D} \mathcal{L}_D$;
  end

  Sample minibatch $x_{\text{gen}} = G(z)$ with $z \sim p_z$;
  $\mathcal{L}_G = \text{mean} \left( \log(1 - D(x_{\text{gen}})) \right)$;
  $\theta_G \leftarrow \theta_G - \gamma \nabla_{\theta_G} \mathcal{L}_G$;
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

(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

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

(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:

$$W(p_{true}, p_{gen}) = \sup_{||f||_{L^1} \leq 1} \mathbb{E}_{x \sim p_{true}}[f(x)] - \mathbb{E}_{x \sim p_{gen}}[f(x)].$$

(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, \( KL(p_{true}||p_{gen}) = \int_{-\infty}^{\infty} p_{true}(x) \log \left( \frac{p_{true}(x)}{p_{gen}(x)} \right) dx \), 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

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

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

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:

\[
\mathcal{L}_D^{GP} = \mathcal{L}_D + \lambda \frac{1}{N} \sum_{i=1}^{N} (||\nabla_{\hat{x}} D(\hat{x}_i)||_2 - 1)^2, \tag{3.9}
\]
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**

$\epsilon \sim \text{Uniform}[0, 1]$;

$\hat{x} = x_{\text{true}} + (1 - \epsilon)x_{\text{gen}}$;

$L_{\text{GP}} = \text{mean} \left( ||\nabla_{\hat{x}} D(\hat{x})||_2 - 1 \right)^2$;

$L_D = L_D + \lambda L_{\text{GP}}$;

**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 $
abla_{\hat{x}} D(\hat{x}) = \epsilon x_{\text{true}} + (1 - \epsilon)x_{\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

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

(3.10)

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:

\[
\mathcal{L}_G^c = \mathcal{L}_G + \lambda d(\Sigma(p_{true}), \Sigma(p_{gen})),
\]

(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:

\[
\mathcal{L}_G^c = \mathcal{L}_G + \lambda \sum_{s=1}^{N_s} \text{KL}(p_{true}(z_s) || p_{gen}(z_s)),
\]

(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 \( z_a \) 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:

\[
\hat{p}_{\text{gen}}(z_s) = \frac{1}{n_{\text{batch}}} \sum_{i=1}^{n_{\text{batch}}} \mathcal{N}(z_s; z_{si}, \sigma_s).
\] (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
3.3. Thesis contribution: probabilistically constrained GANs

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

$$y = \theta_0 + \theta_1 x + \epsilon,$$

(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

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \epsilon.$$  

(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

$$y = \theta^T \phi(x) + \epsilon.$$  

(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

$$\hat{\theta} = \operatorname{arg\,min}_\theta \frac{1}{N} \sum_{i=1}^N \left( \theta^T \phi(x_i) - y_i \right)^2 = \operatorname{arg\,min}_\theta \frac{1}{N} \| \Phi \theta - y \|_2^2,$$

(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:

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

(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

$$\hat{\theta} = \operatorname{arg\,min}_\theta \frac{1}{N} \| \Phi \theta - y \|_2^2 + \lambda \| \theta \|_2^2.$$  

(4.6)
4.1. From linear regression to Gaussian processes

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

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

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

where the push-through matrix identity, \( \Phi (\Phi^T \Phi + I)^{-1} = (\Phi \Phi^T + I)^{-1} \Phi \), 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

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

(4.11)

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

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

(4.12)

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.
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,

$$m(x) = \mathbb{E}[f(x)]$$

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

and the GP can be written as

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

Sampling functions from the GP, evaluated at $n_s$ test inputs $X_s = [x_{s1}, x_{s2}, \ldots, x_{sn_s}]^T$, can then be done by sampling from the multivariate normal distribution

$$f_s = \mathcal{N}(m_s, K_{ss}),$$

where the mean $m_s = m(X_s)$ and the covariance matrix $K_{ss} = k(X_s, X_s)$ are obtained by evaluating the mean and covariance function, respectively, on $X_s$; more precisely, $m_{si} = m(x_{si})$ and $K_{siij} = k(x_{si}, x_{sj}).$

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 $\epsilon \sim \mathcal{N}(0, \sigma_n^2 I)$. We start by considering the joint distribution of the function values $f$ and the function values $f_*$ corresponding to test inputs $X_*$:

$$
\begin{bmatrix}
f \\
f_*
\end{bmatrix} \sim \mathcal{N}
\begin{pmatrix}
\begin{bmatrix} m \\ m_* \end{bmatrix} \\
\begin{bmatrix} K & K_* \\ K_*^T & K_{**} \end{bmatrix}
\end{pmatrix},
$$

(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:

$$
\begin{bmatrix} y \\ f_* \end{bmatrix} \sim \mathcal{N}
\begin{pmatrix}
\begin{bmatrix} m \\ m_* \end{bmatrix} \\
\begin{bmatrix} K + \sigma_n^2 I & K_* \\ K_*^T & K_{**} \end{bmatrix}
\end{pmatrix}.
$$

(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):

$$
f_* | X, y, X_* \sim \mathcal{N}(\bar{f}_*, \text{cov}(f_*)), \text{ where}
$$

(4.18a)

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

(4.18b)

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

(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

\[
m_f(X) = [m_d(x_1) m_t(x_1)^T, \ldots, m_d(x_N) m_t(x_N)^T]^T,
\]

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

where \( m_d(\cdot) \) is the data mean, \( m_t(\cdot) \) 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:

\[
m_f(X) = m_d(X) \otimes m_t,
\]

\[
K_{f,f'}(X, X') = k_d(X, X') \otimes k_t.
\]

The multitask GP is then written as \( f \sim \mathcal{N}(m_f(X), K_{f,f'}(X, X')) \), 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
\[
p(f|y) = \frac{p(y|f)p(f)}{p(y)} \approx q(f).
\] (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 \( f_{\text{max}} \):
\[
0 \overset{!}{=} \nabla_f \log (p(y|f)p(f)) \big|_{f=f_{\text{max}}}
\] (4.26)
\[
= (\nabla_f \log p(y|f) - K^{-1}f) \big|_{f=f_{\text{max}}},
\] (4.27)
\[
\rightarrow f_{\text{max}} = K(\nabla_f \log p(y|f))|_{f=f_{\text{max}}}.
\] (4.28)

For the Hessian, we obtain
\[
\Sigma_{\text{max}}^{-1} = -\nabla_f \nabla_f \log p(f|y)|_{f=f_{\text{max}}}
\] (4.29)
\[
= -\nabla_f \nabla_f \log p(y|f)|_{f=f_{\text{max}}} + K^{-1}.
\] (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 \( p(f|y) \),
\[
\hat{\theta}_q = \arg \min_{\theta_q} D_{\text{KL}}(q(f)||p(f|y)),
\] (4.31)
4.3. Extensions of the Gaussian process framework

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

$$D_{KL}(q(f)||p(f|y)) = \int q(f) \log \left( \frac{q(f)}{p(f|y)} \right) df \quad (4.32)$$

$$= \int q(f) \log \left( \frac{q(f)p(y)}{p(f, y)} \right) df \quad (4.33)$$

$$= \mathbb{E}_{q(f)}[\log q(f)] - \mathbb{E}_{q(f)}[\log p(f, y)] + \log p(y) \quad (4.34)$$

$$= -\text{ELBO}(q) + \log p(y). \quad (4.35)$$

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

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

$$\mathcal{F}_X G_X = 0,$$

(4.39)

and hence $\mathcal{F}_X [G_X [g(x)]] = 0$. The constrained GP on $f$ is then obtained as

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

(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 $\mathcal{F}_X [f] = \frac{\partial f_1}{\partial x_1} + \frac{\partial f_2}{\partial x_2} = 0$ and it follows that $G_X = [-\frac{\partial}{\partial x_2}, \frac{\partial}{\partial x_1}]^T$. Then we find

$$K'_g(x, x') = G_X K_g(x, x') G_X^{T} = \begin{bmatrix} \frac{\partial^2}{\partial x_1 x_2} & \frac{\partial^2}{\partial x_1 x_2} \\ -\frac{\partial^2}{\partial x_1 x_2} & \frac{\partial^2}{\partial x_1 x_2} \end{bmatrix} K_g(x, x').$$

(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

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

(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:

\[ E = \frac{mv^2}{2} + \frac{kz^2}{2}, \quad (4.43) \]

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,

\[ \mathcal{F}[f'(x)] = \sum_i a_i(x) f'_i(x) = C(x). \quad (4.44) \]

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.
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 \( z_{aux} \) is employed, which has been trained on untransformed data, from which the sign can be extracted. The second auxiliary output, \( v_{aux} \), 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

$$ (f' | F_{\text{tot}} f' = S_{\text{tot}}) \sim \mathcal{N}(m', K'), \quad (4.45) $$

where

$$ m' = Am + D^T K, \quad K' = A^T K A, \quad D = (FKF^T)^{-1}FKF^T, \quad A = I_N - D^T F, \quad (4.46) $$

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

$$ \mathcal{N}
\begin{bmatrix}
m \\
m_\ast
\end{bmatrix}
, \begin{bmatrix}
K + \sigma_n^2 I & K_\ast \\
K_\ast^T & K_{\ast\ast}
\end{bmatrix}
\xrightarrow{\text{omit noise}}
\mathcal{N}
\begin{bmatrix}
m \\
m_\ast
\end{bmatrix}
, \begin{bmatrix}
K & K_\ast \\
K_\ast^T & K_{\ast\ast}
\end{bmatrix}
\quad (4.47a) $$

and obtain the constrained prior

$$ \begin{bmatrix}
y' \\
f'_\ast
\end{bmatrix}
\sim \mathcal{N}
\begin{bmatrix}
m' \\
m'_\ast
\end{bmatrix}
, \begin{bmatrix}
K' + \sigma_n^2 I & K'_\ast \\
K'_\ast^T & K'_{\ast\ast}
\end{bmatrix}
\xrightarrow{\text{add noise}}
\mathcal{N}
\begin{bmatrix}
m' \\
m'_\ast
\end{bmatrix}
, \begin{bmatrix}
K' & K'_{\ast} \\
K'_{\ast}^T & K'_{\ast\ast}
\end{bmatrix}
\quad (4.47b) $$

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.


Title
Incorporating sum constraints into multitask Gaussian processes

Authors
Philipp Pilar, Carl Jidling, Thomas B. Schön, and Niklas Wahlström

Edited version of
Incorporating sum constraints into multitask Gaussian processes

Abstract

Machine learning models can be improved by adapting them to respect existing background knowledge. In this paper, we consider multitask Gaussian processes, with background knowledge in the form of constraints that require a specific sum of the outputs to be constant. This is achieved by conditioning the prior distribution on the constraint fulfillment. The approach allows for both linear and nonlinear constraints. We demonstrate that the constraints are fulfilled with high precision and that the construction can improve the overall prediction accuracy as compared to the standard Gaussian process.

1 Introduction

Many real-world problems come with background knowledge known a priori, for instance, that the outputs must be positive at all times or fulfill a certain differential equation. The constraints are often known to near perfect precision. Any model would certainly benefit from having such knowledge hardcoded in advance instead of having to rediscover it, as the additional information would allow for the exclusion of the majority of possible outputs.

In this work, we consider the Gaussian process (GP) [Rasmussen and Williams, 2006], which is a popular and powerful machine learning model. Some assumptions about the underlying function, e.g. regarding its smoothness, can be encoded in a relatively straightforward way into the kernel of the GP. However, it is usually trickier to include more specific prior knowledge and constrained GPs (or, for that matter, constrained machine learning methods) constitute a relevant and active area of research [Willard et al., 2021, Swiler et al., 2020].
In this work, we focus on constraints that take the form of a sum over the outputs of a multitask GP. Constraints of this form arise, for example, when considering conserved quantities in physics such as energy and momentum, where the sum over the energies or momenta of all subcomponents of a closed system must remain constant. As a toy example, we consider the harmonic oscillator, which is ubiquitous in physics; the expression for the energy takes the form

\[ E = E_{\text{pot}}(t) + E_{\text{kin}}(t) = kz(t)^2/2 + mv(t)^2/2, \]  

where \( E_{\text{pot}} \) and \( E_{\text{kin}} \) denote potential and kinetic energy, respectively. We assume that the displacement from the rest position \( z \) and the velocity \( v \) are the outputs of a multitask GP, whereas the time \( t \) serves as input. While the input in this example is one-dimensional, the results we derive in this paper also apply to higher-dimensional inputs.

We have developed a method that allows nonlinear constraints like (1) to be incorporated into the GP. First, we show how nonlinear constraints can be reduced to linear ones via a suitable transformation of the outputs of the GP. Then we proceed to condition the joint prior of the GP on the constraints, which in turn results in a constrained predictive distribution. In the next section, we start by providing a formal definition of the problem.

2 Problem formulation

2.1 Background on the GP

A GP is formally defined as “a collection of random variables, any finite number of which have a joint Gaussian distribution” [Rasmussen and Williams, 2006]. Formally, we write \( f(x) \sim \text{GP}(m(x), k(x, x')) \), where \( m(x) = \mathbb{E}[f(x)] \) and \( k(x, x') = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))] \) are the mean and the covariance function of the GP, respectively. The dataset available for training the GP consists of inputs \( X = \{x_k\}_{k=1}^N \) and noisy outputs \( y_k = f(x_k) + \epsilon_k \), where we assume Gaussian noise \( \epsilon_k \sim \mathcal{N}(0, \sigma_n^2) \). We use \( y \) to denote a vector storing all \( N \) outputs.

In the following, we consider the multitask setting [Bonilla et al., 2008, Skolidis and Sanguinetti, 2011], where a vector \( f(x) \) of \( N_f \) outputs is learned. The overall GP framework remains unchanged but the output vector \( f \) (and observation vector \( y \)) has to be interpreted as an extended vector consisting of the concatenated multitask outputs \( f_k = f(x_k) \) — that is \( f = [f_1^T, f_2^T, \ldots, f_{N_f}^T]^T \) for which it holds that \( f \sim \mathcal{N}(m_f(X), K_{f,f'}(X, X')) \).

When constructing the mean and covariance function, the different tasks need to be taken into account [Alvarez et al., 2012]. We write the mean as

\[ m_f(X) = [m_d(x_1)m_t(x_1)^T, \ldots, m_d(x_N)m_t(x_N)^T]^T, \]
where \( m_d(\cdot) \) is the data mean and \( m_t(\cdot) \) is the task mean. The task mean returns a column vector of length \( N_f \). The covariance matrix becomes

\[
K_{f,f'}(X, X) = \begin{bmatrix}
  k_{d11}k_t(x_1, x_1) & k_{d12}k_t(x_1, x_2) & \ldots \\
  k_{d21}k_t(x_2, x_1) & k_{d22}k_t(x_2, x_2) & \ldots \\
  \vdots & \vdots & \ddots
\end{bmatrix},
\]

where \( k_{dij} = k_d(x_i, x_j) \), and where \( k_d(\cdot, \cdot) \) and \( k_t(\cdot, \cdot) \) denote the data and task kernels, respectively. Note that the task kernel returns a matrix of size \((N_f, N_f)\).

The task mean and kernel are often assumed to be position independent (although this assumption is not necessary for our method to work); then \( m_f \) and \( K_{f,f'} \) can be written as Kronecker products

\[
m_f(X) = m_d(X) \otimes m_t, \quad (4a)
\]
\[
K_{f,f'}(X, X') = k_d(X, X') \otimes \Sigma_t. \quad (4b)
\]

Given the expressions for the mean and the kernel, the predictive distribution is formed through the standard procedure; see Section B.1 in the supplementary material for details. See also Section B.2, for details on how to deal with the case of incomplete measurements, i.e. when there are data points \( y_k \) for which only some of the output tasks have been measured.

2.2 Sum constraint

The main concern of this work is to show how constraints on the sum of some (nonlinear) transformations \( h_i(\cdot) \) of the outputs \( f_i \) can be incorporated into the GP. Formally, we define this class of constraints as

\[
\mathcal{F}[f(x)] = \sum_i a_i(x)h_i(f_i(x)) = C(x), \quad (5)
\]

where the functions \( a_i(x) \) serve as prefactors to the various terms in the sum, \( i \) indexes the outputs of the GP, and \( C(x) \) specifies what value the sum over the outputs should equal at position \( x \) in the input space. In the following we refer to constraints of this form as sum constraint.

In the general case (5), we consider input-dependent constraints \( C(x) \) and \( a_i(x) \). This requires knowledge of the functions \( C(x) \) and \( a_i(x) \), which could be practically infeasible. Hence, an important special case of (5) is the constant sum constraint

\[
\mathcal{F}[f(x)] = \sum_i a_i h_i(f_i(x)) = C, \quad (6)
\]

with constant prefactors \( a_i \) and constant sum \( C \).
One example of a constant sum constraint is the previously mentioned energy conservation for the harmonic oscillator (1). There we have \( a_1 = k/2 \), \( a_2 = m/2 \), \( h_1(z) = z^2 \), \( h_2(v) = v^2 \) and \( C = E \). Other situations where sum constraints arise include learning of probabilities that must sum to one, and the case of mechanical equilibrium where the sum of acting forces must be zero at each point.

3 Method

Let us now develop the methodology required to incorporate sum constraints as defined in Section 2.2 into the GP. In Section 3.1, we consider the case where all the outputs of the GP enter the sum constraint via a monotonic (invertible) nonlinearity and show how to reduce it to a linear sum constraint. In Section 3.1 we extend the procedure to sum constraints with non-monotonic nonlinearities. Finally, we show in Section 3.2 how to include linear sum constraints into the GP and hence, via the aforementioned reductions, also nonlinear sum constraints.

3.1 Reduction to linear constraint

Monotonically increasing nonlinearity

Consider the sum constraint (5) — while the constraint is nonlinear in terms of the outputs, it is linear in terms of the transformed outputs \( h_i(f_i) \); defining \( f_i' = h_i(f_i) \) and substituting it into (5) yields

\[
\mathcal{F}[f'(x)] = \sum a_i(x) f_i'(x) = C(x),
\]

which is linear in the transformed outputs \( f_i' \). Hence, we can train a GP to predict the transformed outputs obeying the linear constraint (7) and backtransform to the original outputs via \( f_i = h_i^{-1}(f_i') \). Note that this GP needs to be trained on transformed data \( y'_i = h_i(y_i) \). This approach requires that the nonlinear functions \( h_i(\cdot) \) are invertible, otherwise it is not possible to unambiguously recover the \( f_i \). See also Snelson et al. [2004].

However, it is not necessary for \( h_i \) to be invertible on its entire domain. Consider the case where it is known that the output \( f_i \) is restricted to an invertible subregion of the domain of \( h_i \); then we solve the problem by choosing the backtransformation \( h_i^{-1} \) corresponding to this subregion. For example, in case of the square function, we can consider the case where \( f_i \) is known to be always positive (or always negative). Then we can just restrict the domain of the nonlinearity \( h_i \) to the positive (negative) half-axis where the function is in fact invertible.
Algorithm 1: The Constrained GP: High-level Procedure

**Step 1**: train an unconstrained GP on the data $\mathbf{y}$ to obtain the auxiliary outputs $f_{\text{aux}}$
- (optional) use the posterior mean of $f_{\text{aux}}$ to create virtual measurements

**Step 2**: train the constrained GP on the transformed data $\mathbf{y}'$ (for details, see Algorithm 2) to obtain $f'$
- (optional) (re)learn the auxiliary outputs together with the constrained outputs

**Step 3**: backtransform the transformed outputs $f'$ using the posterior mean of $f_{\text{aux}}$ from Step 1

When employing the transformation (7), it is important to keep in mind that the GP prior now has to be chosen in a way suitable for the transformed outputs $f'$ instead of $f$; depending on the transformations $h(\cdot)$ involved, this could prove to be more challenging. We recover credible intervals for $f$ in the same way as we recover $f'$, by backtransforming them; for more details, see Section B.7 in the Supplementary material.

Furthermore, the noise corresponding to the transformed data $\mathbf{y}'$ will in general not be normally distributed anymore, which means that GP regression loses its analytical tractability due to the resulting non-Gaussian likelihood. Methods to deal with non-Gaussian likelihoods include the Laplace approximation [Williams and Barber, 1998, Vanhatalo et al., 2009], variational inference [Blei et al., 2017, Tran et al., 2016], and expectation propagation [Minka, 2001]. Due to its simplicity, in this work we use the Laplace approximation to deal with this issue, where applicable. It enables us to approximate non-Gaussian distributions with a Gaussian; see Appendix B.3 for details.

Non-monotonically increasing nonlinearity

In the previous section we showed how to reduce nonlinear sum constraints to linear ones, as long as the nonlinearities are monotonic. However, this is a rather limiting assumption as it would exclude e.g. the square function $h(f) = f^2$ from the admissible transformations. Here we describe a way of circumventing this problem.

The idea underlying our solution is to introduce one (or multiple) auxiliary variables that allow for a unique backtransformation. Typically, the auxiliary variables will keep track of where in the domain of $h(\cdot)$ it is that $f'$ lies, such that the correct local inverse can be chosen when backtransforming.
In case of the square function, we can add the auxiliary output $f_{\text{aux}} = f$ and retrieve the initial output $f$ via $f = \text{sign}(f_{\text{aux}})h^{-1}(f') = \text{sign}(f_{\text{aux}})\sqrt{h(f)}$. While the initial output $f$ is a practical choice here, this is in general not necessary and $f_{\text{aux}}$ can be chosen arbitrarily.

There is no guarantee that learned values $f'$ will always fall within the domain of the backtransformation. If it happens that a predicted value lies outside, a pragmatic solution is to approximate $f'$ with the closest valid value; for example zero in case of negative valued predictions for square values.

Sometimes more information can be extracted from $f_{\text{aux}}$ and used to ameliorate the transformed data $y'$, for instance when the backtransformation switches from one local inverse to another; then we can add virtual measurements for $f'$ at those points and force the constrained GP towards values consistent with $f_{\text{aux}}$, which can significantly reduce artefacts in the backtransformed outputs $f$. Note that this can come at the cost of overconfident credible intervals in the vicinity of the virtual measurements.

In Algorithm 1, we summarize this procedure. In most cases, it is advantageous to learn the auxiliary outputs in a separate GP in Step 1, independently of the constrained outputs; when virtual measurements are to be created, this is required. Optionally, auxiliary outputs can be (re)learned in Step 2; for some examples, this can stabilize the hyperparameter learning of the constrained GP. However, when virtual measurements are involved, the prediction $f_{\text{aux}}$ from Step 1 should also be used for the backtransformation.

We illustrate the approach by returning to the harmonic oscillator (1), with the transformed outputs $f'_1 = z^2$ and $f'_2 = v^2$ (see also the last paragraph in Section 2.2). We choose the auxiliary outputs as $f^1_{\text{aux}} = z$ and $f^2_{\text{aux}} = v$, which we use to extract the sign when backtransforming $f'_1$ and $f'_2$; furthermore, we use the auxiliary outputs to create virtual measurements for $f'_1$ and $f'_2$ at the zero crossings of the posterior mean of $f^1_{\text{aux}}$ and $f^2_{\text{aux}}$. In order to fit the transformed outputs of the GP, the observations $y_k = [z_k, v_k]^T$ are transformed analogously to obtain $y'_k = [z_k^2, v_k^2, z_k, v_k]^T$; $z_k$ and $v_k$ are part of $y'_k$ since we chose to relearn them together with the constrained outputs to improve the performance. The virtual measurements are also included in the transformed data $y'$. In terms of the transformed outputs the constraint can be written compactly as $Ff' = C$, where $F = [a_1, a_2, 0, 0]$. For more details on the harmonic oscillator dataset, see Section C.1 in the supplementary material.

### 3.2 Solving with linear constraints

Having shown how to reduce nonlinear sum constraints to linear ones, we proceed to describe how to incorporate linear sum constraints into the GP.
Algorithm 2: Constraining the GP (Section B.1 refers to the Supplementary material)

Input: mean \( m_f(\cdot) \); kernel \( K_{f,f'}(\cdot, \cdot) \); constraints \( (F, S) \); (transformed) data \( X, y' \); points of prediction \( X_* \)

Output: constrained predictive distribution \( f'_* | X, y', X_* \)

Note: During hyperparameter optimization \( X_* = {} \) and hence \( f_*' = {} \)

Step 1: Construct the joint prior distribution for
\[
\begin{bmatrix} f' \\ f'_* \end{bmatrix} \sim N(\mu_0, \Sigma_0) \text{ according to (B.1)}
\]
- omit noise term \( \sigma_n^2 I \)

Step 2: Construct \( F_{\text{tot}}, S_{\text{tot}} \) according to (9b)

Step 3: Use \( F_{\text{tot}}, S_{\text{tot}} \) to calculate constrained \( \mu', \Sigma' \) according to (8b)

Step 4: Remove entries in \( \mu', \Sigma' \) corresponding to incomplete measurements as detailed in Section B.2

if Hyperparameter optimization then
  Step 5: Calculate the log marginal likelihood according to (B.7c)
  Step 6: Perform optimization step
else if Prediction then
  Step 5: Calculate the predictive distribution \( f'_* | X, y', X_* \)
  according to (B.7a)
end if

The idea is to make use of the fact that sampling from a GP is equivalent to sampling from a multivariate Gaussian distribution, where the mean and covariance are obtained by evaluating the mean and the kernel of the GP at the points of interest.

Let the random vector \( f' \sim N(\mu, \Sigma) \); we are interested in the conditional distribution \( f'| \sum_i a_i f_i' = C \). More generally, to include multiple sum constraints, we want to find the distribution \( f'| Ff' = S \), where the rows of the matrix \( F \) contain the coefficients for each of the \( N_F \) sum constraints to be included, and the elements of the vector \( S \) contain the corresponding sums; compare equation (9a) below.

The required conditional distribution can be calculated analytically [Majumdar and Majumdar, 2019] as

\[
(f'| Ff' = S) \sim N(\mu', \Sigma'), \tag{8a}
\]
where
\[
\mu' = A\mu + D^T S, \quad \Sigma' = A^T \Sigma A,
\]
\[
D = (F\Sigma F^T)^{-1} F \Sigma^T, \quad A = I_n - D^T F.
\] (8b)

Of course, we need to enforce the constraint at all \(N_{tot}\) data points — to that end, we construct the blockdiagonal matrix \(F_{tot}\) and the vector \(S_{tot}\) according to
\[
F(x) = \begin{bmatrix}
a_1(x) & a_2(x) & \cdots \\
b_1(x) & b_2(x) & \cdots \\
\vdots & \vdots & \ddots
\end{bmatrix}, \quad S(x) = \begin{bmatrix}
C_a(x) \\
C_b(x) \\
\vdots
\end{bmatrix},
\] (9a)
\[
F_{tot} = \text{diag}(F(x_1), F(x_2), \ldots), \quad S_{tot} = [S(x_1)^T, \ldots]^T.
\] (9b)

We use \(N_{tot}\) in two different contexts: during the hyperparameter optimization, \(N_{tot}\) denotes the number of data points; whereas during prediction, \(N_{tot}\) denotes the number of both data and predictive points.

Algorithm 2 summarizes the practical procedure of constructing the covariance and the mean, both during hyperparameter optimization and when forming the constrained predictive distribution of the GP. In case of a position dependent constraint, it is important to note that the values of the functions \(C(x)\) and \(a_i(x)\) must be known at all points for which the constraint should be enforced; in our case, this means all \(N_{tot}\) points. Note that in Step 1 of the algorithm we first omit the noise term, since the constraints only hold exactly for noiseless data; the noise then enters in Step 4, after the constraints have been taken into account.

Mathematically, the constraint is enforced by conditioning the Gaussian distribution on it. While the method is not strictly global in the sense of providing a constrained kernel for the GP, it is global for practical purposes as the constraint is enforced at all points of prediction of the GP.

Due to the matrix inversion in (8b), the computational complexity of the algorithm is cubic with leading order term \(\sim O(N_{tot}^3)\), during both hyperparameter optimization and prediction.

**Special case of constant constraints**

In the special case of constant constraints and constant inter-task dependencies of the GP mean and kernel, the constraints can be incorporated more efficiently. Here, the kernel of the GP factorizes into data and task kernel as in (4) and the procedure above simplifies: it now suffices to enforce the constraints \((F, S)\) on the task mean and covariance matrix and to subsequently perform the Kronecker product with the data mean and covariance matrix to obtain the constrained distribution.
Formally, this can be written as follows: let $\mu_t$ and $\Sigma_t$ be the task mean and covariance matrix, respectively; then the constrained quantities $\mu'_t$ and $\Sigma'_t$ are calculated via (8b), using $F$ and $S$ (since the task mean and covariance matrix are constrained directly, it is not necessary to construct $F_{\text{tot}}$ and $S_{\text{tot}}$). Finally, the full constrained mean and covariance matrix are constructed via $\mu' = m \otimes \mu'_t$ and $\Sigma' = K \otimes \Sigma'_t$, where $m$ and $K$ are the data mean and covariance matrix, respectively. Due to the constant constraint, the data mean is also required to be constant. Without loss of generality, we choose it as $m = 1_{N_{\text{tot}}}$ (compare B.6). This procedure is summarized in Algorithm 3 in the Supplementary material. Furthermore, we provide proof that this approach is indeed equivalent to the more general approach from Section 3.2 in Appendix B.6.

Now, the complexity of the matrix inversion involved in (8b) is reduced to $\sim O(N_F^3)$; since $\mu_t$ and $\Sigma_t$ are constrained directly it no longer depends on $N_{\text{tot}}$ (compare also (4)). This constitutes a significant improvement over the general algorithm as usually $N_F \leq N_f \ll N_{\text{tot}}$, where $N_F$ is the number of constraints and $N_f$ the number of tasks. Whenever applicable, it is preferable to use this way of incorporating the constraint, since it is more efficient and numerically more stable than the general procedure given in Algorithm 2.

4 Experimental results

In this section, we demonstrate our method at the hand of two simulation experiments and one real data experiment. They have in common that the constraints involved are constant (see Section 3.2); for examples of the non-constant case, see Sections A.2 and A.3 in the Supplementary material.

4.1 Toy problem revisited

We gave a formulation of the auxiliary variables approach for the harmonic oscillator in Section 3.1 and detailed information on the dataset can be found in Section C.1 in the Supplementary material. Figure 1 illustrates this approach. The constrained GP achieves higher overall accuracy around extremal points, where the prediction is more robust with regard to the influence of random noise. In addition, the constrained GP manages to mitigate the negative effect of incomplete measurements, i.e. data points where only one of the two output dimensions has been measured, better than the unconstrained one (compare left part of $v_{\text{aux}}$ in the figure). This is

---

¹The code used for the experiments is available at https://github.com/ppilar/SumConstraint.
Figure 1: Demonstration of the auxiliary variables approach for the harmonic oscillator. The quantities $z_{\text{aux}}$, $z$ and $v_{\text{aux}}$, $v$ refer to the position and velocity of the harmonic oscillator, respectively. We distinguish between $z_{\text{aux}}$, $z$ and $v_{\text{aux}}$, $v$ to emphasize that, while they aim to approximate the same curve, they are learned by different GPs. The posterior means of the GPs are depicted, together with the $2\sigma$ credible intervals. The dotted lines represent the true curves and the big dots/crosses correspond to the data available to the GPs. **Left:** Results for the unconstrained GP are shown. For this example, these outputs coincide with the auxiliary outputs required for the constrained GP. **Middle:** The transformed outputs learned by the constrained GP are depicted, together with the constraint $2E = kz^2 + mv^2$ (where $k = m = 1$). The results for the auxiliary outputs have been employed to create virtual measurements at zero crossings (differently colored squares) in order to force the quadratic functions towards zero. **Right:** The backtransformed outputs of the constrained GP are shown, where the auxiliary outputs $z_{\text{aux}}$ and $v_{\text{aux}}$ have been used to recover the signs.

natural, since the constrained GP has implicitly added a correlation between the two outputs, which the unconstrained GP is lacking.

The credible intervals in Figure 1 clarify another advantage of the constrained GP: when multiple outputs are learned to a different degree of certainty, information can be transferred from high- to low-credibility outputs, thereby narrowing the credible intervals also for the latter. This is clearly visible in areas with incomplete measurements. On the other hand, credible intervals tend to be overconfident in the vicinity of virtual measurements. Due to the nonlinear, piecewise backtransformation, some discontinuities have been introduced in the credible intervals of the constrained GP near the zero crossings.

In Table 5.1, values for both the root mean squared error (RMSE) and the average absolute violation of the constraint $|\Delta C|$ are given for various noise levels $\sigma_n$, both with complete and incomplete measurements; in case of incomplete measurements, the output components have been omitted at random with probability $f_d = 0.2$. The values have been obtained by
4. Experimental results

<table>
<thead>
<tr>
<th>$f_d$</th>
<th>$\sigma_n = 0.05$</th>
<th>$\sigma_n = 0.1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GP-c</td>
<td>GP-u</td>
</tr>
<tr>
<td>$f_d = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RMSE</td>
<td>2.3±0.6</td>
<td>3.2±0.5</td>
</tr>
<tr>
<td>$</td>
<td>\Delta C</td>
<td>$</td>
</tr>
<tr>
<td>$f_d = 0.2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RMSE</td>
<td>3.4±3.0</td>
<td>4.8±2.8</td>
</tr>
<tr>
<td>$</td>
<td>\Delta C</td>
<td>$</td>
</tr>
<tr>
<td>$f_d = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\sigma_n = 0.3$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_d = 0.2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RMSE</td>
<td>13.7±3.7</td>
<td>18.5±3.3</td>
</tr>
<tr>
<td>$</td>
<td>\Delta C</td>
<td>$</td>
</tr>
</tbody>
</table>

Table 5.1: Comparison of the performance of the constrained GP (GP-c) and the unconstrained GP (GP-u) for the harmonic oscillator. Shown are the root mean squared error (RMSE) of the prediction as well as the mean absolute violation of the constraint, $|\Delta C|$. The standard deviation of the noise is given by $\sigma_n$ whereas $f_d$ is the probability with which output components have been omitted at random from the data. The values have been obtained by averaging over 50 datasets and are given plus-or-minus one standard deviation. Bold font highlights best performance.

averaging over 50 datasets. We observe that the constrained GP fulfills the constraint with up to two orders of magnitude higher accuracy and also performs slightly better in terms of RMSE.

The reason why the constraint is not fulfilled with yet higher accuracy for the constrained GP is that around zero crossings it can occur that invalid values are predicted by the constrained GP (that is, negative values for $z^2$ and $v^2$), which we pragmatically put to zero. This is also the origin of the small artefacts visible in that region of the mean curves in the right plot of Fig. 1.

4.2 Pose estimation

Here we demonstrate how our approach can incorporate length constraints [Perriollat et al., 2011], inspired by applications such as pose estimation. In essence, the length constraint states that the distance $L_{lm}$ between two adjacent points (indexed by $l$ and $m$) in a rigid body is constant, irrespective of position and orientation of the body. When the position is given in terms of Cartesian coordinates $z_i$, the length constraint takes the following form
\[ \sum_{i=1}^{3} z_{li}^2 - 2z_{li}z_{mi} + z_{mi}^2 = L_{lm}^2. \] (10)

This constraint is no longer an instance of the sum constraint as defined in (5), since the middle term depends on multiple outputs. However, with a more elaborate transformation procedure, the sum constraint can still be applied.

To make this more concrete, we consider the example of a triangle in the plane. Here, the outputs of interest are the coordinates of the triangle corners, \( f = [z_{1x}, z_{1y}, z_{2x}, z_{2y}, z_{3x}, z_{3y}] \). The input \( \alpha \) is a continuous parametrization of different poses of the triangle in the plane. Although \( \alpha \) is one-dimensional in this example, the approach generalizes to higher dimensional inputs. In our choice of transformed outputs, we follow the approach by Salzmann and Urtasun [2010a], where pairwise products of the original outputs are learned and subsequently transformed back via a singular value decomposition (SVD); for more details on the technicalities we refer to Section C.5 in the Supplementary material.

A visualization of the problem is provided in Figure 2 where different poses \( \alpha \) of the triangle are depicted; the blue points represent the corners of the triangle, the positions of which are learned by the GP. As can be seen from the data in Table 5.2, our approach here performs best for low noise levels. When the noise is very small, \( \sigma \lesssim 1e^{-3} \), the constrained approach achieves about the same overall accuracy in terms of RMSE as the unconstrained GP, whereas the error in the constraint is reduced by factors of 2-6.

This reduction is not simply a result of the particular parameterization of the problem, which enforces the constraint implicitly for noiseless observations, as shown by Salzmann and Urtasun [2010a]. To see that, we included the results for a GP that is trained on the transformed outputs, but where the constraint is not enforced explicitly. Table 5.2 shows that the result is improved when enforcing the constraint in addition to using the transformed outputs.

4.3 Real data experiment: double pendulum

In this section we consider the ‘Double Pendulum Chaotic’ dataset [Asseman et al., 2018]; this dataset consists of 21 different two dimensional trajectories of a double pendulum and contains annotated positions of the masses attached at the ends of the two pendula. Each trajectory consists of about 17000 measurements, taken at a frequency of 500 Hz. For more information on the parameters of the double pendulum, see Section D.1 in the Supplementary material. We attempt to construct a GP that models both positions
Figure 2: Visualization of the triangle in the plane. The task for the GP is to give the location of the corners of the triangle (blue dots) when given the parameter $\alpha$, which parameterizes different poses of the triangle.

| $\sigma_n$ | RMSE       | $|\Delta C|$   | $|\Delta C|$   |
|------------|------------|----------------|----------------|
| 1e-4       | $3.3\pm0.2$| $0.3\pm0.0$    | $0.3\pm0.0$    |
|            | $4.8\pm0.3$| $1.9\pm0.1$    | $1.8\pm2.7$    |
| 1e-3       | $5.5\pm1.0$| $0.8\pm0.1$    | $0.8\pm0.1$    |
|            | $5.0\pm0.3$| $2.0\pm0.2$    | $2.0\pm0.2$    |
| 1e-2       | $4.2\pm0.8$| $6.2\pm1.0$    | $6.2\pm1.0$    |
|            | $1.6\pm0.2$| $6.4\pm1.4$    | $6.4\pm1.4$    |

Table 5.2: Results for the length constraint applied to the triangle in the plane. We compare results for the constrained GP (GP-c), the unconstrained GP (GP-u) and the unconstrained GP trained on the transformed outputs (GP-tr) [Salzmann and Urtasun, 2010a]. For small values of noise $\sigma_n$, the sum constraint improves the performance of the GP. The values have been obtained by averaging over 50 datasets and are given plus-or-minus one standard deviation.
$z_x$, $z_y$ and velocities $v_x$, $v_y$ of the two masses (i.e. 8 outputs), while at the same time respecting the law of energy conservation; the time $t$ serves as input. As friction is present, we consider a limited section of the trajectory during the second half of the motion where we can assume constant energy (compare Figure 3); energy conservation here takes the form

$$E = m_b g z_{by} + m_g g z_{gy} + \frac{m_b}{2} (v_{bx}^2 + v_{by}^2) + \frac{m_g}{2} (v_{gx}^2 + v_{gy}^2),$$

(11)

where $g$ denotes the gravitational acceleration on earth, and where the indices $b$ and $g$ refer to the blue and the green pendulum, respectively. The constraint is incorporated into the GP in analogy to the harmonic oscillator. In terms of (6), we identify $a_1 = 0$, $a_2 = m_b g$, $a_3 = 0$, $a_4 = m_g g$, $a_5 = m_b/2$, $a_6 = m_b/2$, $a_7 = m_g/2$, $a_8 = m_g/2$, $h_2(z_{by}) = z_{by}$, $h_4(z_{gy}) = z_{gy}$, $h_5(v_{bx}) = v_{bx}^2$, $h_6(v_{by}) = v_{by}^2$, $h_7(v_{gx}) = v_{gx}^2$, $h_8(v_{gy}) = v_{gy}^2$ and $C = E$; note that the coefficients $a_1$, $a_3$ correspond to the outputs $z_{bx}$, $z_{gx}$, which are not part of the constraint (11).

We pick a sequence of 200 data points (which are fairly close together) from one of the trajectories; 15 of these points are used during hyperparameter optimization, and to receive an estimate $\hat{E}$ of the energy. The remaining 185 points are used as test data to compare the performance of constrained and unconstrained GP, both in terms of constraint fulfillment and in terms of RMSE with respect to the data.

Results for one individual sequence are shown in the rightmost plot of Figure 3. We observe that the constrained GP is better at learning the precise shapes of the extrema of the velocity curves, although some artefacts arise close to zero crossings due to inaccurately learned square values. For values close to zero, the credible intervals of the unconstrained GP are often smoother and thinner than those of the constrained GP.

Averaging the results over 50 sequences chosen at random from the second half of the trajectories (with less friction), the RMSE for the constrained GP is 0.31±0.14, whereas for the unconstrained GP it is 0.33±0.16. In terms of constraint fulfillment, the constrained GP clearly performs better with $|\Delta C| = 0.17 \pm 0.14$ as compared to $|\Delta C| = 0.91 \pm 0.61$ for the unconstrained GP. The values here are given plus-or-minus one standard deviation.

5 Related work

Several research projects have considered incorporating constraints into the GP; examples include boundary conditions [Solin and Kok, 2019], inequality constraints [Veiga and A.Marrel, 2012, Maatouk and Bay, 2017] and differential equation constraints [Jidling et al., 2017, Raissi et al., 2017, 2018].
Figure 3: **Left:** Trajectory of the double pendulum. Note that the trajectory shown here is longer than the sequences of motion considered in the plots to the right. **Middle:** Kinetic energy $E_{\text{kin}}$, potential energy $E_{\text{pot}}$ and total energy $E$ of the double pendulum are shown. It is apparent that for the considered segment of the motion the energy is constant for practical purposes, except for fluctuations in the contribution of the kinetic energy due to measurement error. An estimate $\hat{E}$ of the energy is obtained by averaging over $E$. **Right:** Positions $z_0$, $z$ and velocities $v_0$, $v$ of the masses (four components each) as learned by the unconstrained (left inset) and the constrained GP (right inset), respectively; the posterior means of the GPs are depicted together with the $2\sigma$ credible intervals. The dotted lines represent the available data, where the subset of big dots has been used for training.

The recent review by Swiler et al. [2020] provides a good overview of the existing literature on constrained GPs. So far, most of the efforts have been concentrated on the single-task GP. The sum constraint, however, is qualitatively very different from constraints on single-task GPs, in that it explicitly enforces a relationship between different outputs instead of acting on individual outputs. Hence, in this section, we focus on works that consider constraints on the outputs of multitask GPs.

Prior knowledge about vector fields have been imposed into GPs through special divergence-free and curl-free kernels [Wahlström et al., 2013]. Jidling et al. [2017] developed a more general method to include linear operator constraints into the kernel of the GP; this is possible by using the property that GPs are closed under linear transformations [Papoulis and Pillai, 2001] and relating the GP to a suitable latent GP, resembling the use of potential functions in physics. See also Lange-Hegermann [2018] for a discussion of this approach from a more mathematical perspective. Practical applications include modelling of electromagnetic fields [Solin et al., 2018] and reconstruction of strain fields [Jidling et al., 2018, Hendriks et al., 2019b,a, 2020b]. Geist and Trimpe [2020] consider affine constraints on the dynamics of mechanical systems and construct a GP satisfying Gauss’ principle of
least constraint.

There is a connection between our method and the method by Jidling et al. [2017]: while they do not consider affine constraints, their approach can be extended to include those in the context of the constant linear sum constraint (compare also Hendriks et al. [2020a], where the same idea is applied to neural networks). The two works attack the problem from different angles: whereas Jidling et al. [2017] start by directly constructing a covariance matrix out of vectors spanning the nullspace of the constraining operator, we start with the covariance matrix and subsequently constrain it. More details on these parallels are given in Appendix E. An advantage of our approach is that it is straightforward to include additional structure in the task kernel, such as in (B.12). Furthermore, we consider the general case of nonconstant, nonlinear sum constraints.

Constructing kernels that are invariant with respect to certain symmetry transformations has proven fruitful in the fields of atomic and molecular physics. Glielmo et al. [2017] consider GPs to model interatomic force fields; they construct a ‘covariant kernel’ by including symmetries of the force, such as rotation and reflection. Methods for constructing invariant kernels are given by Haasdonk and Burkhardt [2007], whereas Chmiela et al. [2020] use a similar approach to construct a kernel that allows for simultaneous prediction of energies and forces in molecules.

Pose estimation constitutes another area where constrained multitask GPs are of importance; in the case of rigid pose estimation, the lengths are required to be constant. A method to explicitly enforce the constraints during inference is given by Salzmann and Urtasun [2010b], whereas Salzmann and Urtasun [2010a] propose a method to implicitly enforce the fixed-length constraint by learning transformed outputs in which the constraint is linear. We followed this latter approach in the example with the rotated triangle in Section 4.2; in addition to using the transformed outputs we also imposed the length constraint explicitly, which (at least in principle) should allow for training points that do not fulfill the constraint exactly.

6 Conclusions and future work

We have derived a way of incorporating both linear and nonlinear sum constraints into multitask GPs. This is achieved by learning transformed outputs and by conditioning the prior distribution of the GP on the constraint. The toy problem of the harmonic oscillator demonstrated the potential of the method; it showed that the constraint is fulfilled with high accuracy and that the constrained GP can mitigate detrimental effects of noise or of incomplete measurements. Our experiment with the triangle in the plane
showed that the sum constraint improved the method by Salzmann and Urtasun [2010a] of including the length constraint into pose estimation problems; so far, these results are particularly promising in the low-noise setting. The results for the double pendulum dataset showed that our method also works well in case of real-world, noisy data, given a way of estimating the constraint with sufficient accuracy.

In light of the results received for the triangle in the plane in Section 4.2, it appears as if it would be worth investigating the applicability of this approach to pose estimation problems further; especially, in cases where the approach by Salzmann and Urtasun [2010a] gives good results, our constrained GP could potentially improve the performance. To increase the suitability of the approach for big datasets, combining the sum constraint framework with methods such as sparse variational inference [Hensman et al., 2013] appears to be a fruitful direction of inquiry. Finding general methods to incorporate constraints similar to the length constraint (10) into the GP, where nonlinearities may depend on more than one of the outputs at once, constitutes another interesting avenue of future research and would widen the range of possible applications.

Acknowledgements

The work is 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.

Bibliography


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


Figure 4: Demonstration of the auxiliary variables approach for the free fall. The quantities $z_0$, $z$ and $v_{aux}$, $v$ refer to the position and velocity of the mass, respectively. We distinguish between $z_0$, $z$ and $v_{aux}$, $v$ to emphasize that, while they aim to approximate the same curve, they are learned by different GPs. The posterior means of the GPs are depicted, together with the 2-sigma credible intervals. The dotted lines represent the true curves and the big dots/crosses correspond to the data available to the GPs. **Left:** Results for the unconstrained GP are shown. For this example, the $v$-curve coincides with the auxiliary variable $v_{aux}$ required for the constrained GP. **Middle:** The transformed outputs learned by the constrained GP are depicted. The results for the auxiliary variable have been employed to create a virtual measurement for $v^2$ at the zero crossing of $v_{aux}$ (differently colored square) in order to force the quadratic function towards zero. **Right:** The backtransformed outputs of the constrained GP are shown, where the auxiliary output $v_{aux}$ has been used to recover the sign of $v$.

### A Additional examples

In this Section, we take a look at some additional examples where the sum constraint can be applied. The free fall dataset in Section A.1 is another example from physics, where one of the outputs enters linearly into the constraint, instead of quadratically. The damped harmonic oscillator in Section A.2 constitutes a variation of the harmonic oscillator toy example and demonstrates the case of a non-constant constraint. In Section A.3, we investigate an example where the constraint includes different nonlinearities.

#### A.1 Free fall

In addition to the harmonic oscillator (see Section 4.1), we investigated the simple example of a mass in free fall as a second toy problem. Here, the output of the GP consists in position and velocity of the mass, $f^T = [z, v]$, whereas the time $t$ serves as input. Then the constraint takes the following form

$$
\mathcal{F}[f(t)] = mgz(t) + \frac{m}{2}v(t)^2 = E_{pot}(t) + E_{kin}(t) = E. \quad (A.1)
$$
In terms of (6), we identify \( a_1 = mg, a_2 = m/2, h_1(z) = z, h_2(v) = v^2 \) and \( C = E \). Hence, we receive for the transformed outputs \( f'_1 = f_1 = z \) and \( f'_2 = v^2 \). We choose the auxiliary output as \( f'_{\text{aux}} = v \), which we use to extract the sign when backtransforming \( f'_2 \) and to create virtual measurements for \( f'_2 \) at the zero crossings of the posterior mean of \( f'_{\text{aux}} \). In order to fit the transformed outputs of the GP, the observations \( y_k = [z_k, v_k]^T \) are transformed analogously to obtain \( y'_k = [z_k, v_k^2, v_k]^T \); \( v_k \) is part of the constrained outputs, as this improves the performance for this example. The virtual measurements are also included in the transformed data \( y' \). In terms of the transformed outputs the constraint can be written compactly as \( FF' = C \), where \( F = [a_1, a_2, 0] \). For more details on the free fall dataset, see Section C.3.

In Figure 4, results for both constrained and unconstrained GP, applied to the free fall dataset, are depicted. When comparing the left and the right plot, it is apparent, that the constrained GP manages to mitigate detrimental effects of both noise and incomplete measurements, where some of the observed output components have been omitted at random, better than the unconstrained GP (compare area around peak of \( z_0 \) in the figure).

In terms of (6), we identify \( a_1 = mg, a_2 = m/2, h_1(z) = z, h_2(v) = v^2 \) and \( C = E \). Hence, we receive for the transformed outputs \( f'_1 = f_1 = z \) and \( f'_2 = v^2 \). We choose the auxiliary output as \( f'_{\text{aux}} = v \), which we use to extract the sign when backtransforming \( f'_2 \) and to create virtual measurements for \( f'_2 \) at the zero crossings of the posterior mean of \( f'_{\text{aux}} \). In order to fit the transformed outputs of the GP, the observations \( y_k = [z_k, v_k]^T \) are transformed analogously to obtain \( y'_k = [z_k, v_k^2, v_k]^T \); \( v_k \) is part of the constrained outputs, as this improves the performance for this example. The virtual measurements are also included in the transformed data \( y' \). In terms of the transformed outputs the constraint can be written compactly as \( FF' = C \), where \( F = [a_1, a_2, 0] \). For more details on the free fall dataset, see Section C.3.

In Figure 4, results for both constrained and unconstrained GP, applied to the free fall dataset, are depicted. When comparing the left and the right plot, it is apparent, that the constrained GP manages to mitigate detrimental effects of both noise and incomplete measurements, where some of the observed output components have been omitted at random, better than the unconstrained GP (compare area around peak of \( z_0 \) in the figure).

### Table 3: Comparison of the performance of the constrained GP (GP-c) and the unconstrained GP (GP-u) for the free fall. Shown are the root mean squared error (RMSE) of the prediction as well as the mean absolute violation of the constraint, \( |\Delta C| \). The standard deviation of the noise is given by \( \sigma_n \) whereas \( f_d \) is the probability with which output components have been omitted at random from the data. The values have been obtained by averaging over 50 datasets and are given plus-or-minus one standard deviation. Bold font highlights best performance.

<table>
<thead>
<tr>
<th>( f_d )</th>
<th>( \sigma_n = 0.05 )</th>
<th>( \sigma_n = 0.1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GP-c</td>
<td>GP-u</td>
</tr>
<tr>
<td>RMSE</td>
<td>1.9±0.5</td>
<td>2.3±0.4</td>
</tr>
<tr>
<td>(</td>
<td>\Delta C</td>
<td>)</td>
</tr>
<tr>
<td>( f_d = 0 )</td>
<td>RMSE</td>
<td>2.5±0.9</td>
</tr>
<tr>
<td></td>
<td>(</td>
<td>\Delta C</td>
</tr>
<tr>
<td>( f_d = 0.3 )</td>
<td>RMSE</td>
<td>10.1±3.6</td>
</tr>
<tr>
<td></td>
<td>(</td>
<td>\Delta C</td>
</tr>
</tbody>
</table>

For the free fall dataset, the constraint can be written compactly as \( FF' = C \), where \( F = [a_1, a_2, 0] \). For more details on the free fall dataset, see Section C.3.
When looking at the $2\sigma$ credible intervals, we get the same picture as before for the harmonic oscillator: the constrained GP can utilize higher certainty in one output and transfer it to the other one, resulting in overall slimmer intervals. Close to the zero crossing of $v$, however, some artefacts are present due to the piecewise, nonlinear backtransformation, which are absent for the unconstrained GP; furthermore, confidence intervals are stretched a bit due to the backtransformation via the square root.

In Table 3, values for both the root mean squared error (RMSE) and the average absolute violation of the constraint $|\Delta C|$ are given for various noise levels $\sigma_n$, both with complete and incomplete measurements; in case of incomplete measurements, the output components have been omitted at random with probability $f_d = 0.3$. The values have been obtained by averaging over 50 datasets. We observe that the constrained GP fulfills the constraint with up to two orders of magnitude higher accuracy and also performs better in terms of RMSE.

### A.2 Damped harmonic oscillator

Next, we investigate a slight variation of the harmonic oscillator, the damped harmonic oscillator. The formal treatment remains mostly unchanged and details can be found in Section 3.1 in the main paper; the main difference is that damping has been added to the model of the oscillator. As a consequence, the energy is no longer constant and the amplitude of the oscillation decays over time; see Section C.2 for more details. Hence, this example constitutes an instance of the non-constant sum constraint $F[f(t)] = E(t)$, where Algorithm 2 applies.

In Figure 5, results for both constrained and unconstrained GP are depicted. The findings are similar to the undamped harmonic oscillator, and it is apparent that the constrained GP can mitigate the detrimental effects of noisy or incomplete measurements better than the unconstrained GP. In Table 4, the performance on 50 random datasets is evaluated. The outputs of the constrained GP fulfill the constraint with up to two orders of magnitude higher accuracy than the unconstrained one, and also perform slightly better in terms of RMSE. This example demonstrates that, given similar datasets, the performance of our method is very similar, both in case of constant and non-constant constraints (compare Section 4.1).

### A.3 Non-square nonlinearity

Finally, we investigate an example where nonlinearities other than the square nonlinearity are involved in the constraint. We consider the outputs $f =$
Figure 5: Demonstration of the auxiliary variables approach for the damped harmonic oscillator. The quantities \( z_{\text{aux}}, z \) and \( v_{\text{aux}}, v \) refer to the position and velocity of the damped harmonic oscillator, respectively. We distinguish between \( z_{\text{aux}}, z \) and \( v_{\text{aux}}, v \) to emphasize that, while they aim to approximate the same curve, they are learned by different GPs. The posterior means of the GPs are depicted, together with the 2σ credible intervals. The dotted lines represent the true curves and the big dots/crosses correspond to the data available to the GPs. **Left:** Results for the unconstrained GP are shown. For this example, these outputs coincide with the auxiliary outputs required for the constrained GP. **Middle:** The transformed outputs learned by the constrained GP are depicted, together with the constraint \( 2E(t) = k z^2 + m v^2 \) (where \( k = 1 \) and \( m = 1 \)). The results for the auxiliary outputs have been employed to create virtual measurements at zero crossings (differently colored squares) in order to force the quadratic functions towards zero. **Right:** The backtransformed outputs of the constrained GP are shown, where the auxiliary outputs \( z_{\text{aux}} \) and \( v_{\text{aux}} \) have been used to recover the signs.

\[
[f_1, f_2]^T, \text{ on which we want to enforce the constraint}
\]

\[
\mathcal{F}[f(x)] = \log(f_1(x)) + \sin(f_2(x)) = C(x). \tag{A.2}
\]

In terms of (6), we identify \( a_1 = 1, a_2 = 1, h_1(f_1) = \log(f_1), h_2(f_2) = \sin(f_2) \) and \( C = C(x) \). Here, we assume that the true value \( C(x) \) is known. Hence, we receive for the transformed outputs \( f_1' = \log(f_1) \) and \( f_2' = \sin(f_2) \). We choose the auxiliary output as \( f_{\text{aux}}^1 = f_2 \), which we use to disambiguate the backtransformation via the arcsine, that is we keep track of how many multiples of \( \pm \pi/2 \) the output \( f_{\text{aux}}^1 \) has crossed. We also use the auxiliary output to create virtual measurements for \( f_2' \) at points where the posterior mean of \( f_{\text{aux}}^1 \) crosses multiples of \( \pm \pi/2 \), in order to reduce artefacts caused by the discontinuity in the backtransformation.

In Figure 6, results for both constrained and unconstrained GP are depicted. It is apparent that, while not perfect, the constrained GP outperforms the unconstrained one. In Table 5, the results averaged over 50 datasets are given. We see, that the constrained GP outperforms the unconstrained one.
Table 4: Comparison of the performance of the constrained GP (GP-c) and the unconstrained GP (GP-u) for the damped harmonic oscillator. Shown are the root mean squared error (RMSE) of the prediction as well as the mean absolute violation of the constraint, $|\Delta C|$. The standard deviation of the noise is given by $\sigma_n$ whereas $f_d$ is the probability with which output components have been omitted at random from the data. The values have been obtained by averaging over 50 datasets and are given plus-or-minus one standard deviation. Bold font highlights best performance.

<table>
<thead>
<tr>
<th>$f_d$</th>
<th>$\sigma_n = 0.05$</th>
<th>$\sigma_n = 0.1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GP-c</td>
<td>GP-u</td>
</tr>
<tr>
<td>$f_d = 0$</td>
<td>RMSE</td>
<td>3.1±1.3</td>
</tr>
<tr>
<td></td>
<td>$</td>
<td>\Delta C</td>
</tr>
<tr>
<td>$f_d = 0.2$</td>
<td>RMSE</td>
<td>4.1±2.3</td>
</tr>
<tr>
<td></td>
<td>$</td>
<td>\Delta C</td>
</tr>
</tbody>
</table>

Table 4: Comparison of the performance of the constrained GP (GP-c) and the unconstrained GP (GP-u) for the damped harmonic oscillator. Shown are the root mean squared error (RMSE) of the prediction as well as the mean absolute violation of the constraint, $|\Delta C|$. The standard deviation of the noise is given by $\sigma_n$ whereas $f_d$ is the probability with which output components have been omitted at random from the data. The values have been obtained by averaging over 50 datasets and are given plus-or-minus one standard deviation. Bold font highlights best performance.

in terms of RMSE, and it fulfills the constraint with up to 30 times higher accuracy.

A.4 Comparison of approximation methods

In this section, we give a brief comparison of different methods of approximate inference at the example of the harmonic oscillator. The approximation methods under consideration are the Laplace approximation B.3 and variational inference B.4.

Fig. 7 shows the predictive performance of the the unconstrained GP, variational inference and the Laplace approximation. While it is apparent that both approximate GPs fulfill the constraint with high precision, the variational approach tends to overfit to the data. On the other hand, overconfident credible intervals seem to be less of an issue for the variational approach than for the Laplace approximation.

In Table 6, results obtained when averaging over 20 runs are given for different noise settings. It is apparent that the constrained GP with Laplace approximation performs best. While the constrained GP utilizing variational inference performs worst in terms of root-mean-square error, the constraint
B. Technicalities

Figure 6: Demonstration of the auxiliary variables approach for the example with non-square nonlinearity. The quantities $f_1^0$, $f_1$ and $f_2^0$, $f_2$ refer to the same respective outputs of the GPs. We distinguish between $f_1^0$, $f_1$ and $f_2^0$, $f_2$ to emphasize that, while they aim to approximate the same curve, they are learned by different GPs. The posterior means of the GPs are depicted, together with the 2-sigma credible intervals. The dotted lines represent the true curves and the big dots/crosses correspond to the data available to the GPs. **Left:** Results for the unconstrained GP are shown. For this example, the $f_2^0$-curve coincides with the auxiliary output $f_{aux}$ required for the constrained GP. **Middle:** The transformed outputs learned by the constrained GP are depicted, together with the constraint $C = \log(f_1) + \sin(f_2)$. The result for the auxiliary output has been employed to create a virtual measurement at the point where $f_{aux}$ crosses $-\pi/2$ (differently colored square). **Right:** The backtransformed outputs of the constrained GP are shown, where the auxiliary output $f_{aux}$ has been used to disambiguate the backtransformation via the arcsine.

is still fulfilled with high precision. In case of the variational approach, it might be possible to improve upon these results by trying different parameterizations of the variational distribution, or by finding a better suited optimization scheme.

B Technicalities

B.1 Background on GP regression

In this section we give a very brief overview of some important GP regression formulas. For more detailed accounts see e.g. Rasmussen and Williams [2006], Lindholm et al. [2021]. Given the mean function $m(\cdot)$ and kernel $K(\cdot, \cdot)$ of the GP, the predictive distribution of the GP can be calculated by first constructing the joint distribution between observations $y$ and function
Table 5: Comparison of the performance of the constrained GP (GP-c) and the unconstrained GP (GP-u) for the example with non-square nonlinearity. Shown are the root mean squared error (RMSE) of the prediction as well as the mean absolute violation of the constraint, $|\Delta C|$. The standard deviation of the noise is given by $\sigma_n$ whereas $f_d$ is the probability with which output components have been omitted at random from the data. The values have been obtained by averaging over 50 datasets and are given plus-or-minus one standard deviation. Bold font highlights best performance.

<table>
<thead>
<tr>
<th>$f_d$</th>
<th>$\sigma_n = 0.05$</th>
<th>$\sigma_n = 0.1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GP-c</td>
<td>GP-u</td>
</tr>
<tr>
<td></td>
<td>RMSE</td>
<td></td>
</tr>
<tr>
<td>$f_d = 0$</td>
<td>$3.0\pm 1.7$</td>
<td>$3.5\pm 0.5$</td>
</tr>
<tr>
<td>$</td>
<td>\Delta C</td>
<td>$</td>
</tr>
<tr>
<td>$f_d = 0.2$</td>
<td>$3.1\pm 1.0$</td>
<td>$8.6\pm 9.9$</td>
</tr>
<tr>
<td>$</td>
<td>\Delta C</td>
<td>$</td>
</tr>
<tr>
<td>$\sigma_n = 0.15$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$f_d = 0$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$</td>
<td>\Delta C</td>
<td>$</td>
</tr>
<tr>
<td>$f_d = 0.2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$</td>
<td>\Delta C</td>
<td>$</td>
</tr>
</tbody>
</table>

Values at test locations $f_*$,

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

(B.1)

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

Then, the conditional distribution $f_*|X, y, X_*$ is constructed as follows:

$$
f_*|X, y, X_* \sim \mathcal{N}\left( \tilde{f}_*, \text{cov}(f_*) \right), \text{ where}
$$

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

(B.2b)

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

(B.2c)

The log-marginal likelihood, which is used for hyperparameter optimiz-
Figure 7: Comparison of the performance of the unconstrained GP (Left), the variational approach (Middle), and the Laplace approximation (Right) for the example of the harmonic oscillator. The quantities $z_{aux}$, $z$ and $v_{aux}$, $v$ refer to the position and velocity of the harmonic oscillator, respectively. The posterior means of the GPs are depicted, together with the $2\sigma$ credible intervals. The dotted lines represent the true curves and the big dots/crosses correspond to the data available to the GPs.

The Laplace approximation, is given by

$$\log p(y|X) = -\frac{1}{2}(y - m)^T(K + \sigma_n^2 I)^{-1}(y - m)$$

$$- \frac{1}{2}\log|K + \sigma_n^2 I| - \frac{n}{2}\log2\pi.$$ (B.3)

## B.2 Accommodating incomplete measurements

Throughout the paper, we often consider the case of incomplete measurements, i.e. data points where measurements are available only for a subset of the tasks. This can be taken into account by considering equation (B.1) and removing the the rows and columns on the right-hand side corresponding to missing entries in $y$.

To make this more concrete, let us assume that the j-th entry of $y$ is missing on the left-hand side of (B.1). Then we also delete the j-th row of $m$, $(K + \sigma_n^2 I)$ and $K_*$, as well as the j-th column of $(K + \sigma_n^2 I)$ and $K^T_*$, before explicitly constructing the joint distribution. We proceed analogously when training the constrained GP on the transformed data $y'$.

## B.3 Laplace approximation

The Laplace approximation can be employed when the noise distribution corresponding to the (transformed) observations $y'$ is non-Gaussian in order to obtain analytical expressions for the predictive equations and for the log-marginal likelihood. Following again Rasmussen and Williams [2006], we
<table>
<thead>
<tr>
<th>$f_d$</th>
<th>$\sigma_n = 0.05$</th>
<th>$\sigma_n = 0.1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GP-c L</td>
<td>GP-c var</td>
</tr>
<tr>
<td>$f_d = 0$</td>
<td>RMSE</td>
<td>$2.4\pm0.7$</td>
</tr>
<tr>
<td></td>
<td>$</td>
<td>\Delta C</td>
</tr>
<tr>
<td>$f_d = 0.2$</td>
<td>RMSE</td>
<td>$3.0\pm1.2$</td>
</tr>
<tr>
<td></td>
<td>$</td>
<td>\Delta C</td>
</tr>
</tbody>
</table>

Table 6: Comparison of the performance of the constrained GP with Laplace approximation (GP-c L), the constrained GP using variational inference (GP-c var), and the unconstrained GP (GP-u) for the harmonic oscillator. Shown are the root mean squared error (RMSE) of the prediction as well as the mean absolute violation of the constraint, $|\Delta C|$. The standard deviation of the noise is given by $\sigma_n$ whereas $f_d$ is the probability with which output components have been omitted at random from the data. The values have been obtained by averaging over 20 datasets and are given plus-or-minus one standard deviation. Bold font highlights best performance.

approximate the posterior $p(f'|y') \propto p(y'|f')p(f')$ via $p(f'|y') \approx q(f'|y') = \mathcal{N}(\hat{f}',(K^{-1} + W)^{-1})$, where

\[
\hat{f}' = K \left( \nabla_{f'} \log p(y'|f') \right) |_{f' = \tilde{f}'}, \\
W = -\nabla_{f'} \nabla_{f'} \log p(y'|f') |_{f' = \tilde{f}'}. 
\]  

Newton’s method is employed to iteratively determine $\hat{f}'$ from \((B.4a)\) via the update rule

\[
f'^{\text{new}} = f' - \gamma \left( \nabla_{f'} \nabla_{f'} \Psi(f') \right)^{-1} \nabla_{f'} \Psi(f') \\
= \gamma m + (1 - \gamma)f' + \gamma \left( (K^{-1} + W)^{-1} (\nabla_{f'} \log p(y'|f') + W(f' - m) \right), 
\]  

where $\Psi(f') = \log p(y'|f') + \log p(f'|X)$ and where $\gamma$ is the step size. In terms
of these quantities, the expressions (B.2) from the previous section become

\[ \tilde{f}'_s = m + K_s^T K^{-1} (\tilde{f}' - m), \]  
(B.7a)

\[ \text{cov}(f'_s) = K_{**} - K_s^T [K + W^{-1}]^{-1} K_s, \]  
(B.7b)

\[ \log p(y'|X) = -\frac{1}{2} (\tilde{f}' - m)^T K^{-1} (\tilde{f}' - m) + \log p(y'|\tilde{f}'), \]  
(B.7c)

For details on the derivation of these formulas, see Section 3.4 in Rasmussen and Williams [2006].

For the Laplace approximation, the likelihood \( p_{y'}(y'|f') \) of the transformed data \( y' = h(y) \) has to be known. We assume the original data \( y \) to be contaminated by Gaussian noise. In case of the square nonlinearity where \( y' = y^2 \), the likelihood is then given by the pdf of a noncentral chi-squared distribution. In the case of arbitrary nonlinearities \( h_j \), the likelihood can be obtained via

\[ p_{y'}(y'|f') = \prod_{ij} p_{y_{ij}}(y'_{ij}|f'_{ij}) = \prod_{ij} p_{y_{ij}} \left( h^{-1}_j(y'_{ij})|h^{-1}_j(f'_{ij}) \right) \frac{dh^{-1}_j(y'_{ij})}{dy'_{ij}}, \]  
(B.8)

where the indices \( i \) and \( j \) denote data points and tasks, respectively.

There is no guarantee that Newton’s method will determine the correct maximum \( \tilde{f}' \) in case of multimodal distributions, or that the resulting Gaussian distribution will constitute a good approximation of the true posterior. For these reasons, it has to be decided on a case by case basis whether the Laplace approximation should be employed or not. Visual inspection of the GP predictions often gives a good idea on whether the Laplace approximation performs well or not. In cases where it does not perform well, standard GP regression might still produce reasonable results. Alternatively, different methods such as variational inference [Tran et al., 2016] or expectation propagation [Minka, 2001] could be employed; the equations in (B.7) will then need to be replaced by expressions corresponding to these techniques. A brief discussion on variational inference is given in Appendix B.4, as well as a comparison with the Laplace approximation in Appendix A.4.

Throughout the paper, we used the Laplace approximation for the harmonic oscillator 4.1, the free fall A.1, the damped harmonic oscillator A.2, and the example with non-square nonlinearity A.3. In case of the double pendulum 4.3 and the triangle in the plane 4.2, we chose standard GP regression over the Laplace approximation.
B.4 Variational inference

As an alternative to the Laplace approximation (see previous section), variational inference [Blei et al., 2017, Titsias and Lawrence, 2010] can be employed to approximate the posterior \( p(f'|y') \propto p(y'|f')p(f') \) when the likelihood \( p(y'|f') \) is non-Gaussian. The idea is to approximate the posterior with the variational distribution \( q(f') \sim \mathcal{N}(f'|\mu_q, \Sigma_q) \) and to learn the parameters \( \mu_q, \Sigma_q \) by minimizing the Kullback-Leibler divergence KL(\( q(f') || p(f'|y') \)) between variational distribution and posterior. In order to ensure positive definiteness, the entries of the covariance matrix \( \Sigma_q \) are not learned directly, but instead the entries of its Cholesky factor \( L_q \), where it holds that \( \Sigma_q = L_q L_q^T \). Since an exact minimization of the KL divergence is intractable, the evidence lower bound ELBO = \( \log p(y') - \text{KL}(q(f')||p(f'|y')) \) is maximized in its stead.

The predictive equations in terms of the variational parameters are given by

\[ \bar{f}' = m + K_*^T K^{-1}(\mu_q - m), \]  
\[ \text{cov}(f') = K_{**} + K_*^T K^{-1} \left( \Sigma_q K^{-1T} - I \right) K_*, \]

and the ELBO can be rewritten in terms of numerically tractable, one-dimensional integrals

\[
\text{ELBO} = \mathbb{E}_q[\log(p(y'|f'))] - \text{KL}(q(f')||p(f')) \\
= \int \log(p(y'|f'))q(f')df' - \text{KL}(q(f')||p(f')) \\
= \sum_i \int \log(p(y_i'|f_i'))q(f_i')df_i' - \text{KL}(q(f')||p(f')).
\]

Equations B.9 are obtained analogously to (B.7), for a derivation of (B.10), see Titsias and Lawrence [2010]. The parameters \( \mu_q, \Sigma_q \) of the variational distribution and the GP hyperparameters are determined jointly by maximizing the ELBO, which we do by employing gradient descent. Same as for the Laplace approximation, the likelihood for the transformed data \( p(y'|f') \) is required when calculating the ELBO and can be obtained via (B.8).

In our experiments, the variational approach tended to overfit to the data more than the Laplace approximation. It is possible that a different parameterization of the variational covariance matrix, or a different optimization scheme would manage to yield better results. In the paper, we went with the Laplace approximation over variational inference; for a brief comparison of the two approaches at the example of the harmonic oscillator, see Appendix A.4.
B.5 Kernel and mean

Throughout the paper we use a radial basis function (RBF) kernel (also: squared exponential kernel) as data kernel,

$$k_{\text{RBF}}(x, x') = \sigma_f^2 \exp \left( -\frac{||x - x'||^2}{2l^2} \right), \quad (B.11)$$

where $\sigma_f$ is a scale factor and $l$ is the length scale. We use the position independent index kernel provided by gpytorch [Gardner et al., 2018],

$$k_t = BB^T + \text{diag}(v), \quad (B.12)$$

where $B$ is a low-rank matrix and $v$ is a non-negative vector; we chose the rank of $B$ to be equal to the number of tasks of the GP in question. The parameters $\sigma_f, l, B$ and $v$ are to be learned during the training process. For more examples of possible kernels see e.g. Rasmussen and Williams [2006], MacKay [1998]. The Gram matrix is then constructed via the Kronecker product

$$K_{f,f'}(X, X') = k_{\text{RBF}}(X, X') \otimes k_t. \quad (B.13)$$

We chose constant mean functions for all outputs of the multitask GP. All the models have been implemented in python with the library gpytorch [Gardner et al., 2018].

B.6 Special case of constant constraints

In Section 3.2 in the main paper, we detailed the method for incorporating the sum constraint into the GP in the general, non-constant case. Subsequently, in Section 3.2, we pointed out the possibility of implementing the sum constraint in a more efficient way for the case, where all of the constraints are constant and where the kernel of the GP factorizes into data and task kernel, as in (4) and (B.13). The main ideas are discussed in the main paper, here we summarize the modified procedure in Algorithm 3. A proof that the factorization holds also for the constrained GP is given in the next section.

Proof of factorization

In this section we provide formal proof that the claims made in Section 3.2 hold, i.e. that directly constraining the mean and task covariance matrix and subsequently performing the Kronecker product with the data mean and covariance matrix does indeed lead to the constrained GP from Section 3.2.
Algorithm 3: Constraining the GP - Special Case of Constant Task Interdependencies

**Input:** data mean $m_d(\cdot) = 1$; data kernel $k_d(\cdot, \cdot)$; task mean $\mu_t$; task covariance matrix $\Sigma_t$; constraints $(F, S)$; (transformed) data $X, y'$; points of prediction $X_*$

**Output:** constrained predictive distribution $f'_*|X, y', X_*$

**Note:** During hyperparameter optimization $X_*$ = {} and hence $f'_* = {}$

**Step 1:** Use $F, S$ to calculate constrained $\mu'_t, \Sigma'_t$ according to (8b)

**Step 2:** Construct parameters $m, K$ of the (single task) joint prior distribution according to (B.1)
- omit noise term $\sigma^2_n I$

**Step 3:** Use $\mu'_t, \Sigma'_t, m, K$ to construct constrained (multi task) $\mu'_t, \Sigma'_t$ according to (4)

**Step 4:** Remove entries in $\mu'_t, \Sigma'_t$ corresponding to incomplete measurements as detailed in Section B.2

if Hyperparameter optimization then
    **Step 5:** Calculate the log marginal likelihood according to (B.7c)
    **Step 6:** Perform optimization step
else if Prediction then
    **Step 5:** Calculate the predictive distribution $f'_*|X, y', X_*$ according to (B.7a)
end if

To start, let us summarize the objects involved:

$$\Sigma_{\text{tot}} = K \otimes \Sigma_t$$

(C.1)

$$\mu_{\text{tot}} = m \otimes \mu_t$$

$$F_{\text{tot}} = I_{N_{\text{tot}}} \otimes F$$

$$S_{\text{tot}} = 1_{N_{\text{tot}}} \otimes S$$

Here, $K$ and $\Sigma_t$ denote data and task covariance matrix, whereas $m$ and $\mu_t$ denote data and task mean, respectively. $F$ and $S$ define the constraint at a single point. $I_{N_{\text{tot}}}$ denotes identity matrix and $1_{N_{\text{tot}}}$ a vector of only ones of dimension $N_{\text{tot}}$. The quantities with the _tot subscript give the quantities that correspond to the general approach from Section 3.2.

Due to the requirement of constant constraint and inter-task dependencies, we also need to pick a constant data mean $m$, with entries $a = \text{const.}$ We introduce the new quantity $S' = \frac{S}{a}$, which is used when constraining the task mean and covariance matrix. With equation (8), we find the following:
\[ D_{\text{tot}} = (F_{\text{tot}} \Sigma_{\text{tot}} F_{\text{tot}}^T)^{-1} F_{\text{tot}} \Sigma_{\text{tot}}^T \]  
(C.2)

\[ = \left( (I_{N_{\text{tot}}} \otimes F)(K \otimes \Sigma_t)(I_{N_{\text{tot}}} \otimes F)^T \right)^{-1} (I_N \otimes F)(K \otimes \Sigma_t)^T \]

\[ = \left( K \otimes (F \Sigma_t F^T) \right)^{-1} (K^T \otimes F \Sigma_t^T) \]

\[ = \left( K^{-1} \otimes (F \Sigma_t F^T)^{-1} \right) (K^T \otimes F \Sigma_t^T) \]

\[ = K^{-1} K^T \otimes (F \Sigma_t F^T)^{-1} F \Sigma_t^T \]

\[ = I_{N_{\text{tot}}} \otimes D \]

\[ A_{\text{tot}} = I_{N_{\text{tot}}} \otimes I_{N_f} - D_{\text{tot}}^T F_{\text{tot}} \]  
(C.3)

\[ = I_{N_{\text{tot}}} \otimes I_{N_f} - (I_{N_{\text{tot}}} \otimes D)^T (I_{N_{\text{tot}}} \otimes F) \]

\[ = I_{N_{\text{tot}}} \otimes I_{N_f} - I_{N_{\text{tot}}} \otimes D^T F \]

\[ = I_{N_{\text{tot}}} \otimes (I_{N_f} - D^T F) \]

\[ = I_{N_{\text{tot}}} \otimes A \]

\[ \mu'_{\text{tot}} = A_{\text{tot}} \mu_{\text{tot}} + D_{\text{tot}}^T S_{\text{tot}} \]  
(C.4)

\[ = (I_{N_{\text{tot}}} \otimes A)(m \otimes \mu_t) + (I_{N_{\text{tot}}} \otimes D)^T (I_{N_{\text{tot}}} \otimes S) \]

\[ = m \otimes A \mu_t + 1_{N_{\text{tot}}} \otimes D^T S \]

\[ = m \otimes (A \mu_t + D^T S') \]

\[ = m \otimes \mu'_t \]

\[ \Sigma'_{\text{tot}} = A_{\text{tot}} \Sigma_{\text{tot}} A_{\text{tot}}^T \]  
(C.5)

\[ = (I_{N_{\text{tot}}} \otimes A)(K \otimes \Sigma_t)(I_{N_{\text{tot}}} \otimes A)^T \]

\[ = K \otimes (A \Sigma_t A^T) \]

\[ = K \otimes \Sigma'_t \]

Hence we have shown that \( \mu'_{\text{tot}} \) and \( \Sigma'_{\text{tot}} \) of the constrained GP factorize into Kronecker products between the data mean and covariance matrix and the constrained task mean and covariance matrix, respectively.

**B.7 Credible intervals**

While standard deviation and variance for the backtransformed outputs \( f \) cannot be recovered via a simple backtransformation of the corresponding
quantities of $f'$, due to the potentially nonlinear and piecewise backtransformation, it is possible to recover credible intervals in this way: for the transformed outputs $f'$, we generate the upper and lower bounds of the $2\sigma$ credible interval; subsequently those bounds can be backtransformed in the same way as we do for the mean of the GP. That means that the posterior of the constrained $f$ can be a bit skewed, i.e. the mean may not lie exactly in the middle between upper and lower credible interval. When auxiliary outputs are involved in the backtransformation, their respective means should be used also when recovering the credible intervals, for the results to be consistent with the posterior means.

B.8 Training procedure

The models have been trained using the Adam optimizer provided by gpytorch. For each experiment, the corresponding learning rate (lr), number of iterations (iter) and (if applicable) scheduler settings are given in Table 7. The scheduler multiplies the learning rate with s-factor after s-steps iterations. The two different scheduler parameters given for the double pendulum correspond to the constrained and the unconstrained GP, respectively.

During the training of all datasets, we checked for errors in the Cholesky decomposition, which can happen when a matrix becomes singular due to numerical errors; when that happened, hyperparameter optimization was restarted with a new random initialization. For the non-square nonlinearity (logsin) experiment, training of the constrained GP proved to be less stable than for the other datasets. To counteract the issue, we tested for two further failure modes of the GP. First, we checked the learned lengthscale of the GP; if it was unreasonably small (smaller than 0.1), the training was repeated. Very small lengthscales typically correspond to the case where the GP learns an almost constant function with spikes towards all of the training points. Secondly, we confirmed that gradient descent had actually converged during training: to this end, we took the loss values over the last 40 iterations and checked, whether the standard deviation was smaller than 0.1. If either of the two checks failed, the training was repeated with newly initialized hyperparameters.

B.9 Computing power available for the experiments

All experiments have been conducted on a system with NVIDIA GTX 1060, 6GB GPU, Intel Core i7 7700-K @ 4.2GHz CPU and 16GB RAM. The creation of average values for one set of parameters as displayed in Tables 5.1-4 typically took between 15 minutes and three hours.
C. Details on simulated datasets

In this section we provide information on how the data used in the different simulation experiments was generated.

C.1 Harmonic oscillator

The data for the harmonic oscillator toy problem was generated from

\[ z(t) = z_0 \sin(\omega_0 t), \quad (C.1a) \]
\[ v(t) = z_0 \omega_0 \cos(\omega_0 t). \quad (C.1b) \]

The energy is given by

\[ E = \frac{k}{2} z(t)^2 + \frac{m}{2} v(t)^2 = \]
\[ = \frac{k}{2} z_0^2 \sin^2(\omega_0 t) + \frac{m}{2} z_0^2 \omega_0^2 \cos^2(\omega_0 t) = \frac{k}{2} z_0^2. \quad (C.3) \]

We have chosen \( E = 0.8\text{ J} \), \( m = 1\text{ kg} \), \( \omega_0 = 1\text{ s}^{-1} \) and it holds that \( k = m\omega_0^2 \) and \( z_0 = \sqrt{2E/k} \).

Training data has been generated by evaluating the function on the equally spaced grid \( t \in \text{linspace}(0,10,20) \) [s]. Subsequently, random noise \( \epsilon \sim N(0, \sigma_n^2) \) was added to the data and output components were omitted at random with probability \( f_d \); the values for \( \sigma_n \) and \( f_d \) are given in Table 5.1 in the main text. Test data has been generated on the grid \( t \in \text{linspace}(-0.1,10,100) \) [s].
C.2 Damped harmonic oscillator

The data for the damped harmonic oscillator was generated from

\[ z(t) = z'_0(t) \sin(\omega t), \]  
\[ v(t) = z'_0(t) \omega \cos(\omega t) - z'_0(t) \frac{b}{2m} \sin(\omega t), \]

where \( z'_0(t) = z_0 \exp\left(\frac{-bt}{2m}\right) \) and \( \omega = \sqrt{\omega_0^2 - \left(\frac{b}{2m}\right)^2} \). The energy is given by

\[ E(t) = \frac{k}{2} z(t)^2 + \frac{m}{2} v(t)^2, \]

which is now time dependent and no longer yields a constant expression.

We have chosen \( E = 0.8 \text{ J}, m = 1 \text{ kg}, \omega_0 = 1 \text{ s}^{-1}, b = 0.1 \text{ kg s}^{-1} \) and it holds that \( k = m\omega_0^2 \) and \( z_0 = \sqrt{2E/k} \).

Training data has been generated by evaluating the function on the equally spaced grid \( t \in \text{linspace}(0,10,20) \text{ [s]} \). Subsequently, random noise \( \epsilon \sim \mathcal{N}(0, \sigma_n^2) \) was added to the data and output components were omitted at random with probability \( f_d \); the values for \( \sigma_n \) and \( f_d \) are given in Table 4. Test data has been generated on the grid \( t \in \text{linspace}(-0.1,10,100) \text{ [s]} \).

C.3 Free fall

The data for the free fall was generated from

\[ z(t) = v_0 t - \frac{g}{2} t^2, \]  
\[ v(t) = v_0 - gt. \]

The energy is given by

\[ E = mgz(t) + \frac{m}{2} v(t)^2 = \frac{m}{2} v_0^2. \]

We have chosen \( E = 200 \text{ J}, m = 1 \text{ kg} \) and it holds that \( v_0 = \sqrt{2E/m} \), and the gravitational acceleration on earth is \( g = 9.81 \text{ m s}^{-2} \). Training data has been generated by evaluating the function on the equally spaced grid \( t \in \text{linspace}(0,6,20) \text{ [s]} \). Subsequently, random noise \( \epsilon \sim \mathcal{N}(0, \sigma_n^2) \) was added to the data and output components were omitted at random with probability \( f_d \); the values for \( \sigma_n \) and \( f_d \) are given in Table 3. To ensure good visibility and learnability, we scaled the data \( y \) with a factor \( a = 20 \): \( y \rightarrow y/a \). Both in Figure 4 and in Table 3, the results are given in terms of the rescaled data (and noise values); the results in terms of the original scale can be obtained by multiplying with \( a \). Test data has been generated on the grid \( t \in \text{linspace}(-0.1,6,100) \text{ [s]} \).
C. Details on simulated datasets

C.4 Non-square nonlinearity

The data for the experiment with non-square nonlinearities was generated from

\[ f_1(x) = 2e^{-5(x-1)^2} + e^{-5(x+1)^2} + 0.2, \]  
\[ f_2(x) = -\frac{x^3}{2}. \]  

Training data has been generated on the equally spaced grid \( x \in \text{linspace}(-1.2,2,20) \).

Subsequently, random noise \( \epsilon \sim \mathcal{N}(0, \sigma_n^2) \) was added to the data and output components were omitted at random with probability \( f_d \); the values for \( \sigma_n \) and \( f_d \) are given in Table 5. Test data has been generated on the grid \( t \in \text{linspace}(-1.2,2,100) \).

C.5 Triangle in the plane

In terms of the parameter \( \alpha \), the trajectory that we used for the triangle in the plane in Section 4.2 is given by

\[ Z_0 = \begin{bmatrix} 4 & 8 & 8.4 \\ 4 & 4 & 6 \end{bmatrix}, \]  
\[ Z_1 = Z_0 + d(\alpha), \]  
\[ Z = R(\alpha)Z_1 + d(\alpha), \]  

where each column of the matrix \( Z \) contains the coordinates of one corner point of the triangle, and where \( d(\alpha) = \frac{1}{2} \cos(2\alpha) \) and \( R(\alpha) \) is a rotation matrix. Subsequently, random noise \( \epsilon \sim \mathcal{N}(0, \sigma_n^2) \) was added to \( Z \); the values for \( \sigma_n \) are given in Table 5.2 in the main text. We then added an auxiliary point of known position \((4, 4)\) to each datapoint, which will be important for the backtransformation:

\[ Z = \begin{bmatrix} z_{1x} & z_{2x} & z_{3x} & 4 \\ z_{1y} & z_{2y} & z_{3y} & 4 \end{bmatrix}. \]  

Following the approach from Salzmann and Urtasun [2010a], we constructed the matrix \( Q = Z^TZ \) and used the upper triangular elements of \( Q \) as transformed outputs for the constrained GP:

\[ y' = [Q_{11}, Q_{12}, Q_{13}, Q_{14}, Q_{22}, Q_{23}, Q_{24}, Q_{33}, Q_{34}, Q_{44}]. \]  

Then the matrix \( F \) and the corresponding vector \( S \) encoding the length
constraints for all the edges of the triangle become

\[
F = \begin{bmatrix}
1 & -2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & -2 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & -2 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix} , \tag{C.13}
\]

\[
S = \begin{bmatrix}
L_{12}^2 & L_{13}^2 & L_{23}^2 & L_{04}^2
\end{bmatrix}^T , \tag{C.14}
\]

where \(L_{ij}\) denote the distances between the points \(i\) and \(j\). The last row of \(F\) corresponds to the constraint on the distance between the auxiliary point and the origin of the coordinate system. Note, that \(Q_{14}, Q_{24}\) and \(Q_{34}\) could in principle be learned separately from the remaining transformed outputs, as they do not enter into any of the constraints and the corresponding columns in (C.13) are zero. Furthermore, \(Q_{44}\) could be omitted from the learning process entirely, as the value is known.

After training the constrained GP, the predicted values \(f’\) are rearranged into the (symmetric) matrix \(\tilde{Q}\), analogously to (C.12). Then the matrix \(\tilde{Z}\) is recovered via a singular value decomposition (SVD) of \(\tilde{Q}\). This decomposition is not unique and the auxiliary point comes into play: we compare the learned with the known position and determine the angle between them, which enables us to rotate the learned coordinates to their true positions.

Training data was generated on the grid \(\alpha \in [0, 5]\), consisting of 20 uniformly spaced points. Subsequently, random noise \(\epsilon \sim N(0, \sigma_n^2)\) was added to the data; the values for \(\sigma_n\) are given in the main text. Test data was generated over the same range \([0, 5]\), although this time with the grid divided into 100 points.

## D Details on the double pendulum dataset

### D.1 Parameters

In Section 4.3 we demonstrated the applicability of our approach to the ‘Double Pendulum Chaotic’ dataset. A description of the dataset can be found in Asseman et al. [2018]; to prevent confusion, we should mention that the blue and the green marker in our paper correspond to the green and the blue marker in Asseman et al. [2018], respectively (i.e. the colors have been exchanged). The lengths of the two pendula are given as \(l_b = 91\, \text{mm}\) and \(l_g = 70\, \text{mm}\), where the suffix \(b\) refers to the pendulum with blue marker and \(g\) to the one with green marker. However, in order to calculate the energy (up to a constant factor), knowledge of the masses, or at least of the ratio \(m_b/m_g\) is required. From information given by the authors of the paper and the manufacturer of the double pendulum, together with some experimentation
Figure 8: Comparison of the energy of the double pendulum for different frame rates of the camera. It is apparent that a frame rate of 400 Hz is incompatible with the principle of energy conservation; while the energy is decreasing in the long term due to friction, it should never increase. No choice of mass ratio $m_b/m_g$ was able to resolve this issue. On the other hand, a frame rate of 500 Hz together with the mass ratio $m_b/m_g = 6.5$ is compatible with energy conservation, within the bounds of error.

of our own we estimated this ratio as $m_b/m_g \approx 6.5$. Note that in our description of the double pendulum (11), we made the assumption that it consists of two point masses, which is only approximately true.

Another quantity of interest is the frame rate of the camera that was used to create the dataset; it enters into the model when calculating the velocities of the masses. In their paper [Asseman et al., 2018], the authors state a frame rate of 400 Hz. However, our experiments with the dataset and keeping the energy constraint in mind strongly indicate a frame rate of 500 Hz; for 400 Hz there are segments of the motion where the total energy $E$ clearly increases which violates the principle of energy conservation (see Figure 8).

The ‘Double Pendulum Chaotic’ dataset was published under the “Community Data License Agreement - Sharing - Version 1.0”.

D.2 Implementation details

The ‘Double Pendulum Chaotic’ dataset provides data in the form of annotated positions of the masses attached to the ends of the two pendula (together with the position of the top of the apparatus holding the pendulum which does not change and which we therefore omitted). We now have the positions as points on an equally spaced grid; in terms of the camera frame rate $r$ the spacing between two adjacent points is given by $1/r$. To obtain the velocities we numerically take the gradient of the positions on the grid and we receive the data which we use for our GP, with outputs

$$ f = [z_{bx}, z_{by}, z_{gx}, z_{gy}, v_{hx}, v_{hy}, v_{gx}, v_{gy}]^T. \quad \text{(D.1)} $$
To obtain positions and velocities with comparable absolute values, which enhances the performance of the GP and which makes the quantities easier to compare in plots, we scaled positions by a factor of 20 and velocities by a factor of $\sqrt{10}$; the time $t$ was scaled by a factor of 5.

As outlined in Section 4.3, we obtain training data, to be used during hyperparameter optimization, and test data, to evaluate the quality of predictions, by picking a random interval of 200 datapoints from the second half of the trajectories provided by the dataset; out of those we use 15 points as training data and the rest as test data. Note that the value $\hat{E}$ received by evaluating (11) and averaging over the training data will in general be a less accurate estimate than the value of the energy $\hat{E}_0$ received when averaging over all datapoints in the interval, since the average is taken over fewer points in the former case. Hence, when determining the accuracy of the constraint fulfillment, the results in Section 4.3 have been compared to $\hat{E}_0$.

For the double pendulum, we receive the transformed outputs

$$f' = [z_{by}, z_{gy}, v_{bx}^2, v_{by}^2, v_{gx}^2, v_{gy}^2]^T,$$

with corresponding

$$F = [m_b g, m_g g, \frac{m_b}{2}, \frac{m_b}{2}, \frac{m_g}{2}, \frac{m_g}{2}].$$

The auxiliary outputs are

$$f_{aux} = [z_{bx}, z_{gx}, v_{bx}, v_{by}, v_{gx}, v_{gy}]^T;$$

note, that the outputs $z_{bx}$ and $z_{gx}$ are not actually auxiliary outputs, but since they are not involved in the constraint (11) (i.e. the corresponding entry in $F$ would be zero), they can be learned separately from the constrained outputs, together with the auxiliary outputs. Same as for the harmonic oscillator, we created virtual measurements for $v_{bx}^2, v_{by}^2, v_{gx}^2, v_{gy}^2$ at zero crossings of the auxiliary outputs $v_{bx}, v_{by}, v_{gx}, v_{gy}$.

### E Comparison to Jidling et al. [2017]

In this section we will investigate the parallels between the method of Jidling et al. [2017] and our own. While they only consider homogeneous constraints in the paper, in the context of constant linear sum constraints it is simple to extend the method to affine constraints, as we will see below.

In the approach of Jidling et al. [2017], vectors spanning the nullspace of the constraint $\mathcal{F}$ are used to construct the task covariance matrix. Given a
E. Comparison to Jidling et al. [2017] 111

sum constraint $\mathcal{F}(f) = Ff = S$, and vectors $h_i$ spanning the nullspace (i.e. $Fh_i = 0$), we can define the matrix

$$G = [h_1 h_2 \ldots h_n],$$  

(E.1)

where $n$ denotes the dimension of the nullspace. A suitable task covariance matrix can then be constructed via $k_t = GG^T$, and in order to accommodate the non-zero right hand side of the sum constraint, the task mean $m_t$ is chosen such that $Fm_t = S$. Then we obtain the multivariate Gaussian $\mathcal{N}(m_t, k_t)$, samples of which obey the constraint $\mathcal{F}$. Note that $k_t$ is the projector on the nullspace of the constraint; in Matthews et al. [2017], the relationship between constrained multivariate Gaussian distributions and the nullspace of the corresponding linear operator is discussed. Subsequently, the full mean and covariance matrix of the GP can be constructed according to (4).

To make this more concrete, we consider again the example of the harmonic oscillator from Section 3.1. Here, the constraint is given by $F = [k/2, m/2, 0, 0]$ and $S = E$. Then the corresponding matrix $G$ can be constructed as (the choice of null vectors is not unique)

$$G = \begin{bmatrix}
  \frac{m}{\sqrt{m^2+k^2}} & 0 & 0 \\
  0 & \frac{m}{\sqrt{m^2+k^2}} & 0 \\
  0 & 0 & 1 \\
  0 & 0 & 1
\end{bmatrix}. $$

(E.2)

The task mean and task covariance matrix become

$$k_t = GG^T = \begin{bmatrix}
  \frac{m^2}{m^2+k^2} & \frac{-mk}{m^2+k^2} & 0 & 0 \\
  \frac{-mk}{m^2+k^2} & \frac{k^2}{m^2+k^2} & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 1
\end{bmatrix}, \quad m_t = \begin{bmatrix}
  E \\
  \frac{E}{m} \\
  0 \\
  0
\end{bmatrix}. $$

(E.3)

Now if we approach the problem from the other side, as it turns out, starting with the identity matrix as task covariance matrix and then conditioning it according to (8) leads to the same $k_t$ obtained in (E.3). Here we note one advantage of our approach: it is straightforward to include additional correlations into the task covariance matrix, e.g. correlations between constrained outputs and those not involved in the constraint.

For example, when introducing an additional correlation between tasks one and three we obtain (after setting $m = k = 1$)

$$k_t^0 = \begin{bmatrix}
  1 & 0 & 0.5 & 0 \\
  0 & 1 & 0 & 0 \\
  0.5 & 0 & 1 & 0 \\
  0 & 0 & 0 & 1
\end{bmatrix} \quad \longrightarrow \quad k_t = \begin{bmatrix}
  0.5 & -0.5 & 0.25 & 0 \\
  -0.5 & 0.5 & -0.25 & 0 \\
  0.25 & -0.25 & 0.875 & 0 \\
  0 & 0 & 0 & 1
\end{bmatrix},$$

(E.4)
where $k_0$ and $k_t$ are the unconstrained and the constrained task covariance matrix, respectively. So the constraint alters correlations between outputs involved in the constraint and other outputs. In the approach of Jidling et al. [2017], correlations between the nullspace dimensions could be added by introducing a non-diagonal matrix between $G^T$ and $G$ in (E.3). However, that method would not allow us to introduce arbitrary correlations between the tasks as demonstrated in (E.4). Throughout the work, we have used the structure given in (B.12) as the starting point for our task covariance matrices.
Title
Physics-informed neural networks with unknown measurement noise

Authors
Philipp Pilar and Niklas Wahlström

Edited version of
Physics-informed neural networks with unknown measurement noise

Abstract

Physics-informed neural networks (PINNs) constitute a flexible approach to both finding solutions and identifying parameters of partial differential equations. Most works on the topic assume noiseless data, or data contaminated by weak Gaussian noise. We show that the standard PINN framework breaks down in case of non-Gaussian noise. We give a way of resolving this fundamental issue and we propose to jointly train an energy-based model (EBM) to learn the correct noise distribution. We illustrate the improved performance of our approach using multiple examples.

Keywords: physics-informed neural networks, energy-based models, non-Gaussian noise, system identification

1 Introduction

While the idea of using neural networks to solve partial differential equations (PDEs) dates back to the work of Lagaris et al. [1998], the field has received renewed attention [Cuomo et al., 2022, Karniadakis et al., 2021, Markidis, 2021, Blechschmidt and Ernst, 2021] due to the seminal work of Raissi et al. [2019], in which they introduced physics-informed neural networks (PINNs). Both the forward problem, where the solution to a PDE is learned given the boundary conditions, as well as the inverse problem, where parameters of the PDE are to be inferred from measurements, can be solved with the PINN approach.

A multitude of applications for PINNs in science have already been considered: Cai et al. [2021a] review the application of PINNs to fluid mechanics, Cai et al. [2021b] review their application to heat transfer problems,
Yazdani et al. [2020] utilize PINNs to infer parameters for systems of differential equations in systems biology, Mao et al. [2020] investigate the applicability of PINNs to high-speed flows, and Sahli Costabal et al. [2020] utilize PINNs to take into account wave propagation dynamics in cardiac activation mapping.

The main advantage of PINNs over traditional solvers lies in their flexibility, especially when considering the inverse problem [Karniadakis et al., 2021]: as neural networks, they have the capacity for universal function approximation, they are mesh-free, and they can directly be applied to very different kinds of PDEs, without the need to use a custom solver. When dealing with the inverse problem, parameters are learned from data and the question as to the effect of noisy data on the quality of the estimates arises naturally. However, most of the existing work on PINNs assumes either noiseless data, or data contaminated with weak Gaussian noise. While some research has been done on the effects of noisy data in PINN training (see Section 2), they still consider either Gaussian noise or are focused on uncertainty quantification.

In this work, we consider the inverse problem for the case of measurements contaminated by non-Gaussian noise of unknown form. The least-squares loss, which is commonly employed as data loss in PINNs, is known to perform poorly in this case [Constable, 1988, Akkaya and Tiku, 2008]. We give a way of mitigating this issue by suitably modelling the noise distribution. A high-level illustration of our method is given in Fig. 1: we

Figure 1: The PINN-EBM approach: the inputs $t$ are passed through the PINN to obtain the PINN predictions $\hat{x}$. The differential operator $\mathcal{F}(\lambda)$ is then applied to $\hat{x}$ to obtain the PDE residuals, which are subsequently utilized to calculate the PDE loss $L_{\text{PDE}}$. At the same time, the residuals between the noisy measurements $y$ and $\hat{x}$ are formed, serving as noise estimates. The EBM is then trained on these estimates in order to learn the noise PDF, which can in turn be utilized to compute the likelihood of the measurements serving as data loss $L_{\text{data}}^{\text{EBM}}$. Finally, the two loss terms are combined to form the total loss $L_{\text{tot}}$. Both PINN and EBM, as well as the PDE parameters $\lambda$, can be trained by backpropagating $L_{\text{tot}}$. 
employ an energy-based model (EBM) to learn the noise probability density function (PDF) jointly with the PINN. This PDF is then utilized to estimate the likelihood of the measurements under our model, which in turn serves as data loss.

2 Related work

Some research has been done on PINNs in case of noisy measurements, although typically only Gaussian noise is considered. In Yang et al. [2021], the framework of Bayesian neural networks [Goan and Fookes, 2020] is combined with the PINN framework, in order to obtain uncertainty estimates when training PINNs on noisy data. Bajaj et al. [2021] introduce the GP-smoothed PINN, where a Gaussian process (GP) [Rasmussen and Williams, 2006] is utilized to ameliorate noisy initial value data and make the PINN training more robust in this situation. In Chen et al. [2021], the PINN-SR method is introduced which can be employed to robustly determine governing equations from scarce and noisy data. In contrast to these works, we give a way of taking into account unknown, non-Gaussian noise in the PINN and provide a training procedure for this case.

While EBMs are commonly employed for classification [LeCun et al., 2006] and image generation [Du and Mordatch, 2019], they have also been successfully applied to regression problems; applications include object detection [Gustafsson et al., 2020a] and visual tracking [Danelljan et al., 2020]. In Gustafsson et al. [2020b], different methods for training EBMs for regression are discussed. In our work we also consider regression tasks and to the best of our knowledge, our paper is the first to combine EBMs and PINNs. The standard EBM approach to regression would not take the physical knowledge in form of the differential equation into account.

3 Background

In this section we give a brief introduction to the two distinct methods which we will combine in our work: physics-informed neural networks and energy-based models.

3.1 Physics-informed neural networks (PINNs)

The PINN-framework [Raissi et al., 2019] can be used as an alternate approach to solving PDEs, other than the standard, mesh-based solvers. The objective is to numerically determine the solution to a differential equation \( \mathcal{F}x(t) = 0 \), where \( \mathcal{F} \) denotes the differential operator defining the PDE, \( x(t) \)
the solution to the differential equation, and \( t \) the input; both \( x \) and \( t \) can be multidimensional. In the PINN approach, we now employ a neural network to parameterize the numerical solution \( \hat{x}(t) = \hat{x}(t|\theta_{\text{PINN}}) \) of the PDE, where \( \theta_{\text{PINN}} \) denotes the weights of the PINN which are to be optimized.

In this paper, we consider the inverse problem: we are given a dataset \( D_d = \{ t_d, y_d \} \) of \( N_d \) (noisy) measurements of the PINN solution in addition to the parametric form of the differential operator \( F(\lambda) \), which may contain unknown parameters \( \lambda \). Furthermore, we choose another set of \( N_c \) so-called collocation points \( D_c = \{ t_c \} \). They can lie at arbitrary points in the input domain of the PDE and are used to take into account the PDE constraint. As long as collocation points are placed in the areas of interest, they can enable the PINN to extrapolate also to areas without measurements.

When training the PINN, two losses enter into the loss function: the data loss, \( L_{\text{data}} \), evaluating the fit of the PINN prediction with the data, and the PDE loss, \( L_{\text{PDE}} \), a measure of the fulfillment of the PDE by the PINN solution. In the standard PINN, the least-squares loss is used as data loss,

\[
L_{\text{data}}(\hat{x}, \{ t_d, y_d \}_{\text{mb}}) = \frac{1}{N_d'} \sum_{i=1}^{N_d'} (\hat{x}(t_d^i) - y_d^i)^2,
\]

and the squares of the PDE residuals, \( f(t) = F(\lambda)\hat{x}(t) - 0 \), at the collocation points serve as PDE loss,

\[
L_{\text{PDE}}(F, \hat{x}, \{ t_c \}_{\text{mb}}) = \frac{1}{N_c'} \sum_{i=1}^{N_c'} f(t_c^i)^2.
\]

Here, \( N_d' \) and \( N_c' \) denote the number of data points \( t_d \) and collocation points \( t_c \), respectively, in the current mini-batch (mb). The PINN is then trained by minimizing the total loss \( L_{\text{tot}} = L_{\text{data}} + \omega L_{\text{PDE}} \) with respect to the parameters \( \theta_{\text{PINN}} \) and \( \lambda \). Unless stated otherwise, the weighting factor \( \omega = 1 \). Utilizing this loss function, the PINN is optimized via some variation of gradient descent.

### 3.2 Energy-based models (EBMs)

EBMs constitute a powerful method of learning probability densities from data [LeCun et al., 2006]. They are frequently employed for image generation and modeling [Du and Mordatch, 2019, Nijkamp et al., 2020], and have successfully been applied to the domain of regression (see Section 2). The EBM can retain all of the flexibility of a neural network, via the following
parametrization:

\[ p(y|t, \hat{h}) = \frac{e^{\hat{h}(t,y)}}{Z(t, \hat{h})}, \quad Z(t, \hat{h}) = \int e^{\hat{h}(t,y')} \, dy', \]  

(3)

where \( \hat{h}(t, y) = \hat{h}(t, y|\theta_{\text{EBM}}) \) is the (scalar) output of the neural network with weights \( \theta_{\text{EBM}} \).

This expressive capacity, however, comes at the cost of a more complicated training procedure, since the partition function \( Z(t, \hat{h}) \) will typically be analytically intractable. In case of a high dimensional \( y \), it will also be expensive or infeasible numerically; Monte Carlo methods are often employed to approximate the intractable integral in (3). Various ways of training EBMs for regression are given in Gustafsson et al. [2020b].

In this paper, we will choose the approach of directly minimizing the negative log-likelihood (NLL), which can be written as

\[ \text{NLL}(\{t_d, y_d\}, \hat{h}) = -\log \left( \prod_{i=1}^{N_d} p(y_{id}|t_{id}, \hat{h}) \right) = \sum_{i=1}^{N_d} \log Z(t_{id}, \hat{h}) - \hat{h}(t_{id}, y_{id}). \]  

(4)

In our case, the evaluation of the partition function to high accuracy remains tractable by utilizing numerical integration, since the noise distribution remains one-dimensional.

4 Problem formulation

We consider the following setup, consisting of the differential equation and a measurement equation:

\[ \mathcal{F}(\lambda)x(t) = 0, \quad y(t) = x(t) + \epsilon, \]  

(5)

where the parametric form of the differential operator \( \mathcal{F}(\lambda) \) is given, as well as \( N_d \) measurements \( y(t) \) of the corresponding solution \( x(t) \) contaminated by homogeneous measurement noise \( \epsilon \). In this work, we aim to combine the PINN and the EBM framework in order to solve the inverse problem in case of measurements contaminated by non-Gaussian and non-zero mean noise.

4.1 PINNs in case of non-zero mean noise

To see why noise with non-zero mean is problematic in the standard PINN framework, consider the loss function \( \mathcal{L}_{\text{tot}} = \mathcal{L}_{\text{data}} + \omega \mathcal{L}_{\text{PDE}} \) (compare Section 3.1):

\[ \mathcal{L}_{\text{tot}} = \frac{1}{N'_d} \sum_{i=1}^{N'_d} (\hat{x}(t_{id}^i) - y_{id}^i)^2 + \frac{\omega}{N'_c} \sum_{i=1}^{N'_c} f(t_{ci}^i)^2. \]  

(6)
A major challenge when training PINNs, even in case of Gaussian noise, lies in the fact that data and PDE loss counteract each other, since the first term would encourage overfitting towards each data point, whereas the second term would push the solution towards fulfilling the PDE.

Let us first consider the case of a zero-mean noise distribution in the limit of infinite data generated according to (5), and assume that the PINN has sufficient capacity to represent the solution accurately. Then both loss terms would be minimal for \( \hat{x}(t) = x(t) \): in the limit, the data loss would become \( \lim_{N_d \to \infty} L_{\text{data}} = \mathbb{E}[(\hat{x} - x - \epsilon)^2] = \mathbb{E}[(\hat{x} - x)^2] + \text{Var}_\epsilon \), and hence \( \hat{x} = x \).

The PDE loss would then obviously also be minimal (more precisely zero) for \( \hat{x} = x \), due to (5).

For noise with non-zero mean, however, the problem of mismatch between the two loss terms is not simply an effect of finite data, which could be treated via an adequate regularization procedure, but has deeper roots. Considering again the limit of infinite data, we have \( \lim_{N_d \to \infty} L_{\text{data}} = \mathbb{E}[(\hat{x} - x - \epsilon)^2] = \mathbb{E}[(\hat{x} - x - \mu_\epsilon)^2] + \text{Var}_\epsilon \), and the minimizer of the data loss becomes \( \hat{x} = x + \mu_\epsilon \), where \( \mu_\epsilon \) denotes the mean of the noise distribution. For the PDE loss, on the other hand, the minimizer would remain \( \hat{x} = x \) (assuming that the correct parameters \( \lambda \) of \( F(\lambda) \) are known).

The two losses are now counteracting each other and the optimization procedure will produce some compromise between them, even when the parameters \( \lambda \) of the PDE are known exactly a priori. In the case where the parameters of \( F(\lambda) \) are unknown, the optimization procedure will tend to converge to parameter values which result in a lower least-squares loss than the correct ones might.

### 4.2 Reconciling the losses

A simple way of restoring consistency between the two losses in the limit of infinite data is to add an offset parameter \( \theta_0 \) to the PINN prediction \( \hat{x} \) in the data loss term, which is supposed to learn the bias in the noise term. The data loss then reads \( \lim_{N_d \to \infty} L_{\text{data}} = \mathbb{E}[(\hat{x} + \theta_0 - x - \epsilon)^2] = \mathbb{E}[(\hat{x} + \theta_0 - x - \mu_\epsilon)^2] + \text{Var}_\epsilon \) and the optimum \( \hat{x} = x, \theta_0 = \mu_\epsilon \) would be compatible with the PDE loss. Hence the optimization procedure could converge to the correct solution, in principle.

However, since the maximum likelihood estimator (when using the correct likelihood) is asymptotically optimal [Wasserman, 2004], this data loss still does not appear to be the best option. Furthermore, in the practical case of finite data, outliers may have an outsized effect when using the least-squares loss. Hence, a way of taking the non-Gaussianity of the noise into account explicitly would likely further improve the speed of optimization as
well as the final learning outcome. In the next section, we demonstrate how EBMs can be used for this purpose.

4.3 Using EBMs to learn the noise distribution

In order to also take the shape of non-Gaussian noise into account, we can choose to use a different data loss in (6). A natural choice would be the log-likelihood of the measurements, given the noise distribution. In case of Gaussian noise, this would again result in a least-squares loss term, plus a constant. However, since in our problem setup we do not know the form of the noise a priori, we need to determine the shape of the noise distribution jointly with the PINN solution.

To this end, we train an EBM and employ it to obtain the negative log-likelihood of the data, given both our models. For the training procedure, we then utilize the following loss function:

\[ L_{\text{tot}} = L_{\text{EBM data}}(\{y_d - \hat{x}(t_d)\}_{mb}, \hat{h}|\theta_{\text{PINN}}, \theta_{\text{EBM}}) + \omega L_{\text{PDE}}(F, \hat{x}, \{t_c\}_{mb}|\theta_{\text{PINN}}, \lambda), \]

where

\[ L_{\text{EBM data}}(\cdot, \hat{h}) = \frac{1}{N_d} \text{NLL}(\cdot, \hat{h}) = \log Z(\hat{h}) - \frac{1}{N_d} \sum_{i=1}^{N_d} \hat{h}(\cdot), \]

utilizing (4); note that \( \hat{h} \) and hence the NLL are now independent of \( t \), because we consider homogeneous noise. Since we do not know the actual magnitude of the noise at each data point, we use the residuals \( y_d - \hat{x}(t_d) \) between the current PINN prediction and the measurements as our best guess. These estimates of the noise values then serve as training data for the EBM. In other words, we employ an unconditional EBM to model the PDF of the residuals. With the loss (7), both models are trained jointly until convergence.

4.4 Training PINN and EBM jointly

In Algorithm 1, the training procedure is summarized. Since both PINN and EBM need to be trained in parallel, the optimization procedure can be very challenging. In our experiments, it proved advantageous to start with the standard PINN loss in order to obtain a solution close to the data, and to only subsequently, after \( i_{\text{EBM}} \) iterations, switch to the EBM loss, in order to fine-tune the solution. Before the EBM loss is used for the first time, we initialize the EBM by training it for \( N_{\text{EBM}} \) iterations on the current noise estimates \( y_d - \hat{x}(t_d) \), while keeping all parameters except \( \theta_{\text{EBM}} \) fixed. Starting directly with the EBM loss also worked, but it often took significantly longer for the algorithm to converge.

Initializing the EBM only later during the training process is advantageous for two reasons: firstly, the least-squares loss is still a sensible first guess,
Algorithm 1: Training the PINN-EBM

**Input**: PINN $\hat{x}(\cdot|\theta_{\text{PINN}})$, EBM $\hat{h}(\cdot|\theta_{\text{EBM}})$, data points $\{t_d, y_d\}$, collocation points $\{t_c\}$, differential operator $F(\lambda)$, weighting factor $\omega$

**Output**: optimized $\theta_{\text{PINN}}, \theta_{\text{EBM}}, \lambda$

while *Training* do

| Draw a minibatch of data points $\{t_d, y_d\}_{\text{mb}}$ and of collocation points $\{t_c\}_{\text{mb}}$
| if $i < i_{\text{ebm}}$ then
| | Calculate data loss $L_{\text{data}}(\hat{x}, \{t_d, y_d\}_{\text{mb}})$ according to (1)
| else
| | if $i = i_{\text{EBM}}$ then initialize EBM
| | Calculate data loss $L_{\text{data}}^{\text{EBM}}\left(\{y_d - \hat{x}(t_d)\}_{\text{mb}}, \hat{h}\right)$ from (7)
| end if
| if $i \geq i_{\text{EBM}}$ then $\omega' = \omega$ else $\omega' = 1$
| Calculate PDE loss $L_{\text{PDE}}(F, \hat{x}, \{t_c\}_{\text{mb}})$ according to (2)
| Compute total loss $L_{\text{tot}} = L_{\text{data}} + \omega' L_{\text{PDE}}$
| Calculate gradient $\nabla_\theta L_{\text{tot}}$ and update $\theta = \{\theta_{\text{PINN}}, \theta_{\text{EBM}}, \lambda\}$

end

so initializing the EBM on the residuals stemming from the pretrained network will let it start out with a reasonable form of the likelihood instead of a random one. Secondly, it will give us a better idea on the range in which the residuals between PINN prediction and data lie, allowing for a better normalization of the inputs to the EBM and hence more efficient training.

5 Experiments

In this section, we compare the performance of the standard PINN from Section 3.1 to that of the PINN with offset parameter and the combination of PINN and EBM at hand of multiple experiments. In the following, we will refer to these models as PINN, PINN-off and PINN-EBM.

In our experiments, we consider homogeneous noise in a variety of shapes. Amongst them are standard Gaussian noise $p(x) = \mathcal{N}(x|0, 2.5^2)$, a uniform distribution $p(x) = \mathcal{U}[0, 10]$ and a Gaussian mixture of the form $p(x) = \mathcal{N}(x|0, 2^2) + \mathcal{N}(x|4, 4^2) + \mathcal{N}(x|8, 0.5^2)$. The values given here are scaled to the example of the exponential function in Section 5.1. For other experiments, the shapes of the curves are retained, but the magnitude of the noise values is rescaled by a factor corresponding to the measurement values of the dataset at hand. In Fig. 2, the different noise distributions are depicted, together
5. Experiments

Figure 2: The noise distributions considered in the experiments are depicted. From left to right, we have a Gaussian distribution (G), a uniform distribution (u), a mixture of Gaussians (3G), and the same mixture of Gaussians, shifted to obtain zero mean (3G0). The dashed blue curves give examples of PDFs learned by the EBM during the training process, whereas the black curve gives the true PDF.

with a few examples of the PDFs learned by the EBM during experiments.

When evaluating the performance of our models, the following metrics are considered: the absolute error in the learned values of the PDE parameters (|Δλ|), the root-mean-square error (RMSE) on the validation data, the log-likelihood (logL) of the validation data according to the models, and the square values of the PDE residuals (f²) on the training data. In practice, the log-likelihood is the most relevant performance metric: contrary to the RMSE and |Δλ|, it can also be calculated when the true solution is not known. While f² can also be calculated without knowledge of the true solution, it only measures how well the learned PDE is fulfilled and not necessarily the correct one.

More information on the experiments discussed in this section is given in the appendices: details on the training hyperparameters and the model architectures are given in Appendix B. Learning curves for the experiments are provided in Appendix A. In Appendix C, model performance is investigated as a function of the number of training points available, and as a function of the weighting factor ω in (6) and (7).

5.1 Toy problem

As a first toy problem, we consider the following simple differential equation,

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

where \( \lambda = 0.3 \) and with the exponential function, \( x(t) = e^{\lambda t} \), as solution.

In Fig. 3, the results for an individual run of PINN, PINN-off, and PINN-EBM are compared; blue dots represent training data and red dots validation data. From this plot, it is clear that PINN converges to a wrong
solution, and PINN-off gives only a slightly better prediction, whereas the PINN-EBM result is very close to the true curve.

In Table 1, performance metrics of the different models as obtained for different noise forms are given. The results have been averaged over ten runs with different realizations of the measurement noise.

We start by discussing the results for Gaussian mixture noise (3G). It is evident that PINN performs poorly on all metrics. While PINN-off learns the correct parameter $\lambda$ better on average, the variance of its predictions is very high, thereby limiting its use. The PINN-EBM estimate of the parameter is significantly less variable than that of PINN-off, and is very close to the correct value of $\lambda$. In accordance with the quality of the parameter estimates, PINN-EBM clearly outperforms both PINN and PINN-off when considering the RMSE and logL; it achieves the lowest RMSE as well as the highest logL on the validation data.

When considering $f^2$, PINN needs to trade off accuracy in this metric with minimizing the data loss (1), due to the inconsistency between the losses discussed in 4.1. PINN-EBM, on the other hand, does not suffer from this issue and achieves the best fit with the learned PDE.

Three other noise forms are considered in the table: for the uniform noise distribution, the results are very similar to the mixture of Gaussians. In case of Gaussian noise, PINN performs best, which makes sense since it is implicitly using the correct likelihood. PINN-EBM, on the other hand, performs a bit worse as there is some overfit to the noise in the data, compare Fig. 2.

For Gaussian mixture noise with zero mean, where the PDE and data loss are no longer misaligned for the PINN, PINN-EBM still outperforms PINN, although with a lesser margin. This shows that PINN-EBM can successfully
5. Experiments

<table>
<thead>
<tr>
<th>noise</th>
<th>PINN-EBM</th>
<th>PINN-off</th>
<th>PINN</th>
</tr>
</thead>
<tbody>
<tr>
<td>3G</td>
<td>1.22±1.12</td>
<td>3.96±3.62</td>
<td>10.22±1.47</td>
</tr>
<tr>
<td>RMSE</td>
<td>0.29±0.29</td>
<td>1.07±0.87</td>
<td>4.01±0.28</td>
</tr>
<tr>
<td>logL</td>
<td>-3.48±0.97</td>
<td>-7.85±9.0</td>
<td>-8.88±2.75</td>
</tr>
<tr>
<td>100f^2</td>
<td>0.51±0.3</td>
<td>30.85±11.51</td>
<td>48.83±17.07</td>
</tr>
</tbody>
</table>

| u     | 1.27±0.33 | 3.25±2.45 | 12.29±0.66 |
| RMSE  | 0.19±0.12 | 0.79±0.53 | 5.11±0.17  |
| logL  | -4.75±1.14 | -6.49±6.77 | -19.06±3.82 |
| 100f^2 | 0.72±0.29 | 16.25±4.75 | 37.94±12.56 |

| G     | 2.45±1.78 | 2.02±1.34 | 1.08±0.91 |
| RMSE  | 0.6±0.43  | 0.46±0.27 | 0.29±0.07  |
| logL  | -5.13±2.47 | -5.05±3.55 | -4.15±2.48 |
| 100f^2 | 0.42±0.32 | 10.82±5.97 | 11.72±6.58 |

| 3G0   | 1.17±1.03 | 5.28±3.23 | 2.21±2.03 |
| RMSE  | 0.23±0.19 | 1.12±0.69 | 0.48±0.15  |
| logL  | -3.73±1.29 | -9.76±11.15 | -5.44±3.99 |
| 100f^2 | 0.61±0.41 | 32.2±15.05 | 34.02±15.53 |

Table 1: Results for the exponential differential equation (8) in case of different noise forms (compare Fig. 2). The entries in the table are averages obtained over 10 runs plus-or-minus one standard deviation. Bold font highlights best performance.

mitigate the effects of non-Gaussian noise and significantly improves the quality of the parameter estimates, as well as that of the solution to the regression task. While PINN-off manages to improve upon PINN in the case of non-zero mean noise, the high variance in its predictions makes it less reliable than PINN-EBM.

5.2 Bessel equation

To test the framework on a more complicated differential equation, we employ the Bessel equation, a second-order differential equation [Niedziela, 2008, Bowman, 2012],

\[(\lambda t)^2 \dddot{x} + \lambda t \ddot{x} + ((\lambda t)^2 - \nu^2) x = 0,\]

(9)

where we pick \(\nu = 1\) and where we have introduced the parameter \(\lambda = 0.7\), which is to be estimated by the PINN. Results for one run of this experiment are shown in Fig. 4. When comparing the results obtained here (see Table
Figure 4: The Bessel equation (9) with Gaussian mixture noise. Results for one individual run, where the blue dots represent training data and the red dots validation data.

<table>
<thead>
<tr>
<th>noise</th>
<th>PINN-EBM</th>
<th>PINN-off</th>
<th>PINN</th>
</tr>
</thead>
<tbody>
<tr>
<td>3G 100</td>
<td>Δλ</td>
<td></td>
<td>0.27±0.22</td>
</tr>
<tr>
<td>RMSE</td>
<td>0.0±0.0</td>
<td>0.03±0.01</td>
<td>0.04±0.01</td>
</tr>
<tr>
<td>logL</td>
<td>0.94±0.13</td>
<td>0.63±0.08</td>
<td>0.19±0.32</td>
</tr>
<tr>
<td>100f²</td>
<td>0.63±0.14</td>
<td>0.39±0.4</td>
<td><strong>0.19±0.05</strong></td>
</tr>
</tbody>
</table>

Table 2: Results for the Bessel differential equation (8) in case of mixture of Gaussians (3G) noise. The entries in the table are averages obtained over 10 runs plus-or-minus one standard deviation. Bold font highlights best performance.

2) to those from the previous section, we note the following: PINN-EBM again outperforms both PINN and PINN-off on all metrics, and PINN-off performs better than PINN. For this example, the variance of the results for both PINN and PINN-off is smaller than in the previous experiment. One reason for this can be found in the $t^2$ term in (9), which can lead to large values in the PDE loss in (6), thereby putting more weight on PDE fulfillment than on matching the data to a Gaussian.

Another reason why PINN and PINN-off perform better here than for the toy example lies in the shape of the PDE solution: in the previous example from Section 5.1, rather inaccurate estimates of the parameter $\lambda$ could still have been compatible with Gaussian noise of variable strength. Here, on the other hand, the parameter determines the frequency of the oscillation of the curve, leaving less wiggle room for compromise.
5.3 Navier-Stokes equations

For our next experiment, we consider the example of incompressible flow past a circular cylinder and use the dataset from Raissi et al. [2019], to which we add some noise. This setup can be described by the 2D Navier-Stokes equations,

\[ f = u_t + \lambda_1 (uu_x + vu_y) + p_x - \lambda_2 (u_{xx} + u_{yy}) = 0, \quad (10a) \]
\[ g = v_t + \lambda_1 (uv_x + vv_y) + p_y - \lambda_2 (v_{xx} + v_{yy}) = 0, \quad (10b) \]

where \( u \) and \( v \) denote the components of the velocity field and \( p \) the pressure; the subscripts indicate derivatives with respect to the different dimensions of the inputs \( t = (t, x, y) \). Here, the PINN does not output \( u \) and \( v \) directly, but instead models the potential \( \psi \) and the pressure \( p \); then \( u = \psi_y \) and \( v = -\psi_x \). The true values of the parameters are \( \lambda_1 = 1 \) and \( \lambda_2 = 0.01 \).

The results are given in Table 3. It is apparent that PINN-EBM learns \( \lambda_1 \) better than PINN and PINN-off whereas the difference is small for \( \lambda_2 \). PINN-EBM also performs better in terms of RMSE and log-likelihood. In this experiment, we have chosen \( \omega = 50 \) for PINN-EBM since the log-likelihood on the validation data increased when the PDE loss was weighted more strongly (see Fig. 5). For PINN and PINN-off, on the other hand, increasing the parameter did not lead to a notable change in the log-likelihood.

6 Conclusions and future work

In this paper, we demonstrated that the standard PINN fails in case of non-zero mean noise and we proposed the PINN-EBM to resolve this problem;
utilizing an EBM to learn the noise distribution allows the PINN to produce good results also in case of non-zero mean and non-Gaussian noise. Using several examples, ranging from a simple toy problem to the complex Navier-Stokes equations, we demonstrated the capabilities of our method and showed that it outperforms the standard PINN by a significant margin in case of non-Gaussian noise.

In the future, it would be interesting to investigate combining the PINN-EBM with other improvements to the PINN framework, such as adaptive weighting schemes [McClenny and Braga-Neto, 2020], scheduling approaches [Krishnapriyan et al., 2021, Wang et al., 2022] or additional loss terms [Yu et al., 2022]. In principle, our method could also be applied to the problem considered in Bajaj et al. [2021] (see also Section 2): where the standard GP presumably would fail here as well in case of non-Gaussian noise, a representation for the initial conditions could potentially also be learned via PINN-EBM.

**Acknowledgements**

The work is 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*. We would also like to thank Fredrik K. Gustafsson for valuable feedback.
Bibliography


A Learning curves

In Figures 6-8, learning curves for the experiments discussed in Sections 5.1-5.3, with mixture of Gaussians (3G) noise, are provided. From the curves, it becomes apparent that PINN-EBM typically converges quickly to the correct solution, after the EBM has been initialized. In case of the exponential differential equation, PINN and PINN-off also converge fast, albeit towards often significantly less accurate solutions. For the Bessel function, both of them converge towards values close to the correct solution but do so only very slowly.

Figure 6: Learning curves for the exponential differential equation (Section 5.1) with Gaussian mixture noise.

B Implementation details

The datasets employed in Sections 5.1 and 5.2 contained 200 training points, 50 validation points and 2000 collocation points. For the Navier-Stokes example in Section 5.3, 4000 training points, 1000 validation points and 4000 collocation points were used. The collocation points were generated on a uniform grid. For both PINN and EBM, fully connected neural networks with tanh activation function were used, with 4 layers of width 40 in case of

\(^2\)The code for the project is available at https://github.com/ppilar/PINN-EBM
Figure 7: Learning curves for the Bessel equation (Section 5.2) with Gaussian mixture noise.

Figure 8: Learning curves for the Navier Stokes equations (Section 5.3) with Gaussian mixture noise.
the former, and 3 layers of width 5 for the latter. For the EBM, a dropout layer with factor 0.5 was inserted before the last layer. Both inputs and outputs of the networks were normalized to their expected ranges. In case of the Navier-Stokes experiment, 5 layers of width 30 were used for the PINN. The EBM was initialized after \( i_{EBM} = 4000 \) iterations, except for the Navier-Stokes example, where \( i_{EBM} = 10000 \) was chosen. The Adam optimizer with learning rate of \( 2e^{-3} \) was used for both PINN and EBM. The batch size for data points was 200 and the batch size for collocation points was 100. Training the PINN-EBM typically took 30-50% longer than the standard PINN.

C Further experiments

In this appendix, we consider further experiments to assess the impact of the number of training points \( N_{train} \) and the weighting factor \( \omega \) on the model predictions.

C.1 Varying the number of training points

In Fig. 9, we investigate the impact of the number of training points available at the example of the exponential differential equation (8). Starting with a small amount of training points \( N_{train} = 20 \), all of the models perform similarly. As the number of training points increases, the model performances improve as well, although the improvement beyond \( N_{train} = 50 \) seems negligible for PINN and PINN-off. It even appears that PINN-off becomes slightly worse again for larger values \( N_{train} = 20 \); a possible explanation for this (and the volatility of PINN-off in general) may be that outliers in the data have an outsized effect on the offset parameter \( \theta_0 \). The advantage of more training data is most pronounced for PINN-EBM. This makes sense as more training data will enable the EBM to learn a more accurate representation of the noise distribution. The results for smaller \( N_{train} \) show that the EBM can learn reasonable noise distributions also for smaller data sets; thus, PINN-EBM does not depend on an outsized amount of training data to perform well.

C.2 Varying the weighting factor for the PDE loss

In Fig. 10, results for different values of the weighting factor \( \omega \) in (6) and (7) are depicted. In case of PINN-EBM, the training starts with \( \omega = 1 \) and the larger value of \( \omega \) is only used as soon as the EBM has been initialized (compare Algorithm 1). The main questions that this investigation is supposed to answer are the following: do a small value of the PDE residuals \( f^2 \)
correspond to correct solutions? Is there an optimal value of $\omega$ and how can it be determined?

To answer the first question, it is apparent that $f^2$ becomes smaller as $\omega$ increases. However, the corresponding plots of the RMSE and $\Delta \lambda$ show that this does not imply correctness of the results. In case of the Bessel and the Navier Stokes equations, the results can become significantly worse as $\omega$ increases. This also shows that the PDE loss cannot, in general, override the detrimental effects of the non-Gaussian noise for the standard PINN. One may have assumed that the PDE may only be compatible with a very limited range of PDE parameters and hence for the PINN to be pushed towards the correct solution that way; only for PINN-off when applied to the exponential differential equation does this seem to be the place.

To answer the second question, when considering e.g. the plots for the RMSE, it is apparent that no such value $\omega$ exists. Only PINN seems to consistently perform better for small values $\omega$. To determine the best value of $\omega$ for a given case, the log-likelihood on the validation data remains crucial. For PINN and PINN-off, worse predictions do not always translate into lower values of logL, as the example of the Navier Stokes equations shows.
In these cases, it is important to keep an eye on the learned values of \( \lambda \) during training, as values \( \lambda \approx 0 \) can often indicate failure. For the PINN-EBM, however, there is a clear correlation between higher \( \log L \) and better results. Most importantly, the best solution as learned via PINN-EBM consistently results in higher \( \log L \) than PINN and PINN-off for any \( \omega \).
Figure 10: The impact of the weighting factor $\omega$ on model performance is depicted for the different experiments considered in Section 5.
Title
Probabilistic matching of real and generated data statistics in generative adversarial networks

Authors
Philipp Pilar and Niklas Wahlström

Edited version of
Abstract

Generative adversarial networks constitute a powerful approach to generative modeling. While generated samples often are indistinguishable from real data, there is no guarantee that they will follow the true data distribution. In this work, we propose a method to ensure that the distributions of certain generated data statistics coincide with the respective distributions of the real data. In order to achieve this, we add a Kullback-Leibler term to the generator loss function: the KL divergence is taken between the true distributions as represented by a conditional energy-based model, and the corresponding generated distributions obtained from minibatch values at each iteration. We evaluate the method on a synthetic dataset and two real-world datasets and demonstrate improved performance of our method.

1 Introduction

Generative adversarial networks (GANs) [Goodfellow et al., 2014] comprise a generator and a discriminator network trained adversarially until the generator manages to produce samples realistic enough to fool the discriminator. Ever since, GANs have become a popular tool for generative modeling [Hong et al., 2019, Gui et al., 2021].

While the framework is generally applicable, it is probably best known for its successes in image generation [Reed et al., 2016, Isola et al., 2017, Ledig et al., 2017, Mathieu et al., 2016]. Although GANs have proven powerful, challenges such as mode collapse and non-convergence still re-
main [Saxena and Cao, 2021]. In general, the GAN might produce realistic examples, however they may only stem from a limited subspace of the true data distribution. For scientific applications in particular, such as in cosmology [Rodriguez et al., 2018, Villaescusa-Navarro et al., 2021] or high-energy physics [Alanazi et al., 2021, Paganini et al., 2018], where GANs may serve as surrogate models for expensive but highly accurate numerical simulations, having a good match between the distributions is essential [Kansal et al., 2023].

In this work, we focus on matching properties of the generated distribution with those of the real data distribution, with scientific datasets in mind. In particular, we consider statistics of the dataset such as the power spectrum, and match the distributions of the individual components. We incorporate these requirements in the form of probabilistic constraints, since it is not properties of individual samples that are enforced, but collective characteristics of the dataset.

The main ingredients of our approach are the following: we approximate the distribution of the real data in an efficient way by utilizing an energy-based model (EBM) [LeCun et al., 2006], and the generated distribution via kernel density estimation (KDE) [Silverman, 1986] from the current mini-batch. In each iteration, the Kullback-Leibler (KL) divergence between true and generated distribution is then calculated and added as an additional term to the generator loss in order to be minimized. That way, we end up with a constrained generated distribution. In the following, we refer to our method as probabilistically constrained GAN (pcGAN).

2 Related work

The field of physics-informed machine learning, where prior knowledge is introduced into the ML model, has been an active area of research in recent years [Karniadakis et al., 2021, Cuomo et al., 2022]. In the context of GANs, two main approaches to including prior knowledge in the model exist.

In the first approach, the constrained values can be fed as additional inputs into the discriminator, such that it can explicitly use constraint fulfillment as a means to distinguish between real and generated data. In Stinis et al. [2019], GANs are employed for interpolation and extrapolation of trajectories following known governing equations. The generated trajectories are constrained to fulfill these equations by passing the constraint residuals as additional inputs to the discriminator; in order to prevent the discriminator from becoming too strong, some noise is added to the residuals of the real data, which might otherwise be very close to zero. When extrapolating, the GAN is applied iteratively from some initial condition; in order to
train stably, it learns to predict the correct trajectory from slightly incorrect positions of the previous step.

In Yang et al. [2019], a physically-informed GAN (PI-GAN) is developed to model groundwater flow. They make use of the same basic idea as physics-informed neural networks [Raissi et al., 2019] and employ automatic differentiation in order to obtain a partial differential equation (PDE) residual on the GAN output, which is in turn fed into the discriminator. By evaluating the GAN prediction at many different points and comparing to an equivalent ensemble of true values of the corresponding physical field, the GAN is constrained to adhere to a stochastic PDE.

In the second approach, prior knowledge may be taken into account via additional loss terms in either discriminator or generator loss: in Khat-tak et al. [2018, 2019], GANs are employed to simulate detector signals for high-energy physics particle showers. Here, physical constraints such as the particle energy are taken into account via additional generator loss terms.

In Yang et al. [2021], the incorporation of imprecise deterministic constraints into the GAN is investigated; e.g. the case where the GAN output is supposed to follow a PDE, but where the PDE parameters are not known accurately could be formulated as an imprecise constraint. In a first step, deterministic constraints can be included by adding the constraint residuals as an additional loss term to the generator loss; they argue that it is better to add such terms to the generator since this strengthens the weaker party in the adversarial game, instead of giving an even larger advantage to the discriminator. In order to make the constraint imprecise, they do not require that the residuals go to zero, but instead only include residuals above a certain threshold value $\epsilon^2$ in the loss.

The work closest in aim to ours is probably that by Wu et al. [2020], where a statistical constrained GAN is introduced. They add an additional term to the generator loss function in order to constrain the covariance structure of the generated data to that of the true data. This additional term is a measure of similarity between the covariances, and they concluded that the Frobenius norm was the best choice for this purpose. They use their method to obtain better solutions for PDE-governed systems.

Similar to Wu et al. [2020], our method also imposes probabilistic constraints via an additional term to the generator loss. However, there are significant differences: firstly, our method does not consider the covariance structure of the dataset in particular, but instead allows to constrain on arbitrary statistics of the data. Secondly, our method matches the distributions of true and generated data statistics via the KL divergence and takes the complete shapes of the distributions into account, instead of just the second-order moments.
3 Background

3.1 Generative adversarial networks

The basic idea of generative adversarial networks (GANs) [Goodfellow et al., 2014] is to train a generator to generate samples of a given distribution and a discriminator (or critic) to distinguish between real and generated data. During the training, both networks are pitted against each other in a minimax game with value function

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

(1)

Here, \(D\) denotes the discriminator, \(G\) the generator, \(x\) samples drawn from the real data and \(z\) randomly generated latent space vectors serving as input to the generator; \(p_{\text{data}}\) and \(p_z\) denote the real data distribution and the latent vector distribution, respectively. Discriminator and generator are then trained alternatingly; in [Goodfellow et al., 2014], it is shown that a stable equilibrium to the minimax problem (1) exists and that the optimal solution lies in the generator producing samples from the true data distribution.

The standard GAN can be very difficult to train and often suffers from mode collapse. In Arjovsky et al. [2017], the Wasserstein GAN (WGAN) was introduced, where they suggest the earth-mover (EM) distance as a new loss for the GAN. They show that the discriminator and generator losses can then be expressed as

\[
\mathcal{L}_D = D(x_{\text{gen}}) - D(x_{\text{true}}), \tag{2a}
\]

\[
\mathcal{L}_G = -D(x_{\text{gen}}), \tag{2b}
\]

under the condition that the discriminator is Lipschitz continuous. Rather crudely, this is enforced by clipping the weights of the discriminator. In the end, the terms in (2) are approximated as expectations over minibatches.

With this loss function, the discriminator can be interpreted as a critic that assigns scores to both true and generated samples. These scores are not constrained to any specific range and there is no need to keep the discriminator from outperforming the generator. Hence, the critic can be trained for multiple steps in each iteration, in turn yielding more informative gradients for the generator. The advantages of the WGAN include improved learning stability as well as meaningful loss curves [Gui et al., 2021].

3.2 Energy-based models

Energy-based models (EBMs) constitute a flexible approach for modeling probability distributions [LeCun et al., 2006]. Our aim is to employ the
EBM as a regression method to model the probability density functions (PDFs) of $N_s$ statistics of the dataset, which we identify via a class index $s \in [1, \ldots, N_s]$. In order to encode multiple PDFs corresponding to different statistics, we employ a conditional EBM (cEBM) [Gustafsson et al., 2020b,a, Danelljan et al., 2020]; typically, using one cEBM to encode all $N_s$ PDFs jointly is significantly more efficient than training a separate (unconditional) EBM for each statistic.

In the conditional EBM framework, the PDFs can be parameterized as

$$p(z_s|\hat{h}, s) = \frac{e^{\hat{h}(z_s,s)}}{Z(\hat{h}, s)}; \quad Z(\hat{h}, s) = \int e^{\hat{h}(z_s',s)}dz'_s,$$

where $\hat{h}(z_s,s) = \hat{h}(z_s,s|\theta_{\text{EBM}})$ is the (scalar) output of a neural network with weights $\theta_{\text{EBM}}$, and where $z_s \in \mathbb{R}$ denotes the value of the constrained statistic $s$; $Z(\hat{h}, s)$ denotes the corresponding partition functions, ensuring that the PDFs are correctly normalized.

Various ways of training EBMs for regression are given in Gustafsson et al. [2020b]. In this work, we choose the approach of directly minimizing the negative log-likelihood (NLL). It can be written as

$$\sum_{s=1}^{N_s} \text{NLL}((z_s, s), \hat{h}) = -\sum_{s=1}^{N_s} \log \left( \prod_{i=1}^{N_d} p(z_{si}|s, \hat{h}) \right)$$

$$= \sum_{s=1}^{N_s} \left( \log Z(\hat{h}, s) - \sum_{i=1}^{N_d} \hat{h}(z_{si}, s) \right),$$

where $\{(z_s, s)\}$ denotes a dataset of $N_d$ pairs of realizations $z_s$ of the statistic $s$. While the evaluation of the partition function can be challenging for higher dimensional PDFs, it remains tractable in our case; since we only need to solve $N_s$ one-dimensional integrals (3), this can easily be done via numerical integration. For more details on the exact training procedure that we use for the conditional EBM, see Appendix A.1.

4 Method

The aim of our method is to consider the distributions of statistics $s$ of the true dataset, such as e.g. components of the power spectrum (compare Appendix B.1), and to ensure that the same statistics, when extracted from the generated data, are distributed equally.

In order to match true ($p_{\text{true}}$) and generated ($p_{\text{gen}}$) distributions, we add
Algorithm 1: High-level algorithm

Step 1: train the conditional EBM on true data (see Algorithm 3 in the appendix);
Step 2: determine the optimal values $f^*_\sigma$ in (7) (see Algorithm 5 in the appendix);
Step 3: train the pcGAN (see Algorithm 2);

Kullback-Leibler (KL) terms to the generator loss (2b):

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

(5)

where $\lambda$ is a weighting factor. The KL divergence is given by

$$\text{KL}(p_{\text{true}}(z_s)||p_{\text{gen}}(z_s)) = \int_{-\infty}^{\infty} p_{\text{true}}(z'_s) \log \frac{p_{\text{true}}(z'_s)}{p_{\text{gen}}(z'_s)} \, dz'_s.$$  

(6)

The KL divergence is asymmetric and we have chosen the forward KL, also known as zero-avoiding, in order to ensure a complete overlap of areas with non-zero probability of the distributions; in case of the reverse, or zero-forcing, KL, the loss term would typically tend to match $p_{\text{gen}}$ to one of the peaks of $p_{\text{true}}$ and hence fail to match the distributions in a way suitable for our purposes.

In order to evaluate (6), we require means of extracting representations for both true and generated PDF. Note that the representation for $p_{\text{true}}$ will need to be determined only once, in advance of the GAN training, since it remains constant. In contrast to $p_{\text{true}}$, the generated distribution changes during GAN training, and hence $p_{\text{gen}}$ also needs to be determined anew after each generator update.

To obtain the PDFs $p_{\text{true}}$, we train a conditional EBM $\tilde{p}_{\text{true}}(z_s|s)$ parameterized by a neural network; background information on EBMs is given in Section 3.2 and details on the training procedure are given in Appendix A.1.

Training an EBM for $p_{\text{gen}}$ at each iteration would be impractical, since it would require a large number of generated samples for training and could take minutes to train. Kernel methods to obtain a smoothened empirical distribution, on the other hand, seem to be a good fit here, as their computational cost is negligible for small datasets such as minibatches. Note that kernel methods would not be suitable as a means to obtain $\tilde{p}_{\text{true}}$ for the entire dataset, as hundreds of thousands or even millions of data points might then need to be considered for each forward and backward pass.
Hence, we chose to employ kernel density estimation (KDE) [Silverman, 1986] with Gaussian kernels, and approximate the generated distribution at each iteration as a mixture of Gaussians, centered around the current minibatch samples. That is, we obtain the approximate generated PDFs as

\[
\tilde{p}_{\text{gen}}(z_s) = \frac{1}{n_{\text{batch}}} \sum_{i=1}^{n_{\text{batch}}} N(z_s; z_{si}, \sigma_{s}),
\]

where \(N\) corresponds to the PDF of the Normal distribution with mean \(\mu = z_{si}\) and standard deviation \(\sigma_{s}\), and where \(n_{\text{batch}}\) denotes the number of samples in the minibatch. For \(\tilde{p}_{\text{gen}}\) to constitute a reasonable approximation, it is important to use big enough batch sizes.

The standard deviations \(\sigma_{s}\) are chosen separately for each constraint \(s\), under the criterion that \(\tilde{p}_{\text{gen}}\) as obtained from minibatches drawn from the true data should have a KL divergence as small as possible with the true distribution. Since the optimal value of \(\sigma_{s}\) would be expected to depend both on the range of values \(z_s\) in the true dataset and the batch size, we parameterized it as \(\sigma_{s}(n_{\text{batch}}) = \text{std}(z_s)/f_{\sigma}(s, n_{\text{batch}})\), where \(n_{\text{batch}}\) denotes the batch size. We denote the optimal values of \(f_{\sigma}\), which result in minimal KL divergence, as \(f_{\sigma}^{\ast}\). A detailed description of how to determine \(f_{\sigma}^{\ast}\) is given in Appendix A.3.

With representation (7) for the generated distribution and the EBM for the true distribution, the one-dimensional integral in (6) can be carried out numerically. When doing this integration, we want to avoid too much weight being put on long tails of \(p_{\text{true}}\); this would be problematic since then very large standard deviations of the Gaussians in \(\tilde{p}_{\text{gen}}\) would be required to cover the tails, which in turn would lead to a bad resolution of the salient features of the PDF, i.e. the regions where most of the data lie. Hence, we solve this issue by cutting off long tails and assigning a probability density of 0 to points where \(\tilde{p}_{\text{true}} < \max(\tilde{p}_{\text{true}})/20\). After doing this, we renormalize \(\tilde{p}_{\text{true}}\) to integrate to 1, and since \(\lim_{x \to 0} x \log x = 0\), the points which have been set to zero also do not cause any trouble in the integral (6). This approximation will work well as long the probability mass located in the long tails is negligible.

When training the model, it has proven effective to consider only a subset of \(N_{\text{kl}}\) constraints during each iteration. This subset is chosen randomly, where the constraints are sampled according to the preceding values of the respective KL divergences, such that constraints which are as of yet not well fulfilled are more likely to be chosen. More information on this weighting scheme is given in Appendix A.2; a high-level overview of the method is given in Algorithm 1 and the pcGAN training is detailed in Algorithm 2. Note that, while our training algorithm is based on the WGAN, (5) is more...
**Algorithm 2:** Training the probabilistically constrained GAN (pcGAN)

**Input:** Untrained $D$ and $G$; trained cEBM $\tilde{p}_{true}$; data $\{x_{true}\}$; $N_{kl}$

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

\[(a_w, w) = (\text{ones}(N_s), \text{ones}(N_s)/N_s) ;\]

**while** Training **do**

\begin{itemize}
  \item **for** $j \in \text{range}(N_d)$ **do**
    \begin{itemize}
      \item Sample $x_{true}$ ;
      \item Generate $x_{gen}$ ;
      \item $L_D = \text{mean}(D(x_{gen}) - D(x_{true}))$ ;
      \item update $D$ ;
      \item clip weights ;
    \end{itemize}
  \item Generate $x_{gen}$ ;
  \item $L_G^0 = -\text{mean}(D(x_{gen}))$ ;
  \item $L_G^c = 0$ ;
  \item **for** $j \in \text{range}(N_{kl})$ **do**
    \begin{itemize}
      \item sample $s \sim \text{Discrete}(\text{range}(N_s); w)$ ;
      \item calculate statistics $\{z_{si}\}_{i=1}^{n_{\text{batch}}}$ from $x_{gen}$ ;
      \item determine $\tilde{p}_{gen}(z_s)$ according to (7) ;
      \item calculate $l_s = \text{KL}(\tilde{p}_{true}(z_s)||\tilde{p}_{gen}(z_s))$ ;
      \item $L_G^c = L_G^c + \lambda l_s$ ;
      \item update $(a_w, w)$ according to Algorithm 4 in the appendix, with input $(a_w, l_s, s)$ ;
    \end{itemize}
  \end{itemize}

\end{itemize}

$L_G = L_G^0 + L_G^c$ ;

update $G$ ;

\end{itemize}

general and can be used for other types of GANs as well.

5 Results

In this section, we present the results obtained with our model. We consider a synthetic example and two real-world datasets from physics. We compare the pcGAN to both the unconstrained WGAN and the statistical constrained GAN from Wu et al. [2020]. Additional information on the experiments is given in Appendix B.
Figure 1: The distributions of three different power spectrum components are compared, where the orange lines show the PDFs as learned by the cEBM. From left to right, the depicted distributions correspond to the real data, the unconstrained WGAN, the pcGAN and the method of Wu et al. [2020].

5.1 Synthetic example

As our synthetic example, we consider a simple superposition of sine waves. Each wave consists 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 we generate measurements for \( t \in \text{linspace}(0, 20, 200) \). In total, we create 100,000 samples of size 200 to serve as training data. We perform the Fourier transform for real-valued inputs for each time series in the dataset and we use the KL divergence between true and generated distribution of the square roots of the power spectrum components (i.e. the absolute values of the Fourier coefficients) as probabilistic constraints when training the GAN; that is, we have 101 separate constraints (compare Appendix B.1).

In Figure 1, results for the different GAN variants are depicted. It is apparent that data generated by the pcGAN matches the true distributions very well; only for the first component does it exhibit a small additional peak due to the somewhat extended tail. The method of Wu et al. [2020] comes in second, managing to cover the correct area of constraint values, but failing to adhere to the precise shape of the PDFs. The unconstrained
Figure 2: Left The constraint fulfillment for the different models is plotted vs the training iterations. The full lines give the KL divergences averaged over all constraints, whereas the shaded regions indicate the range in which the KL divergences lie for the 101 different constraints, from minimum value to maximum value. For the real data, we bounded this range with dashed lines instead. Right The performance of the models is evaluated at hand of different performance metrics of the generated curves: the mean value, the difference between minimum and maximum value of the curves, and the total spectral energy. For the distributions and curves displayed in these plots, 20,000 generated samples have been considered.

WGAN is distinctly worse than the other methods and tends to assign too much weight to the highest peak of the distribution.

In the left plot of Figure 2, average (full lines), minimum and maximum values (shaded area) of constraint fulfillment as measured by the KL divergence on the different constraints are depicted. Also here, it is clear that the pcGAN matches the constraints best, and the method of Wu et al. [2020] performs reasonably; the unconstrained WGAN is orders of magnitude worse. The lines labeled as real data correspond to the KL divergence as obtained from minibatches sampled from the true data; the pcGAN matches them very well. In the right plot, various performance metrics, which are not included in the constraints, are compared for the different models: the mean value, the difference between minimum and maximum value of the curves, and the total spectral energy (see Appendix B.1). The pcGAN performs comparable to Wu et al. [2020], whereas the unconstrained WGAN is significantly worse.

In Figure 3, the search for the best values $f_\sigma^*$ in (7) is illustrated. It is apparent, that there is a clear, batch size-dependent minimum of the KL-divergence for each constraint, with larger batch sizes tending towards larger values of $f_\sigma^*$; this is due to the fact that more samples in the minibatch allow for a more fine-grained approximation of the generated distribution. In the bottom left plot, optimal values of $f_\sigma^*$ are depicted for all components $p[i]$ of the power spectrum. The spike around $i \approx 10$ is the result of some outliers in the values of the power spectrum components; they lead to a high standard
Figure 3: **Top** The three plots on the top depict the dependency of the KL divergence on the factor $f_\sigma$ for different power spectrum components; the curves have been averaged over 50 minibatches sampled from the original dataset. **Middle** The three plots in the middle row depict the distribution of the constraint values together with their EBM representation, as well as curves obtained via (7) from minibatches of different size (not averaged); it is between them that the KL divergences in the top row have been calculated. **Bottom left** Optimal values of the factor $f_\sigma^*$ are depicted for different batch sizes, where the index $i$ gives the respective component of the power spectrum. **Bottom right** Here, the tails were not cut, resulting in a very large standard deviation due to the zero-avoiding order of the KL-divergence in (6).
deviation of the true distribution, which in turn requires a large $f\sigma^*$ in order to obtain small enough standard deviations for the mixture of Gaussians to resolve the narrow peak well. This is one example where cutting the long tails of the distributions makes a difference (compare the right plot in the bottom row).

In the middle row, approximations of the generated distributions as obtained via the minibatches are depicted. It is apparent that the mixtures of Gaussians approximate them reasonably well, with larger batch sizes typically giving better results.

The architectures used for discriminator and generator were inspired by the DCGAN architecture and a batch size of 256 was used for training, together with the ADAM optimizer [Kingma and Ba, 2015]. A detailed description of the architecture, settings for the training procedure, and samples as obtained from the different models can be found in Appendix B.2.

5.2 IceCube-Gen2 radio signals

The IceCube neutrino observatory [Aartsen et al., 2017] and its planned successor IceCube-Gen2 [Aartsen et al., 2021] are located at the South Pole and make use of the huge ice masses present there in order to detect astrophysical high-energy neutrinos. Deep learning methodology has already been employed to extract information such as shower energy or neutrino direction from radio-detector signals [Glaser et al., 2023, Holmberg, 2022]; Holmberg [2022] also investigated the use of GANs to simulate detector signals. We are going to consider the filtered Askaryan radio signals from Holmberg [2022], which were generated using the NuRadioMC code [Glaser et al., 2020] according to the ARZ algorithm [Alvarez-Muniz et al., 2010]. These signals take the form of 1D waveforms and in our experiments we want to focus solely on the shape of these waves, not their absolute amplitudes; this is achieved by normalizing each signal to its maximum absolute value.

We are going to use the pcGAN again to constrain the generated data on the distributions of the minimum and maximum values of the signals. As additional performance metrics, we use the following signal characteristics: the mean absolute values, the distance between minimum and maximum, and the total spectral energy (compare Appendix B.1).

The results are depicted in Figure 4. We observe that the pcGAN matches the characteristics of both minimum and maximum distribution well, with the maximum distribution being very accurately matched; in the distribution of minimum values, the left peak is notably lower than in the true distribution. The standard WGAN does not match either distribution very well but performs better on the minimum than on the maximum values. The method of Wu et al. [2020] performs better than the standard
WGAN but slightly worse than the pcGAN. In case of the maximum values, it exhibits an odd edge in the middle of the distribution, as well as a lower peak on the right; in case of the minimum values it performs similarly to the pcGAN, only the peak in the middle is more exaggerated.

Figure 4: The distributions of minimum and maximum values as obtained by different models are compared, where the orange lines show the PDFs as learned by the cEBM. From left to right, the depicted distributions correspond to the real data, the unconstrained WGAN, the pcGAN and the method of Wu et al. [2020].

In the left plot of Figure 5, these observations are reflected in the range of KL divergences of the constraints, with the pcGAN achieving the best values. The performance metrics are depicted in the right plot; in terms of these, the pcGAN also performs well consistently, whereas both of the other methods are notably worse for at least one of the metrics.

The network architecture used for the GANs is based on the architecture from Holmberg [2022] and a batch size of 256 was used during training. More details on the training procedure, as well as plots of generated samples are given in Appendix B.3.

5.3 Cosmological temperature maps

The Cosmology and Astrophysics with MachinE Learning Simulations (CAMELS) dataset [Villaescusa-Navarro et al., 2021] has been developed in order to facilitate the development of machine learning models for use in cosmology and astrophysics. While the dataset provides data for a variety of physical fields, here we focus on temperature maps (Tmaps). They can be thought of as 2D images with one channel.
Figure 5: **Left** The constraint fulfillment for the different models is plotted vs the training iterations. The full lines give the KL divergences averaged over all constraints, whereas the shaded regions indicate the range in which the KL divergences lie for the 2 different constraints, from minimum value to maximum value. **Right** The performance of the models is evaluated at hand of the mean absolute value of the detector signals, the distance between minimum and maximum value, and the total spectral energy. For the distributions and curves displayed in these plots, 20 000 generated samples have been considered.

We will use the pcGAN again to see if the power spectrum (see Appendix B.1) of the true data can be matched more accurately. As performance metrics other than the constrained values themselves, we consider the Minkowski functionals [Mecke, 2000], which we implemented using the QuantImPy package [Boelens and Tchelepi, 2021]. The Minkowski functionals characterize the morphology of fields: $M_0$ gives the area of regions above a certain threshold, $M_1$ gives their circumference and $M_2$ is a measure of the curvature of their boundaries.

In Figure 6, the results are evaluated for different components of the power spectrum. For this example, the distribution of the power spectrum components is simple and all of the GANs manage to fit them well. Only the standard WGAN exhibits a clearly noticeable deviation from the true distribution. In the left plot of Figure 7, we plot again mean (full lines) and the intervals between minimum and maximum KL divergence of the constraints (shaded area). The pcGAN and the constrained GAN from Wu et al. [2020] perform almost equally well, with the standard WGAN being slightly less accurate. In the right plot, performance in terms of the Minkowski functionals is evaluated, where we chose the threshold 0 (the temperature maps have been normalized to the range $[-1, 1]$); here, the method of [Wu et al., 2020] outperforms the other GANs by a small margin.

The network architecture used for the GANs is very similar to that in [Villaescusa-Navarro et al., 2021], which is based on the DCGAN architecture [Radford et al., 2015]. The ADAM optimizer and a batch size of 256 were used during the training. More details on the architecture, the training
Figure 6: The distributions of three different power spectrum components as obtained by different models are compared, where the orange lines show the PDFs as learned by the cEBM. From left to right, the depicted distributions correspond to the real data, the unconstrained WGAN, the pcGAN and the method of Wu et al. [2020].

Figure 7: **Left** The constraint fulfillment for the different models is plotted vs the training iterations. The full lines give the KL divergences averaged over all constraints, whereas the shaded regions indicate the range in which the KL divergences lie for the 32 different constraints, from minimum value to maximum value. **Right** The performance of the models is evaluated at hand of the Minkowski functionals with threshold 0. For the distributions and curves displayed in these plots, 20,000 generated samples have been considered.
procedure, as well as plots of generated samples are given in Appendix B.4.

6 Conclusions and future work

We have presented the probabilistically constrained GAN (pcGAN), a method to incorporate probabilistic constraints into GANs. For a given statistic $s$, this is achieved by adding the KL divergence between true and generated distribution as an additional loss term to the generator loss. In order to approximate the true and generated distributions, an energy-based model and kernel density estimation have been employed, respectively. We have evaluated the method at hand of three different datasets, and demonstrated that the match of the true and generated distributions for the constrained statistics improves in comparison to WGAN and Wu et al. [2020]. Other performance metrics and sample quality either improve or are comparable to those of the standard WGAN; when compared to the method of Wu et al. [2020], which method performs best depends on the dataset.

For future work, it would be interesting to find ways of adapting the method such that long tails can also be included in the constraint. Furthermore, it would be interesting to extend the method to consider the joint distribution of different statistics, in order to also include correlations between them in the constraint. Investigating the applicability of the method to other generative models, such as autoencoders [Kingma and Welling, 2014] or denoising diffusion probabilistic models [Ho et al., 2020], constitutes another promising avenue for future research.

Acknowledgements

The work is 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. Our special thanks also go to Christian Glaser, for interesting discussions and for providing access to the IceCube data generator.

Bibliography


Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative


A Methodology

In this appendix, we give additional details on the various components of the pcGAN.

A.1 Training the conditional EBM

Algorithm 3: Training the conditional EBM.

\textbf{Input:} Untrained cEBM ($\hat{h}$); statistics $\bigcup_s \{(z_s, s)\}$ of original dataset

\textbf{Result:} Trained cEBM ($\hat{h}$)

$(a_J, w) = (\text{ones}(N_s), \text{ones}(N_s)/N_s)$;

$N_J = 10$;

\textbf{while Training do}

$J = 0$;

\textbf{for} $j$ in range($N_J$) \textbf{do}

sample $s \sim \text{Discrete}($range($N_s$); $w$);

sample minibatch $\{(z_{si}, s)\}_{i=1}^{n_{\text{batch}}}$ of constraint values;

calculate $J_s = \text{NLL}(\{(z_{si}, s)\}_{i=1}^{n_{\text{batch}}}, \hat{h})/n_{\text{batch}}$ according to $s$-th term in (4);

$J = J + J_s$;

update $(a_J, w)$ according to Algorithm 4, with input $(a_J, J_s, s)$;

\textbf{end}

$J = J/N_J$;

\textbf{end}

Optimization step;

In order to model the distributions $p_{\text{true}}(z_s)$ of the constrained statistics, we employ a conditional EBM $\hat{p}_{\text{true}}(z_s|s, \hat{h})$ (compare Section 3.2). Note, that the neural network does not model the PDF directly, but instead parametrizes the function $\hat{h}$ in (3).

The network architecture for the cEBM is given in Table 1. The input of the network is a 2-tuple, where the first entry gives the constraint value $z_s$ and the second entry the corresponding class index $s$. The former serves as input to the fully connected layer Linear01, whereas the latter is fed into the embedding layer followed by the layer Linear01; subsequently, the resulting outputs are concatenated. The inputs $z_s$ are normalized to lie within the range $[0, 1]$, and outputs are normalized to lie in the range $[0, 5]$; these normalizations are performed separately for each statistic $s$. 
Table 1: Network architecture for the conditional EBM. The tuple \((z_s, s)\) serves as input; \(s\) is passed through the Embedding layer followed by the layer Linear00, whereas \(z_s\) is passed through the layer Linear01. Subsequently, the resulting outputs are concatenated and passed through the remaining network.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output Shape</th>
<th>Activation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>(1,1)</td>
<td></td>
</tr>
<tr>
<td>Embedding</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Linear00</td>
<td>15</td>
<td>ReLU</td>
</tr>
<tr>
<td>Linear01</td>
<td>40</td>
<td>ReLU</td>
</tr>
<tr>
<td>Concatenation</td>
<td>55</td>
<td>ReLU</td>
</tr>
<tr>
<td>Linear</td>
<td>40</td>
<td>ReLU</td>
</tr>
<tr>
<td>Linear</td>
<td>40</td>
<td>ReLU</td>
</tr>
<tr>
<td>Dropout(0.2)</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Linear</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

The training procedure for this network is given in Algorithm 3. The ADAM optimizer with learning rate 0.005, \(\beta_1 = 0.9\), \(\beta_2 = 0.99\) and a batch size of 1024 was used; the network is trained for 20 000 iterations, where the learning rate was halved twice, after 6 000 and after 12 000 iterations. Here, in each iteration, we consider \(N_J\) of the different constraints part of the conditional EBM; choosing \(N_J = 10\) has proven to give good results. In cases where \(N_s < 10\), \(N_J = N_s\) was chosen. The \(N_J\) constraints are sampled randomly out of all the constraints, at first equally weighted and subsequently according to the preceding values of their respective NLL (compare Appendix A.2).

### A.2 Weighting the constraints

**Algorithm 4:** Weighted sampling scheme for different loss terms.

**Input:** array \(a_L\); new loss \(L_i\), index \(i\)

**Result:** (array \(a_L\), weights \(w\))

\[
\begin{align*}
    a_L[i] &= (a_L[i] + L_i) / 2; \\
    w_0 &= a_L - \min(a_L) + 0.1(\max(a_L) - \min(a_L)) + 1e-4; \\
    w &= w_0 / \text{sum}(w_0);
\end{align*}
\]

Both when training the pcGAN (compare Algorithm 2) and when training the cEBM (compare Algorithm 3), we resorted to a weighted sampling scheme in order to put more emphasis on constraints that were as of yet not...
well fulfilled. In case of the cEBM, this procedure determined the statistic $s$, realizations $z_s$ of which would be used as training data in the present training step. In case of the pcGAN, loss terms corresponding to different constrained statistics $s$ were added to the generator loss according to this sampling procedure.

This procedure is detailed in Algorithm 4; the array $a_L$ keeps track of recent loss values and is updated with the new loss $L_i$ every time it is called in the first line of the algorithm. Hence, it constitutes a weighted sum of all loss values obtained so far, where recent values are weighted more strongly and earlier loss values decay exponentially. In the next step, the unnormalized weights $w_0$ are calculated: first, the term $a_L - \text{min}(a_L)$ shifts the losses such that the minimum loss is zero. Then, an offset term $0.1(\max(a_L) - \text{min}(a_L))$ dependent on the range of loss values is added, in order to prevent constraint components from having zero chance of being sampled. Subsequently, another very small offset term of magnitude $1\text{e}^{-4}$ is added; the purpose of this term is simply to prevent errors from arising when all entries of $a_L$ have the exact same values, as is e.g. the case after they have been initialized to an array of ones. Finally, in the third step, the weights are normalized and can be used for sampling.

### A.3 Algorithm to determine $f^*_\sigma$

**Algorithm 5:** Determining the optimal values of $f_\sigma$

---

**Input:** true data $\bigcup_s \{(z_s, s)\}$; trained cEBM $\tilde{p}_{\text{true}}(z_s|s)$

**Result:** $f^*_\sigma$

$N_{\text{avg}} = 50$;

$a_{f_\sigma} = \text{logspace}(-1, 2, 200)$;

for $i_N \in [0, N_{\text{avg}}); s \in [0, N_s); i_{f_\sigma} \in a_{f_\sigma};$ do

sample minibatch $\{(z_s, s)\}$;

determine $\tilde{p}_{\text{gen}}(z_s)$ according to (7);

$\text{KL}[s, i_{f_\sigma}, i_N] = \text{KL}(\tilde{p}_{\text{true}}(z_s|s)||\tilde{p}_{\text{gen}}(z_s))$ according to (6);

end

$i_{f_\sigma} = \text{min}(\text{mean}(\text{KL}, \text{dim} = 2), \text{dim} = 1)$;

$f^*_\sigma = a_{f_\sigma}[i_{f_\sigma}]$;

---

In order to determine the optimal value of $f_\sigma$ for a given constraint distribution, i.e. the distribution of constraint values $z_s$ corresponding to constraint $s$, we perform a grid search over possible values $f_\sigma$. For each value of $f_\sigma$, we evaluate the KL divergence between the approximate true distribution $\tilde{p}_{\text{true}}(z_s|s)$ as obtained via the cEBM, and the generated distribution $\tilde{p}_{\text{gen}}(z_s)$ as obtained from the current minibatch according to (7).
The minibatches are sampled from the true data since the aim is to obtain a KL divergence as small as possible for true data. The obtained values for the KL divergence are then averaged over 50 iterations and subsequently, the value $f_\sigma$ corresponding to the minimum mean value is determined; this value is the desired optimal value $f^*_\sigma$. This procedure is summarized in Algorithm 5, which returns an array of values $f^*_\sigma$ that contains the optimal values of $f_\sigma$ for all $N_s$ constraints under consideration.

B Experiments

In this appendix, we give additional information on the experiments conducted in Sections 5.1-5.3; in particular on the network architectures and the training parameters. The code for the project is available on GitHub\(^4\); note, however, that only the data for the synthetic example is available there.

B.1 Constraints and performance metrics

We start by giving an overview of the different quantities that have been employed either as constraints or performance metrics.

For the 1D signals $x$ of length $N_x = 200$ in Sections 5.1 and 5.2, we used the minimum and maximum values, $\min = \min(x_0, \ldots, x_{N_x-1})$, $\max = \max(x_0, \ldots, x_{N_x-1})$, mean values, $\text{mean}(\text{abs}) = \frac{1}{N_x} \sum_{i=0}^{N_x-1} x_i$, and mean absolute values, $\text{mean}(\text{abs}) = \frac{1}{N_x} \sum_{i=0}^{N_x-1} |x_i|$, of the curves.

The discrete Fourier transform for real-valued inputs (as implemented in torch.fft.rfft) was utilized to obtain the complex Fourier coefficients for positive frequencies $k \in [0, \lfloor N_x/2 \rfloor + 1]$ below the Nyquist frequency,

$$X_k = \sum_{n=0}^{N_x-1} x_n e^{-i2\pi \frac{kn}{N_x}}, \quad (8)$$

and the corresponding power spectrum components are obtained as $S_k = \frac{1}{N_x}|X_k|^2$. The total spectral energy is then calculated as $S = \sum_{k=0}^{\lfloor N_x/2 \rfloor + 1} S_k$. When employed as constraints, we did not constrain on the power spectrum components directly, but instead on $\text{ps}[k] = \sqrt{N_x S_k}$; in case of the spectral energy, we plotted $E = N_x S$.

In case of the 2D temperature maps of size $64 \times 64$ from the CAMELS dataset in Section 5.3, we first calculated the 2D discrete Fourier transform of the inputs (via torch.fft.fft2); then we obtained the 1D power spectrum by considering the radial profile of the 2D power spectrum corresponding to the Tmaps. That is, the vectors from the 2D power spectrum have

\(^4\)The code used for the experiments is available at https://github.com/ppilar/pcGAN.
been binned together into 32 different bins according to their magnitudes $k = \sqrt{|k_x|^2 + |k_y|^2}$, with bin ranges $\{[0.5, 1.5], [2.5, 3.5], \ldots, [31.5, 32.5]\}$ (note that we omitted physical units in our implementation).

As further performance metrics for the Tmaps, we considered the Minkowski functionals $M_k 0$, $M_k 1$, and $M_k 2$, which we calculated using the QuantImPy package [Boelens and Tchelepi, 2021], and precise definitions of which can be found in Mecke [2000] and Boelens and Tchelepi [2021].

B.2 Synthetic example (Section 5.1)

The synthetic dataset consists of 100 000 samples of size 200, generated as described in Section 5.1. For this example, we employed convolutional networks for both the discriminator and generator; details on the corresponding network architectures are given in Tables 4 and 5, respectively. In layers where both batch normalization and an activation function are listed in the column ‘Activation’, batch normalization is applied to the input of the layer whereas the activation function is applied to the output. Padding is employed in each layer such that the given output sizes are obtained; reflection padding is utilized.

In Figure 10, samples from the true distribution as well as generated samples from the different GANs are depicted. It is apparent that the standard WGAN and the method of Wu et al. [2020] produce samples that look slightly worse than for the pcGAN, i.e. they produce somewhat more rugged curves. Apart from that, all three produce reasonable-looking results, although upon closer inspection it becomes apparent that they do not necessarily constitute a superposition of two sine waves.

B.3 IceCube-Gen2 (Section 5.2)

For this example, we considered 50 000 IceCube-Gen2 radio-detector signals of size 200, normalized to their respective maximum absolute values. The networks employed are a mixture of convolutional and fully connected networks, which have been based on the architectures used in Holmberg [2022]; details on discriminator and generator architectures are given in Tables 6 and 7, respectively. For the discriminator, the input to the network is first fed through four convolutional layers in parallel, the outputs of which are subsequently concatenated into one long array. The LeakyReLU activation function with factor 0.2 is applied.

In this example, both of the constraints considered exhibit a spike at values $-1$ or $1$, for the minimum and maximum values, respectively. This can destabilize the cEBM training since the PDFs will be singular at these points, and we resolved this issue by adding a very small amount of ran-
Figure 8: **Top left** The two plots on the top left depict the dependency of the KL divergence on the factor $f_\sigma$; the curves have been averaged over 50 minibatches sampled from the original dataset. **Bottom left** The first two plots in the bottom row depict the distribution of the constraint values together with their EBM representation, as well as curves obtained via (7) from minibatches of different size (not averaged); it is between them that the KL divergences in the top row have been calculated. **Top right** Optimal values of the factor $f_\sigma^*$ are depicted for different batch sizes, where the index $i$ gives the respective constraint.

dom noise in the range between 0 and ±0.01 to the constraint values while training the cEBM.

In Figure 8, a plot on the process of determining optimal values for $f_\sigma^*$ is given. Same as for the synthetic example (compare Fig. 3), the KL divergences as a function of $f_\sigma$ exhibit clear minima that depend on the batch size. For these distributions, cutting the tails was not necessary.

In Figure 11, samples from the true distribution as well as generated samples from the different GANs are depicted. Altogether, most of the generated samples look good, with none of the models clearly outperforming the others.

**B.4 Temperature maps (Section 5.3)**

In this example, we considered 30 000 temperature maps of size $64 \times 64$ from the CAMEL dataset. The data has been augmented by randomly rotating and flipping samples during the training. We employed fully convolutional networks for both discriminator and generator, which have been based on
the architecture employed in Villaescusa-Navarro et al. [2021]. Details on the network architectures are given in Tables 6 and 7, respectively. Batch normalization followed by the LeakyReLU activation function with factor 0.2 is applied after all layers, except for the last one.

In Figure 9, a plot on the process of determining optimal values for $f^*_\sigma$ is given. Same as for the synthetic example (compare Fig. 3), the KL divergences as a function of $f_\sigma$ exhibit clear minima that depend on the batch size. For this example, all of the distributions of the power spectrum components look very similar, and no extremely long tails are present. Still, e.g. for the first power spectrum component, cutting the tails can make a difference, as illustrated in the bottom right plot.

In Figure 12, samples from the true distribution as well as generated samples from the different GANs are depicted. All of the GANs produce samples that look realistic and, from the depicted temperature maps alone, it is hard to tell a difference between the models.

### B.5 Training parameters

Here, we summarize the training parameters used for the different experiments. $N_{it}$ gives the number of training iterations, lr the learning rate, $\lambda$ the weighting factor for the constraints in (5), $\lambda_{Wu}$ the weighting factor for the statistical constraint from Wu et al. [2020]. In the column ‘clamping’, the range is given to which network parameters of the discriminator $D$ were clamped in order to enforce the Lipschitz constraint in WGANs [Arjovsky et al., 2017]. The ADAM optimizer [Kingma and Ba, 2015] was used for training the networks, with hyperparameter values $\beta_1$ and $\beta_2$; after 100 000 iterations, as well as after 150 000 iterations, if applicable, the learning rate was reduced by a factor of 0.2. For the synthetic dataset, the learning rate was kept constant. The parameter $N_d$, which gives the number of discriminator updates per generator update, was chosen as 1.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>$N_{it}$</th>
<th>lr</th>
<th>$\lambda$</th>
<th>$\lambda_{Wu}$</th>
<th>$\beta_1$</th>
<th>$\beta_2$</th>
<th>clamping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthetic (5.1)</td>
<td>150 000</td>
<td>2e-4</td>
<td>30</td>
<td>1</td>
<td>0</td>
<td>0.9</td>
<td>[0, 0.005]</td>
</tr>
<tr>
<td>IceCube-Gen2 (5.2)</td>
<td>200 000</td>
<td>2e-4</td>
<td>0.1</td>
<td>1</td>
<td>0</td>
<td>0.9</td>
<td>[0, 0.01]</td>
</tr>
<tr>
<td>Tmaps (5.3)</td>
<td>120 000</td>
<td>1e-4</td>
<td>0.01</td>
<td>0.001</td>
<td>0</td>
<td>0.9</td>
<td>[0, 0.02]</td>
</tr>
</tbody>
</table>

### B.6 Compute

The models were trained on an Nvidia RTX 3060 Ti. In Table 3, estimated runtimes are given for the experiments described in Sections 5.1 - 5.3; the
Figure 9: **Top** The three plots on top depict the dependency of the KL divergence on the factor $f_\sigma$; the curves have been averaged over 50 mini-batches sampled from the original dataset. **Middle** The first three plots in the bottom row depict the distribution of the constraint values together with their EBM representation, as well as curves obtained via (7) from mini-batches of different size (not averaged); it is between them that the KL divergences in the top row have been calculated. **Bottom left** Optimal values of the factor $f^*_\sigma$ are depicted for different batch sizes, where the index $i$ gives the respective component of the power spectrum. **Bottom right** Here, the tails were not cut, resulting in a larger standard deviation due to the zero-avoiding order of the KL-divergence in (6).
total amount of compute used for the project is probably around 20 times as much. We included the runtimes for training the cEBM (which include the runtime for determining $f^\ast_\sigma$) and for training the GANs.

Table 3: Runtimes for the experiments.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>cEBM</th>
<th>WGAN</th>
<th>pcGAN</th>
<th>Wu et al. [2020]</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthetic (5.1)</td>
<td>0.1h</td>
<td>1.3h</td>
<td>3.0h</td>
<td>2.2h</td>
<td>6.6h</td>
</tr>
<tr>
<td>IceCube-Gen2 (5.2)</td>
<td>0.1h</td>
<td>0.4h</td>
<td>1.1h</td>
<td>0.5h</td>
<td>2.1h</td>
</tr>
<tr>
<td>Tmaps (5.3)</td>
<td>0.2h</td>
<td>4.7h</td>
<td>6.7h</td>
<td>4.9h</td>
<td>16.5h</td>
</tr>
<tr>
<td>total</td>
<td>0.4h</td>
<td>6.4h</td>
<td>10.8h</td>
<td>7.6h</td>
<td>25.2h</td>
</tr>
</tbody>
</table>

Limitations and Societal Impact

**Limitations** One of the main limitations of our method lies in the fact that it is restricted to consider constraints as acting independently and does not include potential correlations between them. Another weakness of the method is that it cannot take long tails of distributions into account accurately and instead, we had to resort to cutting them off. Furthermore, we were not able to perform multiple runs of our experiments, due to the long training time, and hence we are unable to report on their statistical significance.

**Societal impact** In general, generative models can be used for harmful purposes, e.g. in the form of deepfakes; our work, however, aims at complementing scientific simulations, and we do not foresee negative consequences due to our method in particular.
Table 4: Discriminator architecture for the synthetic example.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output Size</th>
<th>Kernel Size</th>
<th>Stride</th>
<th>Activation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>1 × 200</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conv</td>
<td>32 × 99</td>
<td>3</td>
<td>2</td>
<td>BatchNorm, ReLU</td>
</tr>
<tr>
<td>Conv</td>
<td>32 × 99</td>
<td>3</td>
<td>1</td>
<td>BatchNorm, ReLU</td>
</tr>
<tr>
<td>Conv</td>
<td>32 × 99</td>
<td>3</td>
<td>1</td>
<td>ReLU</td>
</tr>
<tr>
<td>Conv</td>
<td>64 × 48</td>
<td>3</td>
<td>2</td>
<td>BatchNorm, ReLU</td>
</tr>
<tr>
<td>Conv</td>
<td>64 × 48</td>
<td>3</td>
<td>1</td>
<td>BatchNorm, ReLU</td>
</tr>
<tr>
<td>Conv</td>
<td>64 × 48</td>
<td>3</td>
<td>1</td>
<td>ReLU</td>
</tr>
<tr>
<td>Conv</td>
<td>128 × 23</td>
<td>3</td>
<td>2</td>
<td>ReLU</td>
</tr>
<tr>
<td>Conv</td>
<td>128 × 23</td>
<td>3</td>
<td>1</td>
<td>ReLU</td>
</tr>
<tr>
<td>Conv</td>
<td>128 × 23</td>
<td>3</td>
<td>1</td>
<td>ReLU</td>
</tr>
<tr>
<td>Conv</td>
<td>256 × 10</td>
<td>3</td>
<td>2</td>
<td>ReLU</td>
</tr>
<tr>
<td>Conv</td>
<td>256 × 10</td>
<td>3</td>
<td>1</td>
<td>ReLU</td>
</tr>
<tr>
<td>Conv</td>
<td>256 × 10</td>
<td>3</td>
<td>1</td>
<td>ReLU</td>
</tr>
<tr>
<td>Flatten</td>
<td>2560</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linear</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Generator architecture for the synthetic example.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output Size</th>
<th>Kernel Size</th>
<th>Stride</th>
<th>Activation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>1 × 5</td>
<td></td>
<td></td>
<td>BatchNorm</td>
</tr>
<tr>
<td>ConvTransp</td>
<td>256 × 25</td>
<td>3</td>
<td>16</td>
<td>BatchNorm, Tanh</td>
</tr>
<tr>
<td>Conv</td>
<td>256 × 25</td>
<td>3</td>
<td>1</td>
<td>BatchNorm, Tanh</td>
</tr>
<tr>
<td>Conv</td>
<td>256 × 25</td>
<td>3</td>
<td>1</td>
<td>BatchNorm, Tanh</td>
</tr>
<tr>
<td>ConvTransp</td>
<td>128 × 50</td>
<td>3</td>
<td>2</td>
<td>BatchNorm, Tanh</td>
</tr>
<tr>
<td>Conv</td>
<td>128 × 50</td>
<td>3</td>
<td>1</td>
<td>BatchNorm, Tanh</td>
</tr>
<tr>
<td>Conv</td>
<td>128 × 50</td>
<td>3</td>
<td>1</td>
<td>BatchNorm, Tanh</td>
</tr>
<tr>
<td>ConvTransp</td>
<td>64 × 100</td>
<td>3</td>
<td>2</td>
<td>BatchNorm, Tanh</td>
</tr>
<tr>
<td>Conv</td>
<td>64 × 100</td>
<td>3</td>
<td>1</td>
<td>BatchNorm, Tanh</td>
</tr>
<tr>
<td>Conv</td>
<td>64 × 100</td>
<td>3</td>
<td>1</td>
<td>BatchNorm, Tanh</td>
</tr>
<tr>
<td>ConvTransp</td>
<td>32 × 200</td>
<td>3</td>
<td>2</td>
<td>BatchNorm, Tanh</td>
</tr>
<tr>
<td>Conv</td>
<td>32 × 200</td>
<td>3</td>
<td>1</td>
<td>BatchNorm, Tanh</td>
</tr>
<tr>
<td>Conv</td>
<td>32 × 200</td>
<td>3</td>
<td>1</td>
<td>Tanh</td>
</tr>
<tr>
<td>Conv</td>
<td>1 × 200</td>
<td>3</td>
<td>1</td>
<td>Tanh</td>
</tr>
</tbody>
</table>
Table 6: Discriminator architecture for the IceCube-Gen2 data. The input is first fed through the layers Conv01-Conv04 in parallel and the outputs are subsequently concatenated into one long array.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output Shape</th>
<th>Kernel Size</th>
<th>Stride</th>
<th>Activation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>$1 \times 200$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conv01</td>
<td>$32 \times 49$</td>
<td>5</td>
<td>4</td>
<td>LeakyReLU</td>
</tr>
<tr>
<td>Conv02</td>
<td>$32 \times 47$</td>
<td>15</td>
<td>4</td>
<td>LeakyReLU</td>
</tr>
<tr>
<td>Conv03</td>
<td>$32 \times 44$</td>
<td>25</td>
<td>4</td>
<td>LeakyReLU</td>
</tr>
<tr>
<td>Conv04</td>
<td>$32 \times 42$</td>
<td>35</td>
<td>4</td>
<td>LeakyReLU</td>
</tr>
<tr>
<td>Concatenate</td>
<td>$32 \times 182$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conv</td>
<td>$1 \times 182$</td>
<td>1</td>
<td>1</td>
<td>LeakyReLU</td>
</tr>
<tr>
<td>Linear</td>
<td>92</td>
<td></td>
<td></td>
<td>LeakyReLU</td>
</tr>
<tr>
<td>Linear</td>
<td>45</td>
<td></td>
<td></td>
<td>LeakyReLU</td>
</tr>
<tr>
<td>Linear</td>
<td>20</td>
<td></td>
<td></td>
<td>LeakyReLU</td>
</tr>
<tr>
<td>Linear</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7: Generator architecture for the IceCube-Gen2 data.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output Size</th>
<th>Kernel Size</th>
<th>Stride</th>
<th>Activation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linear</td>
<td>24</td>
<td></td>
<td></td>
<td>ReLU</td>
</tr>
<tr>
<td>Conv</td>
<td>$48 \times 24$</td>
<td>3</td>
<td>1</td>
<td>ReLU</td>
</tr>
<tr>
<td>Conv</td>
<td>$48 \times 24$</td>
<td>3</td>
<td>1</td>
<td>ReLU</td>
</tr>
<tr>
<td>ConvTransp</td>
<td>$24 \times 49$</td>
<td>3</td>
<td>2</td>
<td>ReLU</td>
</tr>
<tr>
<td>Conv</td>
<td>$24 \times 49$</td>
<td>3</td>
<td>1</td>
<td>ReLU</td>
</tr>
<tr>
<td>ConvTransp</td>
<td>$12 \times 99$</td>
<td>3</td>
<td>2</td>
<td>ReLU</td>
</tr>
<tr>
<td>Conv</td>
<td>$12 \times 99$</td>
<td>3</td>
<td>1</td>
<td>ReLU</td>
</tr>
<tr>
<td>ConvTransp</td>
<td>$6 \times 199$</td>
<td>3</td>
<td>2</td>
<td>ReLU</td>
</tr>
<tr>
<td>Conv</td>
<td>$1 \times 200$</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Table 8: Discriminator architecture for the CAMELS data.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output Size</th>
<th>Kernel Size</th>
<th>Stride</th>
<th>Activation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>$1 \times 64 \times 64$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conv</td>
<td>$64 \times 32 \times 32$</td>
<td>$4 \times 4$</td>
<td>2</td>
<td>BatchNorm2d, LeakyReLU</td>
</tr>
<tr>
<td>Conv</td>
<td>$128 \times 16 \times 16$</td>
<td>$4 \times 4$</td>
<td>2</td>
<td>BatchNorm2d, LeakyReLU</td>
</tr>
<tr>
<td>Conv</td>
<td>$256 \times 8 \times 8$</td>
<td>$4 \times 4$</td>
<td>2</td>
<td>BatchNorm2d, LeakyReLU</td>
</tr>
<tr>
<td>Conv</td>
<td>$512 \times 4 \times 4$</td>
<td>$4 \times 4$</td>
<td>2</td>
<td>BatchNorm2d, LeakyReLU</td>
</tr>
<tr>
<td>Conv</td>
<td>$1 \times 1 \times 1$</td>
<td>$4 \times 4$</td>
<td>1</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 9: Generator architecture for CAMELS data (where BN2d = BatchNorm2d).

<table>
<thead>
<tr>
<th>Layer</th>
<th>Output Size</th>
<th>Kernel Size</th>
<th>Stride</th>
<th>Activation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>$100 \times 1 \times 1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ConvTransp2d</td>
<td>$512 \times 4 \times 4$</td>
<td>$4 \times 4$</td>
<td>1</td>
<td>BN2d, LeakyReLU</td>
</tr>
<tr>
<td>ConvTransp2d</td>
<td>$256 \times 8 \times 8$</td>
<td>$4 \times 4$</td>
<td>2</td>
<td>BN2d, LeakyReLU</td>
</tr>
<tr>
<td>ConvTransp2d</td>
<td>$128 \times 16 \times 16$</td>
<td>$4 \times 4$</td>
<td>2</td>
<td>BN2d, LeakyReLU</td>
</tr>
<tr>
<td>ConvTransp2d</td>
<td>$64 \times 32 \times 32$</td>
<td>$4 \times 4$</td>
<td>2</td>
<td>BN2d, LeakyReLU</td>
</tr>
<tr>
<td>ConvTransp2d</td>
<td>$1 \times 64 \times 64$</td>
<td>$4 \times 4$</td>
<td>2</td>
<td>Tanh</td>
</tr>
</tbody>
</table>
Figure 10: Samples for the synthetic example as obtained from the different models.
Figure 11: Samples for the IceCube-Gen2 dataset as obtained from the different models.
Figure 12: Samples for the CAMELS dataset as obtained from the different models.
Recent licentiate theses from the Department of Information Technology

2023-001 Håkan Runvik: *Modeling and Estimation of Impulsive Biomedical Systems*

2022-003 Camille Clouard: *Computational Statistical Methods for Genotyping Biallelic DNA Markers from Pooled Experiments*

2022-002 Gustaf Borgström: *Making Sampled Simulations Faster by Minimizing Warming Time*

2022-001 Sam Hylamia: *Secure In-body Communication and Sensing*

2021-002 Karl Bengtsson Bernander: *Improving Training of Deep Learning for Biomedical Image Analysis and Computational Physics*

2021-001 Niklas Gunnarsson: *On the Registration and Modeling of Sequential Medical Images*

2020-006 David Widmann: *Calibration of Probabilistic Predictive Models*

2020-005 Anna Wigren: *Exploiting Conjugacy in State-Space Models with Sequential Monte Carlo*

2020-004 Muhammad Osama: *Machine Learning for Spatially Varying Data*

2020-003 Christos Sakalis: *Securing the Memory Hierarchy from Speculative Side-Channel Attacks*

2020-002 Ulrika Sundin: *Global Radial Basis Function Collocation Methods for PDEs*


2019-006 Kristiina Ausmees: *Efficient Computational Methods for Applications in Genomics*