Learning Probabilistic Representations for Inference, Training and Interpretability



Gabriele Abbati
Linacre College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Hilary 2020

To my family

Acknowledgements

First and foremost, I would like to thank my family for everything they did during these long and intense university years. Their efforts and sacrifices allowed me to study in some of the best universities in the world without a single worry. This thesis would not exist without their love and support. I will be forever grateful to my mother Paola, my father Giuseppe, my uncle Gabriele and my aunt Nives.

I would like to thank my supervisors, Michael Osborne and Tom Rainforth, for the help and support I received during my DPhil. In addition to academic guidance, I would like to thank Tom for the light-hearted dinners and evenings we enjoyed in my last year at Oxford. I would also like to thank Deepmind and the University of Oxford for providing the necessary funding for this research.

Along the way, I met many people who helped me both academically and personally, and I would like to express my gratitude for them. I would like to thank Stefan Bauer for having invited me to the Max Planck Institute for Intelligent Systems in Tübingen, a research stay that proved crucial for the success of my DPhil. Thanks to Francesco and Philippe for having been fantastic collaborators and friends. Also, thanks to Matteo for the good times we had in Germany, Spain and Zürich, and for having made the long days in the lab more enjoyable. Most importantly, I would like to thank my Oxford colleagues Bernardo, Kyriakos, Henry and Tom, for all the discussions, the laughs, the drinks and the dances (and for having being exceptional proofreaders).

Finally, I would like to deeply thank Federica, who has always been close to me during my last months as a student. I will always be immensely grateful for her love, support and understanding.

Abstract

For decades, the holy grail of many fields of science, mathematics and statistics has been the ability to predict the future or, more quantitatively, to forecast the behaviours of quantities of interest. Examples range from predicting the value of financial products to forecasting energy consumption and several other applications.

While historically practitioners have focused on designing handcrafted methods for specific prediction tasks, the tools of Machine Learning (ML) promise to automate data analyses that extract patterns and structures and leverage this information in future forecasts. The addition of the probabilistic framework of Bayesian Machine Learning enables the inclusion of useful prior knowledge and provides predictive uncertainty quantification.

This thesis focuses on generative modeling, an important class of algorithms that stands at the core of Bayesian ML: by making structural assumptions on the generative processes that produced the data, generative models intend to explain the observed variability and yield a deep, interpretable understanding of data.

We present three original contributions, which leverage model-based machine learning to gain insights about the data-generating process and subsequently leverage them to provide reliable predictions. First, we propose a novel probabilistic framework based on constrained Gaussian Processes that incorporates derivative information in the form of ordinary differential equations. Second, we propose a novel noise model for

Gaussian Processes that, in conjunction with state-of-the-art moment matching and adversarial techniques, addresses the problem of parameter inference in systems of stochastic differential equations. Finally, we move our attention to the optimization and sampling algorithms employed for Bayesian inference. In particular, we use Gaussian Process latent variable models to learn lower-dimensional space embeddings wherein optimizers and samplers can operate more effectively, thereby producing a meta-algorithm that can be applied to these inference methods to improve efficiency.

Contents

1 Introduction **1**

1.1 Contributions and Outline 4

1.2 Publications 7

1.3 Code Implementation 8

2 Literature Review **9**

2.1 Probabilistic Machine Learning 10

2.2 Latent Variable Models 15

2.3 Computational Techniques for Inference and Training 19

2.4 Model Learning 34

2.5 Gaussian Processes 43

2.6 Machine Learning for Differential Equations 57

3 ODIN: ODE-Informed Regression for Parameter and State Inference in Time-Continuous Dynamical Systems **65**

4 AReS and MaRS - Adversarial and MMD-Minimizing Regression for SDEs **83**

5 AdaGeo: Adaptive Geometric Learning for Optimization and Sampling **97**

6 Conclusions **109**

6.1 Achievements 109

6.2 Discussion and Future Work 111

A Code Implementation **115**

A.1 ODIN: ODE-Informed Regression for Parameter and State Inference in Time-Continuous Dynamical Systems 115

A.2 AReS and MaRS - Adversarial and MMD-Minimizing Regression for SDEs 118

A.3 AdaGeo: Adaptive Geometric Learning for Optimization and Sampling 120

Bibliography **123**

1

Introduction

Predicting the future behaviour of certain quantities of interest is a task of crucial importance in many sciences and industrial applications. Examples range from forecasting energy consumption in order to regulate production, to predicting the future value of financial products to guide investments, to optimizing industrial manufacturing production. When tackling these challenges, practitioners historically developed handcrafted, *ad-hoc* computational pipelines explicitly designed to address specific prediction tasks. Nowadays, with data being often complex and high-dimensional, not infrequently such approaches prove ineffective. Indeed, the limited nature of algorithms that are designed by experts to address a particular problem prevents these methods to capture the entirety of relevant information contained in data. At the same time, it is not rare that unwanted prior knowledge and structural assumptions get irremediably injected into the computational pipeline.

It is thus easy to see the potential benefits which automated data analyses can offer: for this reason we look at the tools of Machine Learning. With the term Machine Learning (ML), we denote a set of methods that can automatically extract

patterns in data and exploit this knowledge to forecast quantities of interest, make informed decisions and many other similar tasks [Murphy, 2012]. ML methods employ automation to unravel the hidden structures in data and go beyond the limits of approaches that are tailored to specific applications.

However, in many settings data can be noisy, messy and/or limited, in which cases the results produced by algorithms relying on it can be inaccurate. Furthermore, there are circumstances in which it would be beneficial to augment the problem formulation with prior knowledge, especially if included in a soft, weak way, to correct for the effects of low-quality data. In this view, the algorithms and methods of Machine Learning can be enhanced with the tools of probability and statistics: the set of methods that this process yields is denoted as Bayesian Machine Learning (an introduction can be found in Gelman et al. [2013]; Ghahramani [2015]). This probabilistic framework can be used to express a notion of confidence on predictions: the purpose of such uncertainty quantification, modelled through probabilistic formulations, consists in assessing how much we believe a certain solution to be the "correct" one. Moreover, Bayesian ML allows us to encode and account for the prior knowledge that the model designer may have about the problem.

Throughout this work, we will mainly focus on *generative models*, a family of methods which stands at the core of Bayesian Machine Learning and that can contribute to a deeper understanding of the data generating process. As their name suggests, they leverage probabilistic assumptions to model the generative mechanisms that caused the data and explain in this way the main features of the available observations. Modeling how data originated can help decipher important properties that have a fundamental role both in understanding the problem and improving performance within a subsequent downstream task (such as the prediction challenge mentioned above).

Within the class of generative probabilistic methods, we are mainly interested in a specific subset of models: latent variable models [Bishop, 2006]. When used in a generative fashion, latent variable models (or LVMs for the sake of brevity)

can be employed to reason about the variability in the observed data and the processes that generated it through a set of latent (or unobserved) variables. For this reason, they are often applied in the context of representation learning, a particular subclass of ML that aims at learning embeddings of the data that allow to extract information more easily and improve performance of downstream predictors [Bengio et al., 2013]. Intuitively, LVMs assume the existence of probabilistic dependencies between these two sets of variables to uncover important underlying, hidden structures. Additionally, the latent set, together with the mapping, can yield a simpler, lower-dimensional representation for the observations. Thanks to their generality and wide applicability, latent variable models can be employed in several settings, such as indicating regime changes in time series, as in Hidden Markov Models (HMM), clustering data points by describing multi-modality, as in Mixture Models, and performing dimensionality reduction through lower-dimensional representations.

The contents of this thesis are organized on the basis of the main applications the LVMs are employed for. Chapters 3 through 5 present the novel contributions of this work, which span three main areas:

- **Inference** and **Interpretability** for **ODEs**: in the context of state and parameter inference in systems of ordinary differential equations (ODEs), we incorporate model-based knowledge into Gaussian Process (GP) regression [Rasmussen and Williams, 2006], whose denoised state space can be seen in this view as a latent space.

- **Inference** and **Interpretability** for **SDEs**: we extend the framework introduced for ODEs to the more challenging setting of stochastic differential equations (SDEs). Incorporating once again model-based knowledge into GP regression, we construct a latent space in which the different sources of noise are separated, for better parameter inference and predictions.

- **Model Training**: we investigate the assumption that optimizers and samplers, such as Stochastic Gradient Descent and Markov Chain Monte Carlo [Hastings, 1970], act on a lower-dimensional manifold. We do this through the help of Gaussian Process Latent Variable Models (GPLVM) [Lawrence, 2005], in the attempt of speeding up sampling- and optimization-based inference.

In the next section we describe in more detail the contributions introduced above.

1.1 Contributions and Outline

In this section we outline the remainder of the thesis.

In Chapter 2 we provide the context and background necessary for this work. We start by introducing the ideas behind probabilistic ML by discussing discriminative and generative modeling and looking at how these two paradigms approach prediction tasks. In the context of generative modeling, we proceed to give a high-level discussion of Bayesian Machine Learning that includes the concepts of prior knowledge, updating beliefs and probabilistic prediction. We also include a brief introduction to Latent Variable Models. Next, we presents some of the optimization and sampling tools that are needed to perform approximate inference in the setting of Bayesian ML. We then introduce the concept of model learning, point out the connections with model-data divergence minimization and discuss methods based on moment matching and adversarial techniques that address this problem. Finally, in order to provide a more detailed background to the single contributions of this thesis, we look at a particular class of Bayesian methods: Gaussian Processes [Rasmussen and Williams, 2006]. Here we give an overview of their formulation, the role that mean and covariance functions play within regression, the connections between GPs and reproducing kernel Hilbert spaces (RKHS) and we further discuss their unsupervised extension known as Gaussian Process-Latent Variable Models [Rasmussen and Williams, 2006; Lawrence, 2005]. Finally, we show how ML in general, and GPs in particular, can be combined with (ordinary and

stochastic) differential equations to build interpretable models to describe many phenomena in science and engineering. Last, we provide a high-level overview of the methods and contributions which in the past tackled the problem of state and parameter inference.

In its central section, following the format of the integrated thesis, every chapter of this thesis contains one of the main novel contributions here presented. Each of them is shown in the camera-ready format of the venue where the corresponding paper has been published.

Chapter 3 presents the first of two papers that interpret the denoised states yielded by Gaussian Process regression as a latent space. Here we address the important problem of parameter inference in ordinary differential equations which, especially in a data-scarce setting, appears in engineering and many applied sciences. This problem can be stated as follows: we receive noisy observations of a time series, along with a set of plausible ODEs that could have generated the latter data. These equations, derived by a practitioner through expert knowledge and first principles, are in turn governed by a set of unknown parameters. The objective is to infer the denoised states of the time series and estimate the parameters that best describe the time series, in order to be able to accurately predict future values. For this purpose, we introduce a novel generative modeling approach based on Gaussian Processes, in which we incorporate the derivative information, given by the ODE system, as additional constraints. We then go ahead and formulate an appropriate likelihood that is optimized in the context of a computationally and data efficient algorithm for state and parameter inference. We validate our scheme (named *ODIN*, as in ODE-Informed GP regression) through an extensive set of experiments, in which our approach shows state-of-the-art results for state and parameter inference, both in terms of accuracy and computational cost. Thanks to the virtual lack of hyperparameters that need to be manually tuned, it also shows promising results for the much more challenging problem of model selection.

The second contribution is found in Chapter 4. Here we move from ordinary differential equations to the more challenging setting of *stochastic* differential equations. Informally, the main difference that separates SDEs from ODEs is a further source of noise in the derivative term, which effectively makes the system dynamics chaotic and random: the solution of an SDE is indeed a stochastic process. The problem is fairly similar to the one presented in the previous chapter: we are given noisy observations of a the time series, together with a set of SDEs which we assume to have generated the data. The objective is to infer the drift and diffusion matrices that parameterize the above SDE system, in order to be able to model its dynamics in their fullness and make predictions. Since SDEs represent a widely used modeling tool, many methods that rely on various discretization and numerical integration schemes were developed in the past. In this chapter, we extend the gradient matching framework for parameter inference in systems of ODEs [Varah, 1982; Calderhead et al., 2009; Dondelinger et al., 2013] to the SDE case. To the best of our knowledge, this is the first attempt at such task. We first propose to leverage the Doss-Sussmann transformation [Doss, 1977; Sussmann, 1978] and construct a latent space for GP regression in which dynamics and observational noise are separated. We then estimate the desired parameters by deploying state-of-the-art adversarial [Goodfellow et al., 2014] and moment matching [Gretton et al., 2012] inference techniques (respectively named *AReS* and *MaRS*), circumventing the need for the expensive discretization schemes of more classical approaches. This yields significant improvements in accuracy and robustness for parameter inference given random initial guesses. On four established benchmark systems, we validate and compare the performance of our algorithms to state-of-the-art solutions based on extended Kalman filtering and Gaussian processes.

Next, we leave this heavily model-based perspective to focus instead on the inference methods that support the tools of Bayesian Machine Learning. The performances of the algorithms presented above, together with all Bayesian approaches, rely on the success of sampling- or optimization-based procedures that are employed in inference and training. Examples range from the various flavours of

gradient-based optimization [Duchi et al., 2011; Zeiler, 2012; Kingma and Ba, 2014] to Markov Chain Monte Carlo sampling [Hastings, 1970]. In high-dimensional settings, typically found nowadays when tackling real-world challenges, issues such as slow-mixing, non-convexity and correlations hinder efficiency. In Chapter 5, we investigate the assumption that, in these settings, optimization and sampling effectively take place on an underlying, lower-dimensional manifold. We propose *AdaGeo*, an algorithm that uses Gaussian Process Latent Variable Models (GP-LVM) to identify such embeddings and adaptively learn the geometry of the parameter space during optimization and sampling. Subsequent samples or optimization steps are then proposed on the basis of the geometry of the manifold. We show how this meta-framework can be applied to optimizers such as stochastic gradient descent and samplers such as stochastic gradient Langevin dynamics [Welling and Teh, 2011] and stochastic gradient Riemannian Langevin dynamics [Patterson and Teh, 2013]. In a series of experiments, we show how this method can in principle achieve performance improvements for both optimization and sampling.

Finally, in Chapter 6, we restate the results achieved in the original contributions of this thesis and suggest possible directions for future work.

1.2 Publications

The contributions described above collect the work that has been published in the following conference papers:

- Philippe Wenk*, **Gabriele Abbati***, Michael A Osborne, Andreas Krause, Bernhard Schölkopf and Stefan Bauer. ODIN: ODE-Informed Regression for Parameter and State Inference in Time-Continuous Dynamical Systems. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*. New York, NY, USA (2020).

* denotes equal contribution.

- **Gabriele Abbati***, Philippe Wenk*, Michael A Osborne, Andreas Krause, Bernhard Schölkopf and Stefan Bauer. AReS and MaRS - Adversarial and MMD-Minimizing Regression for SDEs. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*. Long Beach, California (2019).

- **Gabriele Abbati**, Alessandra Tosi, Michael A Osborne and Seth Flaxman. AdaGeo: Adaptive Geometric Learning for Optimization and Sampling. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS)*. Lanzarote, Spain (2018).

This thesis is presented in *integrated* format: the central chapters contain the camera-ready versions of the above papers, each followed a Statement of Authorship that details my personal contributions to the papers.

1.3 Code Implementation

For each of the above contributions, the code that implements the corresponding algorithms and produces all the experimental results is publicly available. At the time of publication, all relevant software was uploaded to online repositories on the `github` website, to promote transparency and to facilitate further research.

For the three papers in this thesis, the code implementation can be found at the following URLs:

- **ODIN**: `https://github.com/gabb7/ODIN`
- **AReS and MaRS**: `https://github.com/gabb7/AReS-MaRS`
- **AdaGeo**: `https://github.com/gabb7/AdaGeo`

This dissertation further includes a basic *user guide* that shows the high-level code workflow and helps extending the code for comparison and future research efforts. This can be found in appendix A.

2

Literature Review

In this chapter we provide a general overview of the background needed for the works later presented. We first introduce the reader to probabilistic Machine Learning, highlight the differences between discriminative and generative learning and present the Bayesian perspective within ML. Then, we discuss some of the main algorithms for optimization and sampling that are needed to perform approximate inference in the context of Bayesian ML. Afterwards, we introduce the problem of model learning and its connections with model-data divergence. Next, in order to provide a solid background for the single contributions found in the next chapters, we present a particular class of Bayesian models, namely Gaussian Processes, together with their unsupervised extension Gaussian Process Latent Variable Models. Last, we show how Machine Learning models can be augmented by incorporating model-based knowledge based on dynamical differential equations.

2.1 Probabilistic Machine Learning

2.1.1 The Prediction Problem

The tools and algorithms of Machine Learning automatically extract and exploit patterns in data to predict future behaviours of quantities of interest. More precisely, one of the main scopes of ML methods is to leverage known information to provide accurate forecasts in previously unseen circumstances, i.e. a data point that has not been observed before. Examples where these settings can be found range from image understanding (for instance image subject classification) to time series forecasting. Within the context of *supervised learning* [Murphy, 2012], the prediction problem can be formulated as follows. Assume we are given a training dataset $\mathcal{D} = \{\mathbf{x}_n, y_n\}_{n=1}^N$, where each $\mathbf{x}_n \in \mathcal{X}$ represents the input data as seen by the algorithm and $y_n \in \mathcal{Y}$ encodes the desired (usually scalar) output that has to be connected to the corresponding $\mathbf{x}_n$. This input-output framework that includes a clear target (the variables y_n) is what defines learning with supervision.

2.1.2 Discriminative and Generative Models

A common choice to tackle the problem described above is to use *discriminative* learning approaches [Vapnik et al., 1998; Breiman et al., 2001], such as neural networks [Rumelhart et al., 1988; Bishop et al., 1995; Goodfellow et al., 2016] and support vector machines [Cortes and Vapnik, 1995; Scholkopf and Smola, 2001]. In essence, these algorithms learn a parameterized mapping $f_{\boldsymbol{\theta}}$, such that $y = f_{\boldsymbol{\theta}}(\mathbf{x})$, while avoiding further assumptions about the generative mechanism behind data. The vector $\boldsymbol{\theta} \in \Theta$ denotes the parameters that govern the mapping $f_{\boldsymbol{\theta}}$ and define the model itself (for example, the weights in a neural network or the kernel hyperparameters in a support vector machine). The process of identification of the optimal parameters $\boldsymbol{\theta}^*$ that best describe the dataset $\mathcal{D} = \{\mathbf{x}_n, y_n\}_{n=1}^N$ is referred to as *training*. Usually, $\boldsymbol{\theta}^*$ is found by direct minimization of an empirical predictive loss function $\mathcal{L}$ over the training set. As a simple example, neural networks can be

trained for regression problems by minimization of the mean squared error over the training set:

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(f_{\boldsymbol{\theta}}(\mathbf{x}_n), y_n) = \frac{1}{N} \sum_{n=1}^{N} (y_n - f_{\boldsymbol{\theta}}(\mathbf{x}_n))^2 \,. \tag{2.1}$$

Discriminative methods focus on flexibility and broad applicability and present reduced parameter tuning required from the user when compared to the generative approaches we later discuss. Of course, these approaches are not free from downsides. Because of the weak assumptions they make in their formulations, it is hard to include useful prior knowledge in the problem; settings where only little or skewed data is available can thus become highly problematic. Moreover, focusing only on the prediction task can be restrictive when trying to extract a deeper understanding from data. Finally, because of their nature, most discriminative methods fail to provide effective measures of uncertainty quantification at prediction time: in other words, the user cannot know a priori how likely the predicted $f_{\boldsymbol{\theta}^*}(\mathbf{x}_n)$ will be close to the true y_n.

The pitfalls listed above motivate the need for methods that enable the user to have more information and subsequently control on the model. These can be found within *generative* modelling approaches [Ng and Jordan, 2002; Bishop, 2006]. In contrast to discriminative learning, where we focus on learning a predictor, these algorithms assume the existence of a probabilistic model that explains the underlying generative mechanisms that gave rise to the observed data. Put differently, they model the joint likelihood $p(\mathbf{x}, y)$ of the input data $\mathbf{x}$ and the target outputs y. From the joint likelihood, it is straightforward to compute the conditional probability $p(y \mid \mathbf{x})$. The latter can be interpreted as a probabilistic variant of the mapping $f : \mathcal{X} \to \mathcal{Y}$.

While it is often reasonable that one should address the prediction problem directly rather than look at a more general problem as an intermediate step [Vapnik et al., 1998], this generative methodology arguably has its perks. The stronger assumptions, typical of generative approaches, may limit modeling flexibility but at

the same time they allow for prior, domain-specific knowledge to be incorporated into the model and effectively boost performances. Moreover, the same assumptions can achieve a deeper understanding of the data generating process and better uncertainty quantification than discriminative approaches. Naturally, generative models show some drawbacks as well: when data is particularly complex, it is difficult for the user to explicitly describe probabilistic dependencies; at the same time, in big data settings, training becomes hard and approximations are needed, inevitably affecting predictive performances.

2.1.3 Bayesian Machine Learning

In the previous section, we motivate the need for a probabilistic framework within the tools of Machine Learning, highlighting the practical relevance of uncertainty quantification and prior knowledge. As already mentioned in Chapter 1, we now proceed to introduce Bayesian Machine Learning.

The philosophy behind Bayesian learning consists of a few steps: first, we need to choose a generative model (or a set of models) that we consider suitable to describe the system we are interested in, and that encodes the prior information we have on the problem. Second, we *condition* the model on the observed data given in the training set $\mathcal{D}$. Finally, we derive a new model that will assimilate the information contained in the data and adjust its predictions accordingly. The process of characterization of this new, data-informed model is referred to as *Bayesian inference*.

We now introduce the Bayesian framework in a more quantitative fashion. Consider the prediction problem as presented in Section 2.1.1. In the case of discriminative models, in order to achieve a satisfying predictive power we need to identify a good model configuration, i.e. the optimal parameters $\boldsymbol{\theta}^*$. On the contrary, the Bayesian paradigm proposes to ignore point estimates to instead look at the whole parameter space Θ at once. First, we encode the domain-specific prior knowledge potentially available with a *prior distribution* $p(\boldsymbol{\theta})$: this probability measure expresses our prior beliefs about the different values of $\boldsymbol{\theta}$, before looking

at the data. Second, given the training dataset $\mathcal{D}$, we need to describe how likely it has been generated by the model corresponding to the parameter set $\boldsymbol{\theta}$. We do this through the *likelihood* $p(\mathcal{D} \mid \boldsymbol{\theta})$. Finally, the prior distribution and the model likelihood can be combined with Bayes' rule, which yields the *posterior distribution* over $\boldsymbol{\theta}$:

$$p(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})} = \frac{p(\mathcal{D} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})}{\int_{\Theta} p(\mathcal{D} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})\mathrm{d}\boldsymbol{\theta}}. \tag{2.2}$$

The posterior in Eq. 2.2 indicates how well a certain model configuration (parameterized by $\boldsymbol{\theta}$) explains the observed set $\mathcal{D}$. The denominator is called *marginal likelihood* (or *model evidence*) and is necessary for the function $p(\boldsymbol{\theta} \mid \mathcal{D})$ to represent a valid probability distribution. It is important to notice that the two factors in the numerator can compensate each other: when we have a strong prior skewed towards certain regions of the space Θ, it takes clear evidence from data to correct that belief. At the same time, if the specialist is not certain about the outcome of the training process, they can use a non-informative prior that will give priority to the data in $\mathcal{D}$.

Bayes' rule can be applied in an iterative fashion to progressively improve the posterior distribution as new data becomes available. This is the concept of *updating beliefs*. Intuitively, the Bayesian framework allows to continuously update the posterior as new data is received, without losing past information or wasting computation. Consider a training dataset $\mathcal{D}$ that is split into two parts, i.e. $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2\}$; our model will be first trained on $\mathcal{D}_1$ and only subsequently on $\mathcal{D}_2$. This is a classic online learning setting, where the dataset is streamed in chunks. After the first inference, executed through Eq. 2.2, we can use the resulting posterior as prior for the second one and incorporate in this way both parts of the dataset into a single model:

$$p(\boldsymbol{\theta} \mid \mathcal{D}_1, \mathcal{D}_2) = \frac{p(\mathcal{D}_2 \mid \boldsymbol{\theta}, \mathcal{D}_1)p(\boldsymbol{\theta} \mid \mathcal{D}_1)}{p(\mathcal{D}_2 \mid \mathcal{D}_1)} = \frac{p(\mathcal{D}_2 \mid \boldsymbol{\theta}, \mathcal{D}_1)p(\mathcal{D}_1 \mid \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D}_2 \mid \mathcal{D}_1)p(\mathcal{D}_1)}. \tag{2.3}$$

After having defined the Bayesian posterior distribution over the model parameters $\boldsymbol{\theta}$, we now discuss how to use it in prediction tasks. Depending on

the philosophy embraced by the user, there are different methodologies for this purpose.

After the definition of the model and subsequently of the posterior distribution, there are several ways in which inference can be executed. Since leveraging the posterior can be computationally demanding (more details on this can be found in the later sections), there are some approximations and shortcuts that can be used. Closer to discriminative approaches, we can instead identify the set of optimal parameters $\boldsymbol{\theta}^*$ that best describe the data according to the model. For unseen points $\tilde{\mathbf{x}}$, predictions will then be computed through the probabilistic mapping $p(\tilde{y} \mid \tilde{\mathbf{x}}, \boldsymbol{\theta}^*)$.

First and foremost, the simplest plan of action to estimate the optimal $\boldsymbol{\theta}^*$ consists in maximizing the model likelihood $p(\mathcal{D} \mid \boldsymbol{\theta})$ by optimization. The resulting $\boldsymbol{\theta}^*$ is denoted to as the *maximum likelihood estimator (MLE)*:

$$\boldsymbol{\theta}^*_{\text{MLE}} = \underset{\boldsymbol{\theta}}{\text{argmax}}\, p(\mathcal{D} \mid \boldsymbol{\theta}). \tag{2.4}$$

Even if straightforward to compute, there are downsides to the MLE: first, since it ignores the prior term, it does not actually take advantage of the posterior distribution (which is why using the MLE is not considered a Bayesian practice). Second, there are many settings in which a point estimate might not suffice.

Alternatively to the MLE, one can maximize the full posterior distribution instead of restricting the estimation to the model likelihood. The result is called *Maximum A Posteriori (MAP)*:

$$\boldsymbol{\theta}^*_{\text{MAP}} = \underset{\boldsymbol{\theta}}{\text{argmax}}\, p(\mathcal{D} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta}). \tag{2.5}$$

While the MAP incorporates the available prior information into the objective function, it is also true that we are left once again with a single point estimate, losing in this way much information about the posterior's landscape. Moreover, it is worth noticing that the MAP estimate is strongly dependent on the parameterization of the model.

In contrast to the previous simplistic approaches, a more traditional Bayesian paradigm does not plan to limit the predictive power of the model to single point estimates, but instead leverages the full posterior distribution $p(\boldsymbol{\theta} \mid \mathcal{D})$. The latter can then be used to build the *predictive posterior distribution*:

$$p(\tilde{y} \mid \tilde{\mathbf{x}}, \mathcal{D}) = \int_{\Theta} p(\tilde{y} \mid \boldsymbol{\theta}, \tilde{\mathbf{x}}, \mathcal{D}) p(\boldsymbol{\theta} \mid \mathcal{D}) \mathrm{d}\boldsymbol{\theta}. \tag{2.6}$$

This modus operandi allows to retain information about crucial features of the posterior distributions such as different modes, variance (i.e. uncertainty), etc. Alas, computing the predictive distribution can be troublesome. Indeed, if we recall the denominator in the Bayes' rule in Eq. 2.2:

$$p(\mathcal{D}) = \int_{\Theta} p(\mathcal{D} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta}, \tag{2.7}$$

we can see that the exact solution of this model evidence term is often not analytically tractable. For this reason, estimating the above integral requires sampling-based approximation techniques, which are discussed, among other inference techniques, in Section 2.3.

In the following, we introduce briefly a class of Bayesian models which will have a fundamental role in this thesis, namely Latent Variable Models (LVM). Indeed, the common theme that is visible across each of the three original contributions presented here consists in exploiting the advantages yielded by strong structural assumptions made through LVMs.

2.2 Latent Variable Models

As previously mentioned, a model-based perspective in Machine Learning can present several advantages. In particular, such approaches promise to incorporate prior knowledge on the task, together with a deep, useful understanding of the data-generating process.

In this work we focus on a specific family of probabilistic models, namely Latent Variable Models (LVMs). It is quite difficult to precisely define LVMs as in

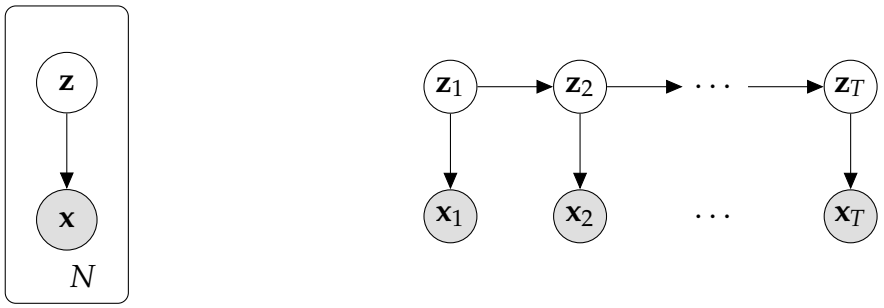
different ways all Bayesian models have some sort of unobserved variable which we are interested in learning. In general, the term LVM is used to indicate the existence, for each data point in the training dataset, of a corresponding latent or hidden variable associated with it. The main assumption behind LVMs is that the variability observed in the realizations of a variable $\mathbf{x}$ can be explained by a mapping between those and a (usually lower-dimensional) latent variable $\mathbf{z}$. A general graphical model (Figure 2.1a) would induce a joint likelihood between these two sets of variables, usually factorized as

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x} \mid \mathbf{z})p(\mathbf{z}) = p(\mathbf{z}) \prod_{n=1}^{N} p(\mathbf{x}_n \mid \mathbf{z}_n). \tag{2.8}$$

There are several reasons why the LVM methodology is convenient. First, since $\mathbf{z}$ is often lower-dimensional than $\mathbf{x}$, LVMs have few parameters that express correlations between the observations in an interpretable and explicit fashion. It is then easy to understand some of the fundamental properties of a data point $\mathbf{x}$ by looking at the corresponding latent representation $\mathbf{z}$. Second, the same $\mathbf{z}$ can be used as a bottleneck, a compressed representation that will maintain the main features of the data, possibly in an interpretable way, and often allow for effective clustering (for example in the case of a one-to-many mapping from $\mathbf{z}$ to $\mathbf{x}$). Depending on the choice of the form of the likelihood $p(\mathbf{x} \mid \mathbf{z})$ and the prior $p(\mathbf{z})$, different models can be derived.

In this thesis, LVMs are used in different settings to compute a latent but interpretable representation for the training dataset. In Chapter 3, the latent variables are the denoised states of GP regression and the parameters of the ODE system. In Chapter 4, in addition to the parameters of the SDE systems, we build a latent state space in which two sources of noise are explicitly separated. Finally, in Chapter 5, LVMs are used to identify a lower-dimensional manifold (with known geometry) to improve efficiency of samplers and optimizers.

In the following we give a quick overview of two of the most popular LVM algorithms, the Mixture Model and the Hidden Markov Model (HMM).



(a) General LVM. (b) First-order HMM.

Figure 2.1: Probabilistic graphical models describing a general Latent Variable Model (left) and a first-order Hidden Markov Model.

Gaussian Mixture Model The mixture model assumes that an observed variable $\mathbf{x}$ is generated by a normalized sum of K distributions, usually of the same family but with different parameters. If the parameters $\boldsymbol{\theta}_k$ of all K components are collected in the vector $\boldsymbol{\theta}$, the resulting model has the form

$$p(\mathbf{x}_n \mid \boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k p_k(\mathbf{x}_n \mid \boldsymbol{\theta}_k). \tag{2.9}$$

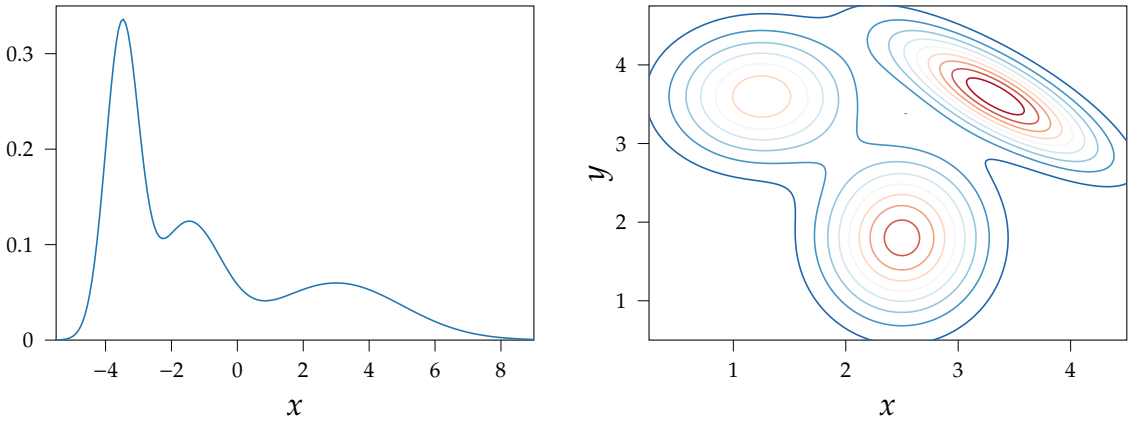
In this case, the corresponding latent variable $\mathbf{z}$ can be interpreted as a categorical indicator that tells us which distribution the corresponding observation has been drawn from.

A popular instance of this framework can be found when the K components have a Gaussian form. We refer to this particular model as Gaussian Mixture Model (GMM). The likelihood of the resulting model can be written as

$$p(\mathbf{x}_n \mid \boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}\left(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right), \tag{2.10}$$

where the vector $\boldsymbol{\theta}$ collects all the parameters $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$, respectively the mean and covariance matrix of the K components. Some examples of GMM densities can be found in Figure 2.2.

Hidden Markov Model Informally, a first-order Markov chain can be thought of as a series of states that show the Markov property, i.e. the probability of the next



(a) 1D GMM (b) 2D GMM

Figure 2.2: Examples of Gaussian Mixture Models. On the left, we can see the probability density function (pdf) of a 1-dimensional GMM; on the right, a contour plot of the pdf generated by a 2-dimensional GMM. Both example mixtures have three components.

state only depends on the current one. For a sequence of T states $\mathbf{x}_1, \ldots, \mathbf{x}_T$, the joint probability distribution can then be factorized as

$$p(\mathbf{x}_1, \ldots, \mathbf{x}_T) = p(\mathbf{x}_1) \prod_{t=1}^{T-1} p(\mathbf{x}_{t+1} \mid \mathbf{x}_t). \tag{2.11}$$

In case there is the concern that the Markov property could limit the expressiveness of the model, it is possible to augment the predictive power of the model by considering states beyond the previous one when computing the probability for the next state, e.g. using $p(\mathbf{x}_{t+1} \mid \mathbf{x}_t, \mathbf{x}_{t-1})$ instead of $p(\mathbf{x}_{t+1} \mid \mathbf{x}_t)$. However, this approach needs a growing number of parameters to model the higher orders.

An alternative method consists in assuming the existence of an underlying, latent Markov chain $\mathbf{z}_1, \ldots, \mathbf{z}_T$ that generated the sequence of observed states $\mathbf{x}_t$. This is the main idea behind HMM: we have a latent (hidden) Markov chain $\mathbf{z}$ and an observation model $p(\mathbf{x} \mid \mathbf{z})$ that describes the relationship between the latent and the observed variables. The joint distribution has the form

$$p(\mathbf{x}, \mathbf{z}) = \left(p(\mathbf{z}_1) \prod_{t=1}^{T-1} p(\mathbf{z}_{t+1} \mid \mathbf{z}_t) \right) \left(\prod_{t=1}^{T} p(\mathbf{x}_t \mid \mathbf{z}_t) \right) \tag{2.12}$$

and the corresponding graphical model can be found in Figure 2.1b.

2.3 Computational Techniques for Inference and Training

In the previous sections we discuss the introduction of a probabilistic framework in the context of Machine Learning and illustrate the main differences between discriminative and generative models, particularly within inference and training. Moreover, we introduce Latent Variable Models as a crucial point of view for this thesis.

As already mentioned in Section 2.1.3, Bayesian (but also non-Bayesian) ML methods often require numerical approximations when training or performing inference. In such cases, the performance of the algorithms deeply rely on the successful application of optimization or sampling techniques.

In order to provide a solid background for this dissertation, in this section we introduce the most well-established algorithms for these purposes. Some of the methods here presented will then be subject of research in Chapter 5, where we propose a meta-algorithm framework that can be applied to virtually any optimization or sampling method to improve efficiency.

2.3.1 Optimization Algorithms

First, we discuss the optimization algorithms that support the deployment of discriminative methods. Let us recall how these approaches address the prediction problem. Given the inputs $\mathbf{X} = \{\mathbf{x}_n\}_{n=1}^N$ and the desired outputs $\mathbf{Y} = \{y_n\}_{n=1}^N$, discriminative methods learn a parameterized mapping $f_{\boldsymbol{\theta}}(\mathbf{x})$ such that $y = f_{\boldsymbol{\theta}}(\mathbf{x})$. In this case, learning corresponds to identifying the optimal set of parameters $\boldsymbol{\theta}^*$ that minimizes an empirical predictive loss function $\mathcal{L}(f_{\boldsymbol{\theta}}(\mathbf{x}_n), y_n)$ over the training set:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{n=1}^{N} \mathcal{L}\left(f_{\boldsymbol{\theta}}(\mathbf{x}_n), y_n\right). \tag{2.13}$$

The final objective consists in building a mapping that performs well on new data $\tilde{\mathbf{x}}$, unseen at training time, by predicting the outputs $\tilde{y} = f_{\boldsymbol{\theta}^*}(\tilde{\mathbf{x}})$. In this view, the

learning problem is expressed as an optimization problem. In the following, we show some of the main techniques currently used in Machine Learning for this purpose. Without loss of generality, we strictly consider minimization settings. Our focus is on gradient-based optimization approaches, as these will be of most relevance to this work.

First Order Methods

We can start this discussion from a simple observation: the gradient of a scalar function $\mathcal{L}$ with respect to the optimization parameters $\boldsymbol{\theta}$, denoted as $\nabla_{\boldsymbol{\theta}}\mathcal{L}$, indicates the direction of steepest ascent of the function. Intuitively, when starting from an initial guess $\boldsymbol{\theta}_0$, it makes sense to iteratively follow the negative direction of the gradient until convergence to a minimum: this method is called *gradient descent* and the corresponding update rule can be written as

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \sum_{n=1}^{N} \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(f_{\boldsymbol{\theta}}(\mathbf{x}_n), y_n\right), \tag{2.14}$$

where η is the *learning rate* and determines the size of the gradient step.

Crucially, we can observe that every iteration of gradient descent requires a sweep over the entire dataset, necessary when computing the sum in Eq. 2.14. This becomes inconvenient in settings where we need to deal with large datasets, often encountered in industrial applications. For this reason, *Stochastic Gradient Descent* (SGD) [Robbins and Monro, 1951] proposes to estimate the gradients by evaluation on a randomly selected mini-batch of M data points, where $M \ll N$. The update rule then becomes

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \frac{N}{M} \sum_{m=1}^{M} \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(f_{\boldsymbol{\theta}}(\mathbf{x}_m), y_m\right). \tag{2.15}$$

Crucially, SGD promises almost sure convergence, as long as an opportune diminishing schedule η_t is developed; keeping it constant would not be enough. The necessary conditions are

$$\sum_t \eta_t^2 < \infty, \qquad \sum_t \eta_t = \infty. \tag{2.16}$$

A typical example for η_t is the exponential decay: $\eta_t = \eta_0 e^{-k_0 t}$, where η_0 and k_0 are hyper-parameters that need to be chosen appropriately.

Because of the recent successes that derived from SGD applications [Bottou, 2010], there have been several efforts towards the development of more sophisticated, faster-converging variants of stochastic gradient descent [Ruder, 2016].

One of the flaws addressed by the modern evolutions of SGD is the ineffectiveness to navigate landscapes where the surface of the objective function curves more steeply in one direction than others [Sutton, 1986], a scenario that is quite common around local optima. To overcome this issue, it is possible to add *momentum* to the gradient by adding a fraction γ (commonly seen in the literature as equal to 0.9) of the past update vector to the current one [Nesterov, 1983]:

$$\boldsymbol{\nu}_t = \gamma \boldsymbol{\nu}_{t-1} + \eta_t \nabla_{\boldsymbol{\theta}} \mathcal{L}, \tag{2.17}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \boldsymbol{\nu}_t. \tag{2.18}$$

For brevity, the term $\nabla_{\boldsymbol{\theta}} \mathcal{L}$ refers to the estimation of the gradient at time t, which can be obtained using a mini-batch procedure as seen above. Intuitively, momentum encourages descent along directions for which the gradients point in the same direction for several iterations while penalizing gradients that change often.

Alternatively to adding momentum, one can think of automatically adjusting the learning rate, component-wise with respect to the optimization parameters. This is the rationale behind *AdaGrad* [Duchi et al., 2011], widely employed in large networks training. This approach accumulates gradient magnitudes and uses them to adjust the learning rate accordingly, eliminating the need for manual updates. More formally, let us define the gradient of the loss function $\mathcal{L}$ with respect to each parameter θ_i at time step t as

$$g_{t,i} = [\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t)]_i. \tag{2.19}$$

Then, the update rule for AdaGrad is the following:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i} + \epsilon}} \cdot g_{t,i}, \tag{2.20}$$

where $G_{t,i} = \sum_{j=1}^{t} g_{j,i}^2$ and ϵ is a smoothing parameter. Unfortunately, the squared accumulator term in the denominator increases monotonically during training, to the point that updates becomes infinitesimally small.

There are different approaches that address this problem. For instance, Zeiler [2012] propose *AdaDelta*, which replaces the term $\mathbf{G}_t$ with the decaying average of all past squared gradients:

$$\mathbb{E}[\mathbf{g}^2]_t = \gamma \mathbb{E}[\mathbf{g}^2]_{t-1} + (1-\gamma)\mathbf{g}_t^2, \tag{2.21}$$

where $\gamma \simeq 0.9$. A similar approach has been developed independently by Hinton et al. [2012].

Finally, we arrive to the SGD variant that is the most popular today for black-box gradient-based optimization, adaptive moment estimation (Adam) [Kingma and Ba, 2014]. Together with a decaying average of the past squared gradients $\mathbf{G}_t$, Adam additionally saves a decaying average of the past gradients $\mathbf{m}_t$:

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t, \tag{2.22}$$

$$\mathbf{G}_t = \beta_2 \mathbf{G}_{t-1} + (1-\beta_2)\mathbf{g}_t^2. \tag{2.23}$$

In line with previous methods, the update rule is

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{\mathbf{G}_t + \epsilon}}\mathbf{m}_t. \tag{2.24}$$

Adam computes individual adaptive learning rates for the different components of $\boldsymbol{\theta}$ from estimates of the first and second moments of the gradients.

Second Order Methods

So far we discussed first order methods, where the optimization process only considers the gradients of the objective function. Despite their computational convenience, these approaches present some fundamental flaws: correlations between the single parameters in $\boldsymbol{\theta}$ are ignored by mere gradients; only sublinear convergence is promised; parameter tuning is often required as first order methods are not scale-invariant.

It is possible to address these issues and go beyond the first order by including in the update rule the Hessian matrix of the objective function with respect to the parameters $\boldsymbol{\theta}$. This denotes a class of optimization algorithms called *second order methods*. In the following, we illustrate some of the most popular approaches falling under this classification.

The classic example of a second order method is *Newton's method*, which attempts to find the roots of $\nabla_{\boldsymbol{\theta}}\mathcal{L}$, where $\nabla_{\boldsymbol{\theta}}\mathcal{L} = \mathbf{0}$. At each iteration, the second order Taylor expansion of $\mathcal{L}$ around $\boldsymbol{\theta}_t$ can be written as:

$$\mathcal{L}(\boldsymbol{\theta}_t + \Delta\boldsymbol{\theta}) \simeq \mathcal{L}(\boldsymbol{\theta}_t) + \nabla\mathcal{L}(\boldsymbol{\theta}_t)^\top \Delta\boldsymbol{\theta} + \frac{1}{2}\Delta\boldsymbol{\theta}^\top \mathbf{H}\mathcal{L}(\boldsymbol{\theta}_t)\Delta\boldsymbol{\theta}, \tag{2.25}$$

where $\mathbf{H}\mathcal{L}(\boldsymbol{\theta}_t)$ is the Hessian matrix of $\mathcal{L}$ evaluated at $\boldsymbol{\theta}_t$. We then look for a stationary point for which $\frac{\mathrm{d}}{\mathrm{d}\Delta\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_t + \Delta\boldsymbol{\theta}) = 0$. With some algebraic effort we can derive the iterative scheme

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \left[\mathbf{H}\mathcal{L}(\boldsymbol{\theta}_t)\right]^{-1} \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_t). \tag{2.26}$$

As we can see, there is no hyper-parameter needed for the execution of Newton's algorithm, apart from the choice of the initial guess $\boldsymbol{\theta}_0$. Nevertheless, in many settings the estimation of the Hessian matrix is not feasible, either analytically or computationally.

The observation above motivates the need for second order optimization algorithms that include approximation methods for the Hessian matrix and avoid its actual calculation. These are called *quasi-Newton* methods: one of the most popular examples is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm, which we will illustrate in the following.

Similarly to the previously described Newton method, BFGS starts from an initial guess $\boldsymbol{\theta}_0$, this time coupled with a guess of the Hessian matrix $\mathbf{B}_0$. At each iteration t, we first need to choose the direction $\mathbf{p}_t$ in which we will search for the optimum by solving the equation

$$\mathbf{B}_t \mathbf{p}_t = -\nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_t). \tag{2.27}$$

Next, we perform a one-dimensional line search to find the right step size α_t:

$$\alpha_t = \underset{\alpha}{\operatorname{argmin}}\, \mathcal{L}(\boldsymbol{\theta}_t + \alpha \mathbf{p}_t), \tag{2.28}$$

which we then use to compute the intermediate secant $\mathbf{s}_t = \alpha_t \mathbf{p}_t$. At this point, we can update the estimation of the optimal $\boldsymbol{\theta}^*$ and the approximation of the Hessian $\mathbf{B}_t$:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \mathbf{s}_t, \tag{2.29}$$

$$\mathbf{r}_t = \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_{t+1}) - \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t), \tag{2.30}$$

$$\mathbf{B}_{t+1} = \mathbf{B}_t + \frac{\mathbf{r}_t^\top \mathbf{r}_t}{\mathbf{r}_t^\top \mathbf{s}_t} - \frac{\mathbf{B}_t \mathbf{s}_t \mathbf{s}_t^\top \mathbf{B}_t}{\mathbf{s}_t^\top \mathbf{B}_t \mathbf{s}_t}. \tag{2.31}$$

BFGS is not guaranteed to converge unless the function admits a good quadratic approximation near the optimum, but practically it performs satisfyingly well even when this assumption is relaxed.

After looking at the main algorithms that tackle the optimization problems within Machine Learning, we will now move our attention to more Bayesian settings, where sampling-based approximate inference techniques are needed.

2.3.2 Sampling Methods for Bayesian Inference

Let us recall what the inference process entails in the context of Bayesian Machine Learning. As illustrated in Section 2.1.3, the application of Bayes' rule to the prior distribution and the model likelihood yields the analytical expression for the posterior distribution over the parameters of interest $\boldsymbol{\theta}$:

$$p(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(\mathcal{D})}. \tag{2.32}$$

In the traditional Bayesian practice, we would like to take full advantage of the information carried by the posterior. The main obstacle to this purpose is represented by the denominator in the above ratio, which we can write as

$$p(\mathcal{D}) = \int_{\Theta} p(\mathcal{D} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta} \tag{2.33}$$

and acts as a normalization constant in the posterior. The distribution $p(\mathcal{D})$ is the expectation of the model likelihood under the prior $p(\boldsymbol{\theta})$, hence the name *marginal likelihood*.

When $p(\mathcal{D})$ is known, then the posterior $p(\boldsymbol{\theta} \mid \mathcal{D})$ can be expressed analytically. This is often not the case, as unfortunately the calculation of the integral in Eq. 2.33 is not trivial. Sometimes (but quite rarely) it is possible to exploit tricks based on *conjugate priors*. In those cases, the prior $p(\boldsymbol{\theta})$ is chosen on the basis of the likelihood $p(\mathcal{D} \mid \boldsymbol{\theta})$, in such a way that the marginal likelihood is analytically tractable but, more importantly, so that the resulting posterior would be in the same *probability distribution family* (e.g. exponential) as the prior. This strategy is mainly of algebraic convenience: the choice of the prior is indeed hardly based on the problem but rather than on the model likelihood. As a consequence, priority is not put on the modeling performances.

When analytical integration is not available, we need to resort to *numerical methods* and settle for an approximation of the posterior distribution. We denote the process of approximating the posterior as *approximate inference*. There are several ways to tackle this problem: in the next section we focus on sampling-based methods, starting from Monte Carlo integration.

Monte Carlo

Monte Carlo (MC) methods [Metropolis and Ulam, 1949] represent one of the most popular class of algorithms whose applications span all fields of science and engineering. The core idea behind Monte Carlo consists in developing strategies to compute randomized samples that can be used to characterize a probability distribution and avoid the approximations needed to formulate an analytical solution. In the context of Bayesian inference, MC applications focus on the approximation of the posterior distribution $p(\boldsymbol{\theta} \mid \mathcal{D})$ by proposing samples that will be used to numerically estimate expectations.

Let us start by considering the problem of computing the expectation of a target function $f(\boldsymbol{\theta})$ under a posterior distribution $p(\boldsymbol{\theta} \mid \mathcal{D})$:

$$I \triangleq \mathbb{E}_{p(\boldsymbol{\theta}|\mathcal{D})}[f(\boldsymbol{\theta})] = \int_{\Theta} f(\boldsymbol{\theta}) p(\boldsymbol{\theta} \mid \mathcal{D}) \mathrm{d}\boldsymbol{\theta}. \tag{2.34}$$

For instance, this integral has to be evaluated when performing prediction in a Bayesian setting. In the context of Gaussian Processes, more details about how the posterior distribution is used at prediction time can be found in Section 2.5.2. When the integral I cannot be solved analytically, it can be approximated using the unbiased Monte Carlo estimator $\hat{I}$, defined as

$$I \approx \hat{I} \triangleq \frac{1}{S} \sum_{s=1}^{S} f(\boldsymbol{\theta}_s) \qquad \text{where} \qquad \boldsymbol{\theta}_s \sim p(\boldsymbol{\theta} \mid \mathcal{D}). \tag{2.35}$$

It is important to notice that most MC methods do not require the normalized posterior $p(\boldsymbol{\theta} \mid \mathcal{D})$. Instead, we can provide the unnormalized distribution $p(\mathcal{D} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})$, which is relatively easy to compute.

Perhaps the simplest MC method that addresses the above problem is *rejection sampling* [Murphy, 2012], where we introduce a proposal distribution $q(\boldsymbol{\theta})$ that is easy to sample from, satisfies $Mq(\boldsymbol{\theta}) > p(\mathcal{D} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})$, and provides an upper envelope for the unnormalized posterior distribution. The algorithm then goes as follows: we sample $\boldsymbol{\theta} \sim q(\boldsymbol{\theta})$ and $u \sim U(0, 1)$. If $u > \frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{Mq(\boldsymbol{\theta})}$ the sample is rejected, otherwise it is accepted. A graphical representation of the rejection sampling algorithm can be found in Figure 2.3.

The idea behind rejection sampling consists in exploiting a proposal q that can be easily sampled from: the closer it is to $p(\mathcal{D} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})$ (or equivalently the smaller M), the more efficiently the method can perform. In order to partially automate the choice of the proposal, Gilks and Berzuini [2001] introduce *adaptive rejection sampling* (ARS), where q is updated at runtime to improve the tightness of the envelope.

Another popular MC integration method for computing the estimator in Eq. 2.35 is *importance sampling* (IS) [Murphy, 2012], illustrated in Figure 2.4. The main

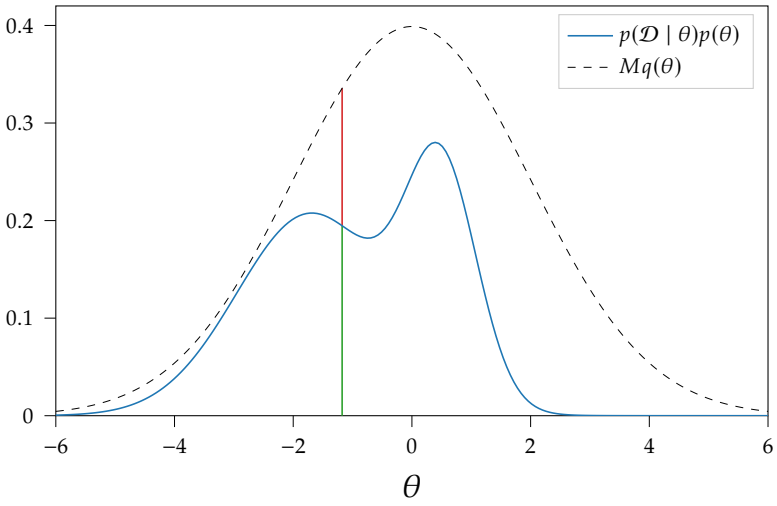


Figure 2.3: Illustration of one-dimensional rejection sampling. At each iteration, a candidate sample is drawn from the proposal distribution, i.e. $\theta \sim q(\theta)$. Assuming the drawn value of θ is the one corresponding to the vertical line in the figure, we can then visualize the ratio between the probabilities of acceptance (in green) and rejection (in red) when sampling the variable $u \sim U(0, 1)$.

idea behind this algorithm consists in achieving efficiency by drawing samples from areas of Θ where the density $p(\mathcal{D} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})$ is high but also where the function $f(\boldsymbol{\theta})$ is large. Fewer samples would then be highly representative for the integral calculation. First, we introduce a proposal distribution $q(\boldsymbol{\theta})$, whose samples are used to estimate the integral

$$\mathbb{E}_{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}\left[f(\boldsymbol{\theta})\right] = \int_{\Theta} f(\boldsymbol{\theta}) \frac{p(\mathcal{D} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})}{q(\boldsymbol{\theta})} q(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta} \tag{2.36}$$

by approximate the expectation as

$$\mathbb{E}_{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}\left[f(\boldsymbol{\theta})\right] \approx \frac{1}{S} \sum_{s=1}^{S} \omega_s f(\boldsymbol{\theta}_s) = \hat{I}, \tag{2.37}$$

where we define the importance weights $\omega_s \triangleq \frac{p(\mathcal{D}|\boldsymbol{\theta}_s)p(\boldsymbol{\theta}_s)}{q(\boldsymbol{\theta}_s)}$.

The choice of the proposal distribution $q(\boldsymbol{\theta})$ needs to be practical for the computation of the integral, but how should we choose it? For example, one can minimize the variance of the MC estimator

$$\hat{I} = \frac{1}{S} \sum_{s=1}^{S} \frac{p(\mathcal{D} \mid \boldsymbol{\theta}_s)p(\boldsymbol{\theta}_s)f(\boldsymbol{\theta}_s)}{q(\boldsymbol{\theta}_s)} \tag{2.38}$$

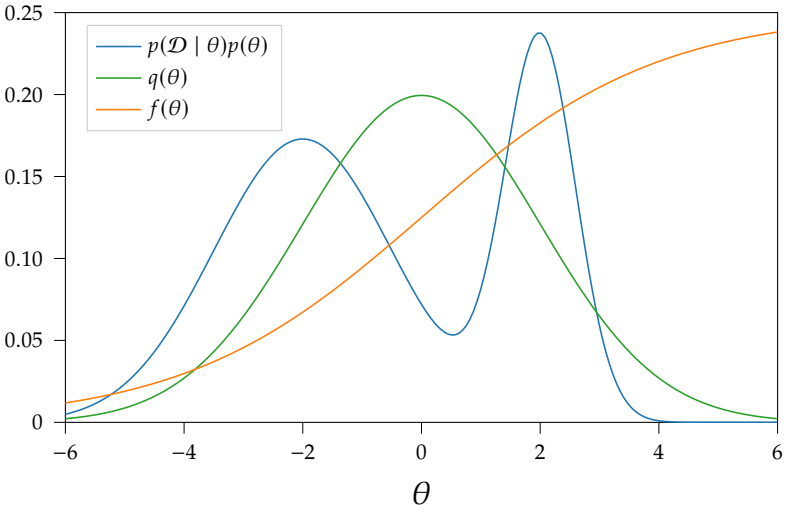


Figure 2.4: Illustration of a one-dimensional example of importance sampling. The objective here is to integrate the function $f(\theta)$ under the unnormalized distribution $p(\mathcal{D} \mid \theta)p(\theta)$. In this case, samples are proposed starting from the simpler proposal distribution $q(\theta)$ and their contribution towards the integral estimation is re-weighted through the ratio $p(\mathcal{D} \mid \theta)p(\theta)/q(\theta)$.

and find the resulting optimal q^* in

$$q^*(\boldsymbol{\theta}) = \frac{|f(\boldsymbol{\theta})|p(\mathcal{D} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})}{\int |f(\boldsymbol{\theta}')|p(\mathcal{D} \mid \boldsymbol{\theta}')p(\boldsymbol{\theta}')\mathrm{d}\boldsymbol{\theta}'}. \tag{2.39}$$

A full derivation can be found in Murphy [2012]. Even though in most settings computing $q^*(\boldsymbol{\theta})$ would not be practically feasible, we can observe that a "good" proposal should make the ratio $\frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})|f(\boldsymbol{\theta})|}{q(\boldsymbol{\theta})}$ as close to a constant as possible. Similarly to ARS, one can think of adapting the proposal distribution during sampling to improve the approximation, as in *adaptive importance sampling* (AIS) [Oh and Berger, 1992].

All the methods we discussed in this section suffer from a problem that is common to many inference algorithms, the *curse of dimensionality* [Bellman, 2015]. Informally, it refers to the fact that all numerical and modeling routines tend to perform poorly in high-dimensional settings. For instance, rejection and importance sampling are actually employed only in very low-dimensional inference settings, as their performances become deeply unsatisfactory otherwise. This is mainly due to distances and algebraic spaces becoming hard to be interpreted by algorithms. The

challenges that arise from the curse of dimensionality are addressed by Markov chains Monte Carlo methods, as explained in the following section.

Markov Chain Monte Carlo

Markov chain Monte Carlo (MCMC) methods [Metropolis et al., 1953; Hastings, 1970; Gilks et al., 1995] are among the most popular sampling approaches that tackle the curse of dimensionality and are arguably the most widely used family of algorithms in Bayesian inference. In addition to their popularity in statistics and Machine Learning, MCMC methods have countless applications in many other fields of science, engineering and physics. The core idea behind MCMC is the construction of a *Markov chain*, a stochastic process whose stationary (or equilibrium) distribution is equal to the (potentially unnormalized) target distribution.

The strategy adopted here to alleviate the sampling routine from the detrimental effects of the curse of dimensionality is to craft *local* updates in order to encourage the sampler to explore areas close to its current position within Θ, as opposed to sample independently from the target distribution. In accordance to the *Markov property*, which we discuss below, the current state only depends on the previous one. The prior knowledge inserted through this framework can be translated to the assumption that in high dimensions the portion of the space Θ with relevant probability is usually very limited.

Let us start by introducing a more formal definition of Markov chain.

Definition 2.3.1 (Markov chain)**.** *A (discrete-time) Markov chain is a sequence of countably infinite random variables (or states) $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \ldots$ with the Markov property, which states that the probability distribution of the next state only depends on the current one and not on previous states:*

$$p\left(\boldsymbol{\theta}_{n+1} = \bar{\boldsymbol{\theta}}_{n+1} \mid \boldsymbol{\theta}_n = \bar{\boldsymbol{\theta}}_n, \boldsymbol{\theta}_{n-1} = \bar{\boldsymbol{\theta}}_{n-1}, \ldots, \boldsymbol{\theta}_1 = \bar{\boldsymbol{\theta}}_1\right) = p\left(\boldsymbol{\theta}_{n+1} = \bar{\boldsymbol{\theta}}_{n+1} \mid \boldsymbol{\theta}_n = \bar{\boldsymbol{\theta}}_n\right). \tag{2.40}$$

The quantities $p\left(\boldsymbol{\theta}_{n+1} = \bar{\boldsymbol{\theta}}_{n+1} \mid \boldsymbol{\theta}_n = \bar{\boldsymbol{\theta}}_n\right)$ are called *transition probabilities*. If every transition follows the same distribution, i.e.

$$p\left(\boldsymbol{\theta}_{n+1} = \bar{\boldsymbol{\theta}}_{n+1} \mid \boldsymbol{\theta}_n = \bar{\boldsymbol{\theta}}_n\right) = p\left(\boldsymbol{\theta}_n = \bar{\boldsymbol{\theta}}_n \mid \boldsymbol{\theta}_{n-1} = \bar{\boldsymbol{\theta}}_{n-1}\right), \tag{2.41}$$

the chain is said to be *homogeneous*. Most MCMC methods rely on homogeneous Markov chains (some exceptions being adaptive MCMC algorithms [Gilks and Wild, 1992; Cappé et al., 2008; Andrieu and Thoms, 2008]).

As mentioned before, we are interested in the *stationary distribution* to which the Markov chain eventually converges. If we want a Markov chain to converge to a target distribution $\pi(\boldsymbol{\theta})$, we require

$$\lim_{n\to\infty} p(\boldsymbol{\theta}_n = \boldsymbol{\theta}) = \pi(\boldsymbol{\theta}), \tag{2.42}$$

for every possible starting state. Two conditions are thus necessary: first, $\pi(\boldsymbol{\theta})$ has to be a stationary distribution for the Markov chain; second, convergence must be theoretically possible from every starting point $\boldsymbol{\theta}_1$ (the *ergodicity* property).

A sufficient but not necessary condition to guarantee that a Markov chain converges to the desired stationary distribution is *detailed balance*. A chain satisfies detailed balance if

$$\pi(\boldsymbol{\theta})T(\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}) = \pi(\boldsymbol{\theta}')T(\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}'), \tag{2.43}$$

where the quantities $T(\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}) = p(\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}' \mid \boldsymbol{\theta}_t = \boldsymbol{\theta})$ are called *transition kernels*. In this case the chain is said to be *reversible*. An ergodic Markov process that satisfies the previous conditions will converge to the target distribution $\pi(\boldsymbol{\theta})$.

The first, simplest and probably most popular example of MCMC is the Metropolis Hastings (MH) algorithm [Hastings, 1970]. Consider an unnormalized target distribution $\tilde{p}(\boldsymbol{\theta})$; at each iteration n, the MH Markov chain proposes updates $\boldsymbol{\theta}'$ according to a proposal distribution $q(\boldsymbol{\theta}' \mid \boldsymbol{\theta}_n)$, crafted over Θ and conditioned on the current state $\boldsymbol{\theta}_n$. The proposal $\boldsymbol{\theta}'$ is then accepted as the new sample $\boldsymbol{\theta}_{n+1}$ with probability

$$p(\text{accept}) = \min\left(1, \frac{\tilde{p}(\boldsymbol{\theta}')q(\boldsymbol{\theta}_n \mid \boldsymbol{\theta}')}{\tilde{p}(\boldsymbol{\theta}_n)q(\boldsymbol{\theta}' \mid \boldsymbol{\theta}_n)}\right). \tag{2.44}$$

It is to be noticed that the choice of the proposal distribution $q(\boldsymbol{\theta}' \mid \boldsymbol{\theta})$ greatly affects the performances of the sampler. For example, if $q(\boldsymbol{\theta}' \mid \boldsymbol{\theta})$ becomes almost independent of the current state $\boldsymbol{\theta}$ (i.e. $q(\boldsymbol{\theta}' \mid \boldsymbol{\theta}) \approx q(\boldsymbol{\theta}')$), the updates would not be local anymore and the issues described above for rejection and importance sampling would reappear. On the other hand, if the proposals are too close to the current state, correlations between samples increase, the chain might get stuck near single modes of the target distribution or show other pathological behaviours [Andrieu et al., 2003].

The challenges that MCMC samplers encounter have been the subject of study of various research efforts: as a consequence, there are several algorithms that build upon the logic of the Metropolis Hastings and improve proposal distributions, for example considering alternative parameterizations through auxiliary variables [Higdon, 1998; Andrieu et al., 2010]. A popular implementation is the Hamiltonian Monte Carlo [Duane et al., 1987; Neal et al., 2011], a particular MCMC scheme that uses Hamiltonian dynamics to reduce inter-sample correlations and explore the objective landscape more efficiently.

Finally, we would like to briefly mention a further class of MCMC methods called Metropolis-adjusted Langevin algorithms (MALA) [Roberts and Rosenthal, 1998; Roberts and Stramer, 2002]. These methods combine Langevin dynamics (a certain technique used for modeling molecular dynamics) to MH-like algorithms. A popular example from this family is Stochastic Gradient Langevin Dynamics (SGLD) sampler [Welling and Teh, 2011]: samples are proposed on the base of a SGD update rule to which appropriately decaying Gaussian noise is added; subsequently, proposals are accepted with MH acceptance probability. After an initial *burn-in* period, the sampler will switch dynamics and access a Langevin regime, during which the Markov chain will directly sample from the (unnormalized) posterior distribution. In SGLD, stochastic optimization seamlessly transitions into posterior sampling. Nevertheless, a certain degree of parameter tuning is required.

As a final remark, extensions of these algorithms on Riemannian manifolds have been recently proposed for example by Girolami and Calderhead [2011]; Patterson and Teh [2013].

2.3.3 Variational Inference

Even though the focus of the last section on Bayesian inference is predominantly on Monte Carlo methods, it is worth mentioning that these do not represent the only possible approach to the inference problem. Indeed, there are alternative solutions that can be preferable in settings when MC convergence is problematic.

In particular, variational inference (VI) [Jordan et al., 1999; Blei et al., 2017] has become an increasingly popular choice when operating in Bayesian settings. The core strategy of VI consists in recasting the inference problem as an optimization problem: after assuming the existence of a valid approximation within a parameterized distribution family, the best candidate is found by iteratively minimizing the distance between the approximate and the actual posterior distributions.

More precisely, consider the following joint likelihood between a set of observed variables $\mathbf{x}$ and a set of latent variables $\mathbf{z}$:

$$p(\mathbf{z}, \mathbf{x}) = p(\mathbf{z})p(\mathbf{x} \mid \mathbf{z}). \tag{2.45}$$

In the Bayesian framework, the latent variables $\mathbf{z}$ (on which we impose a prior $p(\mathbf{z})$) help to parameterize and control the modeling distribution.

As mentioned above, the process of Bayesian inference consists in characterizing in some useful way the posterior distribution $p(\mathbf{z} \mid \mathbf{x})$. The logic behind VI is the following: we first define a *family* of approximate densities $\mathcal{Q}$, where we need to consider a trade-off between convenient analytical properties (for optimization and integration) and flexibility in the representational capabilities. Then, we select the best approximation $q^*(\mathbf{z}) \in \mathcal{Q}$ that minimizes the Kullback-Leibler (KL) divergence [Kullback and Leibler, 1951] to the exact posterior:

$$q^*(\mathbf{z}) = \underset{q(\mathbf{z}) \in \mathcal{Q}}{\operatorname{argmin}} \operatorname{KL}\left(q(\mathbf{z}) || p(\mathbf{z} \mid \mathbf{x})\right), \tag{2.46}$$

where

$$\mathrm{KL}\left(q(\mathbf{z})||p(\mathbf{z} \mid \mathbf{x})\right) = \mathbb{E}_{q(\mathbf{z})}\left[\log q(\mathbf{z})\right] - \mathbb{E}_{q(\mathbf{z})}\left[\log p(\mathbf{z} \mid \mathbf{x})\right]. \tag{2.47}$$

Evaluating the KL divergence implies the computation of the evidence term $p(\mathbf{x})$. This can be easily seen by expanding the integral:

$$\mathrm{KL}\left(q(\mathbf{z})||p(\mathbf{z} \mid \mathbf{x})\right) = \mathbb{E}_{q(\mathbf{z})}\left[\log q(\mathbf{z})\right] - \mathbb{E}_{q(\mathbf{z})}\left[\log p(\mathbf{z}, \mathbf{x})\right] + \log p(\mathbf{x}). \tag{2.48}$$

We can get around this by noticing that the minimization of the KL divergence is equivalent to the maximization of an alternative objective, the evidence lower bound (ELBO):

$$\mathrm{ELBO}(q) = \mathbb{E}_{q(\mathbf{z})}\left[\log p(\mathbf{z}, \mathbf{x})\right] - \mathbb{E}_{q(\mathbf{z})}\left[\log q(\mathbf{z})\right]. \tag{2.49}$$

Looking at Eq. 2.48, it is easy to see that the ELBO is defined as the negative KL divergence plus the evidence term $\log p(\mathbf{x})$. The same quantity can be also rewritten as

$$\begin{aligned} \mathrm{ELBO}(q) &= \mathbb{E}_{q(\mathbf{z})}\left[\log p(\mathbf{z})\right] + \mathbb{E}_{q(\mathbf{z})}\left[\log p(\mathbf{x} \mid \mathbf{z})\right] - \mathbb{E}_{q(\mathbf{z})}\left[\log q(\mathbf{z})\right] && (2.50) \\ &= \mathbb{E}_{q(\mathbf{z})}\left[\log p(\mathbf{x} \mid \mathbf{z})\right] - \mathrm{KL}\left(q(\mathbf{z})||p(\mathbf{z})\right), && (2.51) \end{aligned}$$

which is difference between the expected log-likelihood of the data and the KL divergence between the prior $p(\mathbf{z})$ and the approximation $q(\mathbf{z})$. Many approaches have been proposed to address the problem of the optimization scheme, such as expectation maximization [Dempster et al., 1977], coordinate ascent [Bishop, 2006] and expectation propagation [Minka, 2001; Opper and Winther, 2005].

Variational inference, intended in this classical sense, presents some pitfalls, such as the need for restrictive assumptions over the approximations space $\boldsymbol{Q}$ and long ELBO analytical derivations. A more modern version of VI [Hoffman et al., 2013] tackles these issues by reformulating the problem in a stochastic fashion, leveraging stochastic gradient descent and Monte Carlo estimators for the optimization problem.

2.4 Model Learning

In the previous section we discuss the tools of Bayesian ML and the most popular inference algorithms that are necessary to make predictions about new data.

Nevertheless, when learning a Machine Learning model, we often need to make crucial modeling decisions that range from the choice of distribution class of the model likelihood, to the shape of prior distributions or the important feature of model complexity. Unfortunately, it is easy to encounter pathological behaviours such as overfitting (the model performs well at training time but poorly when predicting unseen data points) and underfitting (the model is too simple and lacks the necessary expressivity to capture the main features in data). The problem of identifying the model that best describes the training data is called *model learning*, or *model selection* [Murphy, 2012].

In settings where enough data is available, a first approach to model selection consists in splitting the overall dataset into *training*, *validation* and *test* sets. First, we train all candidate models on the training set; then, we tune their hyper-parameters on the validation set; finally, we pick the one that scores the lowest error on the validation set and estimate its *generalization error* by measuring their performance on the test set.

When data for training and testing is limited, an alternative can be found in *k-fold cross-validation*. The training dataset is divided into k equal parts, or folds; each model is trained on the first $k-1$ folds and tested on the k-th one; this procedure is repeated k times, by iterating on the folds. At this point, we would choose the model with the best average score over the k test folds. Cross-validation can be easily implemented but, not surprisingly, comes with a significant computational burden (each model needs to be trained k times).

A different path to model selection can be found in *probabilistic model selection*. Its main objective is to design a measure of performance that can be computed with training data only, and that would not be predisposed towards over-fitting. In the

following, we are going to take a look at some paradigms for probabilistic model selection.

2.4.1 Bayesian Model Selection

The Bayesian framework offers a mathematically elegant way to quantitatively compare models' performances. Thanks to Bayes' rule, we can compute the posterior distribution over a model m given a dataset $\mathcal{D}$:

$$p(m \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid m)p(m)}{\sum_{m_i \in \mathcal{M}} p(\mathcal{D} \mid m_i)p(m_i)}, \tag{2.52}$$

where $p(\mathcal{D} \mid m_i)$ is the model likelihood and $p(m_i)$ is the prior distribution assigned by the user to a specific model configuration.

Intuitively, using the likelihood $p(\mathcal{D} \mid m)$ in model selection may favor models that are over-parameterized, but these effects can be dampened by integration and employment of the posterior distribution (thanks to the *Bayesian Occam's razor* [MacKay, 1995; Murray and Ghahramani, 2005]).

Bayesian model selection is only deceptively simple, as there are several obstacles to its application on real world problems. First, it is difficult to define an appropriate prior distribution over the model space, whose choice can heavily influence the effectiveness of the Bayesian approach. Thus, the most common and reasonable resolution consists in deploying an uninformative, uniform prior distribution. The main issue with the latter method is that this distribution will be uniform with respect to models, but not to model characteristics and complexity, hiding implicit assumptions over the hyper-parameters; moreover, the notion of "proximity" between models is canceled. Very similar conclusions can be reached when looking at parameter priors (formally $p(\boldsymbol{\theta} \mid m_i)$). The second hurdle to overcome, once the prior has been defined, is of course the actual computation of the marginal likelihood $p(\mathcal{D})$.

2.4.2 Information Criteria

In the attempt of solving the model selection problem, several *information criteria* have been proposed over the years: the main scope is to measure a model's performance while circumventing the need of prior definition and evidence computation.

A classic method is the Akaike Information Criterion (AIC) [Akaike, 1974]:

$$\mathrm{AIC} = \ln p(\mathcal{D} \mid \boldsymbol{\theta}^*_{\mathrm{MLE}}) - M, \tag{2.53}$$

where $p(\mathcal{D} \mid \boldsymbol{\theta}^*_{\mathrm{MLE}})$ is the likelihood of the data computed with the maximum likelihood estimator $\boldsymbol{\theta}^*_{\mathrm{MLE}}$ and M is the number of adjustable parameters in the model.

Leveraging the Laplace approximation within the Bayesian framework, it is possible to formulate an alternative information criterion. From Bayes' theorem the model evidence is given by

$$p(\mathcal{D}) = \int p(\mathcal{D} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta} \tag{2.54}$$

which, by taking the logarithm and applying the Laplace approximation, can be re-written as

$$\ln p(\mathcal{D}) \simeq \ln p(\mathcal{D} \mid \boldsymbol{\theta}^*_{\mathrm{MAP}}) + \ln p(\boldsymbol{\theta}^*_{\mathrm{MAP}}) + \frac{M}{2} \ln(2\pi) - \frac{1}{2} \ln |\mathbf{A}|. \tag{2.55}$$

Here $\mathbf{A}$ is the Hessian matrix of the negative log-posterior:

$$\mathbf{A} = -\nabla\nabla \ln p(\mathcal{D} \mid \boldsymbol{\theta}^*_{\mathrm{MAP}}) p(\boldsymbol{\theta}^*_{\mathrm{MAP}}) = -\nabla\nabla \ln p(\boldsymbol{\theta}^*_{\mathrm{MAP}} \mid \mathcal{D}), \tag{2.56}$$

and the last three terms of Eq. 2.55 comprise the *Occam factor* that penalizes model complexity.

Assuming a broad, Gaussian prior distribution over the parameters and a full-rank Hessian, Eq. 2.55 can be further approximated as

$$\ln p(\mathcal{D}) \simeq \ln p(\mathcal{D} \mid \boldsymbol{\theta}^*_{\mathrm{MAP}}) - \frac{1}{2} M \ln N, \tag{2.57}$$

where N is the number of data points and additive constants were omitted. The latter equation is known as the Bayesian Information Criterion (BIC) or Schwarz

Criterion [Schwarz et al., 1978], which penalizes model complexity more aggressively than AIC.

Despite seemingly easy to understand and apply, AIC and BIC can become non-effective when used to quantify the performance of singular statistical models, i.e. learning algorithms with hidden variables and/or a hierarchical structure (virtually all models currently used in machine intelligence tasks) [Watanabe, 2009]. Such models are non-identifiable and present a singular Fisher information matrix. Therefore, the conventional relationship between generalization errors and training errors may not hold as expected, either for the maximum likelihood or Bayes estimation. Recently, Watanabe [2010] addresses this problem and proposes a Widely Applicable Information Criterion (WAIC), which generalizes the AIC and promises to hold true for any *true* distribution, parametric model, or prior distribution.

Similarly to the AIC, even the BIC can suffer when dealing with singular models, as the Bayes free energy cannot be approximated effectively by Eq. 2.57 (maximizing the marginal likelihood is equivalent to minimizing Bayes free energy). For this reason, Watanabe [2013] proposes a Widely applicable Bayesian Information Criterion (WBIC) that generalizes the BIC and corrects its asymptotic behaviour.

2.4.3 A Note on Variational Inference and Model Selection

As we showed in Section 2.3.3, alternatively to the minimization of the negative log-likelihood, models can be trained by maximizing the variational expected lower bound (ELBO), described by Eq. 2.51.

From Eq. 2.51, we can formally outline the relationship between the ELBO and the model evidence:

$$\text{ELBO}(q) = \text{Evidence} - \text{KL}\left(q(\mathbf{z})||p(\mathbf{z})\right). \tag{2.58}$$

Even though the ELBO may be employed in model selection [Blei et al., 2017], there is a crucial consideration to be made. When fitting the approximation $q(\mathbf{z})$ to the true posterior, there might exist several equivalently performing modes in the parameter space. If $q(\mathbf{z})$ is fitted through optimization, the algorithm will hopefully

converge to one of the modes while ignoring the others, and no action would need to be taken in this case. On the contrary, if Bayesian inference is employed, all modes would be considered in the integral over the parameter space. Hence, the numerical value of the ELBO would not take into account the different modes and appropriate corrections would be required [Bishop, 2006].

2.4.4 Model Learning and Divergences

In the previous sections, we look at several ways to choose the best model from a pool of candidates on the basis of generalization error estimates, or information criteria. However, it is useful to point out a further formulation of model learning that recalls the generative modeling paradigm as illustrated in Section 2.1.2.

Let us assume that we are given a dataset $\mathbf{X} = \{\mathbf{x}_n\}_{n=1}^N$, where the $\mathbf{x}_n$ are realizations of a variable $\mathbf{x} \in \mathcal{X}$, produced according to some unknown underlying generative process with density $p_{\mathcal{D}}(\mathbf{x})$. For a given model parameterized by a vector $\boldsymbol{\theta}$, we can identify the parameters that best describe the dataset by maximization of the corresponding model marginal likelihood under the data distribution:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x})\right]. \tag{2.59}$$

At this point we can make a crucial observation: solving the problem in Eq. 2.59 is equivalent to minimizing the Kullback-Leibler divergence between the true density of the data generative process and the model evidence, i.e.

$$\underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \mathbb{E}_{p_{\mathcal{D}}(\mathbf{x})} \left[\log p_{\boldsymbol{\theta}}(\mathbf{x})\right] = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \mathrm{KL}\left(p_{\mathcal{D}}(\mathbf{x}) \,||\, p_{\boldsymbol{\theta}}(\mathbf{x})\right). \tag{2.60}$$

This methodology for the learning process is reflected in many approaches that aim at learning a good approximation of $p_{\mathcal{D}}(\mathbf{x})$ on the basis of the samples given in the training dataset. The common strategy consists in parameterizing a family of distributions that can approximate the target $p_{\mathcal{D}}(\mathbf{x})$ through a vector $\boldsymbol{\theta}$ and subsequently identifying the best candidate $p_{\boldsymbol{\theta}^*}(\mathbf{x})$ during training. In this case, to train a model correspond to the characterization of the distribution within the search set that is the "closest" to the objective density. There are several ways

to measure the distance between probability distributions and consequently to minimize it. Among those, we would like to introduce the reader to Maximum Mean Discrepancy, as found in the next section.

2.4.5 Maximum Mean Discrepancy

The Maximum Mean Discrepancy (MMD) [Gretton et al., 2012] is a metric on probabilistic measures that belongs to the class of integral probability metrics [Müller, 1997]. Informally, it quantifies how close two probability densities are to each other, based on the moments of the distributions.

Let us consider two random variables $\mathbf{u}$ and $\mathbf{v}$, which we respectively associate to the probability distributions $\mathbb{P}$ and $\mathbb{Q}$. Integral probability metrics employ a class $\mathcal{F}$ of witness functions to compute a distance between $\mathbb{P}$ and $\mathbb{Q}$:

$$\mathcal{D}_{\mathcal{F}}(\mathbb{P}, \mathbb{Q}) = \sup_{f \in \mathcal{F}} \left(\mathbb{E}_{\mathbb{P}} \left[f(\mathbf{u}) \right] - \mathbb{E}_{\mathbb{Q}} \left[f(\mathbf{v}) \right] \right). \tag{2.61}$$

A specific probability metric is then determined by the choice of the functional family $\mathcal{F}$.

In the case of the MMD, $\mathcal{F}$ is the unit ball in a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$ with positive definite kernel $k(\mathbf{u}, \mathbf{u}')$. The reproducing property of RKHS is the following: $\forall f \in \mathcal{H}, f(\mathbf{u}) = \langle f, k(\mathbf{u}, \cdot) \rangle_{\mathcal{H}}$ [Rudin, 2006]. More details about RKHS and its connections with Gaussian Processes can be found in Section 2.5.4. The Maximum Mean Discrepancy is then defined as:

$$MMD(\mathbb{P}, \mathbb{Q}, \mathcal{H}) = \sup_{f \in \mathcal{H}, ||f||_{\mathcal{H}} < 1} \left(\mathbb{E}_{\mathbb{P}} \left[f(\mathbf{u}) \right] - \mathbb{E}_{\mathbb{Q}} \left[f(\mathbf{v}) \right] \right). \tag{2.62}$$

Gretton et al. [2012] showed that the witness function f^* corresponding to the supremum has the form:

$$f^*(\mathbf{w}) \propto \mathbb{E}_{\mathbb{P}} \left[k(\mathbf{u}, \mathbf{w}) \right] - \mathbb{E}_{\mathbb{Q}} \left[k(\mathbf{v}, \mathbf{w}) \right]. \tag{2.63}$$

Although computing the MMD directly using the formulas above can seem daunting, it can be estimated from samples [Gretton et al., 2007, 2012]. Let us consider a more concrete setting, where we have a random variable $\mathbf{x}$, associated with

two probability distributions, the true data generative process $p_{\mathcal{D}}(\mathbf{x})$ and the model evidence $p_{\boldsymbol{\theta}}(\mathbf{x})$. Notice that we can directly sample from these two densities: in particular, the dataset $\mathcal{D}$ will contain samples from $p_{\mathcal{D}}(\mathbf{x})$, while various sampling techniques (depending on the model complexity) can yield samples from $p_{\boldsymbol{\theta}}(\mathbf{x})$. Given samples $\mathbf{X}^{(\mathcal{D})} = \left\{\mathbf{x}_i^{(\mathcal{D})}\right\}_{i=1}^{M}$ drawn i.i.d. from $p_{\mathcal{D}}(\mathbf{x})$ and $\mathbf{X}^{(\boldsymbol{\theta})} = \left\{\mathbf{x}_i^{(\boldsymbol{\theta})}\right\}_{i=1}^{M}$ drawn i.i.d. from $p_{\boldsymbol{\theta}}(\mathbf{x})$, the empirical witness function is

$$\hat{f}(\mathbf{x}) \propto \frac{1}{M}\sum_{i=1}^{M} k\left(\mathbf{x}_i^{(\mathcal{D})}, \mathbf{x}\right) - \frac{1}{M}\sum_{i=1}^{M} k\left(\mathbf{x}_i^{(\boldsymbol{\theta})}, \mathbf{x}\right). \tag{2.64}$$

An unbiased estimator for the squared MMD is

$$\begin{aligned} M\hat{M}D^2\left(\mathbf{X}^{(\mathcal{D})}, \mathbf{X}^{(\boldsymbol{\theta})}\right) &= \frac{1}{M(M-1)}\sum_{i\neq j}^{M} k\left(\mathbf{x}_i^{(\mathcal{D})}, \mathbf{x}_j^{(\mathcal{D})}\right) + \frac{1}{M(M-1)}\sum_{i\neq j}^{M} k\left(\mathbf{x}_i^{(\boldsymbol{\theta})}, \mathbf{x}_j^{(\boldsymbol{\theta})}\right) \\ &\quad -\frac{2}{M^2}\sum_{i=1}^{M}\sum_{j=1}^{M} k\left(\mathbf{x}_i^{(\mathcal{D})}, \mathbf{x}_j^{(\boldsymbol{\theta})}\right). \end{aligned} \tag{2.65}$$

The kernel function k needs to be chosen a priori. Two of the most popular choices are the Gaussian radial basis function (RBF) kernel:

$$k^{\text{rbf}}(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{||\mathbf{u}-\mathbf{v}||^2}{2l^2}\right) \tag{2.66}$$

and the rational quadratic kernel:

$$k_\alpha^{\text{rq}}(\mathbf{u}, \mathbf{v}) = \left(1 + \frac{||\mathbf{u}-\mathbf{v}||^2}{2\alpha l^2}\right)^{-\alpha}. \tag{2.67}$$

In both cases, the lengthscale l is a hyper-parameter that needs a certain degree of tuning. A common practice consists in setting the lengthscale to be equal to the median pairwise distance among the joint data. More information about kernels and their connections to Gaussian Processes can be found in Section 2.5.

Despite being formulated in the view of statistical test applications, the MMD has been successful in generative modeling as well. Li et al. [2015] propose to train a generative neural network by iteratively minimizing the MMD between the data and the samples from the model (estimated through mini-batches in a stochastic fashion as illustrated in Section 2.3.1). The resulting method is called generative moment matching network (GMMN).

2.4.6 Adversarial Losses

Recently, within the research community there has been a deep interest in methods that minimize the divergence between the model and the data distributions through adversarial approaches. The most popular example in this direction is the generative adversarial network (GAN) [Goodfellow et al., 2014]. In this framework, a generative model is pitted against a discriminative model that learns to distinguish between samples from the model distribution $p_{\boldsymbol{\theta}}(\mathbf{x})$ and samples from the data distribution $p_{\mathcal{D}}(\mathbf{x})$.

In more detail, let us consider a generative model (the *generator*) which induces the model probability $p_{\boldsymbol{\theta}}(\mathbf{x})$ on the data points $\mathbf{x}$. In GANs, we first define a prior $p_{\mathbf{z}}(\mathbf{z})$ over input noise variable $\mathbf{z} \in \mathcal{Z}$ (often standard Gaussian), then we add a mapping from $\mathcal{Z}$ to the data space $\mathcal{X}$ denoted as $G(\mathbf{z};\boldsymbol{\omega})$, where G is a differentiable function represented for example by a multilayer perceptron with parameteters $\boldsymbol{\omega}$. Likewise, the discriminator can be modelled through another multilayer perceptron $D(\mathbf{x},\boldsymbol{\phi})$ that outputs a single scalar. $D(\mathbf{x})$ represents the probability that $\mathbf{x}$ came from the data rather than the generator's distribution $p_{\boldsymbol{\theta}}$.

During training, D tries to maximize the probability of assigning the correct label to both training examples and samples from G, while G tries to minimize $\log[1 - D(G(\mathbf{z}))]$. Alternatively, we can say that G and D play the minimax game with the value function $V(D,G)$:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[\log D(\mathbf{x})\right] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}\left[\log(1 - D(G(\mathbf{z})))\right]. \tag{2.68}$$

Goodfellow et al. [2014] also show that the solution to the minimax problem is unique and minimizes the KL divergence between $p_{\boldsymbol{\theta}}(\mathbf{x})$ and $p_{\mathcal{D}}(\mathbf{x})$.

Due to the success of GANs, there are many methods that build upon these foundations to improve the performance of the generative model and achieve faithful resemblance with the data distribution. One of the most well-known extensions of GANs incorporates elements of optimal transport [Villani, 2008], such as

the Wasserstein distance into this framework: the result is the Wasserstein GAN (WGAN) [Arjovsky et al., 2017].

First, let us define the Wasserstein distance between probability densities.

Definition 2.4.1. *The Earth-Mover (EM) distance or Wasserstein-1 distance between two probability distributions $\mathbb{P}$ and $\mathbb{Q}$ is defined as*

$$W(\mathbb{P}, \mathbb{Q}) = \inf_{\gamma \in \prod(\mathbb{P}, \mathbb{Q})} \mathbb{E}_{(\mathbf{u}, \mathbf{v}) \sim \gamma} \left[||\mathbf{u} - \mathbf{v}|| \right], \tag{2.69}$$

where $\prod(\mathbb{P}, \mathbb{Q})$ denotes the set of all joint distributions $\gamma(\mathbf{u}, \mathbf{v})$ whose marginals are respectively $\mathbb{P}$ and $\mathbb{Q}$.

Intuitively, $\gamma(\mathbf{u}, \mathbf{v})$ indicates how much mass should be moved to transform $\mathbb{P}$ into $\mathbb{Q}$. The EM distance can then be interpreted as the cost of the optimal mass transport plan between the two distributions. Unfortunately the infimum in Eq. 2.69 is highly intractable, but the Kantorovich - Rubinstein duality [Villani, 2008] yields

$$W(\mathbb{P}, \mathbb{Q}) = \sup_{||f||_L \leq 1} \Big(\mathbb{E}_{\mathbf{u} \sim \mathbb{P}} \left[f(\mathbf{u}) \right] - \mathbb{E}_{\mathbf{v} \sim \mathbb{Q}} \left[f(\mathbf{v}) \right] \Big), \tag{2.70}$$

where the sup is over all the 1-Lipschitz functions $f : \mathcal{X} \to \mathbb{R}$. Notice that if we replace $||f||_L \leq 1$ with $||f||_L \leq K$, then we end up with $K \cdot W(\mathbb{P}_r, \mathbb{P}_g)$. Thus, with a parameterized family of functions $\{f_w\}_{w \in \mathcal{W}}$ we can equivalently solve the problem

$$\max_{w \in \mathcal{W}} \Big(\mathbb{E}_{\mathbf{u} \sim \mathbb{P}} \left[f_w(\mathbf{u}) \right] - \mathbb{E}_{\mathbf{v} \sim \mathbb{Q}} \left[f_w(\mathbf{v}) \right] \Big). \tag{2.71}$$

The main idea behind the Wasserstein GAN consists in replacing the discriminator of the classical GAN with a "critic" trained to learn the function f_w that solves the maximization problem in Eq. 2.71. Following the same notation as before, we need to find the right $\boldsymbol{\omega}$ that parameterizes the critic and solves

$$\max_{\boldsymbol{\omega}} \Big(\mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \left[f_{\boldsymbol{\omega}}(\mathbf{x}) \right] - \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}} \left[f_{\boldsymbol{\omega}}(G(\mathbf{z})) \right] \Big). \tag{2.72}$$

As an alternative to the Wasserstein GAN, there is another method that aims at combining the MMD with the adversarial learning approach. Although MMD has

nice theoretical guarantees that were studied in the past [Gretton et al., 2012; Li et al., 2015; Dziugaite et al., 2015], the performance of algorithms based on it, such as GMMN, are inferior to GAN's [Liu et al., 2015; Yu et al., 2015]. Li et al. [2017] proposes to narrow this gap with their MMD-GAN, in which the kernel for the computation of the MMD is learned adversarially.

2.5 Gaussian Processes

From this point on, we focus on a particular class of Bayesian models, namely Gaussian Processes, which represent the main theme in the contributions presented here. As a matter of fact, all three original works see the applications of GPs to different settings.

Gaussian Processes are stochastic processes that can be used to define non-parametric probabilistic distributions over functions. GPs are extremely effective when used for regression tasks, especially in a data-scarce setting: indeed, in the best Bayesian tradition, they allow for the use of prior distributions and provide a measure for uncertainty quantification. After training, the predictive procedure for a Gaussian Processes amounts to the computation of the mean and variance of the Normal predictive distribution over the target y. Inference for GPs is made possible by the extraordinary analytical properties that derive from the central role played by the Gaussian distribution. In particular, the fact that the conditioning and marginalizing operations are closed for Gaussians is crucial.

Unfortunately, when employing a maximum likelihood approach, the training process for GPs is computationally demanding and scales cubically with the number of data points. For this reason, they do not scale effectively on large datasets.

We start the section by introducing GPs as distributions over functional spaces; we then show analytically how they can be trained on data and used for prediction tasks; we provide an overview of the most popular kernel used for GP regression; we discuss their connections with Hilbert Space Reproducing Kernels; finally, we

introduce their unsupervised learning extension (Gaussian Process-Latent Variable Models).

2.5.1 A Function-Space View

First of all, let us introduce Gaussian Processes as probability distributions over functional spaces, in the context of regression.

There are two main approaches that one can follow when trying to describe a predictive function f. The first method, mentioned earlier in the thesis, consists in restricting the class of functions in which we look for a viable solution (for example, considering only linear functions of the inputs). The second method, loosely speaking, associates a prior probability with every possible function, where higher probabilities are given to functions that are a priori considered to be likely to be close to the true solution. In this view, *Gaussian Processes* [Rasmussen and Williams, 2006] are powerful tools that describe distributions over functional spaces. The formal definition of a Gaussian Process is the following:

Definition 2.5.1 (Gaussian Process)**.** *A Gaussian Process (GP) is a collection of random variables, any finite number of which have a joint Gaussian distribution.*

A GP is fully specified by its mean and covariance functions. The mean function $m(\mathbf{x})$ and the covariance function $k(\mathbf{x}, \mathbf{x}')$ of a real process $f(\mathbf{x})$ are defined as

$$\begin{aligned} m(\mathbf{x}) &= \mathbb{E}[f(\mathbf{x})], \\ k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))], \end{aligned} \tag{2.73}$$

and the GP is denoted as

$$f(\mathbf{x}) \sim \mathcal{GP}\left(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\right). \tag{2.74}$$

The covariance function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ must be a bounded ($k(\mathbf{x}, \mathbf{x}') < \infty \, \forall \, \mathbf{x}, \mathbf{x}' \in \mathcal{X}$), symmetric ($k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x}) \, \forall \, \mathbf{x}, \mathbf{x}' \in \mathcal{X}$) and reproducing *kernel*. More detail about this are given in Section 2.5.4.

The choice of employing a Gaussian distribution in this context is easily justified by its attractive analytical properties, the most convenient being the "marginalization property": the marginal distributions in a multivariate Gaussian are themselves Gaussian. Since a GP is defined as a collection of variables, it automatically implies a notion of consistency: if the GP describes a multidimensional variable such as $(y_1, y_2) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, then it must also specify $y_1 \sim \mathcal{N}(\mu_1, \Sigma_{11})$, where Σ_{11} is the corresponding element of $\boldsymbol{\Sigma}$. Intuitively, this translates to the examination of a larger set of variables not influencing the distribution of the smaller set. Furthermore, the distribution of any subset of covariates conditioned upon any other subset is Gaussian. These properties are crucial when it comes to the predictive process, as illustrated in the next section.

2.5.2 Predictions with Gaussian Processes

Consider an observation process in which the observations $\mathbf{y}$ are corrupted by identically distributed Gaussian noise with variance σ_y^2:

$$y = f(\mathbf{x}) + \epsilon, \qquad \text{where} \qquad \epsilon \sim \mathcal{N}(0, \sigma_y^2), \tag{2.75}$$

From this point on, we collect all the input points $\{\mathbf{x}_n\}_{n=1}^N$ in the matrix $\mathbf{X}$, of size $N \times D$, where D is the dimensionality of $\mathbf{x}_n$, and the target output points $\{y_n\}_{n=1}^N$ in the vector $\mathbf{y}$. Notice that under these assumptions the system is homoscedastic, i.e. the measurement noise does not depend on the input. The prior on the noisy observations is

$$\operatorname{cov}(y_p, y_q) = k(\mathbf{x}_p, \mathbf{x}_q) + \sigma_y^2 \delta_{pq} \quad \text{or} \quad \operatorname{cov}(\mathbf{y}) = \mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_y^2 \mathbf{I}_N, \tag{2.76}$$

where $\mathbf{I}_N$ is the identity matrix of size $N \times N$. The elements of the covariance matrix $\mathbf{K}$ are computed using the kernel k as in $K_{pq} = k(\mathbf{x}_p, \mathbf{x}_q)$.

Let us now consider a prediction task, same as the previously discussed settings. Given the dataset $\mathcal{D} = \{\mathbf{X}, \mathbf{y}\}$, we would like to predict the noise-free outputs $\mathbf{f}_*$ at the new locations $\mathbf{X}_*$. For simplicity of notation and without loss of generality, we consider the outputs $\mathbf{y}$ and the inputs $\mathbf{X}$ to be normalized.

First and foremost, we compute the joint distribution of the observed targets $\mathbf{y}$ and the predicted function values $\mathbf{f}_*$ under the prior:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_y^2 \mathbf{I}_N & \mathbf{K}(\mathbf{X}, \mathbf{X}_*) \\ \mathbf{K}(\mathbf{X}_*, \mathbf{X}) & \mathbf{K}(\mathbf{X}_*, \mathbf{X}_*) \end{bmatrix}\right). \tag{2.77}$$

Then, we can use the useful analytical properties of Gaussian distributions to get the following conditional that we can use for prediction:

$$\begin{aligned} \mathbf{f}_* \mid \mathbf{X}, \mathbf{y}, \mathbf{X}_* \ &\sim \ \mathcal{N}(\mathbb{E}[\mathbf{f}_*], \mathbb{V}[\mathbf{f}_*]), \ \text{ where} \\ \mathbb{E}[\mathbf{f}_*] &= \mathbf{K}(\mathbf{X}, \mathbf{X}_*)^\top \left[\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_y^2 \mathbf{I}_N\right]^{-1} \mathbf{y}, \\ \mathbb{V}[\mathbf{f}_*] &= \mathbf{K}(\mathbf{X}_*, \mathbf{X}_*) - \mathbf{K}(\mathbf{X}, \mathbf{X}_*)^\top \left[\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_y^2 \mathbf{I}_N\right]^{-1} \mathbf{K}(\mathbf{X}_*, \mathbf{X}). \end{aligned} \tag{2.78}$$

It will be important for future sections to introduce the marginal likelihood $p(\mathbf{y} \mid \mathbf{X})$, which can be computed through the following integral:

$$p(\mathbf{y} \mid \mathbf{X}) = \int p(\mathbf{y} \mid \mathbf{f}, \mathbf{X}) p(\mathbf{f} \mid \mathbf{X}) \mathrm{d}\mathbf{f}. \tag{2.79}$$

Under the Gaussian prior we can write the distribution for $\mathbf{f} \mid \mathbf{X} \sim \mathcal{N}(\mathbf{0}, \mathbf{K}(\mathbf{X}, \mathbf{X}))$, or equivalently the likelihood

$$\log p(\mathbf{f} \mid \mathbf{X}) = -\frac{1}{2}\mathbf{f}^\top \mathbf{K}^{-1} \mathbf{f} - \frac{1}{2}\log |\mathbf{K}| - \frac{N}{2}\log 2\pi. \tag{2.80}$$

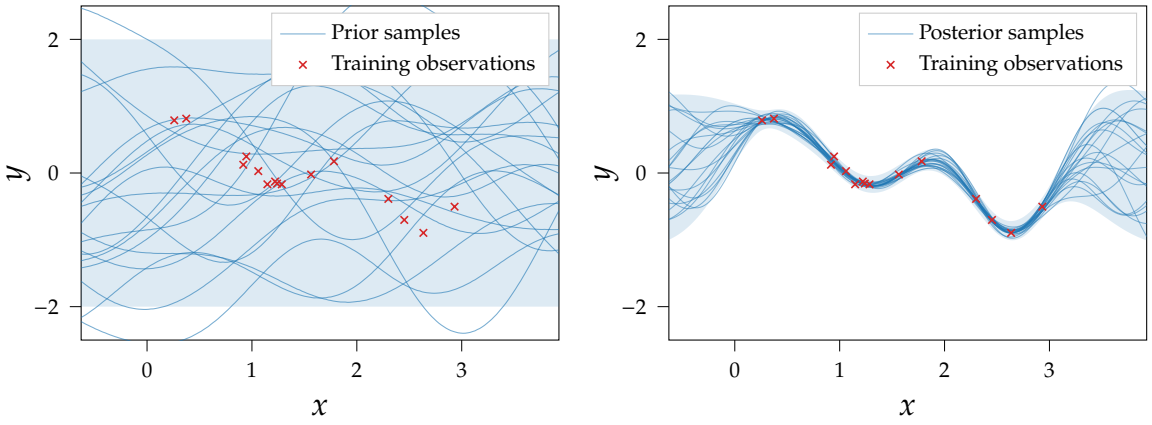
Finally, once again using Gaussian properties:

$$\log p(\mathbf{y} \mid \mathbf{X}) = -\frac{1}{2}\mathbf{y}^\top \left(\mathbf{K} + \sigma_y^2 \mathbf{I}_N\right)^{-1} \mathbf{y} - \frac{1}{2}\log \left|\mathbf{K} + \sigma_y^2 \mathbf{I}_N\right| - \frac{N}{2}\log 2\pi. \tag{2.81}$$

The last equation also follows directly from the observation that $\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \mathbf{K} + \sigma_y^2 \mathbf{I}_N)$.

2.5.3 Mean and Covariance Functions

The choice of mean and covariance function can significantly affect the predictive performance of the model. In particular, together with prior distributions, we can use the covariance function of a GP to encode our knowledge on the features and ultimately on the functional space of possible solutions. The inference process will take this into account when looking for the functions that best describe the training set.



(a) Before Training

(b) After Training

Figure 2.5: Samples from a Gaussian Process distribution before and after training. On the left, we show samples coming from a zero-mean GP prior with a radial basis function kernel. On the right, samples from a GP that has been trained on the data displayed as red crosses. The area between $\mu - 2\sigma$ and $\mu + 2\sigma$, corresponding to the 95% confidence interval, has been shaded.

Mean and covariance functions are in turn parameterized by a set of hyper-parameters. Performing inference with Gaussian Processes consists in: either identifying the best fitting parameters by maximization of the likelihood in Eq. 2.81; or marginalizing the parameters out through sampling-based approximation techniques.

Mean Functions

The most popular choice for the prior mean is to assume a zero function $\mu_\phi(\mathbf{x}) = 0$: our prior guess is that the target function will be null at any input location. Alternatively, we could choose the mean to be equal to a constant, which we would treat as a hyper-parameter during training, i.e. $\mu_\theta(\mathbf{x}) = \theta$. Naturally, if more prior knowledge on the problem is available, one can employ more tailored mean functions (typically polynomials, periodic or trigonometric functions, etc.).

Covariance Functions

As mentioned in previous sections, the covariance function is a crucial component of GP regression. It encodes assumptions about the function we wish to learn, expressed through limitations of the functional space in which we search for solutions and through a notion of *similarity* (or distance) between data points.

The most popular class of covariance functions is given by the class of *stationary* kernels, i.e. $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x} - \mathbf{x}')$. Stationary covariance functions are invariant to translations in the input space and thus implicitly assume that the system dynamics do not change within the domain $\mathcal{X}$. The most widely used stationary kernel is the *squared exponential* (SE):

$$k_{\text{SE}}(\mathbf{x}, \mathbf{x}') = \sigma \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2l^2}\right) \tag{2.82}$$

where σ and l, respectively variance and lengthscale, are hyper-parameters that model correlations. The strong smoothness assumptions encoded with this covariance function are unrealistic in many applied settings, yet its attractive analytical properties lead to its adoption in the Gaussian Process community.

Another important group of covariance functions is the *Matérn class*, given by

$$k_{\text{Matern}}(\mathbf{x}, \mathbf{x}') = \sigma \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}||\mathbf{x} - \mathbf{x}'||}{l}\right)^{\nu} K_{\nu}\left(\frac{\sqrt{2\nu}||\mathbf{x} - \mathbf{x}'||}{l}\right), \tag{2.83}$$

with positive hyper-parameters σ and l, and a further parameter ν that is not considered at inference time but instead fixed a priori. ν defines the shape of the modified Bessel function K_{ν} and regulates the smoothness assumptions that are intrinsic to the kernel (we point the reader at Rasmussen and Williams [2006] for more information on the subject).

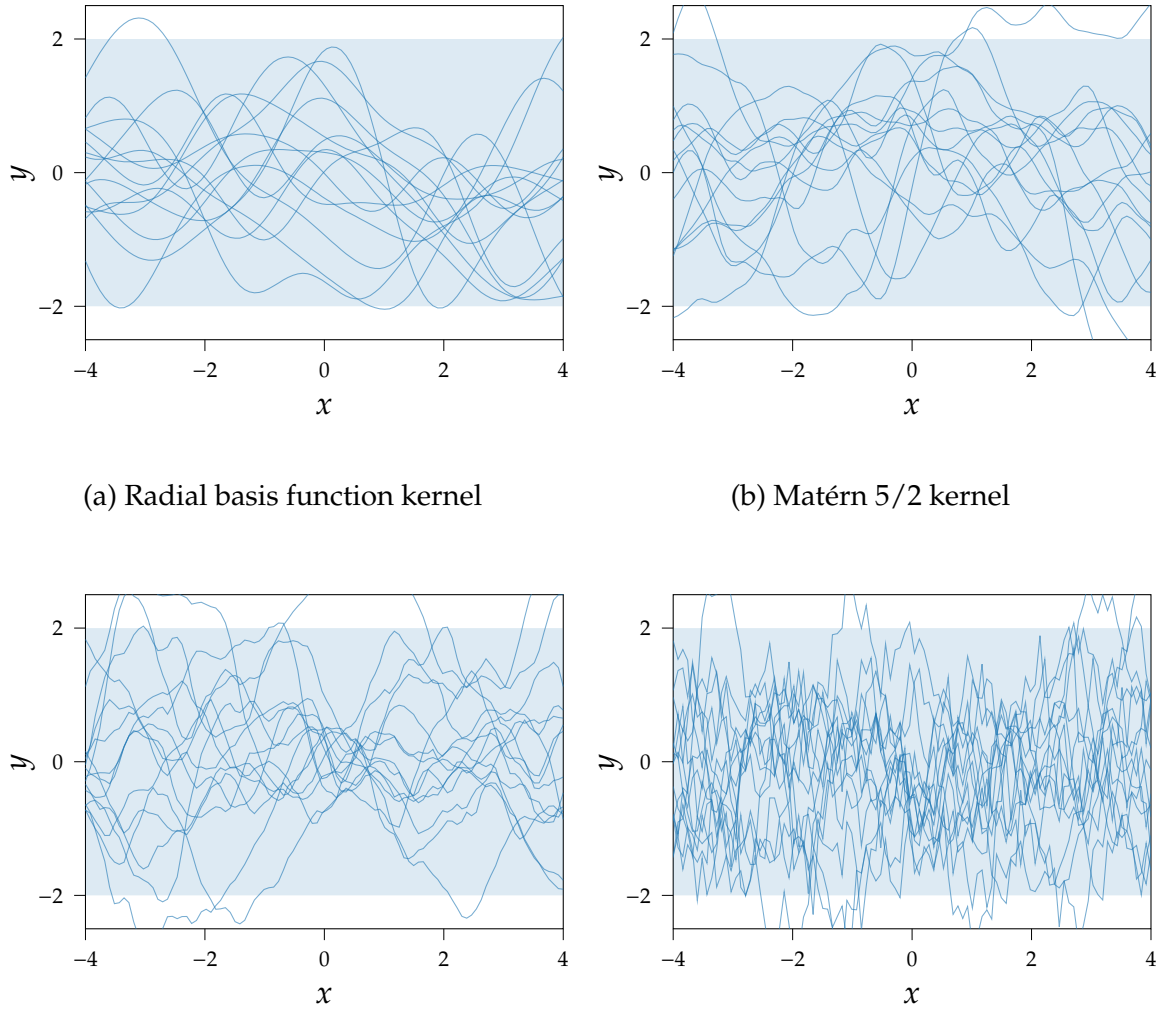
A further widely used and highly flexible covariance function is the *rational quadratic* (RQ) kernel:

$$k_{\text{RQ}}(\mathbf{x}, \mathbf{x}') = \sigma \left(1 + \frac{||\mathbf{x} - \mathbf{x}'||^2}{2\alpha l^2}\right)^{-\alpha}. \tag{2.84}$$

Again, σ and l are the same hyper-parameters described above and α has a similar function as the ν of the Matérn kernel.

As a final example that does not belong to the family of stationary covariance function, we show the *dot product* kernel, that can also be obtained from linear regression:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_0 + \frac{\mathbf{x} \cdot \mathbf{x}'}{l}. \tag{2.85}$$



(a) Radial basis function kernel

(b) Matérn 5/2 kernel

(c) Matérn 3/2 kernel

(d) Exponential kernel

Figure 2.6: Samples from Gaussian process priors with different covariance functions. The area between $\mu - 2\sigma$ and $\mu + 2\sigma$, corresponding to the 95% confidence interval, has been shaded.

2.5.4 A Weight-Space View

Behind the superficially simple derivations shown in the previous sections, Gaussian Processes possess intricate structures and assumptions that are not obvious to the naked eye. An alternative view and derivation that can potentially shine a light on these is based on reproducing kernels [Hofmann et al., 2008].

Mathematical Preliminaries

First and foremost, we formally define the mathematical construct of a *Hilbert space*. In this view, we first need to introduce the concepts of a *vector space* and an *inner product*. Here we will strictly review only the concepts necessary for this work, but we point the reader at Rudin [2006] for a more extensive discussion of functional analysis.

Definition 2.5.2 (Vector Space)**.** *A vector space V over a field F is a set V augmented by the vector addition ($+ : V \times V \rightarrow V$) and the scalar multiplication ($\cdot : F \times V \rightarrow V$) operations that satisfy the following properties for all $\mathbf{u}, \mathbf{v}, \mathbf{w} \in V$ and $a, b \in F$:*

- *Associativity of addition:* $\mathbf{u} + (\mathbf{v} + \mathbf{w}) = (\mathbf{u} + \mathbf{v}) + \mathbf{w}$.
- *Commutativity of addition:* $\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$.
- *Existence of additive identity:* $\exists\, \mathbf{0} \in V : \mathbf{v} + \mathbf{0} = \mathbf{v}$.
- *Existence of additive inverse:* $\exists\ -\mathbf{v} \in V : \mathbf{v} + (-\mathbf{v}) = \mathbf{0}$.
- *Compatibility of scalar multiplication with field multiplication:* $a(b\mathbf{v}) = (ab)\mathbf{v}$.
- *Identity element of scalar multiplication:* $1\mathbf{v} = \mathbf{v}$.
- *Distributivity of scalar multiplication with respect to vector addition:* $a(\mathbf{u} + \mathbf{v}) = a\mathbf{u} + a\mathbf{v}$.
- *Distributivity of scalar multiplication with respect to field addition:* $(a + b)\mathbf{v} = a\mathbf{v} + b\mathbf{v}$.

Elements of V are commonly called vectors.

Building on the example of $\mathbb{R}$, a vector space over the field $\mathbb{R}$ is the d-dimensional space $\mathbb{R}^d$.

Definition 2.5.3 (Inner Product)**.** *An inner product on a vector space V over a field F is a function $\langle \cdot, \cdot \rangle_V : V \times V \to \mathbb{R}$ that satisfies the following properties for all $\mathbf{u}, \mathbf{v}, \mathbf{w} \in V$ and $a \in F$:*

- *Conjugate symmetry:* $\langle \mathbf{u}, \mathbf{v} \rangle_V = \overline{\langle \mathbf{v}, \mathbf{u} \rangle_V}$.
- *Linearity in the first argument:* $\langle \mathbf{u} + \mathbf{v}, \mathbf{w} \rangle_V = \langle \mathbf{u}, \mathbf{w} \rangle_V + \langle \mathbf{v}, \mathbf{w} \rangle_V$.
- *Positive-definiteness:* $\langle \mathbf{v}, \mathbf{v} \rangle_V \geq 0$ *and* $\langle \mathbf{v}, \mathbf{v} \rangle_V = 0 \iff \mathbf{v} = \mathbf{0}$.

An inner product space can be defined as a vector space V over a field F augmented by an inner product operation.

Finally, we can introduce the definition of a Hilbert space.

Definition 2.5.4 (Hilbert Space)**.** *A Hilbert space $\mathcal{H}$ is a real or complex inner product space that is also complete with respect to the norm induced by the inner product, i.e.* $||\mathbf{v}||_{\mathcal{H}} = \sqrt{\langle \mathbf{v}, \mathbf{v} \rangle_{\mathcal{H}}}$, *where* $\mathbf{v} \in \mathcal{H}$.

The Hilbert space can be intuitively thought of as a generalization of the notion of Euclidean space to infinite dimensions. An example of a Hilbert space is the set of all functions on $\mathbb{R}$ that are measurable with respect to the L^2 norm, also called Lebesgue space L^2.

From Linear Models to Gaussian Processes: the Kernel Trick

The next step towards a deeper understanding of GPs is to consider a simple linear model and extend it with the tools above. A prediction model which is linear with respect to the weights $\mathbf{w}$ can be written as

$$f(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + \mu(\mathbf{x}), \qquad \mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{C}) \tag{2.86}$$

where $\boldsymbol{\phi}(\cdot) = [\phi_1(\cdot), \ldots, \phi_m(\cdot)]$, $\phi_i : \mathcal{X} \to \mathbb{R}\ \forall i = 1, \ldots, m$ is a feature map in an m-dimensional Hilbert space. Assume we are given a dataset of points $\mathbf{x}_1, \ldots, \mathbf{x}_N$, then the predictive model yields the output $\mathbf{F} = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N)]$ which is Gaussian-distributed, with mean $\boldsymbol{\mu} = [\mu(\mathbf{x}_1), \ldots, \mu(\mathbf{x}_N)]$ and covariance $k(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\phi}(\mathbf{x}_i)^\top \mathbf{C} \boldsymbol{\phi}(\mathbf{x}_j)$, $\forall\ i, j = 1, \ldots, N$. It is important to notice that $\mathbf{C}$ must be positive semi-definite for the Gaussian above to be well-defined. Indeed we require

$$\boldsymbol{\phi}(\mathbf{x})^\top \mathbf{C} \boldsymbol{\phi}(\mathbf{x}') = \sum_{a=1}^{m} \sum_{b=1}^{m} C_{ab} \phi_a(\mathbf{x}) \phi_b(\mathbf{x}') \qquad \forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}. \tag{2.87}$$

From the positive-definiteness property it follows that the Cholesky decomposition $\mathbf{C} = \mathbf{L}\mathbf{L}^\top$ exists. At this point we can further define the mapping $\boldsymbol{\psi}(\cdot) = \mathbf{L}\boldsymbol{\phi}(\cdot) = [\psi_1(\cdot), \ldots, \psi_m(\cdot)]$ and from that the covariance function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ can be alternatively written as

$$k(\mathbf{x}, \mathbf{x}') = \langle \boldsymbol{\psi}(\mathbf{x}), \boldsymbol{\psi}(\mathbf{x}') \rangle_{\mathcal{H}} = \sum_{a=1}^{m} \psi_a(\mathbf{x}) \psi_a(\mathbf{x}'). \tag{2.88}$$

From the Cauchy-Schwartz inequality it derives

$$k(\mathbf{x}, \mathbf{x}') \leq \sqrt{\sum_{a=1}^{m} (\psi_a(\mathbf{x}))^2} \sqrt{\sum_{a=1}^{m} (\psi_a(\mathbf{x}'))^2}. \tag{2.89}$$

If now we consider the case in which $\mathcal{H}$ is infinite dimensional and $\boldsymbol{\psi}(\cdot)$ stays in the bounded space L^2 mentioned above, i.e.

$$\sum_{a=1}^{\infty} (\psi_a(\mathbf{x}))^2 < \infty \qquad \forall\ \mathbf{x} \in \mathcal{X}, \tag{2.90}$$

then the covariance function will be valid and bounded, since $k(\mathbf{x}, \mathbf{x}') < \infty\ \forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}$, with infinitely many basis functions.

With this parameterization, it is not necessary to compute the potentially infinite representation $\boldsymbol{\psi}$ and the inner product $\langle \boldsymbol{\psi}(\mathbf{x}), \boldsymbol{\psi}(\mathbf{x}') \rangle_{\mathcal{H}}$ if k has the *reproducing property*, i.e.

$$\langle l(\cdot), k(\cdot, \mathbf{x}) \rangle_{\mathcal{H}} = l(\mathbf{x}), \qquad \forall\ \mathbf{x} \in \mathcal{X}, l(\cdot) \in \mathcal{H}. \tag{2.91}$$

The above is referred to as the *kernel trick*.

Gaussian Processes and Reproducing Kernel Hilbert Spaces

With the preliminaries introduced previously, we can now define the Reproducing Kernel Hilbert Space (RKHS).

Definition 2.5.5 (Reproducing Kernel Hilbert Space)**.** *Let $\mathcal{X}$ be an arbitrary set and $\mathcal{H}$ a Hilbert space of real-valued functions over $\mathcal{X}$. Then a function $k : \mathcal{X} \times \mathcal{X} \to \mathcal{H}$ is a reproducing kernel and $\mathcal{H}$ is a reproducing kernel Hilbert space if $k(\cdot, \mathbf{x}) \in \mathcal{H}\ \forall\ \mathbf{x} \in \mathcal{X}$ and the property in Eq. 2.91 is satisfied.*

The theory behind RKHS becomes relevant for Gaussian Processes when we highlight that their covariance functions can be seen as reproducing kernels. Furthermore, the kernel choice shapes the set of the possible solutions for the regression function. Remembering that in order to produce a valid covariance matrix the kernel k is required to be positive definite, we have:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} \mathbf{v}_i \mathbf{v}_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0, \qquad \forall \mathbf{v}_i \in \mathbb{R}^d, \mathbf{x}_i \in \mathcal{X}, i, j = 1, \ldots, N. \tag{2.92}$$

From the RKHS:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} \mathbf{v}_i \mathbf{v}_j k(\mathbf{x}_i, \mathbf{x}_j) = \sum_{i=1}^{N} \mathbf{v}_i \psi(\mathbf{x}_i)^\top \sum_{j=1}^{N} \mathbf{v}_j \psi(\mathbf{x}_i) = \left\| \sum_{i=1}^{N} \mathbf{v}_i \psi(\mathbf{x}_i) \right\|_{\mathcal{H}}^2 \geq 0. \tag{2.93}$$

Thus the condition in Eq. 2.90 is sufficient for the distribution of $\mathbf{F}$ to be a valid Gaussian even when an infinite number of basis functions is considered.

When looking into more general settings, we refer to the Moore-Aronszajin theorem [Aronszajn, 1950].

Theorem 2.5.1 (Moore-Aronszajin)**.** *Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a symmetric, positive definite kernel on a set $\mathcal{X}$. Then there is a unique Hilbert space $\mathcal{H}$ of functions on $\mathcal{X}$ for which k is a reproducing kernel.*

The latter theorem is crucial in the formulation of GPs. Let k be any Gaussian Process kernel function that is symmetric and positive-definite. Then at least one feature map $\psi : \mathcal{X} \to \mathcal{H}$ exists, together with corresponding $\phi : \mathcal{X} \to \mathcal{H}$

and $\mathbf{C}$. If we then go on and require k to be bounded, some finite realizations $[f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N)]$ will be Gaussian distributed with mean $[\mu(\mathbf{x}_1), \ldots, \mu(\mathbf{x}_N)]$ and covariance $k(\mathbf{x}_i, \mathbf{x}_j)$, $\forall i, j = 1, \ldots, N$.

Hence, it is possible to interpret Gaussian Process regression as a generalization of Bayesian linear regression that leverages an infinite dimensional feature mapping to an RKHS.

2.5.5 Gaussian Process Latent Variable Models

A Note on Unsupervised Learning

So far, we discussed the framework of supervised learning, described in Section 2.1.1, where the objective is to learn a relation (or a predictor) between inputs $\mathbf{x}_n$ and outputs y_n. Now we drastically change this view by moving towards an alternative paradigm in Machine Learning: assume that now we are given a dataset $\mathcal{D} = \{\mathbf{X}\} = \{\mathbf{x}_n\}_{n=1}^N$, which contains only the inputs $\mathbf{x}_n$. In this case the automated analysis tools of ML can help to discover hidden structures in data by looking at similarities and differences between the data points, maybe after making careful assumptions on the generative processes that caused the variability observed in the dataset $\mathcal{D}$. This setting is called *unsupervised learning*.

One of the classic approaches within unsupervised learning can be found in the latent variable models (LVMs) described in Section 2.2. LVMs can be used in the context of unsupervised learning for visualization [Lawrence, 2004], clustering [Bishop, 2006] and general learning from data. In particular, in this section we will turn our attention to Gaussian Process Latent Variable Models (GPLVMs) [Lawrence, 2005], which extend the classic GP model towards unsupervised frameworks.

Probabilistic Principal Component Analysis

First, we need to introduce probabilistic principal component analysis [Tipping and Bishop, 1999], also probabilistic PCA or PPCA, a latent variable model that extends the classic PCA with a probabilistic framework.

Consider a dataset $\mathcal{D} = \{\mathbf{x}_n\}_{n=1}^N$, where $\mathbf{x}_n \in \mathbb{R}^d$. We define as $\mathbf{X}$ the matrix with dimensions $N \times d$ in which every row contains a data point, i.e. $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_N]^\top$. We denote the n-th q-dimensional latent point corresponding to $\mathbf{x}_n$ with $\mathbf{z}_n$ (where $q \leq d$). Then, we assume a linear relationship between the two variables, corrupted by noise:

$$\mathbf{x}_n = \mathbf{W}\mathbf{z}_n + \boldsymbol{\epsilon}_n, \qquad \boldsymbol{\epsilon}_n \sim \mathcal{N}\left(\mathbf{0}, \beta^{-1}\mathbf{I}\right) \tag{2.94}$$

where the matrix $\mathbf{W}$ with size $d \times q$ contains the weights of the linear model. The likelihood for each data point, under these assumptions, is Gaussian:

$$p(\mathbf{x}_n \mid \mathbf{z}_n, \mathbf{W}, \beta) = \mathcal{N}(\mathbf{W}\mathbf{z}_n, \beta^{-1}\mathbf{I}). \tag{2.95}$$

Assuming a standard prior distribution over the latent space, i.e. $\mathbf{z}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, we can obtain the marginal likelihood by integrating over the latent variables:

$$p(\mathbf{x}_n \mid \mathbf{W}, \beta) = \int p(\mathbf{x}_n \mid \mathbf{z}_n, \mathbf{W}, \beta) p(\mathbf{z}_n) \mathrm{d}\mathbf{z}_n = \mathcal{N}\left(\mathbf{0}, \mathbf{W}\mathbf{W}^\top + \beta^{-1}\mathbf{I}\right). \tag{2.96}$$

Further assuming statistical independence between the data points $\mathbf{x}_n$, the likelihood of the whole dataset can be written as

$$p(\mathbf{X} \mid \mathbf{W}, \beta) = \prod_{n=1}^{N} p(\mathbf{x}_n \mid \mathbf{W}, \beta), \tag{2.97}$$

and the optimal parameters $\mathbf{W}$ can be found by maximization of the above likelihood. Tipping and Bishop [1999] show that the solution is analytical and is reached when $\mathbf{W}$ spans the optimal sub-space of the data, hence the probabilistic interpretation of PCA.

Gaussian Process Latent Variable Models

In the previous section we show how PPCA can be derived with a simple process that assumes a linear relationship between the observed and latent variables, marginalizes out the latter and optimizes the linear weights. Lawrence [2005] proposes a fundamentally opposite approach, which consists in marginalizing the weights and optimizing the latent variables.

Following the Bayesian methodology, we consider the weights in $\mathbf{W}$ as random variables on which we assume a standard Gaussian prior:

$$p(\mathbf{W}) = \prod_{i=1}^{d} \mathcal{N}(\mathbf{w}_i \mid \mathbf{0}, \mathbf{I}), \tag{2.98}$$

$\mathbf{w}_i$ being the i-th column of $\mathbf{W}$. The conjugate prior allows for marginalization of $\mathbf{W}$, which yields the likelihood

$$p(\mathbf{X} \mid \mathbf{Z}, \beta) = \prod_{i=1}^{d} p(\mathbf{x}_{:,d} \mid \mathbf{Z}, \beta) = \prod_{i=1}^{d} \mathcal{N}\left(\mathbf{x}_{:,d} \mid \mathbf{0}, \mathbf{Z}\mathbf{Z}^\top + \beta^{-1}\mathbf{I}\right), \tag{2.99}$$

where with $\mathbf{x}_{:,d}$ we denoted the d-th column of $\mathbf{X}$. Optimizing the latent variables can be translated into maximizing the likelihood

$$L = -\frac{dN}{2}\ln 2\pi - \frac{d}{2}\ln|\mathbf{K}| - \frac{1}{2}\mathrm{Tr}\left(\mathbf{K}^{-1}\mathbf{X}\mathbf{X}^\top\right) \tag{2.100}$$

where $\mathbf{K} = \mathbf{Z}\mathbf{Z}^\top + \beta^{-1}\mathbf{I}$. The solution is given by

$$\mathbf{X} = \mathbf{U}\mathbf{L}\mathbf{V}^\top \tag{2.101}$$

where $\mathbf{U}$ is an $N \times q$ matrix that contains the first q eigenvectors of $\mathbf{Z}\mathbf{Z}^\top$, $\mathbf{L}$ is a $q \times q$ diagonal matrix in which $L_{jj} = \left(\lambda_j - \frac{1}{\beta}\right)^{-\frac{1}{2}}$, λ_j being the eigenvalue associated with the j-th eigenvector of $d^{-1}\mathbf{Z}\mathbf{Z}^\top$, and $\mathbf{V}$ is an arbitrary $q \times q$ rotation matrix.

GPLVMs can now be recovered by observing that a simple Gaussian Process provided with the kernel function

$$k(\mathbf{z}_i, \mathbf{z}_j) = \mathbf{z}_i^\top \mathbf{z}_j + \beta^{-1}\delta_{ij} \tag{2.102}$$

would return, for the N latent points $\mathbf{z}_n$, the covariance matrix $\mathbf{K} = \mathbf{Z}\mathbf{Z}^\top + \beta^{-1}\mathbf{I}$. This covariance is equal to the one associated with each factor of the marginal likelihood for probabilistic PCA as presented in Eq. 2.99, that can thus be seen as the product of d independent Gaussian Processes. This dual interpretation of PPCA points at a class of models that employs GPs as mappings from the latent space $\mathcal{Z}$ to the observed $\mathcal{X}$. PPCA itself is a special case in which dimensions are assumed to be linear, independent and identically distributed. Of course all of these assumptions can be relaxed, in particular through the use of non-linear kernels like the ones introduced in Section 2.5.3.

2.6 Machine Learning for Differential Equations

In this chapter, we first discuss Bayesian Machine Learning and some of the inference techniques that are necessary to make predictions about data. Subsequently, we take a deep dive into Gaussian Processes, a popular regression framework with appealing analytical properties used for functional modeling.

In this final part of the literature review, we would like to motivate the industrial and scientific relevance that Bayesian models and especially GPs can have in modern modelling settings, in particular when combined with mathematical tools.

Assume we want to model a particular system with a generative approach, for example a chemical reaction or a financial security, that evolves over time and for which we have (noisy) observations. After what we discussed in this chapter, we can think about using Gaussian Processes for regression, as illustrated in Section 2.5. Nevertheless, let us make a crucial observation: before the tools of ML became so popular, for decades mathematicians and physicists have dedicated several research efforts to the development of handcrafted models that would describe such phenomena, for instance through differential equations.

In this section we would like to introduce the reader to the connections between ML (and especially GPs) and differential equations, in their ordinary and stochastic form. In particular, we look at how these two modeling tools can be combined together to produce reliable predictions that take explicitly into account the generative processes that gave rise to the observed data, on the base of mathematical models. In chapters 3 and 4 we present two original contributions that propose novel ways to incorporate respectively ODE- and SDE-based information into Gaussian Processes, interpreting the resulting state space as a latent variable model.

2.6.1 Ordinary Differential Equations

Mathematical models have proven to be essential tools when tackling the numerical challenges that arise in many fields of science. In particular, with first prin-

ciples and expert knowledge it is possible to work out a parametric form for the *ordinary differential equations* (ODEs) that describe many dynamical systems. Applications and successes of ODE-based modeling span the many different realms of science, including but not limited to control theory [Ogunnaike and Ray, 1994], robotics [Murray, 2017], economics [Zhang, 2005] and medicine, where differential equations have been applied to gene sequencing [Chen et al., 1999], patient outcomes predictions [Fey et al., 2015] and tumoral development forecasting [Hass et al., 2017].

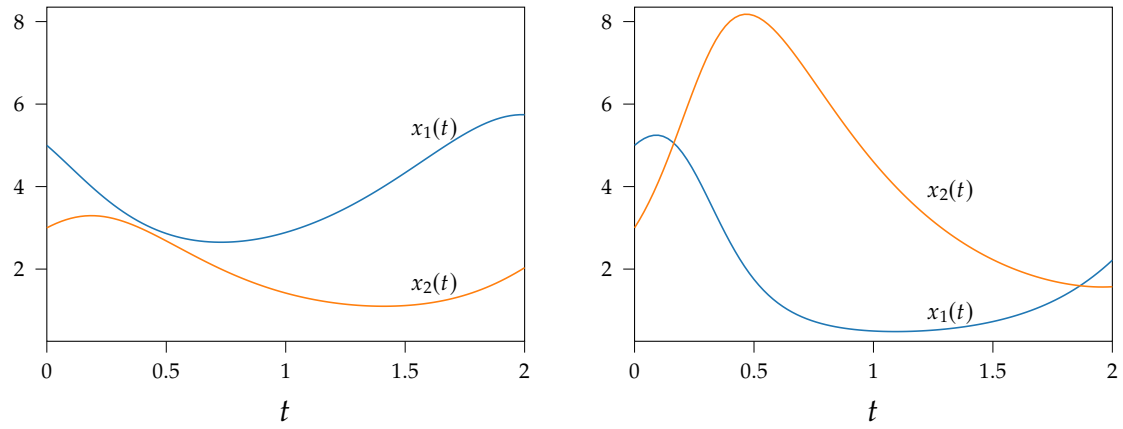
Although deriving the necessary equations directly from physics is a hard task, the advantages that ODE models can bring, both in terms of predictions and interpretability, are well established. One of the ways in which Machine Learning can help ODE-based predictions is through parameter inference, i.e. by automatically identifying the equation parameters that best describe the systems of interest.

More formally, let us $\mathbf{x} \in \mathbb{R}^K$ be a K-dimensional state vector collecting all variables that fully define a dynamical system. Consider a time variable $t \in \mathbb{R}_0^+$, taking values in the continuous interval $[0, T]$; we denote the function that maps the time stamp t to the corresponding state $\mathbf{x}$ as $\mathbf{x}(t)$. Then, the corresponding ODEs can be written as

$$\dot{\mathbf{x}}(t) = \frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{f}(\mathbf{x}(t), \boldsymbol{\theta}), \tag{2.103}$$

where $\mathbf{f}$ is a parameterized, vector-valued and potentially highly nonlinear function that expresses the interactions and dependencies between the components of $\mathbf{x}$, while the vector $\boldsymbol{\theta} \in \mathbb{R}^D$ collects the parameters that regulate the state dynamics.

The problem of parameter inference can be stated as follows. We are given noisy observations $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_N]$ of a time series, corresponding to the time stamps $\mathbf{t} = [t_1, \ldots, t_N]$. At the same time, we are given a parametric form for the system's dynamics $\mathbf{f}(\mathbf{x}, t; \boldsymbol{\theta})$, which we assume to have been derived through first principles and expert knowledge. The objective consists in inferring the denoised states $\mathbf{x}$ and



(a) $\boldsymbol{\theta} = [2, 1, 4, 1]$ (b) $\boldsymbol{\theta} = [4, 1, 2, 1]$

Figure 2.7: Example trajectories generated by the Lotka-Volterra equations (Eq. 2.106, 2.107) with two different sets of parameters $\boldsymbol{\theta}$. As we can see, the system parameters can highly affect dynamics.

the set of parameters $\boldsymbol{\theta}$ that best models the observed dynamics. In other words:

$$\text{Given} \qquad \begin{cases} \dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}, \boldsymbol{\theta}) \\ \mathbf{y} = \mathbf{x} + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathbb{P}_n \end{cases} \tag{2.104}$$

$$\text{Infer} \qquad \mathbf{x}, \boldsymbol{\theta}. \tag{2.105}$$

Here $\boldsymbol{\epsilon}$ is a vector modeling the observational noise, distributed according to $\mathbb{P}_n$. We omitted the direct dependency of $\mathbf{f}$ on t for clarity of notation.

Probably the most classic example of a 2-dimensional dynamical system is represented by the Lotka-Volterra equations [Lotka, 1932]:

$$\dot{x}_1(t) = \quad \theta_1 x_1(t) - \theta_2 x_1(t) x_2(t) \tag{2.106}$$

$$\dot{x}_2(t) = -\theta_3 x_2(t) + \theta_4 x_1(t) x_2(t). \tag{2.107}$$

Together, they model the dynamics of a biological system in which two species, respectively the prey x_1 and the predator x_2, interact with each other. The parameters $\boldsymbol{\theta} = [\theta_1, \theta_2, \theta_3, \theta_4]$ are positive real numbers that describe how the two populations affect one another and they greatly define the trajectories of the system (as seen in Figure 2.7). A graphical representation of the parameter inference problem for the Lotka-Volterra equations can be found in Figure 2.8.

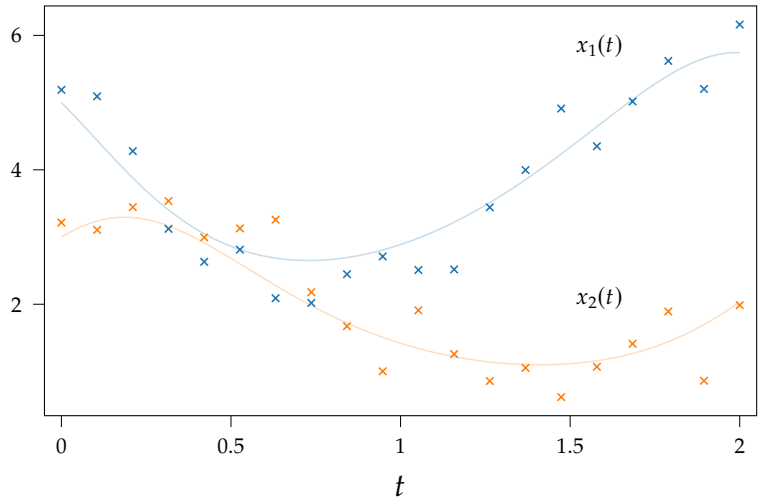


Figure 2.8: Graphical illustration of the parameter inference problem for the Lotka-Volterra equations. The dataset consists of the time series (drawn as crosses) generated by the system corresponding to the parameter vector $\boldsymbol{\theta} = [2, 1, 4, 1]$, together with the functional parameterized form as found in Eq. 2.106, 2.107. The task is to recover the true parameters $\boldsymbol{\theta}$ and the denoised states $x_1(t)$, $x_2(t)$.

The first attempts at tackling parameter inference were based on the integrated trajectory $\mathbf{x}(t)$, solution of

$$\begin{aligned} \mathbf{x}(t) &= \int_0^t \mathbf{f}(\mathbf{x}(s), \boldsymbol{\theta}) \mathrm{d}s \\ \mathbf{x}(0) &= \mathbf{x}_0. \end{aligned} \tag{2.108}$$

Since the above integral rarely admits analytical solutions, classically parameter inference has been tackled through numerical integration schemes. In particular, Bard [1974] and Benson [1979] minimize a loss function based on the divergence between the integrated trajectories and the data, as a measure of the goodness-of-fit. In other words, these approaches iteratively propose a new set of parameters and evaluate them by numerical integration by comparison with the observed data.

Alas, numerical integration is computationally expensive, especially when repeated until parameter convergence. In order to circumvent the need for numerical quadrature, Varah [1982] proposes to invert the logic behind the above approaches. In principle, adequate parameters will lead to trajectories that are close to the ground truth, i.e. searching for the good-quality $\boldsymbol{\theta}$ is equivalent to denoising the

states. First, Varah [1982] fits a spline to the observations in order to get an approximations of the denoised trajectories, then differentiates it with respect to the time variable analytically and matches state and derivative estimates of said splines to obtain the ODE parameters. This seminal paper gave rise to the class of *gradient matching* algorithms.

In a Bayesian setting, the parameter inference problem in the context of gradient matching has been addressed with Gaussian Processes, because of their convenient analytical properties. The first pioneering work of Calderhead et al. [2009] proposes a probabilistic modeling framework that relies on GPs, on which inference is performed through MCMC. Dondelinger et al. [2013] change the probabilistic setup and formulate a more efficient inference scheme. Moreover, Gorbach et al. [2017] introduce an alternative formulation based on Variational Inference (VI). All these works are based on the Product of Experts (PoE) heuristic, which has been shown to lead to theoretical issues in the case of gradient matching algorithms [Wenk et al., 2018].

In Chapter 3, we propose a novel generative modeling approach for gradient matching that circumvents the need for the Product of Experts in the context of parameter inference for ODEs. The resulting algorithm, *ODE-Informed Regression (ODIN)*, leverages constrained Gaussian Processes to build a computationally and data efficient algorithm for state and parameter inference. We also show how our method improves the current state of the art in an extensive set of experiments and how it can help with the challenging problem of model selection.

2.6.2 Stochastic Differential Equations

As we saw in the previous section, ordinary differential equations are a crucial modeling tool in many disciplines. Nevertheless, their form do not allow to model uncertain or stochastic dynamics. To this purpose ODEs can be extended with the addition of a further noise term: the resulting model is called stochastic differential equation (SDE) and its solutions are stochastic processes. They are widely

applied to many fields of science, such as economics and finance [Black and Scholes, 1973; Eraker, 2001], biology [Gillespie, 2000; Golightly and Wilkinson, 2011], physics [Van Kampen, 1992] and epidemiology [Fuchs, 2013].

In the Itô form, SDEs differ from ODEs because of one extra term that captures the stochasticity of the process dynamics:

$$\mathrm{d}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), \boldsymbol{\theta})\mathrm{d}t + \mathbf{g}(\mathbf{x}(t), \boldsymbol{\theta})\mathrm{d}\mathbf{w}(t). \tag{2.109}$$

Here the function $\mathbf{g}(\mathbf{x}(t), \boldsymbol{\theta})$ expresses the dependencies in the stochastic term of the differential equation and $\mathbf{w}(t)$ is a Wiener process with the same dimension as $\mathbf{x}(t)$. The formulation of the parameter inference problem in this case does not differ from the ODE case. It is well known that, because of the Wiener term, parameter inference in this context is not a trivial task, with most methods being sensitive to initialization [Picchini, 2007], depending strongly on the choice of hyperparameters [Bhat et al., 2015] or requiring large amounts of computation [Ryder et al., 2018].

The most common example of SDEs is certainly the Ornstein-Uhlenbeck process, a 1-dimensional, linear stochastic differential equation that reverts back to the mean:

$$\mathrm{d}x(t) = \theta_1(\theta_2 - x(t))\mathrm{d}t + \theta_3\mathrm{d}w(t). \tag{2.110}$$

Some realizations corresponding to different sets of parameters can be found in Figure 2.9.

Arguably, it would be a very arduous task to provide a comprehensive overview of the methods that address the problem of parameter inference (or state smoothing) in stochastic differential equations. We would like to point the reader at Tronarp and Särkkä [2018] for a review of the current state of the art for smoothing schemes and Sørensen [2004]; Nielsen et al. [2000]; Hurn and Lindsay [1999] for the more traditional approaches. Most the latter methods rely on estimating the probability of the sample paths $\mathbf{x}$ conditioned on the system parameters $\boldsymbol{\theta}$, i.e. $p(\mathbf{x} \mid \boldsymbol{\theta})$, which is usually intractable. To overcome this issue, Elerian et al. [2001] and Eraker

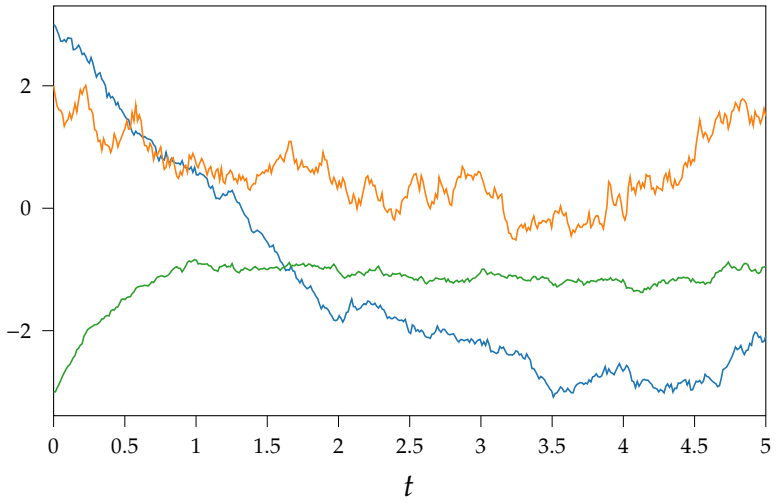


Figure 2.9: Example trajectories of the Ornstein-Uhlenbeck process, with different sets of parameters $\boldsymbol{\theta}$.

[2001] use the Euler-Maruyama discretization to produce an artificial grid of observations (that interpolates the actual $\mathbf{y}$) and subsequently employ MCMC-based integration; this approach is later refined with improved bridge constructs and support for partial observability [Pieschner and Fuchs, 2018; van der Meulen et al., 2017].

In addition, there are several research efforts that address the same problems by leveraging signal processing and Machine Learning techniques, such as Kalman filtering [Särkkä et al., 2015] and variational approximation methods in combination with Gaussian Processes [Archambeau et al., 2007; Vrettas et al., 2015]. On the other side, GPs can be used to directly model the functions $\mathbf{f}(\mathbf{x}(t), \boldsymbol{\theta})$ and $\mathbf{g}(\mathbf{x}(t), \boldsymbol{\theta})$ in Eq. 2.109, ignoring any prior knowledge about their parametric form [Ruttor et al., 2013; Yildiz et al., 2018].

In Chapter 4, we propose a novel statistical framework that combines the Doss-Sussman transformation [Doss, 1977; Sussmann, 1978] with Gaussian Processes to tackle the problem of parameter inference in systems of SDEs, for general drift $\mathbf{f}(\mathbf{x}(t), \boldsymbol{\theta})$ and linear diffusion $\mathbf{g}(\mathbf{x}(t), \boldsymbol{\theta})$ functions. This probabilistic model is then introduced into a grid-free, computationally efficient and robust parameter inference scheme that minimizes adversarial and moment matching loss functions.

With an extensive set of experiments, we show that our method improves the current state-of-the-art of parameter inference for SDEs that give rise to multimodal posterior distributions.

3

ODIN: ODE-Informed Regression for Parameter and State Inference in Time-Continuous Dynamical Systems

ODIN: ODE-Informed Regression for Parameter and State Inference in Time-Continuous Dynamical Systems

Philippe Wenk,[*1,2] **Gabriele Abbati,**[*3] **Michael A Osborne,**[3]
Bernhard Schölkopf,[4] **Andreas Krause,**[1] **Stefan Bauer**[4]
[1]Learning and Adaptive Systems Group, ETH Zürich
[2]Max Planck ETH Center for Learning Systems
[3]Department of Engineering Science, University of Oxford
[4]Empirical Inference Group, Max Planck Institute for Intelligent Systems
Correspondence to: wenkph@ethz.ch, gabb@robots.ox.ac.uk

Abstract

Parameter inference in ordinary differential equations is an important problem in many applied sciences and in engineering, especially in a data-scarce setting. In this work, we introduce a novel generative modeling approach based on constrained Gaussian processes and leverage it to build a computationally and data efficient algorithm for state and parameter inference. In an extensive set of experiments, our approach outperforms the current state of the art for parameter inference both in terms of accuracy and computational cost. It also shows promising results for the much more challenging problem of model selection.

Introduction

Ordinary differential equations (ODEs) represent a ubiquitous tool for modeling problems in many quantitative sciences and engineering. While with first principles and expert knowledge it is often possible to work out a parametric form for the equations that model a system of interest, in most cases there is no closed-form solution, making parameter identification problematic. One needs to rely on numerical schemes, which can be sub-optimal given that the exact system trajectory is usually unknown: typically, we observe noisy measurements of the true trajectory only at some discrete time points. This problem has been extensively studied in the past with classical approaches based on numerical integration (e.g. Bard (1974) and Benson (1979)), also in the relevant context of model selection (Chen, Shojaie, and Witten 2017; Wu et al. 2014).

Classical approaches iteratively propose new sets of parameters and then evaluate them by numerical integration: the estimated trajectory can then be compared against the observed data. Among others, Varah (1982) argue that this procedure can be turned on its head to improve computational performance. In principle, finding good parameters is equivalent to denoising the states, since adequate ODE parameters will lead to a trajectory that is close to ground truth. In particular, Varah (1982) first fit a spline curve to the observations to approximate the true trajectory, and subsequently match state and derivative estimates of said splines to obtain the ODE parameters. This idea gave rise to a class of *gradient matching* algorithms that rely on spline regression, kernel regression and, in a Bayesian setting, Gaussian process regression (GPR).

Gaussian processes (GPs) provide a very natural and theoretically appealing way to smooth time series, especially because they are very closely related to Kalman filtering (Hartikainen and Särkkä 2010). Thus, there has been significant interest in incorporating them into the gradient matching framework, starting from the pioneering theoretical work of Calderhead, Girolami, and Lawrence (2009): they propose a GP-based probabilistic modeling scheme on which they perform inference using MCMC. Dondelinger et al. (2013) change this probabilistic setup to achieve a more efficient MCMC sampling procedure (AGM - Adaptive Gradient matching), while Gorbach, Bauer, and Buhmann (2017) introduce a computationally efficient inference scheme based on variational inference (VGM - Variational Gradient Matching). Crucially, all these methods rely on a product of experts (PoE) heuristic (an alternative approach is formulated by Barber and Wang (2014), later questioned by Macdonald, Higham, and Husmeier (2015)). However, Wenk et al. (2018) show that the PoE leads to theoretical issues: indeed, in the graphical models proposed by Calderhead, Girolami, and Lawrence (2009), Dondelinger et al. (2013) and Gorbach, Bauer, and Buhmann (2017), the ODE parameters become statistically independent of the observations. Thus, Wenk et al. (2018) propose a new graphical model that circumvents this issue and present an efficient MCMC-based inference scheme (FGPGM - Fast Gaussian Process based Gradient Matching). A further formulation that is based on variational inference and allows for additional inequality constraints on the derivatives is provided by Lorenzi and Filippone (2018).

Similarly to Gaussian process-based gradient matching, González, Vujačić, and Wit (2014) and Niu et al. (2016) use kernel ridge regression in a frequentist setting (RKG2/RKG3 - Reproducing Kernel based Gradient Matching). Aiming directly for point estimates of the parameters, their approaches are naturally faster than alternatives that build on the use of MCMC and Gaussian processes.

[*]Equal contribution

Copyright © 2020, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Nevertheless, they rely on several trade-off parameters to be tuned via cross-validation, which can turn out to be impractical to data-scarce environments.

In our work, we extend and blend both Bayesian and frequentist viewpoints to obtain a computationally efficient algorithm that can learn states and parameters in a low-data setting. In particular, we

- Present a novel generative model, rephrasing the parameter inference problem as constrained Gaussian process regression;
- Provide a data-efficient algorithm that concurrently estimates states and parameters;
- Show how all hyperparameters can be learned from data and how they can be used as an indicator for model mismatch;
- Provide an efficient Python implementation for public use, with a publicly available code base at https://github.com/gabb7/ODIN

Background

Problem Setting

Throughout this work, we consider K-dimensional dynamical systems whose evolution is described by a set of differential equations parameterized by a vector $\boldsymbol{\theta}$, i.e.

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \boldsymbol{\theta}). \tag{1}$$

The system is observed under additive, zero-mean Gaussian noise at N discrete time points $\mathbf{t} = [t_1, \dots, t_N]$. We assume the standard deviation of the noise to be constant over time but it may differ for each state dimension. The noise is further assumed to be uncorrelated across dimensions and time points. This leads to the following observation model:

$$y_k(t_i) = x_k(t_i) + \epsilon_k(t_i), \qquad \epsilon_k(t_i) \sim \mathcal{N}(0, \sigma_k). \tag{2}$$

Previous research efforts propose to model $\mathbf{f}$ itself as a Gaussian process (e.g. Heinonen et al. (2018)) or to extend the framework to SDEs (e.g. Ryder et al. (2018), Abbati et al. (2019)). While it might be interesting to investigate how these frameworks could be combined with ODIN, this falls outside the scope of this paper. We thus restrict ourselves explicitly to the case where we have deterministic differential equations with known parametric form.

Temporal Regression

In the context of dynamical systems, Gaussian processes are employed mostly to model directly the system dynamics (i.e. the function $\mathbf{f}$). GP-based gradient matching approaches the problem differently. Here GPs model the states $\mathbf{x}$ and provide an approximation for the function $\mathbf{x}$ that maps a time point t_i to the corresponding state vector $\mathbf{x}(t_i)$. For the sake of readability, we assume $K = 1$ and denote the values of $x(t)$ stacked across time points as $\mathbf{x} = [x(t_1), \dots, x(t_N)]$. As shown in the experiment section, the extension to $K > 1$ is straightforward: K independent Gaussian processes can be stacked to model each state independently.

While our method can theoretically work with any nonlinear, differentiable regression technique, we choose to use Gaussian processes. GPs have superb analytical properties (Rasmussen and Williams 2006) and they recently showed remarkable empirical results in the context of parameter inference for ODEs (Lorenzi and Filippone 2018; Wenk et al. 2018). Moreover, thanks to the representer theorem (Schölkopf, Herbrich, and Smola 2001), Gaussian processes are closely related to kernel ridge regression: this connects GP-based gradient matching approaches to the reproducing-kernel-based ones (e.g. Niu et al. (2016)).

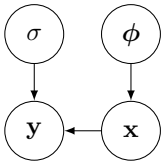


Figure 1: Generative model for standard Gaussian process regression. Given kernel hyperparameters $\boldsymbol{\phi}$ and observation noise standard deviation σ, the probability densities for the states $\mathbf{x}$ and their noisy observations $\mathbf{y}$ are fully determined.

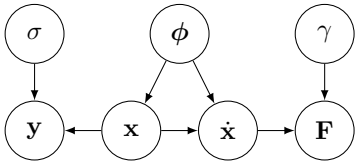


Figure 2: Generative model for GP regression with derivative observations $\mathbf{F}$, for which we use a Gaussian observation model with variance γ (as in ODIN). Due to the GP prior, $\mathbf{x}$ and $\dot{\mathbf{x}}$ are jointly Gaussian with known probability densities once the kernel hyperparameters $\boldsymbol{\phi}$ are determined.

Gaussian Process Regression

As in standard GP regression, we start by choosing a covariance function (or kernel) $k_{\boldsymbol{\phi}}$, which is parameterized by a set of hyperparameters $\boldsymbol{\phi}$. The kernel is used to compute a covariance matrix $\mathbf{C}_{\boldsymbol{\phi}}$, whose elements are given by $[\mathbf{C}_{\boldsymbol{\phi}}]_{i,j} = k_{\boldsymbol{\phi}}(t_i, t_j)$. $\mathbf{C}_{\boldsymbol{\phi}}$ can be used to define a zero-mean prior over the true states $\mathbf{x}$ at the observation times $\mathbf{t}$:

$$p(\mathbf{x} \mid \boldsymbol{\phi}) = \mathcal{N}(\mathbf{x} \mid \mathbf{0}, \mathbf{C}_{\boldsymbol{\phi}}). \tag{3}$$

The noise model from Equation (2) yields a Gaussian likelihood for the observations $\mathbf{y}$:

$$p(\mathbf{y} \mid \mathbf{x}, \sigma) = \mathcal{N}(\mathbf{y} | \mathbf{x}, \sigma^2 \mathbf{I}). \tag{4}$$

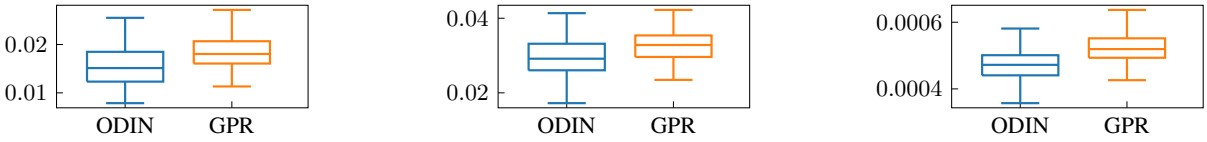
Using Bayes rule and observing the fact that a product of two Gaussians in the same variables is again a Gaussian, we obtain the classic GP posterior

$$p(\mathbf{x} \mid \mathbf{y}, \sigma, \boldsymbol{\phi}) = \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_{\mathbf{x}}, \boldsymbol{\Sigma}_{\mathbf{x}}), \tag{5}$$

$$\text{where} \qquad \boldsymbol{\mu}_{\mathbf{x}} = \mathbf{C}_{\boldsymbol{\phi}}(\mathbf{C}_{\boldsymbol{\phi}} + \sigma^2 \mathbf{I})^{-1} \mathbf{y} \tag{6}$$

$$\text{and} \qquad \boldsymbol{\Sigma}_{\mathbf{x}} = \sigma^2 (\mathbf{C}_{\boldsymbol{\phi}} + \sigma^2 \mathbf{I})^{-1} \mathbf{C}_{\boldsymbol{\phi}} \tag{7}$$

A graphical representation of this generative model can be found in Figure 1. Throughout this paper, we assume that k_{ϕ} is *differentiable* w.r.t. both of its arguments.



(a) Lotka-Volterra (b) FitzHugh-Nagumo (c) Protein Transduction

Figure 3: RMSE of state estimates using vanilla GP regression and ODIN on the benchmark systems (low noise case). While GPR can only access the noisy observations $\mathbf{y}$, ODIN considers the parametric form of $\mathbf{f}(\mathbf{x}, \boldsymbol{\theta})$ (with no information about $\boldsymbol{\theta}$). This additional regularization contributes towards more accurate estimations.

Gaussian Process Regression with Derivatives

As previously noted e.g. by Solak et al. (2003), the estimate of the posterior distribution of the states given by Equation (5) can be further refined if we consider access to noisy observations of the derivatives. Let us then assume we have additional observations $\mathbf{F}$ that are generated by $F(t_i) = \dot{x}(t_i) + \delta(t_i)$, where $\delta(t_i) \sim \mathcal{N}(0, \gamma)$. Incorporating such derivatives is straightforward: since Gaussian processes are closed under linear operations, the distribution over the derivatives is again a Gaussian process. Following the notation of Wenk et al. (2018) and the argumentation in their appendix, we obtain

$$p(\dot{\mathbf{x}} \mid \mathbf{x}, \boldsymbol{\phi}) = \mathcal{N}(\dot{\mathbf{x}} \mid \mathbf{D}\mathbf{x}, \mathbf{A}), \tag{8}$$

where the exact form of the matrices $\mathbf{D}$ and $\mathbf{A}$ is omitted for simplicity, but can be found in the supplementary material. Equation (8) can now be combined with the likelihood for the derivative observations

$$p(\mathbf{F} \mid \dot{\mathbf{x}}, \gamma) = \mathcal{N}(\mathbf{F} \mid \dot{\mathbf{x}}, \gamma\mathbf{I}). \tag{9}$$

This leads to the generative model shown in Figure 2. Just like in standard Gaussian process regression, all posteriors of interest can be calculated analytically, since all probability densities are Gaussian distributions in $\mathbf{x}$, $\dot{\mathbf{x}}$ or linear transformations thereof.

ODE-Informed Regression

Gaussian Process-based Gradient Matching

Given the model in Figure 2, the main challenge consists in including the mathematical expressions of the ODEs in a meaningful way. In traditional GP-based gradient matching, the ODEs are introduced as a second generative model for $\mathbf{F}$ or $\dot{\mathbf{x}}$. The latter is then combined with the Gaussian process model of Figure 2 to establish a probabilistic link between the observations $\mathbf{y}$ and the parameters $\boldsymbol{\theta}$. However, the GP model fully determines the probability densities of $\mathbf{F}$ and $\dot{\mathbf{x}}$. Thus, the two generative models have to be combined using some heuristic, like the product of experts (Calderhead, Girolami, and Lawrence 2009; Dondelinger et al. 2013; Gorbach, Bauer, and Buhmann 2017) or an additional Dirac delta function forcing equality (Wenk et al. 2018).

The resulting, unified generative model is then used to approximate the posterior of $\mathbf{x}$ and $\boldsymbol{\theta}$ through Bayesian inference techniques, e.g. MCMC (Calderhead, Girolami, and Lawrence 2009; Dondelinger et al. 2013; Wenk et al. 2018) or variational mean field (Gorbach, Bauer, and Buhmann 2017). Inference for these algorithms consists in computing mean and standard deviation of an approximate posterior to get estimates that include uncertainty. As we shall see in the experiment section, this works well for sufficiently tame dynamics and identifiable systems, but struggles to produce meaningful results for multi-modal posteriors. Crucially, practical systems often produce multi-modal posteriors and suffer from unidentifiability without strong priors (e.g. Stephan et al. (2008), Hass et al. (2017)). As we shall see, ODE-informed regression can overcome this issue.

ODIN: ODE-Informed Regression

To avoid the problems associated with the two probabilistic models of traditional GP-based gradient matching, ODIN does not include the ODEs via a separate generative model. Instead, they are introduced at inference time in the form of *constraints*, essentially solving a constrained MAP problem.

We start with the joint density of the Gaussian process described in Figure 2, denoted by $p(\mathbf{y}, \mathbf{x}, \dot{\mathbf{x}}, \mathbf{F} \mid \sigma, \gamma, \boldsymbol{\phi})$. As a result of the Gaussian observation model for $\mathbf{F}$, $\dot{\mathbf{x}}$ can be marginalized out analytically, leading to

$$p(\mathbf{y}, \mathbf{x}, \mathbf{F} \mid \sigma, \gamma, \boldsymbol{\phi}) = \tag{10}$$
$$\mathcal{N}(\mathbf{y} \mid \mathbf{x}, \sigma^2\mathbf{I})\, \mathcal{N}(\mathbf{x} \mid \mathbf{0}, \mathbf{C}_{\boldsymbol{\phi}})\, \mathcal{N}(\mathbf{F} \mid \mathbf{D}\mathbf{x}, \mathbf{A} + \gamma\mathbf{I}).$$

Assuming fixed values for $\sigma, \boldsymbol{\phi}$ and γ, this equation can be simplified by taking the logarithm, discarding all terms that do not explicitly depend on the states $\mathbf{x}$ and the derivative observations $\mathbf{F}$ and ignoring multiplicative factors to obtain

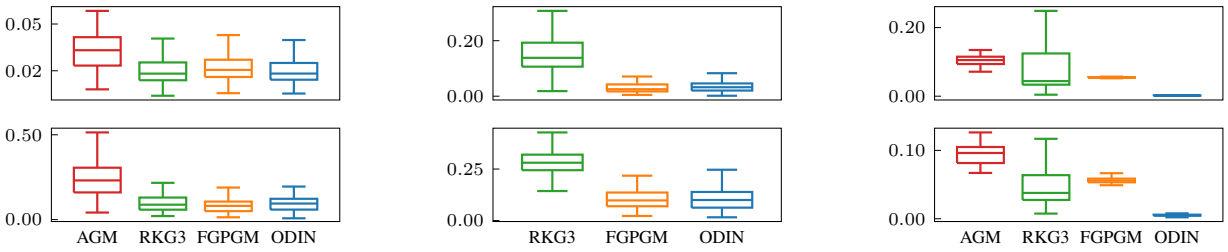
$$\tilde{\mathcal{R}}(\mathbf{x}, \mathbf{F}, \mathbf{y}) = \tag{11}$$
$$||\mathbf{x}||^2_{\mathbf{C}_{\boldsymbol{\phi}}^{-1}} + ||\mathbf{x} - \mathbf{y}||^2_{\sigma^{-2}\mathbf{I}} + ||\mathbf{F} - \mathbf{D}\mathbf{x}||^2_{(\mathbf{A}+\gamma\mathbf{I})^{-1}},$$

where $||\mathbf{u}||^2_{\mathbf{M}} := \mathbf{u}^T\mathbf{M}\mathbf{u}$ is the norm of the vector $\mathbf{u}$ weighted by a positive-definite matrix $\mathbf{M}$.

The key mechanism behind ODIN lies in how we obtain values for $\mathbf{F}$. In principle, $\mathbf{F}$ could be marginalized out to recover standard GP regression. However, this is not desirable, as we would ignore the ODE information. Instead, ODIN includes the ODEs as additional constraints in the optimization problem: rather than keeping $\mathbf{F}$ completely flexible, we assume the existence of a parameter vector $\boldsymbol{\theta}$ that links the derivative observations to the ODEs. More formally,

$$\mathbf{x}, \mathbf{F} = \arg\min_{\mathbf{x}, \boldsymbol{\theta}} \tilde{\mathcal{R}}(\mathbf{x}, \mathbf{F}, \mathbf{y}) \tag{12}$$

$$\text{s. t.} \quad \exists\boldsymbol{\theta} \quad \text{with} \quad \mathbf{f}(\mathbf{x}(t_i), \boldsymbol{\theta}) = \mathbf{F}_i \quad \text{for all } i. \tag{13}$$



(a) Lotka-Volterra (b) FitzHugh-Nagumo (c) Protein Transduction

Figure 4: Trajectory RMSE for parameter inference on the benchmark systems. The top row shows the low noise case with $\sigma = 0.1$ for LV, $SNR = 100$ for FHN and $\sigma = 0.001$ for PT. The bottom row shows the high noise case with $\sigma = 0.5$ for LV, $SNR = 10$ for FHN and $\sigma = 0.01$ for PT.

As it turns out, these constraints can be incorporated in the optimization problem by directly substituting $\mathbf{F}$ with the corresponding contribution from the ODEs, leading to

$$\mathbf{x}, \boldsymbol{\theta} = \arg\min_{\mathbf{x},\boldsymbol{\theta}} \mathcal{R}(\mathbf{x}, \boldsymbol{\theta}, \mathbf{y}), \tag{14}$$

where $\mathcal{R}(\mathbf{x}, \boldsymbol{\theta}, \mathbf{y}) := \tilde{\mathcal{R}}(\mathbf{x}, \boldsymbol{f}(\mathbf{x}, \boldsymbol{\theta}), \mathbf{y})$.

This constitutes the key idea behind ODIN. Instead of providing direct observations of the derivatives, we generate them via the ODEs after fixing the ODE parameters $\boldsymbol{\theta}$.

Similarly to classical frequentist methods (Varah 1982; Niu et al. 2016), $\mathcal{R}(\mathbf{x}, \boldsymbol{\theta}, \mathbf{y})$ punishes divergence between the states $\mathbf{x}$ and the observations $\mathbf{y}$, as well as between the output of the ODEs, $\mathbf{f}(\mathbf{x}, \boldsymbol{\theta})$, and the derivatives estimated by the regressing GP; the regularization term avoids overfitting. However, in sharp contrast to frequentist approaches, all trade-off parameters are naturally provided by the GP framework, once the hyperparameters $\boldsymbol{\phi}$ and noise levels σ and γ are fixed. As we will see later, the absence of cross-validation for hyperparameter learning crucially improves accuracy in sparsely observed systems.

Algorithm 1 ODIN

Input: $\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(K)}, \mathbf{f}(\mathbf{x}, \boldsymbol{\theta})$
Step 1: State-independent GP regression
for all $k \in K$ **do**
  Standardize time $\mathbf{t}$ and observations $\mathbf{y}_k$.
  Fit $\boldsymbol{\phi_k}$ and σ_k using empirical Bayes, i.e. maximize $p(\mathbf{y}^{(k)}|\mathbf{t}, \boldsymbol{\phi}_k, \sigma_k)$.
  Initialize $\mathbf{x}_k$ using the mean $\boldsymbol{\mu}_k$ of the trained GP.
end for
Step 2: ODE Information Incorporation
Initialize $\boldsymbol{\theta}$ randomly.
Initialize $\gamma_1, \ldots, \gamma_K = 1.0$
Apply L-BFGS-B to solve the optimization problem (15) and obtain $\hat{\mathbf{x}}, \hat{\boldsymbol{\theta}}, \hat{\gamma}_1, \ldots, \hat{\gamma}_K$.
Return: $\hat{\mathbf{x}}, \hat{\boldsymbol{\theta}}, \hat{\gamma}_1, \ldots, \hat{\gamma}_K$

Derivative Observation Model

Let us recall that we conceptually substitute the ODE outputs with derivative observations that are subjected to Gaussian noise, with variance γ. In this process, we can interpret in an intuitive but meaningful way: during and after training, the ODE outputs and the GP derivative estimates can deviate from the ground truth and thus differ from each other. This divergence is accounted for by γ. In classical GP-based approaches (Dondelinger et al. 2013; Gorbach, Bauer, and Buhmann 2017; Wenk et al. 2018), γ is treated as a random variable whose values are independent of the inference procedure; sometimes it is fixed a priori (Gorbach, Bauer, and Buhmann 2017; Wenk et al. 2018). However, we can expect that the divergence between ODEs and GP derivatives would be larger in the early steps of training, while it should decrease when the ODEs describe well to the ground truth. Thus, it is sensible to automatically adapt γ to reflect the current quality of the estimates.

The ODIN framework can be adjusted to reflect this reasoning. To obtain Equation (11), we implicitly assumed γ to be constant. If we rather treat it as an optimization parameter, the objective of Equation (14) changes to

$$\mathbf{x}, \boldsymbol{\theta}, \gamma = \arg\min_{\mathbf{x},\boldsymbol{\theta},\gamma} \mathcal{R}(\mathbf{x}, \boldsymbol{\theta}, \mathbf{y}, \gamma) \tag{15}$$

where

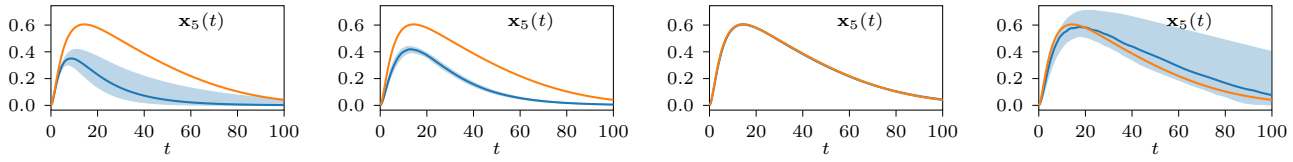
$$\mathcal{R}(\mathbf{x}, \boldsymbol{\theta}, \mathbf{y}, \gamma) = \mathcal{R}(\mathbf{x}, \boldsymbol{\theta}, \mathbf{y}) + \log(\det(\mathbf{A} + \gamma\mathbf{I})). \tag{16}$$

If γ is part of the optimization procedure, the contribution of the normalization constant in Equation (10) can not be ignored when deriving the risk appearing in Equation (16). In practice, similarly to the log-determinant in standard GP regression, this term acts as an Occam's razor by preventing an excessive growth of γ if the GP derivatives and the ODE outputs differ significantly.

The final ODIN routine is summarized as Algorithm 1.

Remarks

Throughout this work, we assume to have access to observations $\mathbf{y}$ that are subjected to the noise model described in Equation (2). However, the Gaussian noise assumption is only needed when deriving the term $||\mathbf{x} - \mathbf{y}||^2_{\sigma^{-2}\mathbf{I}}$ in $\mathcal{R}$. Thus, it could be straightforward to accommodate for alternative noise models by adjusting the corresponding term in the risk formula. On the other side, a Gaussian noise model (with variance γ) is a strict requirement for the derivative observations, as it is necessary to marginalize $\dot{\mathbf{x}}$ analytically.



(a) AGM (b) FGPGM (c) ODIN (d) RKG3

Figure 5: Comparison of the trajectories obtained by numerically integrating the inferred parameters of the Protein Transduction system for $\sigma = 0.01$. The solid blue line is the median trajectory, while for clarity we shaded the area between the 25% and 75% quantiles. The orange trajectory represents the ground truth.

As demonstrates in the experiments section, in the case of perfect ODEs (i.e. when we know the true parametric form a priori), γ can in principle be set to zero; nevertheless, when that is not the case it provides an effective mechanism for detecting model mismatch and helps with the challenging problem of model selection.

Experiments

In this section, we demonstrate the versatility of ODIN and compare its performance to various state-of-the-art methods. We start by comparing its parameter inference capabilities to various state-of-the-art inference schemes on three commonly used benchmark systems: the Lotka-Volterra (LV) predator-prey model (Lotka 1932); the FitzHugh-Nagumo (FHN) neuronal model (FitzHugh 1961; Nagumo, Arimoto, and Yoshizawa 1962); the chemical protein transduction (PT) system as seen in Vyshemirsky and Girolami (2007). All three systems have already been studied extensively in the context of gradient matching (Calderhead, Girolami, and Lawrence 2009; Dondelinger et al. 2013; Gorbach, Bauer, and Buhmann 2017; Wenk et al. 2018) and thus represent a clear benchmark. For completeness, we restate the concrete parametric form in the supplementary material, together with the ground truth for all parameters. In addition to state and parameter inference, we show how ODIN can be used for model selection, a missing feature for every comparison method here considered. Finally, we prove linear scaling behavior of ODIN in the state dimension K by investigating its performance on a high-dimensional, fourth benchmark system with up to 1000 states.

Evaluation Details and Data Creation

All experimental datasets are generated using numerical simulations. Thus, the ground truth for both the states $\mathbf{x}^*$ and parameters $\boldsymbol{\theta}^*$ is always available. Following Wenk et al. (2018), we employ the trajectory RMSE as a metric to compare the quality of parameter estimates. For ease of reference, we restate the definition in Definition 1.

Definition 1 (Trajectory RMSE) *Let $\hat{\boldsymbol{\theta}}$ be the parameters estimated by an inference algorithm. Let $\mathbf{t}$ be the vector collecting the observation times. Define $\tilde{x}(t)$ as the trajectory one obtains by integrating the ODEs using the estimated parameters, but the true initial value, i.e.*

$$\tilde{x}(0) = \mathbf{x}^*(0) \tag{17}$$

$$\tilde{x}(t) = \int_0^t f(\tilde{x}(s), \hat{\boldsymbol{\theta}}) ds \tag{18}$$

and define $\tilde{\mathbf{x}}$ element-wise as its evaluation at observation times $\mathbf{t}$, i.e. $\tilde{x}_i = \tilde{x}(t_i)$. The trajectory RMSE is then defined as

$$\textit{tRMSE} := \frac{1}{N} ||\tilde{\mathbf{x}} - \mathbf{x}||_2, \tag{19}$$

where $||.||_2$ denotes the standard Euclidean 2-norm.

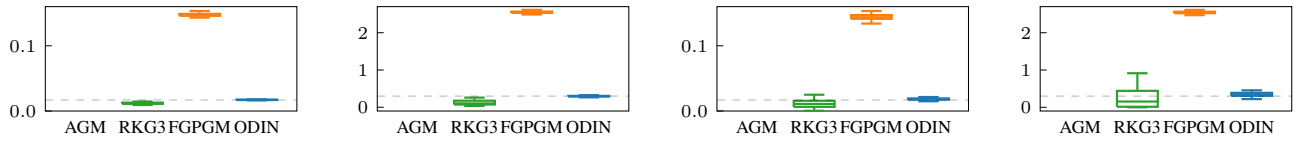
To evaluate the robustness of each algorithm w.r.t. different observation noise realizations, we always run 100 repetitions for every experimental setting. In each repetition, we keep $\mathbf{x}^*$ and $\boldsymbol{\theta}^*$ fixed and only sample the noise on $\mathbf{y}$. Results are then reported as quantiles over these 100 runs.

State and Parameter Inference

In the parameter inference setting, the true parametric form of the dynamical system is assumed to be provided by a practitioner, derived through first principles or expert knowledge. Thus, together with the noisy observations $\mathbf{y}$, we have access to the true parametric form $\dot{\mathbf{x}} = f(\mathbf{x}, \boldsymbol{\theta})$. The goal is to recover the true states $\mathbf{x}$ and parameters $\boldsymbol{\theta}^*$ at observation time. While smoothing is important, estimating $\boldsymbol{\theta}^*$ is of greater practical importance.

Out of the three comparison algorithms we chose, AGM (Dondelinger et al. 2013) and FGPGM (Wenk et al. 2018) rely on Gaussian processes and MCMC inference, while RKG3 (Niu et al. 2016) chooses a frequentist, kernel-regression-based approach. For all comparisons, implementations provided by the respective authors are used. Once more in accordance to the gradient matching literature, evaluations include both a low and a high noise setting for every system.

As shown by Solak et al. (2003), including direct observations of $\mathbf{F}$ can improve the accuracy of GP regression. ODIN does not have access to such observations, but it leverages the parametric form of the ODEs as a regularizer when performing state inference. As can be seen in Figure 3, this regularization actually improves the estimates of the states. This fact motivates a key difference to (Calderhead, Girolami, and Lawrence 2009), who propose to first fit the states using GPR and then perform gradient matching while keeping the states fixed. In the following, we show how



(a) θ_5, $\sigma = 0.001$ (b) θ_6, $\sigma = 0.001$ (c) θ_5, $\sigma = 0.01$ (d) θ_6, $\sigma = 0.01$

Figure 6: Parameter estimates for Protein Transduction for $\sigma = 0.001$ (a-b) and $\sigma = 0.01$ (c-d). Showing median, 50% and 75% quantiles over 100 independent noise realizations. The dashed line indicates the ground truth.

Table 1: Median and standard deviation of computation time (in seconds) for parameter inference over 100 independent noise realizations.

	AGM$[s]$	RKG3$[s]$	FGPGM$[s]$	ODIN$[s]$
LV, $\sigma = 0.1$	4548.0 ± 453.8	79.0 ± 19.0	3169.5 ± 90.1	$\mathbf{13.4 \pm 5.1}$
LV, $\sigma = 0.5$	4545.0 ± 558.5	76.5 ± 15.8	3187.5 ± 340.9	$\mathbf{11.4 \pm 5.1}$
FHN, $SNR = 100$	/	74.5 ± 14.3	8678.0 ± 482.7	$\mathbf{5.8 \pm 3.5}$
FHN, $SNR = 10$	/	77.5 ± 12.3	8677.0 ± 487.8	$\mathbf{4.4 \pm 3.8}$
PT, $\sigma = 0.001$	29776.5 ± 4804.7	469.0 ± 21.6	20291.5 ± 435.3	$\mathbf{8.9 \pm 1.5}$
PT, $\sigma = 0.01$	30493.0 ± 1470.4	480.0 ± 42.0	20437.0 ± 713.2	$\mathbf{20.6 \pm 3.75}$

ODIN can learn reliable parameters, improving the current state of the art in terms of accuracy and run time.

Accuracy In Figure 4 we compare the trajectory RMSE for the three benchmark systems. While the total tRMSE is an effective indicator for the overall performance, we also include the state-wise tRMSE in the supplementary material. Unfortunately, AGM was unstable on FitzHugh-Nagumo despite serious hyper-prior tuning efforts on our side. We thus do not have any results for this case. To help visualizing the raw numbers obtained by the tRMSE, we also report in Figure 5 the trajectories obtained by numerically integrating the inferred parameters. While here we report only one state for the high noise case of Protein Transduction, a full set of plots can be found in the supplement.

Run time In Table 1, we list the median training times (in seconds) and the corresponding standard deviation of all algorithms on the three parameter inference benchmark systems. It is evident (and not unexpected) that the optimization-based algorithms ODIN and RKG3 are orders of magnitude faster than the MCMC-based FGPGM and AGM. Furthermore, the need for cross-validation schemes in RKG3 seems to increase its run time roughly by an order of magnitude when compared to ODIN.

Identifiability While both LV as well as FHN models are relatively simple, Protein Transduction (PT) still represents a considerable challenge. Amongst others, both Dondelinger et al. (2013) and Wenk et al. (2018) claim that the two parameters θ_5 and θ_6 are only weakly identifiable. However, a quick experiment with different numerical values for those ODE parameters shows that they are actually identifiable. Indeed, neither RKG3 and ODIN seem to suffer from identifiability problems. For ODIN, this can be attributed to two key differences, the inference scheme and the flexible γ. Both AGM and FGPGM ultimately return the posterior mean of the parameter marginals. In Figure 7, we show example marginals for fixed γ. While these distributions are Gaussian-shaped for Lotka-Volterra, they are much wilder for PT. If we were to keep the γ fixed, ODIN would converge to an optimum instead of an expectation, which might be more appropriate in a multi-modal setting. However, ODIN does not keep γ fixed. Instead, its γ evolves during optimization according to the quality of the current parameters estimation, leading to an overall smoother inference. Consequently, the final parameter estimates are significantly more accurate (see Figure 6). For AGM, while the ratio between θ_5 and θ_6 is fairly stable and reasonably not far from the correct number, the absolute parameter values have median magnitudes of roughly 10^{12}: thus they do not appear in this figure.

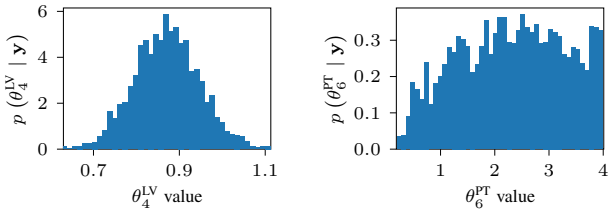


Figure 7: Parameter marginal distributions of θ_4 of Lotka-Volterra and θ_6 of Protein Transduction for one sample roll-out with fixed γ. While the LV marginal is nicely Gaussian, the PT marginal is much wilder.

Robustness Besides accurate parameter estimates, ODIN also exhibits more contained variance, especially compared to AGM and RKG3. This is a direct consequence of the underlying GP structure, which enables efficient and stable calculation of all parameters. Furthermore, a flexible γ seems to smooth out the optimization surface, avoiding the rugged landscapes reported by Dondelinger et al. (2013).

Table 2: Median and standard deviation of γ for different model misspecifications of the Lotka-Volterra system and 100 independent noise realizations.

	$\mathcal{M}_{1,1}$	$\mathcal{M}_{0,1}$	$\mathcal{M}_{1,0}$	$\mathcal{M}_{0,0}$
γ_1	$10^{-6} \pm 0.00$	3.01 ± 0.23	$10^{-6} \pm 0.00$	3.03 ± 0.24
γ_2	$10^{-6} \pm 0.04$	$10^{-6} \pm 0.00$	1.51 ± 0.31	1.53 ± 0.35

Priors While in a Bayesian inference setting it is common to introduce a prior over $\boldsymbol{\theta}$, our graphical model in Figure 2 does not treat $\boldsymbol{\theta}$ as a random variable. In a practical setting, we might not even know the parametric form of the ODEs: it thus seems quite difficult to justify the use of a prior. However, it should be noted that our framework can easily accommodate any prior without major modifications. An additional factor $p(\boldsymbol{\theta})$ in Equation (10) directly leads to an additional summand $-\log(p(\boldsymbol{\theta}))$ in Equation (16). From a frequentist perspective, this could be interpreted as an additional regularizer, similarly to LASSO or ridge regression. Since all other summands in Equation (16) grow linearly with the amount of observations N and the prior contribution stays constant, the regularization term would eventually have minor influence in an asymptotic setting.

Model Selection

In practice, domain experts might not be able to provide one single true model. Instead, they might indicate a set of plausible models that they would like to test against the observed data. In this section we investigate this problem, known as model selection. For empirical evaluation, we use the Lotka-Volterra system as ground truth to simulate our empirical data. We then create four different candidate models via the following two additional ODEs

$$\dot{x}_1(t) = \quad \theta_1 x_1^2(t) + \theta_2 x_2(t), \tag{20}$$
$$\dot{x}_2(t) = -\theta_3 x_2(t). \tag{21}$$

Each model is indexed as $\mathcal{M}_{i,j}$, where $i, j \in \{0, 1\}$. Here, $i = 0$ indicates that the wrong equation (i.e. 20) is used to model the dynamics of the first state, while if $i = 1$ we provide the true parametric form in that specific candidate model. In a similar fashion, $j = 0$ indicates that the wrong equation (i.e. 21) is used to model the dynamics of the second state, otherwise $j = 1$. ODIN is run independently for each $\mathcal{M}_{i,j}$. Besides state and parameter estimates, we thus obtain final values for γ, which are presented in Table 2. For numerical stability, γ was lower bounded to 10^{-6} in all experiments. For the correct model $\mathcal{M}_{1,1}$, γ settles at this lower bound, while it converges to a much larger value in case a wrong model is used. This justifies the intuitive interpretation of γ as a mean to account for model mismatch between the GP regressor and the ODE model. This last result proves that γ is indeed an efficient tool for identifying true parametric forms. Interestingly, this also works dimension-wise for the mixed models $\mathcal{M}_{0,1}$ and $\mathcal{M}_{1,0}$, even though the states x_1 and x_2 are coupled via wrong ODEs. This can be explained by the GP regressor prioritizing states $\mathbf{x}$ close to the observations $\mathbf{y}$. Indeed, while incorrect ODEs might deteriorate the accuracy of the state estimates with wrong regularization, their detrimental effects are limited by the observation-dependent partial objective, effectively decoupling the model mismatch across dimensions.

Linear Scaling in State Dimension

A key feature of gradient matching algorithms is the linear scaling in the state dimension K. Following Gorbach, Bauer, and Buhmann (2017), we demonstrate this for ODIN by using the Lorenz '96 system with $\theta = 8$, using 50 observations equally spaced over $t = [0, 5]$. The results are shown in Figure 8, including a linear regressor fitted to the means with least squares.

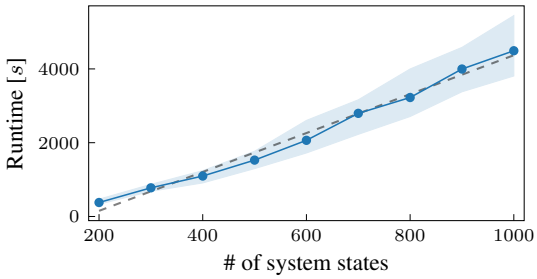


Figure 8: Run time for parameter inference on Lorenz '96 for different state dimension, with a (dashed) linear regressor fitted to the data. For each system size, we report the mean (dots) +- one standard deviation (shaded area) over 100 independent noise realizations.

Discussion

Parametric ODE systems are at the backbone of many practical applications, settings where Gaussian processes and kernel regression have shown to be efficient inference tools. In this paper, we demonstrate how to combine the advantages of both approaches by using theoretical insights to extend standard GP regression. The resulting algorithm, ODIN, significantly improves the current state of the art in terms of accuracy and runtime for parameter inference tasks and provides an appealing framework for model selection. Unlike other methods, ODIN does not require hyperparameter tuning and represents an out-of-the-box applicable tool for parameter inference and model selection for parametric ODE models.

Acknowledgments. This research was supported by the Max Planck ETH Center for Learning Systems. GA acknowledges funding from Google DeepMind and University of Oxford. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme grant agreement No 815943.

References

Abbati, G.; Wenk, P.; Osborne, M. A.; Krause, A.; Schölkopf, B.; and Bauer, S. 2019. Ares and mars adversarial and mmd-minimizing regression for sdes. In *International Conference on Machine Learning*, 1–10.

Barber, D., and Wang, Y. 2014. Gaussian processes for bayesian estimation in ordinary differential equations. In *International Conference on Machine Learning*, 1485–1493.

Bard, Y. 1974. Nonlinear parameter estimation.

Benson, M. 1979. Parameter fitting in dynamic models. *Ecological Modelling* 6(2):97–115.

Calderhead, B.; Girolami, M.; and Lawrence, N. D. 2009. Accelerating bayesian inference over nonlinear differential equations with gaussian processes. In *Advances in neural information processing systems*, 217–224.

Chen, S.; Shojaie, A.; and Witten, D. M. 2017. Network reconstruction from high-dimensional ordinary differential equations. *Journal of the American Statistical Association* 112(520):1697–1707.

Dondelinger, F.; Husmeier, D.; Rogers, S.; and Filippone, M. 2013. Ode parameter inference using adaptive gradient matching with gaussian processes. In *Artificial Intelligence and Statistics*, 216–228.

FitzHugh, R. 1961. Impulses and physiological states in theoretical models of nerve membrane. *Biophysical journal* 1(6):445–466.

González, J.; Vujačić, I.; and Wit, E. 2014. Reproducing kernel hilbert space based estimation of systems of ordinary differential equations. *Pattern Recognition Letters* 45:26–32.

Gorbach, N. S.; Bauer, S.; and Buhmann, J. M. 2017. Scalable variational inference for dynamical systems. In *Advances in Neural Information Processing Systems*, 4806–4815.

Hartikainen, J., and Särkkä, S. 2010. Kalman filtering and smoothing solutions to temporal gaussian process regression models. In *Machine Learning for Signal Processing (MLSP), 2010 IEEE International Workshop on*, 379–384. IEEE.

Hass, H.; Masson, K.; Wohlgemuth, S.; Paragas, V.; Allen, J. E.; Sevecka, M.; Pace, E.; Timmer, J.; Stelling, J.; MacBeath, G.; et al. 2017. Predicting ligand-dependent tumors from multi-dimensional signaling features. *NPJ systems biology and applications* 3(1):27.

Heinonen, M.; Yildiz, C.; Mannerström, H.; Intosalmi, J.; and Lähdesmäki, H. 2018. Learning unknown ode models with gaussian processes. *arXiv preprint arXiv:1803.04303*.

Lorenz, E. N., and Emanuel, K. A. 1998. Optimal sites for supplementary weather observations: Simulation with a small model. *Journal of the Atmospheric Sciences* 55(3):399–414.

Lorenzi, M., and Filippone, M. 2018. Constraining the dynamics of deep probabilistic models. *arXiv preprint arXiv:1802.05680*.

Lotka, A. J. 1932. The growth of mixed populations: two species competing for a common food supply. *Journal of the Washington Academy of Sciences* 22(16/17):461–469.

Macdonald, B.; Higham, C.; and Husmeier, D. 2015. Controversy in mechanistic modelling with gaussian processes. In *International Conference on Machine Learning*, 1539–1547.

Nagumo, J.; Arimoto, S.; and Yoshizawa, S. 1962. An active pulse transmission line simulating nerve axon. *Proceedings of the IRE* 50(10):2061–2070.

Niu, M.; Rogers, S.; Filippone, M.; and Husmeier, D. 2016. Fast inference in nonlinear dynamical systems using gradient matching. In *Journal of Machine Learning Research: Workshop and Conference Proceedings*, volume 48, 1699–1707. Journal of Machine Learning Research.

Rasmussen, C., and Williams, C. 2006. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. Cambridge, MA, USA: MIT Press.

Ryder, T.; Golightly, A.; McGough, A. S.; and Prangle, D. 2018. Black-box variational inference for stochastic differential equations. *arXiv preprint arXiv:1802.03335*.

Schölkopf, B.; Herbrich, R.; and Smola, A. J. 2001. A generalized representer theorem. In *International conference on computational learning theory*, 416–426. Springer.

Solak, E.; Murray-Smith, R.; Leithead, W. E.; Leith, D. J.; and Rasmussen, C. E. 2003. Derivative observations in gaussian process models of dynamic systems. In *Advances in neural information processing systems*, 1057–1064.

Stephan, K. E.; Kasper, L.; Harrison, L. M.; Daunizeau, J.; den Ouden, H. E.; Breakspear, M.; and Friston, K. J. 2008. Nonlinear dynamic causal models for fmri. *Neuroimage* 42(2):649–662.

Varah, J. M. 1982. A spline least squares method for numerical parameter estimation in differential equations. *SIAM Journal on Scientific and Statistical Computing* 3(1):28–46.

Vyshemirsky, V., and Girolami, M. A. 2007. Bayesian ranking of biochemical system models. *Bioinformatics* 24(6):833–839.

Wenk, P.; Gotovos, A.; Bauer, S.; Gorbach, N.; Krause, A.; and Buhmann, J. M. 2018. Fast gaussian process based gradient matching for parameter identification in systems of nonlinear odes. *arXiv preprint arXiv:1804.04378*.

Wu, H.; Lu, T.; Xue, H.; and Liang, H. 2014. Sparse additive ordinary differential equations for dynamic gene regulatory network modeling. *Journal of the American Statistical Association* 109(506):700–716.

Supplementary Material

GP Derivatives

For completeness, we restate the parametric form of the GP derivative mean and covariance as introduced by Calderhead, Girolami, and Lawrence (2009).

$$p(\dot{\mathbf{x}} \mid \mathbf{x}, \boldsymbol{\phi}) = \mathcal{N}(\dot{\mathbf{x}} \mid \mathbf{D}\mathbf{x}, \mathbf{A}), \tag{22}$$

where

$$\mathbf{D} \coloneqq {}'\mathbf{C}_{\boldsymbol{\phi}}\mathbf{C}_{\boldsymbol{\phi}}^{-1}, \tag{23}$$

$$\mathbf{A} \coloneqq \mathbf{C}''_{\boldsymbol{\phi}} - {}'\mathbf{C}_{\boldsymbol{\phi}}\mathbf{C}_{\boldsymbol{\phi}}^{-1}\mathbf{C}'_{\boldsymbol{\phi}} \tag{24}$$

and

$$\left[{}'\mathbf{C}_{\boldsymbol{\phi}}\right]_{i,j} \coloneqq \frac{\partial}{\partial a} k_{\boldsymbol{\phi}}(a,b)|_{a=t_i,b=t_j}, \tag{25}$$

$$\left[\mathbf{C}'_{\boldsymbol{\phi}}\right]_{i,j} \coloneqq \frac{\partial}{\partial b} k_{\boldsymbol{\phi}}(a,b)|_{a=t_i,b=t_j}, \tag{26}$$

$$\left[\mathbf{C}''_{\boldsymbol{\phi}}\right]_{i,j} \coloneqq \frac{\partial^2}{\partial a \partial b} k_{\boldsymbol{\phi}}(a,b)|_{a=t_i,b=t_j}. \tag{27}$$

Benchmarking Systems

To demonstrate the performance of ODIN, we use four commonly used benchmarking systems. To guarantee a fair comparison, we follow the established parameter settings as adopted among others by Calderhead, Girolami, and Lawrence (2009), Dondelinger et al. (2013), Gorbach, Bauer, and Buhmann (2017) and Wenk et al. (2018).

Lotka-Volterra The Lotka-Volterra predator-prey model was originally introduced by Lotka (1932) to describe population dynamics. It is a 2D system whose dynamics are determined by the ODEs

$$\dot{x}_1(t) = \quad \theta_1 x_1(t) - \theta_2 x_1(t) x_2(t) \tag{28}$$

$$\dot{x}_2(t) = -\theta_3 x_2(t) + \theta_4 x_1(t) x_2(t). \tag{29}$$

Using $\boldsymbol{\theta} = [2, 1, 4, 1]$ and initial conditions $\mathbf{x}(0) = [5, 3]$, the system is in a stable limit cycle with very smooth trajectories. The training data in this case consists of 20 equally spaced observations on the interval $[0, 2]$. It should be noted that the ODEs are linear in one state or parameter variable if all other state and parameter variables are kept constant. In the context of Gaussian process-based gradient matching, this means that the posterior marginals of the parameters and states are Gaussian distributed, which makes this system rather easy to solve. The VGM algorithm introduced by Gorbach, Bauer, and Buhmann (2017) is an excellent showcase of how to use this fact to derive an efficient variational approximation.

FitzHugh-Nagumo The FitzHugh-Nagumo model was originally introduced by FitzHugh (1961) and Nagumo, Arimoto, and Yoshizawa (1962) to model the activation of giant squid neurons. It is a 2D system whose dynamics are described by the ODEs

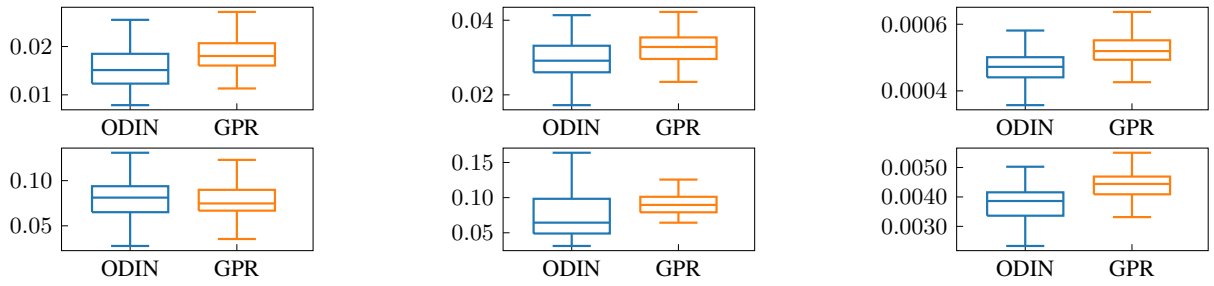
$$\dot{V} = \theta_1 (V - \frac{V^3}{3} + R) \tag{30}$$

$$\dot{R} = \frac{1}{\theta_1}(V - \theta_2 + \theta_3 R). \tag{31}$$

Using $\boldsymbol{\theta} = [0.2, 0.2, 3]$ and initial conditions $\mathbf{x}(0) = [-1, 1]$, this system is in a stable limit cycle. However, the trajectories of this system are quite rough with rapidly changing lengthscales, which is a significant challenge for any smoothing-based scheme. Furthermore, both V and θ_1 appear nonlinearly in the ODEs, leading to non-Gaussian posteriors. The dataset for this case consists of 20 equally spaced observations on the interval $[0, 10]$.

Protein Transduction The Protein Transduction model was originally introduced by Vyshemirsky and Girolami (2007) to model chemical reactions in a cell. It is a 5D system whose dynamics are described by the ODEs

$$\begin{aligned}
\dot{S} &= -\theta_1 S - \theta_2 S R + \theta_3 R_S \\
\dot{dS} &= \theta_1 S \\
\dot{R} &= -\theta_2 S R + \theta_3 R_S + \theta_5 \frac{R_{pp}}{\theta_6 + R_{pp}} \\
\dot{R}_S &= \theta_2 S R - \theta_3 R_S - \theta_4 R_S \\
\dot{R}_{pp} &= \theta_4 R_S - \theta_5 \frac{R_{pp}}{\theta_6 + R_{pp}}.
\end{aligned} \tag{32}$$



(a) Lotka-Volterra (b) FitzHugh-Nagumo (c) Protein Transduction

Figure 9: Comparing the RMSE of state estimates using vanilla GP regression and ODIN. All systems were evaluated on 100 independent noise realizations and parameter initializations. The top row shows the low noise case, the bottom row shows the high noise case.

The parameters and initial conditions of this system were set respectively to $\boldsymbol{\theta} = [0.07, 0.6, 0.05, 0.3, 0.017, 0.3]$ and $\mathbf{x}(0) = [1, 0, 1, 0, 0]$. Due to the dynamics changing rapidly at the beginning of the trajectories, the training data is generated by sampling the system at $\mathbf{t} = [0, 1, 2, 4, 5, 7, 10, 15, 20, 30, 40, 50, 60, 80, 100]$. This system has highly nonlinear terms and many claim it to be only weakly identifiable (Dondelinger et al. 2013; Gorbach, Bauer, and Buhmann 2017; Wenk et al. 2018).

Lorenz '96 The Lorenz '96 system was originally introduced by Lorenz and Emanuel (1998) for weather forecasting. The dimensionality $K > 3$ of this model can be chosen arbitrarily, with the k-th state being governed by the differential equation

$$\dot{x}_k = (x_{k+1} - x_{k-2})x_{k-1} - x_k + \theta, \tag{33}$$

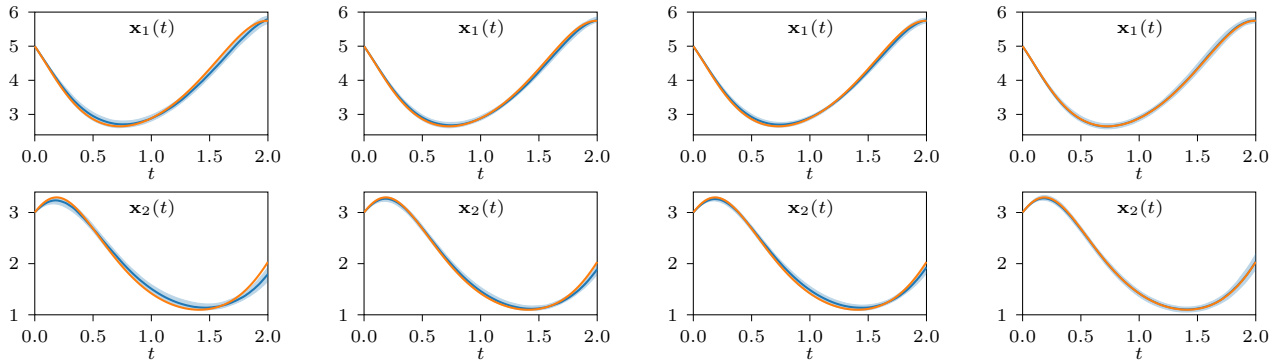
where all indices should be read modulo K. Introduced to the Gaussian process-based gradient matching community by Gorbach, Bauer, and Buhmann (2017), this system is used to demonstrate the scaling of any algorithm in the amount of states K, by keeping everything but K fixed. Following Gorbach, Bauer, and Buhmann (2017), we use $\theta = 8$ and 50 equally spaced observations over $t = [0, 5]$.

State Inference

Problem Setting Let us assume we are provided with a set of noisy observations $\mathbf{y}$ measured at the times $\mathbf{t}$ and the true parametric form $\dot{\mathbf{x}} = f(\mathbf{x}, \boldsymbol{\theta})$. The objective consists in recovering the true states $\mathbf{x}^*$ at observation times $\mathbf{t}$ without any information about the true parameters $\boldsymbol{\theta}^*$.

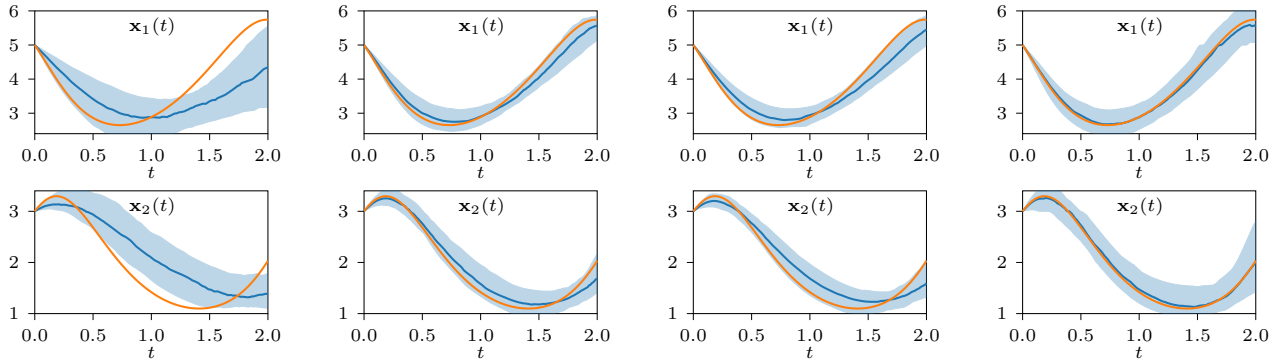
ODEs Provide Useful Information As shown in Figure 9, ODIN provides more accurate state estimates in all but the high noise LV case compared to standard GP regression. The different behavior can be explained by the quality of the GP prior. For Lotka Volterra, the RBF kernel provides a perfectly suited prior for the sinusoidal form of its dynamics. It is thus not surprising that the GP regression estimates are already quite good, especially in a high noise setting. However, for both FitzHugh Nagumo and Protein Transduction, the GP prior is slightly off. Thus, including the additional information provided by the ODEs leads to significant improvements in state estimation.

Median Trajectories



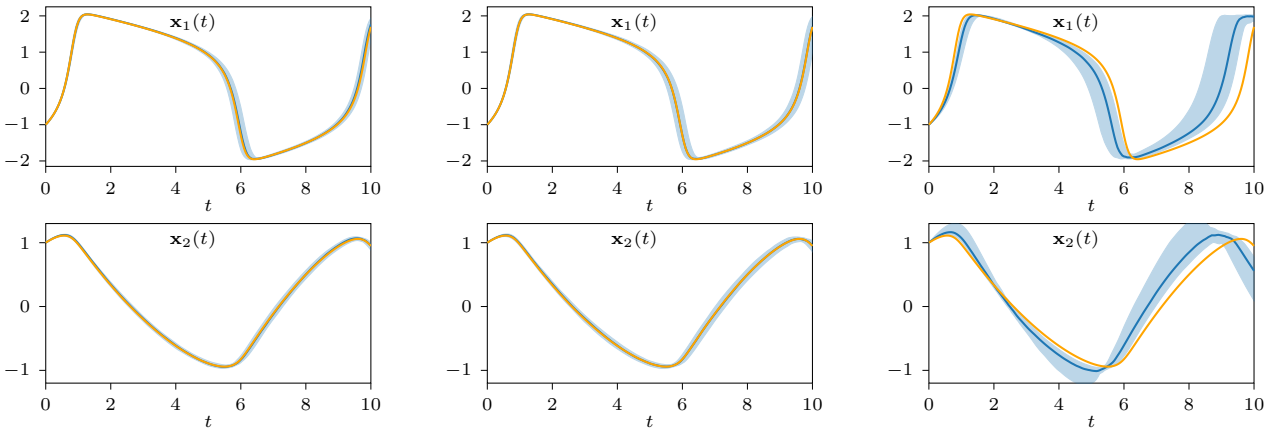
(a) AGM (b) FGPGM (c) ODIN (d) RKG3

Figure 10: Comparison of the trajectories obtained by numerically integrating the inferred parameters of the Lotka-Volterra system for $\sigma = 0.1$. The plot was created using 100 independent noise realizations, where the solid blue line is the median trajectory and the shaded areas denote the 25% and 75% quantiles. While the upper row shows the results for state x_1, the lower one does the same for state x_2. The orange trajectory is the ground truth. The difference between the four algorithms is barely visible.



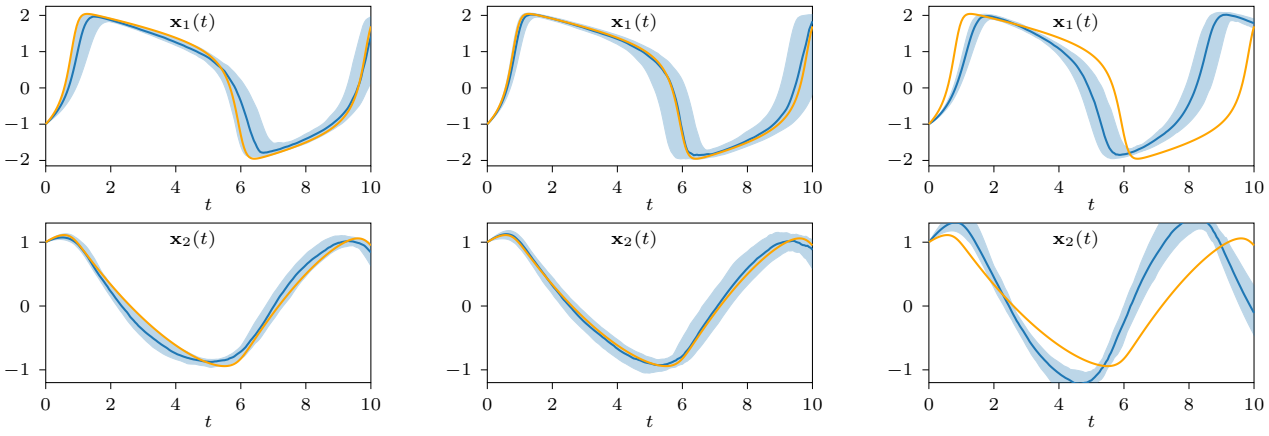
(a) AGM (b) FGPGM (c) ODIN (d) RKG3

Figure 11: Comparison of the trajectories obtained by numerically integrating the inferred parameters of the Lotka-Volterra system for $\sigma = 0.5$. The plot was created using 100 independent noise realizations, where the solid blue line is the median trajectory and the shaded areas denote the 25% and 75% quantiles. While the upper row shows the results for state x_1, the lower one does the same for state x_2. The orange trajectory is the ground truth. While all algorithms seem to perform reasonably well, the perfect match between the sinusoidal dynamics and the RBF kernel lead to a well performing RKG3, while the more flexible Gaussian process based schemes seem to suffer more strongly from a smoothing bias.



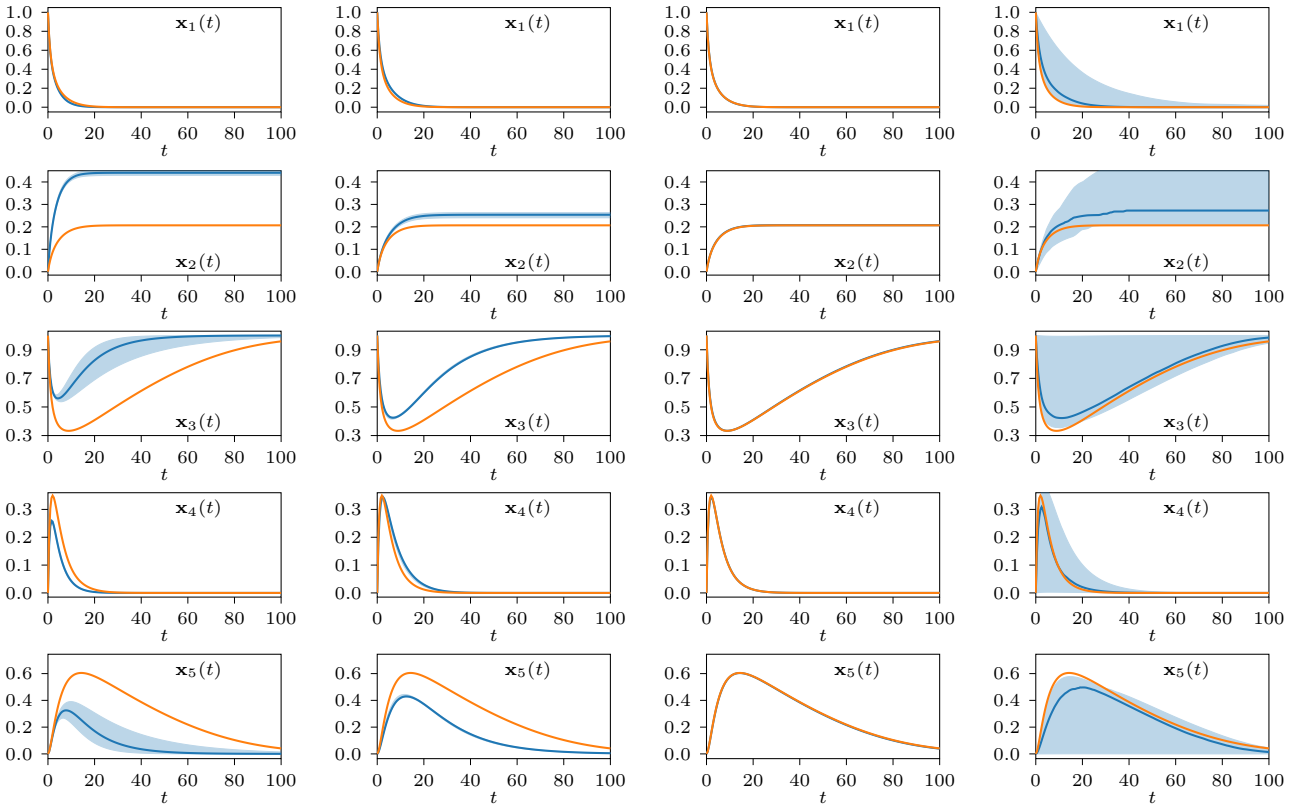
(a) FGPGM (b) ODIN (c) RKG3

Figure 12: Comparison of the trajectories obtained by numerically integrating the inferred parameters of the FitzHugh-Nagumo system for a SNR of 100. The plot was created using 100 independent noise realizations, where the solid blue line is the median trajectory and the shaded areas denote the 25% and 75% quantiles. While the upper row shows the results for state x_1, the lower one does the same for state x_2. The orange trajectory indicates the ground truth. This experiment demonstrates how ODIN can comfortably deal with dynamics with rapidly changing lengthscales.



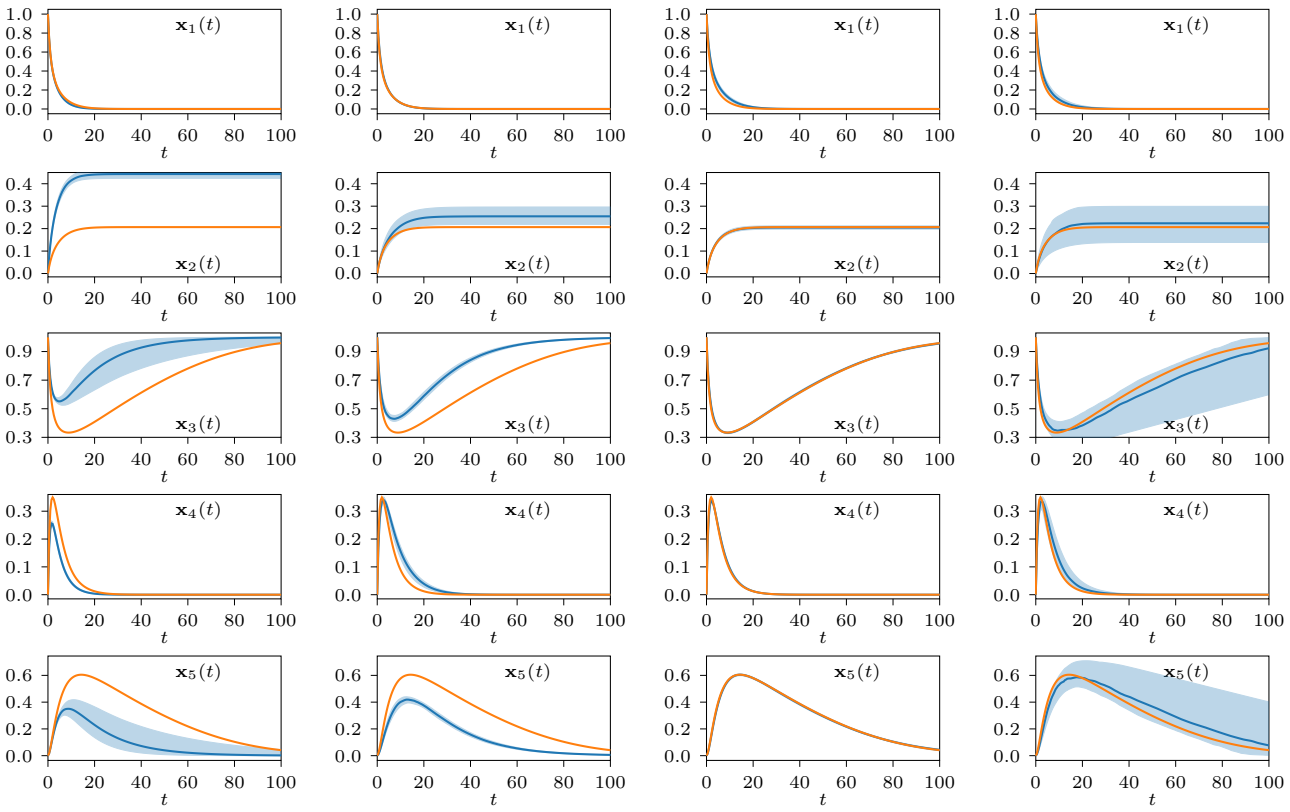
(a) FGPGM (b) ODIN (c) RKG3

Figure 13: Comparison of the trajectories obtained by numerically integrating the inferred parameters of the FitzHugh-Nagumo system for a SNR of 10. The plot was created using 100 independent noise realizations, where the solid blue line is the median trajectory and the shaded areas denote the 25% and 75% quantiles. While the upper row shows the results for state x_1, the lower one does the same for state x_2. The orange trajectory indicates the ground truth. This experiment demonstrates how ODIN can comfortably deal with dynamics with rapidly changing lengthscales.



(a) AGM (b) FGPGM (c) ODIN (d) RKG3

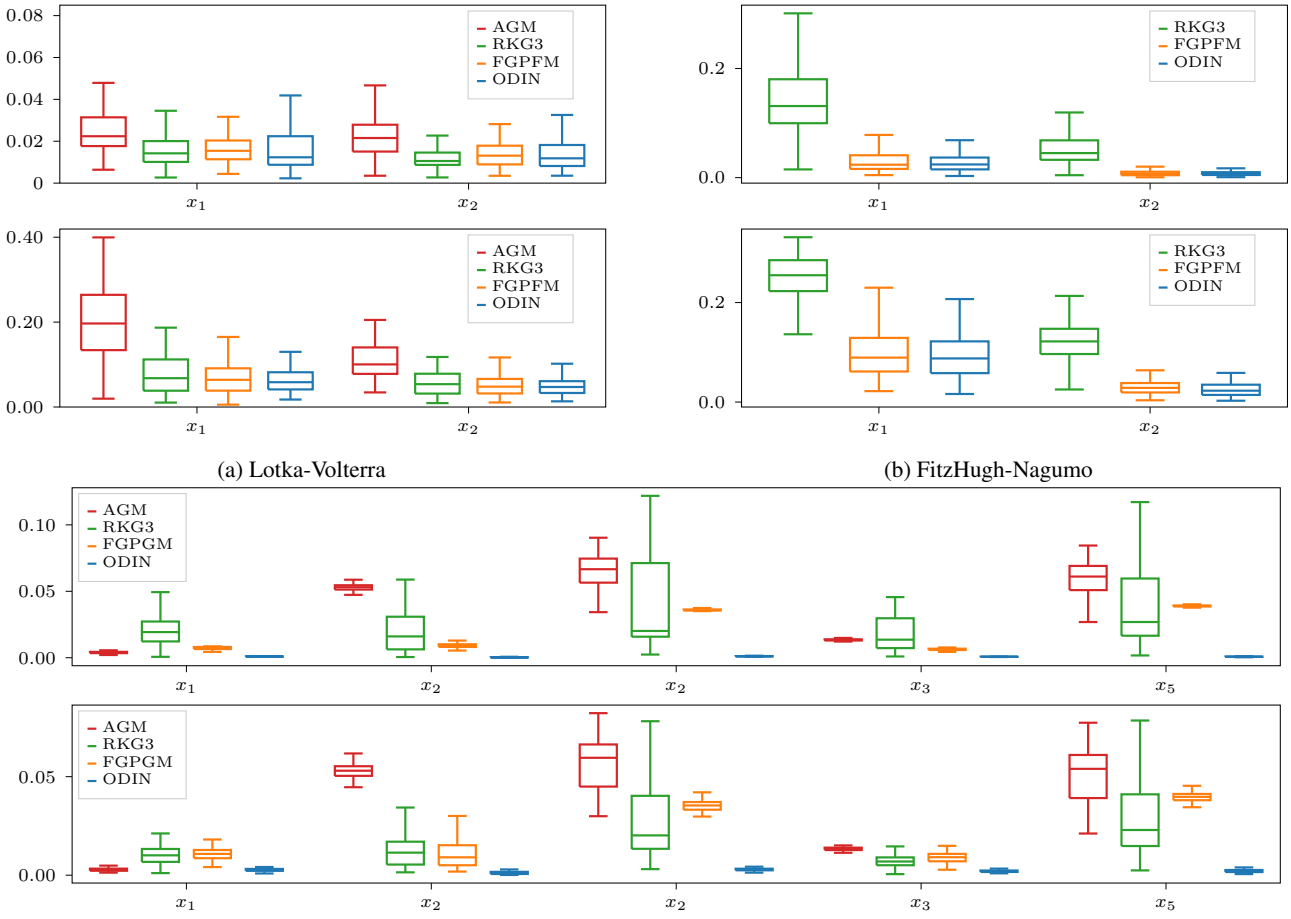
Figure 14: Comparison of the trajectories obtained by numerically integrating the inferred parameters of the Protein Transduction system for $\sigma = 0.001$. The plot was created using 100 independent noise realizations, where the solid blue line is the median trajectory and the shaded areas denote the 23% and 75% quantiles. The orange trajectory is the ground truth. This experiment proves how ODIN is the only algorithm among the displayed ones that can handle non-Gaussian posterior marginals.



(a) AGM (b) FGPGM (c) ODIN (d) RKG3

Figure 15: Comparison of the trajectories obtained by numerically integrating the inferred parameters of the Protein Transduction system for $\sigma = 0.01$. The plot was created using 100 independent noise realizations, where the solid blue line is the median trajectory and the shaded areas denote the 23% and 75% quantiles. The orange trajectory is the ground truth. As in the low noise case, this experiment proves how ODIN is the only algorithm among the displayed ones that can handle non-Gaussian posterior marginals.

Statewise tRMSE



(a) Lotka-Volterra

(b) FitzHugh-Nagumo

(c) Protein Transduction

Figure 16: Statewise trajectory RMSE for all benchmark systems in the parameter inference problem. For each pair of plots, the top shows the low noise case with $\sigma = 0.1$ for LV, $\sigma = 0.001$ for PT and $SNR = 100$ for FHN. The bottom shows the high noise case with $\sigma = 0.5$ for LV, $\sigma = 0.01$ for PT and $SNR = 10$ for FHN.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	ODIN: ODE-Informed Regression for Parameter and State Inference in Time-Continuous Dynamical Systems
Publication Status	☒ Published ☐ Accepted for Publication ☐ Submitted for Publication ☐ Unpublished and unsubmitted work written in a manuscript style
Publication Details	Philippe Wenk*, Gabriele Abbati*, Michael A Osborne, Andreas Krause, Bernhard Schölkopf and Stefan Bauer. ODIN: ODE-Informed Regression for Parameter and State Inference in Time-Continuous Dynamical Systems. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence.* New York, NY, USA (2020). (* denotes equal contribution)

Student Confirmation

Student Name:	Gabriele Abbati		
Contribution to the Paper	• Helped to develop the original idea by co-first author • Conceived the algorithmic procedure and the optimization routine • Added the model selection feature, originally not present • Coded the software package and released it on github • Performed all experiments • Wrote the paper together with co-first author		
Signature	Gabriele Abbati	Date	23/01/2020

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title:	Mike Osborne, Professor		
Supervisor comments	I endorse the description.		
Signature	MOsborne	Date	24 Jan 2020

4

AReS and MaRS - Adversarial and MMD-Minimizing Regression for SDEs

AReS and MaRS - Adversarial and MMD-Minimizing Regression for SDEs

Gabriele Abbati [*1] **Philippe Wenk** [*2 3] **Michael A Osborne** [1] **Andreas Krause** [2] **Bernhard Schölkopf** [4] **Stefan Bauer** [4]

Abstract

Stochastic differential equations are an important modeling class in many disciplines. Consequently, there exist many methods relying on various discretization and numerical integration schemes. In this paper, we propose a novel, probabilistic model for estimating the drift and diffusion given noisy observations of the underlying stochastic system. Using state-of-the-art adversarial and moment matching inference techniques, we avoid the discretization schemes of classical approaches. This leads to significant improvements in parameter accuracy and robustness given random initial guesses. On four established benchmark systems, we compare the performance of our algorithms to state-of-the-art solutions based on extended Kalman filtering and Gaussian processes.

1. Introduction

Modeling discretely observed time series is a challenging problem that arises in quantitative sciences and many fields of engineering. While it is possible to tackle such problems with difference equations or ordinary differential equations (ODEs), both approaches suffer from serious drawbacks. Difference equations are difficult to apply if the observation times are unevenly distributed. Furthermore, they do not generalize well across observation frequencies, while natural laws do not care about this artificial construct. ODEs can deal with these challenges, but fail to incorporate the inherent stochastic behavior present in many physical, chemical or biological systems. These effects can be captured by introducing stochasticity in the dynamics model, which brings to stochastic differential equations (SDEs). In this paper, we use exclusively the Itô-form

$$\mathrm{d}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), \boldsymbol{\theta})\mathrm{d}t + \mathbf{g}(\mathbf{x}(t), \boldsymbol{\theta})\mathrm{d}\mathbf{w}(t), \qquad (1)$$

where $\mathbf{x}(t)$ is the time-dependent vector of states we would like to model, $\boldsymbol{\theta}$ collects the parameters of the model, $\mathbf{f}$ is the drift term, $\mathbf{g}$ is the matrix-valued diffusion function and $\mathbf{w}(t)$ is a Wiener-process of the same dimension as the state vector $\mathbf{x}$.

While SDEs can efficiently capture stochasticity and deal with unevenly spaced observation times and frequency, inference is rather challenging. Due to the stochasticity of $\mathbf{w}(t)$, the state vector $\mathbf{x}(t)$ is itself a random variable. Except for few special cases, it is not possible to find an analytic solution for the statistics of $\mathbf{x}(t)$ for general drift and diffusion terms. The problem is even more challenging if we were to condition on or state-estimate some discrete time observations $\mathbf{y}$ (filtering/smoothing) or infer some statistics for the parameters $\boldsymbol{\theta}$ (parameter inference). It is well known that the parameter inference problem is a difficult task, with most approaches either being very sensitive to initialization (Picchini, 2007), strongly dependent on the choice of hyperparameters like the spacing of the integration grid (Bhat et al., 2015) or using excessive amount of computational resources even for small scale systems and state-of-the-art implementation (Ryder et al., 2018).

The difficulty of the parameter estimation problem of estimating parameters of drift and diffusion under observational noise is readily exemplified by the fact that even major scientific programming environment providers like MATLAB are still lack an established toolbox for practical use. In this paper, we will take a step into a novel direction tackling this open and exciting research question.

1.1. Related Work

While it is impossible to cover all related research efforts, we would like to give a quick overview by mentioning some of the most relevant. For a more in-depth discussion, we recommend Tronarp & Särkkä (2018), who provide an excellent review of the current state-of-the-art smoothing schemes. Moreover, Sørensen (2004), Nielsen et al. (2000) and Hurn et al. (2007) provide extensive explanations of the more traditional approaches.

[*]Equal contribution [1]Department of Engineering Science, University of Oxford [2]Learning and Adaptive Systems Group, ETH Zürich [3]Max Planck ETH Center for Learning Systems [4]Empirical Inference Group, Max Planck Institute for Intelligent Systems. Correspondence to: Gabriele Abbati <gabb@robots.ox.ac.uk>, Philippe Wenk <wenkph@ethz.ch>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

Most classical methods rely on calculating the probability of sample paths $\boldsymbol{x}$ conditioned on the system parameters $\boldsymbol{\theta}$, denoted as $p(\mathbf{x}|\boldsymbol{\theta})$. Since $p(\mathbf{x}|\boldsymbol{\theta})$ is usually analytically intractable, approximation schemes are necessary. Elerian et al. (2001) and Eraker (2001) use the Euler-Maruyama discretization to approximate $p(\mathbf{x}|\boldsymbol{\theta})$ on a fixed, fine grid of artificial observation times later to be leveraged in a MCMC sampling scheme. Pieschner & Fuchs (2018) and van der Meulen et al. (2017) subsequently refine this approach with improved bridge constructs and incorporated partial observability. Ryder et al. (2018) follow up on this idea by combining discretization procedures with variational inference. Särkkä et al. (2015) investigate different approximation methods based on Kalman filtering, while Archambeau et al. (2007) and Vrettas et al. (2015) use a variational Gaussian process-based approximation. Finally, it should be mentioned that $p(\mathbf{x}|\boldsymbol{\theta})$ can be inferred by solving the Fokker-Planck-Kolmogorov equations using standard methods for PDEs (Hurn & Lindsay, 1999; Aït-Sahalia, 2002).

Instead of approximating $p(\mathbf{x}, \boldsymbol{\theta})$ in a variational fashion, Gaussian processes can as well be used to directly model $\mathbf{f}(\mathbf{x}, \boldsymbol{\theta})$ and $\mathbf{g}(\mathbf{x}, \boldsymbol{\theta})$, ignoring in this way any prior knowledge about their parametric form. This approach was investigated by Ruttor et al. (2013), whose linearization and discretization assumptions which were later relaxed by Yildiz et al. (2018). While we will show in our experiments that these methods can be used for parameter estimation if the parametric form of drift and diffusion are known, it should be noted that parameter inference was not the original goal of their work.

1.2. Our Work

To the best of our knowledge, there are only very few works that try to circumvent calculating $p(\mathbf{x}|\boldsymbol{\theta})$ at all. Our approach is probably most closely related to the ideas presented by Riesinger et al. (2016). Our proposal relies on the Doss-Sussman transformation (Doss, 1977; Sussmann, 1978) to reduce the parameter inference problem to parameter inference in an ensemble of random ordinary differential equations (RODEs). These equations can then be solved path-wise using either standard numerical schemes or using the computationally efficient gradient matching scheme of Gorbach et al. (2017) as proposed by Bauer et al. (2017).

The path-wise method by Bauer et al. (2017) has natural parallelization properties, but there is still an inherent approximation error due to the Monte Carlo estimation of the expectation over the stochastic element in the RODE. Furthermore, their framework imposes severe linearity restrictions on the functional form of the drift $\mathbf{f}(\mathbf{x}, \boldsymbol{\theta})$, while it is unable to estimate the diffusion matrix $\mathbf{g}(\mathbf{x}, \boldsymbol{\theta})$.

While we will keep their assumption of a constant diffusion matrix, i.e. $\mathbf{g}(\mathbf{x}, \boldsymbol{\theta}) = \mathbf{G}$, our approach gets rid of the linearity assumptions on the drift $\mathbf{f}$. Furthermore, we substitute the Monte Carlo approximation by embedding the SDE into a fully statistical framework, allowing for efficient estimation of both $\mathbf{G}$ and $\boldsymbol{\theta}$ using state-of-the-art statistical inference methods.

Despite a constant diffusion assumption might seem restrictive at first, such SDE models are widely used, e.g. in chemical engineering (Karimi & McAuley, 2018), civil engineering (Jiménez et al., 2008), pharmacology (Donnet & Samson, 2013) and of course in signal processing, control and econometrics. While we believe that this approach could be extended approximately to systems with general diffusion matrices, we leave this for future work.

The contributions of our framework are the following:

- We derive a new statistical framework for diffusion and drift parameter estimation of SDEs using the Doss-Sussmann transformation and Gaussian processes.
- We introduce a grid-free, computationally efficient and robust parameter inference scheme that combines a non-parametric Gaussian process model with adversarial loss functions.
- We demonstrate that our method is able to estimate constant but non-diagonal diffusion terms of stochastic differential equations without any functional form assumption on the drift.
- We show that our method significantly outperforms the state-of-the-art algorithms for SDEs with multi-modal state posteriors, both in terms of diffusion and drift parameter estimation.
- We share and publish our code to facilitate future research at `https://github.com/gabb7/AReS-MaRS`.

2. Background

In this section, we formalize our problem and introduce the necessary notation and background drawn from Gaussian process-based gradient matching for ODEs.

2.1. Problem Setting

We consider SDEs of the form

$$\mathrm{d}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), \boldsymbol{\theta})\mathrm{d}t + \mathbf{G}\mathrm{d}\mathbf{w}(t), \tag{2}$$

where $\mathbf{x}(t) = [\mathbf{x}_1(t), \dots, \mathbf{x}_K(t)]^\top$ is the K-dimensional state vector at time t; $\mathrm{d}\mathbf{w}(t)$ are the increments of a standard Wiener process; $\mathbf{f}$ is an arbitrary, potentially highly nonlinear function whose parametric form is known, save for the unknown parameter vector $\boldsymbol{\theta}$; $\mathbf{G}$ is the unknown but

constant diffusion matrix. Without loss of generality, we can assume $\mathbf{G}$ to be a lower-diagonal, positive semi-definite matrix.

The system is observed at N arbitrarily spaced time points $\mathbf{t} = [t_1, \dots, t_N]$, subjected to Gaussian observation noise:

$$\mathbf{y}(t_n) = \mathbf{x}(t_n) + \mathbf{e}(t_n) \qquad \forall\, n = 1, \dots, N, \tag{3}$$

where we assume the noise variances to be state-dependent but time-independent, i.e.

$$p(\mathbf{e}(t_n)) = \prod_{k=1}^{K} \mathcal{N}\left(\mathbf{e}_k(t_n) \mid 0, \sigma_k\right), \tag{4}$$

for $n = 1, \dots, N$ and $k = 1, \dots, K$.

2.2. Deterministic ODE Case

In the context of Bayesian parameter inference for deterministic ordinary differential equations, Calderhead et al. (2009) identify numerical integration as the main culprit for bad computational performance. Thus, they propose to turn the parameter estimation procedure on its head: instead of calculating $p(\mathbf{y} \mid \boldsymbol{\theta})$ using numerical integration, they extract two probabilistic estimates for the derivatives, one using only the noisy observations $\mathbf{y}$ and one using the differential equations. The main challenge is then to combine these two distributions, such that more information about $\mathbf{y}$ can guide towards better parameter estimates $\boldsymbol{\theta}$. For this purpose, Calderhead et al. (2009) propose a product of experts heuristics that was accepted and reused until recently Wenk et al. (2018) showed that this heuristic leads to severe theoretical issues. They instead propose an alternative graphical model, forcing equality between the data based and the ODE based model save for a Gaussian distributed slack variable.

In this paper, we use another interpretation of gradient matching, which is aimed at finding parameters $\boldsymbol{\theta}$ such that the two distributions over $\dot{\mathbf{x}}$ match as closely as possible. Acknowledging the fact that standard methods like minimizing the KL divergence are not tractable, we use robust moment matching techniques while solving a much harder problem with $\mathbf{G} \neq \mathbf{0}$. However, it should be clear that our methodology could easily be applied to the special case of deterministic ODEs and thus provides an additional contribution towards parameter estimation for systems of ODEs.

2.3. Notation

Throughout this paper, bold, capital letters describe matrices. Values of a time-dependent quantities such as the state vector $\mathbf{x}(t)$ can be collected in the matrix $\mathbf{X} = [\mathbf{x}(t_1), \dots, \mathbf{x}(t_N)]$ of dimensions $K \times N$, where the k-th row collect the N single-state values at times $\mathbf{t} = [t_1, \dots, t_N]$ for the state k. The matrix $\mathbf{X}$ can be vectorized by concatenating its rows and defining in this way the vector $\mathbf{x} = [\mathbf{x}_1, \dots, \mathbf{x}_K]^\top$. This vector should not be confused with $\mathbf{x}(t)$, which is still a time-dependent vector of dimension K.

As we work with Gaussian processes, it is useful to standardize the state observations by subtracting the mean and dividing by the standard deviation, in a state-wise fashion. We define the vector of the data standard deviation $\boldsymbol{\sigma}_\mathbf{y} = [\sigma_{\mathbf{y}_1}, \dots, \sigma_{\mathbf{y}_K}]$, and the matrix $\mathbf{S}$ as:

$$\mathbf{S} = \boldsymbol{\sigma}_\mathbf{y} \otimes \mathbf{I}_N \tag{5}$$

where $\otimes$ indicates the Kronecker product and $\mathbf{I}_N$ is the identity matrix of size $N \times N$. Similarly for the means, we can define the $N \times K$ vector $\boldsymbol{\mu}_\mathbf{y}$ that contains the K state-wise means of the observations, each repeated N times. Thus the standardize vector $\tilde{\mathbf{x}}$ can be defined as:

$$\tilde{\mathbf{x}} = \mathbf{S}^{-1}(\mathbf{x} - \boldsymbol{\mu}_\mathbf{y}). \tag{6}$$

For the sake of clarity, we omit the normalization in the following sections. It should be noted however that in a finite sample setting, standardization strongly improves the performance of GP regression. In our implementation and all the experiments in section 4, we assume a GP prior on the states standardized using the state-wise mean and standard deviation of the observations $\mathbf{y}$.

For coherence with the current Gaussian process-based gradient matching literature, we follow the notation introduced by Calderhead et al. (2009) and Wenk et al. (2018) wherever possible.

3. Methods

In the deterministic case with $\mathbf{G} = \mathbf{0}$, the GP regression model can be directly applied to the states $\mathbf{x}$. However, if $\mathbf{G} \neq \mathbf{0}$, the derivatives of the states with respect to time t no longer exist due to the contributions of the Wiener process. Thus, performing direct gradient matching on the states is not feasible.

3.1. Latent States Representation

We propose to tackle this problem by introducing a latent variable $\mathbf{z}$, defined via the linear coordinate transformation

$$\mathbf{z}(t) = \mathbf{x}(t) - \mathbf{o}(t), \tag{7}$$

where $\mathbf{o}(t)$ is the solution of the following SDE:

$$\mathrm{d}\mathbf{o}(t) = -\mathbf{o}(t) + \mathbf{G}\mathrm{d}\mathbf{w}(t). \tag{8}$$

Without loss of generality, we set $\mathbf{z}(0) = \mathbf{x}(0)$ and thus $\mathbf{o}(0) = 0$. While in principle the framework supports any

initial condition as long as $\mathbf{z}(0) + \mathbf{o}(0) = \mathbf{x}(0)$, the reasons for this choice will become clear in section 3.2.3.

Using Itô's formula, we obtain the following SDE for $\mathbf{z}$

$$\mathrm{d}\mathbf{z}(t) = \{\mathbf{f}(\mathbf{z}(t) + \mathbf{o}(t), \boldsymbol{\theta}) + \mathbf{o}(t)\}\,\mathrm{d}t \tag{9}$$

This means that for a given realization of $\mathbf{o}(t)$, we obtain a differentiable latent state $\mathbf{z}(t)$. In principle, we could sample realizations of $\mathbf{o}(t)$ and solve the corresponding deterministic problems, which is equivalent to approximately marginalizing over $\mathbf{o}(t)$. However, it is actually possible to treat this problem statistically, completely bypassing such marginalization. We do this by creating probabilistic generative models for observations and derivatives analytically. The equations are derived in this section, while the final models are shown in Figure 1.

3.2. Generative Model for Observations

Let us define $\mathbf{e}(t)$ as the Gaussian observation error at time t. Using the matrix notation introduced in section 2.3, we can write

$$\mathbf{Y} = \mathbf{X} + \mathbf{E} = \mathbf{Z} + \mathbf{O} + \mathbf{E}, \tag{10}$$

where $\mathbf{Z}$ and $\mathbf{O}$ are the matrices corresponding to the lowercase variables introduced in the previous section. In contrast to standard GP regression, we have an additional noise term $\mathbf{O}$, which is the result of the stochastic process described by equation (8). As in standard GP regression, it is possible to recover a closed form Gaussian distribution for each term.

3.2.1. GP Prior

We assume a zero-mean Gaussian prior over the latent states $\mathbf{z}$, whose covariance matrix is given by a kernel function $k(x, y)$, in turn parameterized by the hyperparameter vector $\boldsymbol{\phi}$:

$$p(\mathbf{z} \mid \boldsymbol{\phi}) = \mathcal{N}\left(\mathbf{z} \mid \mathbf{0}, \mathbf{C}_{\boldsymbol{\phi}}\right). \tag{11}$$

We treat all state dimensions as independent, meaning that we put independent GP priors with separate hyperparameters $\boldsymbol{\phi}_k$ on the time evolution of each state. Consequently, $\mathbf{C}_{\boldsymbol{\phi}}$ is a block diagonal matrix with K blocks each of dimension $N \times N$. The blocks model the correlation over time introduced by the GP prior.

3.2.2. Error Model

In equation (4), we assume that observational errors are i.i.d. Gaussians uncorrelated over time. The joint distribution of all errors is thus still a Gaussian distribution, whose covariance $\mathbf{T}$ has only diagonal elements given by the GP likelihood variances $\boldsymbol{\sigma} = \{\sigma_k^2\}_{k=1}^K$. More precisely:

$$\mathbf{T} = \boldsymbol{\sigma} \otimes \mathbf{I}_N. \tag{12}$$

and

$$p(\mathbf{e} \mid \boldsymbol{\sigma}) = \mathcal{N}\left(\mathbf{e} \mid \mathbf{0}, \mathbf{T}\right). \tag{13}$$

3.2.3. Ornstein-Uhlenbeck Process

Through the coordinate transformation in equation (7), all stochasticity is captured by the stochastic process $\mathbf{o}(t)$ described by equation (8). Such mathematical construct has a closed-form, Gaussian solution and is called Ornstein-Uhlenbeck process. For the one-dimensional case with zero initial condition and unit diffusion

$$\mathrm{d}\hat{o}(t) = -\hat{o}(t) + \mathrm{d}w(t), \tag{14}$$

we get the following mean and covariance:

$$\mathbb{E}[\hat{o}(t)] = 0 \tag{15}$$

$$\mathrm{cov}[\hat{o}(t_i), \hat{o}(t_j)] = \frac{1}{2}e^{-|t_i - t_j|} - \frac{1}{2}e^{-(t_i + t_j)}. \tag{16}$$

Sampling $\hat{o}(t)$ at the N points $\mathbf{t} = [t_1, \ldots, t_N]$ yields the vector $\hat{\mathbf{o}}(\mathbf{t}) = [\hat{o}(t_1), \ldots, \hat{o}(t_N)]$, which is Gaussian distributed:

$$p(\hat{\mathbf{o}}(\mathbf{t})) = \mathcal{N}(\hat{\mathbf{o}}(\mathbf{t}) \mid \mathbf{0}, \boldsymbol{\Omega}_{\text{one}}), \tag{17}$$

where $[\boldsymbol{\Omega}_{\text{one}}]_{ij} = \mathrm{cov}[\hat{o}(t_i), \hat{o}(t_j)]$ according to (16). In the case of a K-dimensional process with identity diffusion, i.e.

$$\mathrm{d}\hat{\mathbf{o}}(t) = -\hat{\mathbf{o}} + \mathbf{I}_K \mathrm{d}\mathbf{w}(t), \tag{18}$$

we can just treat each state dimension as an independent, one-dimensional OU process. Thus, after sampling $\hat{\mathbf{o}}(t)$ K times at the N time points in $\mathbf{t}$ and unrolling the resulting matrix as described in section 2.3, we get

$$p(\hat{\mathbf{o}}) = \mathcal{N}(\hat{\mathbf{o}} \mid \mathbf{0}, \boldsymbol{\Omega}), \tag{19}$$

where $\boldsymbol{\Omega}$ is a block diagonal matrix with one $\boldsymbol{\Omega}_{\text{one}}$ for each state dimension.

Using Itô's formula, we can show that the samples of the original Ornstein-Uhlenbeck process $\mathbf{o}$ at each time point can be obtained via the linear coordinate transformation

$$\mathbf{o}(t) = \mathbf{G}\hat{\mathbf{o}}(t). \tag{20}$$

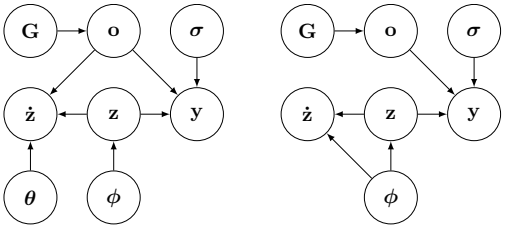
Let $\mathbf{B}$ be defined as the matrix that performs this linear transformation for the unrolled vectors $\mathbf{o} = \mathbf{B}\hat{\mathbf{o}}$. We can then write the density of the original OU process as

$$p(\mathbf{o} \mid \mathbf{G}) = \mathcal{N}\left(\mathbf{o} \mid \mathbf{0}, \mathbf{B}\boldsymbol{\Omega}\mathbf{B}^\top\right). \tag{21}$$

3.2.4. Marginals of the Observations

Using equation (10), the marginal distribution of $\mathbf{y}$ can be computed as the sum of three independent Gaussian-distributed random variables with zero mean, described respectively by equations (11), (13) and (21). Thus, $\mathbf{y}$ is again Gaussian-distributed, according to

$$p(\tilde{\mathbf{y}} \mid \boldsymbol{\phi}, \mathbf{G}, \boldsymbol{\sigma}) = \mathcal{N}(\mathbf{y} \mid \mathbf{0}, \boldsymbol{\Sigma}), \tag{22}$$



(a) SDE-based model (b) Data-based model

Figure 1. Generative models for the two different ways to compute the derivatives of the latent states $\mathbf{z}$.

where

$$\mathbf{\Sigma} = \mathbf{C}_{\boldsymbol{\phi}} + \mathbf{T} + \mathbf{B}\boldsymbol{\Omega}\mathbf{B}^T. \tag{23}$$

Thanks to the latent state representation, the diffusion matrix $\mathbf{G}$ is now a part of the hyperparameters of the observation model. It can then be inferred alongside the hyperparameters of the GP using maximum evidence (Rasmussen, 2004). Using a stationary kernel k, $\mathbf{C}_{\boldsymbol{\phi}} + \mathbf{T}$ captures the stationary part of $\mathbf{z}$ as in standard GP regression, while the parameters in $\mathbf{G}$ describe the non-stationary part due deriving from $\boldsymbol{\Omega}$. This ultimately leads to an identifiable problem.

3.3. Generative Model for Derivatives

Similarly to gradient matching approaches, we define two generative models for the derivatives $\dot{\mathbf{z}}$, one based on the data and one based on the SDE model.

3.3.1. Data-based Model

As shown e.g. in the appendix of Wenk et al. (2018), the prior defined in equation (11) automatically induces a GP prior on the conditional derivatives of $\mathbf{z}$. Defining

$$\mathbf{D} \coloneqq {}'\mathbf{C}_{\boldsymbol{\phi}}\mathbf{C}_{\boldsymbol{\phi}}^{-1}, \tag{24}$$

$$\mathbf{A} \coloneqq \mathbf{C}''_{\boldsymbol{\phi}} - {}'\mathbf{C}_{\boldsymbol{\phi}}\mathbf{C}_{\boldsymbol{\phi}}^{-1}\mathbf{C}'_{\boldsymbol{\phi}} \tag{25}$$

where

$$[{}'\mathbf{C}_{\boldsymbol{\phi}}]_{i,j} \coloneqq \frac{\partial}{\partial a} k_{\boldsymbol{\phi}}(a,b)\Big|_{a=t_i, b=t_j}, \tag{26}$$

$$\left[\mathbf{C}'_{\boldsymbol{\phi}}\right]_{i,j} \coloneqq \frac{\partial}{\partial b} k_{\boldsymbol{\phi}}(a,b)\Big|_{a=t_i, b=t_j}, \tag{27}$$

$$\left[\mathbf{C}''_{\boldsymbol{\phi}}\right]_{i,j} \coloneqq \frac{\partial^2}{\partial a \partial b} k_{\boldsymbol{\phi}}(a,b)\Big|_{a=t_i, b=t_j}, \tag{28}$$

we can write

$$p(\dot{\mathbf{z}} \mid \mathbf{z}, \boldsymbol{\phi}) = \mathcal{N}\left(\dot{\mathbf{z}} \mid \mathbf{D}\mathbf{z}, \mathbf{A}\right). \tag{29}$$

3.3.2. SDE-based Model

There also is a second way of obtaining an expression for the derivatives of $\mathbf{z}$, namely using equation (9):

$$p(\dot{\mathbf{z}} \mid \mathbf{o}, \mathbf{z}, \boldsymbol{\theta}) = \delta(\dot{\mathbf{z}} - \mathbf{f}(\mathbf{z} + \mathbf{o}, \boldsymbol{\theta}) - \mathbf{o}), \tag{30}$$

where δ represents the dirac delta.

3.4. Inference

Combined with the modeling paradigms introduced in the previous sections, this yields the two generative models for the observations in Figure 1. The graphical model in Figure 1a represents the derivatives we get via the generative process described by the SDEs, in particular the nonlinear drift function $\mathbf{f}$. The model in Figure 1b represents the derivatives yielded by the generative process described by the Gaussian process. Assuming a perfect GP fit and access to the true parameters $\boldsymbol{\theta}$, intuitively these two distributions should be equal. We thus want to find parameters $\boldsymbol{\theta}$ that minimizes the difference between these two distributions.

Compared to the deterministic ODE case, the graphical models in Figure 1 contain additional dependencies on the contribution of the OU process $\mathbf{o}$. Furthermore, the SDE-driven probability distribution of $\dot{\mathbf{z}}$ in Figure 1a depends on $\mathbf{z}$, $\mathbf{o}$, and $\boldsymbol{\theta}$ through a potentially highly nonlinear drift function $\mathbf{f}$. Thus, one cannot do analytical inference without making restrictive assumptions on the functional form of $\mathbf{f}$.

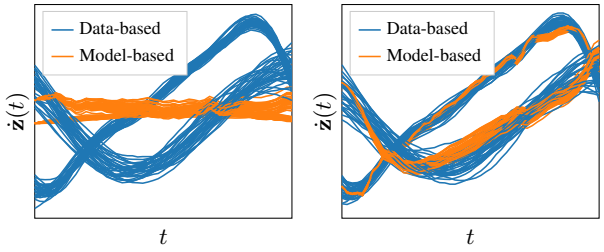
However, as shown in section A.3 of the appendix , it is possible to derive computationally efficient ancestral sampling schemes for both models, as summarized in Algorithm 1. While this rules out classical approaches like analytically minimizing the KL divergence, we can now deploy likelihood-free algorithms that were designed for matching two probability densities based on samples.

Algorithm 1 Ancestral sampling for $\dot{\mathbf{z}}$

Input: $\mathbf{y}, \mathbf{f}(\mathbf{z}, \boldsymbol{\theta}), \mathbf{t}, \boldsymbol{\sigma}, \mathbf{G}$
Ancestral sampling the SDE model
Sample $\mathbf{o}_\text{s}$ by drawing from $p(\mathbf{o} \mid \mathbf{G})$
Sample $\mathbf{z}_\text{s}$ by drawing from $p(\mathbf{z} \mid \mathbf{y}, \mathbf{o}, \boldsymbol{\sigma})$ using $\mathbf{o}_\text{s}$
Sample $\dot{\mathbf{z}}_\text{s}$ by drawing from $p(\dot{\mathbf{z}} \mid \mathbf{o}, \mathbf{z}, \boldsymbol{\theta})$ using $\mathbf{o}_\text{s}, \mathbf{z}_\text{s}$
Ancestral sampling the Data model
Sample $\mathbf{z}_\text{d}$ by drawing from $p(\mathbf{z} \mid \mathbf{y}, \mathbf{G}, \boldsymbol{\sigma})$
Sample $\dot{\mathbf{z}}_\text{d}$ by drawing from $p(\dot{\mathbf{z}} \mid \mathbf{z}, \boldsymbol{\phi})$
Return: $\dot{\mathbf{z}}_\text{s}, \dot{\mathbf{z}}_\text{d}$

3.5. Adversarial Sample-based Inference

Arguably, generative adversarial networks (GANs) (Goodfellow et al., 2014) are amongst the most popular algorithms of this kind; here a parametric neural network is trained to match the unknown likelihood of the data. The basic GAN setup consists of a fixed data set, a generator that tries to create realistic samples of said dataset and a discriminator that tries to tell apart the fake samples from the true ones. As recently shown by Yang et al. (2018), GANs have the potential to solve stochastic partial differential equations



(a) Before Training (b) After Training

Figure 2. Comparing gradients sampled from the graphical model in Figure 1a (Model-based) and the graphical model in Figure 1b (Data-based) before and after adversarial training on Lotka Volterra.

(SPDEs). Yang et al. (2018) assume a fixed data set consisting of observations (similar to the $\mathbf{y}$ in this paper) and use an SPDE-inspired neural network as a generator for realistic observations. In the case of SDEs however, this would still involve a lot of numerical integration. Thus, we modify the GAN setup by leaving behind the idea of having a fixed data set. Instead of relying on bootstrapped samples of the observations $\mathbf{y}$, we sample the derivatives from the data-based model shown in Figure 1b. For a sufficiently good model fit, these samples represent the true derivatives of the latent variable $\mathbf{z}$. We then use the SDE-based model shown in Figure 1a as a generator. To avoid standard GAN problems such as training instability and to improve robustness, we choose to replace the discriminator with a critic $\mathcal{C}_{\boldsymbol{\omega}}$. As proposed by Arjovsky et al. (2017), this critic is trained to estimate the Wasserstein distance between the derivative samples. The resulting algorithm, summarized in Algorithm 2, can be interpreted as performing Adversarial Regression for SDEs and will thus be called AReS. In Figure 2, we show the derivatives sampled from the two models both before and after training for one example run of the Lotka Volterra system (cf. Section 4.4). While not perfect, the GAN is clearly able to push the SDE gradients towards the gradients of the observed data.

3.6. Maximum Mean Discrepancy

Even though they work well in practical settings, during training GANs need ad hoc precautions and careful balancing between their generator and discriminator. Dziugaite et al. (2015) propose to solve this problem using Maximum Mean Discrepancy (MMD) (Gretton et al., 2012) as a metric to substitute the discriminator. As proposed by Li et al. (2015), we choose the rational quadratic kernel to obtain a robust discriminator that can be deployed without fine-tuning on a variety of problems. The resulting procedure, summarized in Algorithm 3, can be interpreted as performing Maximum mean discrepancy-minimizing Regression for SDEs and will thus be called MaRS.

Algorithm 2 AReS

Input: Observations $\mathbf{y}$ at times $\mathbf{t}$, a model $\mathbf{f}$, learning rate α, number of total iterations N_{it}, the clipping parameter c, the batch size M, the number of iterations of the critic per generator iteration n_{critic}.
Train the Gaussian process on the data to recover the hyperparameters $\boldsymbol{\phi}$, $\boldsymbol{\sigma}$ and the diffusion $\mathbf{G}$
Initialize the critic parameters $\boldsymbol{\omega}$ and the SDE parameters $\boldsymbol{\theta}$ respectively with $\boldsymbol{\omega}_0$ and $\boldsymbol{\theta}_0$
for $n_{\text{it}} = 1, \dots, N_{\text{it}}$ **do**
  for $n_{\text{c}} = 1, \dots, n_{\text{critic}}$ **do**
    Sample $\dot{\mathbf{z}}_{\text{s}} \sim p_{\text{s}}(\dot{\mathbf{z}})$ and $\dot{\mathbf{z}}_{\text{d}} \sim p_{\text{d}}(\dot{\mathbf{z}})$ as described in algorithm 1. Each batch contains M elements
    $g_{\boldsymbol{\omega}} \leftarrow \nabla_{\boldsymbol{\omega}} \left[\frac{1}{M} \sum_{i=1}^{M} \mathcal{C}_{\boldsymbol{\omega}}(\dot{\mathbf{z}}_{\text{d}}^{(i)}) - \frac{1}{M} \sum_{i=1}^{M} \mathcal{C}_{\boldsymbol{\omega}}(\dot{\mathbf{z}}_{\text{s}}^{(i)}) \right]$
    $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \alpha \cdot \text{Adam}(\boldsymbol{\omega}, g_{\boldsymbol{\omega}})$
    $\boldsymbol{\omega} \leftarrow \text{clip}(\boldsymbol{\omega}, -c, c)$
  end for
  $g_{\boldsymbol{\theta}} \leftarrow -\nabla_{\boldsymbol{\theta}} \frac{1}{M} \sum_{i=1}^{M} f_{\boldsymbol{\omega}}(\dot{\mathbf{z}}_{\text{s}}^{(i)})$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \cdot \text{Adam}(\boldsymbol{\theta}, g_{\boldsymbol{\theta}})$
end for

Algorithm 3 MaRS

Input: Observations $\mathbf{y}$ at times $\mathbf{t}$, SDE model $\mathbf{f}$, learning rate α, number of iterations N_{it}, batch size M
Train the Gaussian process on the data to recover the hyperparameters $\boldsymbol{\phi}$, $\boldsymbol{\sigma}$ and the diffusion $\mathbf{G}$
Initialize the SDE parameters with $\boldsymbol{\theta}_0$
for $n_{\text{it}} = 1, \dots, N_{\text{it}}$ **do**
  Sample $\dot{\mathbf{z}}_{\text{s}} \sim p_{\text{s}}(\dot{\mathbf{z}})$ and $\dot{\mathbf{z}}_{\text{d}} \sim p_{\text{d}}(\dot{\mathbf{z}})$ as described in algorithm 1. Each batch contains M elements
  $g_{\boldsymbol{\theta}} \leftarrow \nabla_{\boldsymbol{\theta}} MMD_u^2 [\dot{\mathbf{z}}_{\text{s}}, \dot{\mathbf{z}}_{\text{d}}]$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \cdot \text{Adam}(\boldsymbol{\theta}, g_{\boldsymbol{\theta}})$
end for

4. Experiments

4.1. Setups

To evaluate the empirical performance of our method, we conduct several experiments on simulated data, using four standard benchmark systems and comparing against the EKF-based approach by Särkkä et al. (2015) and two GP-based approaches respectively by Vrettas et al. (2015) and Yildiz et al. (2018).

The first system is a simple Ornstein-Uhlenbeck process as shown in Figure 3a, given by the SDE

$$\mathrm{d}x(t) = \theta_0(\theta_1 - x(t))\mathrm{d}t + G\mathrm{d}w(t). \tag{31}$$

As mentioned in Section 3.2.3, this system has an analytical Gaussian process solution and thus serves more academic purposes. We use $\boldsymbol{\theta} = [0.5, 1.0]$, $G = 0.5$ and $x(0) = 10$.

The second system is the Lorenz '63 model given by the

SDEs

$$\begin{aligned}
\mathrm{d}x_1(t) &= \theta_1(x_2(t) - x_1(t))\mathrm{d}t &&+ \sigma_1 \mathrm{d}w_1(t)\\
\mathrm{d}x_2(t) &= (\theta_2 x_1(t) - x_2(t) - x_1(t)x_3(t))\mathrm{d}t &&+ \sigma_2 \mathrm{d}w_2(t)\\
\mathrm{d}x_3(t) &= (x_1(t)x_2(t) - \theta_3 x_3(t))\mathrm{d}t &&+ \sigma_3 \mathrm{d}w_3(t).
\end{aligned}$$

In both systems, the drift function is linear in one state or one parameter conditioned on all the others (cf. Gorbach et al., 2017). Furthermore, there is no coupling across state dimensions in the diffusion matrix. This leads to two more interesting test cases.

To investigate the algorithm's capability to deal with off-diagonal terms in the diffusion, we introduce the two dimensional Lotka-Volterra system shown in Figure 3b, given by the SDEs

$$\mathrm{d}\mathbf{x}(t) = \begin{bmatrix} \theta_1 x_1(t) - \theta_2 x_1(t)x_2(t) \\ -\theta_3 x_2(t) + \theta_4 x_1(t)x_2(t) \end{bmatrix} \mathrm{d}t + \mathbf{G}\mathrm{d}\mathbf{w}(t), \quad (32)$$

where $\mathbf{G}$ is, without loss of generality, assumed to be a lower triangular matrix. The true vector parameter is $\boldsymbol{\theta} = [2, 1, 4, 1]$ and the system is simulated starting from $\mathbf{x}(0) = [3, 5]$. Since its original introduction by Lotka (1932), the Lotka Volterra system has been widely used to model population dynamics in biology. The system is observed at 50 equidistant points in the interval $t = [0,\ 20]$. As it turns out, this problem is significantly challenging for all algorithms, despite the absence of observation noise.

To investigate the effect of strong non-linearities in the drift, we introduce the Ginzburg-Landau double-well potential shown in Figure 3c, defined by the SDE

$$\mathrm{d}x(t) = \theta_0 x(\theta_1 - x^2)\mathrm{d}t + G\mathrm{d}w(t). \quad (33)$$

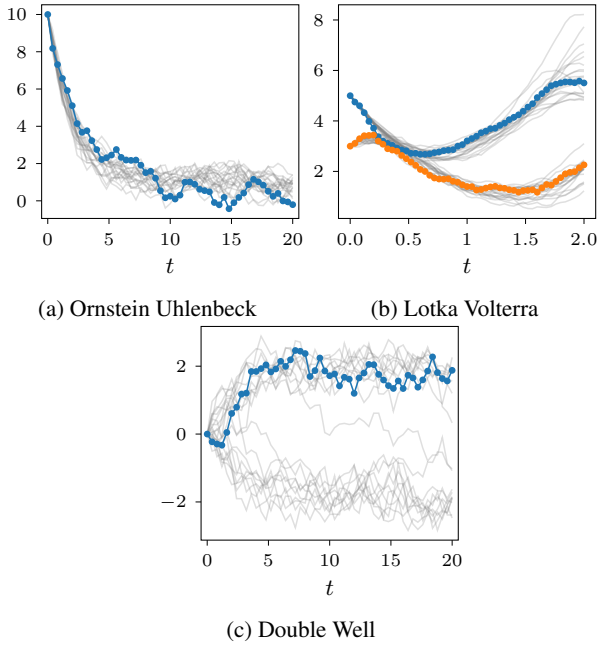
Using $\boldsymbol{\theta} = [0.1, 4]$, $G = 0.5$ and $x(0) = 0$, this system exhibits an interesting bifurcation effect. While there are two stable equilibria at $x = \pm 2$, the one the system will end up in is completely up to noise. For this reason it represents a fitting framework to test how well an algorithm can deal with multi-modal SDEs. The potential value is observed at 50 equidistant points in the interval $t = [0,\ 20]$, subjected to observational noise with $\sigma = 0.2$.

Lastly, some implementation details are constant throughout each experiment: the critic in the adversarial parameter estimation is a 2-layer fully connected neural network, with respectively 256 and 128 nodes. Every batch, for both MMD and adversarial training contains 256 elements. While the Ornstein-Uhlenbeck process and the double-well potential were modeled with a sigmoid kernel, for Lotka-Volterra and Lorenz '63 we used a common RBF (we point at Rasmussen (2004) for more information about kernels and GPs).

4.2. Evaluation

For all systems, the parameters $\boldsymbol{\theta}$ turn out to be identifiable. Thus, the parameter value is a good indicator of how well an algorithm is able to infer the drift function. However, since the components of $\mathrm{d}\mathbf{w}(t)$ are independent, there are multiple diffusion matrices $\mathbf{G}$ that generate the same trajectories. We thus directly compare the variance of the increments, i.e. the elements of $\mathbf{H} := \mathbf{G}^T\mathbf{G}$.

To account for statistical fluctuations, we use 100 independent realizations of the SDE systems and compare the mean and standard deviation of $\boldsymbol{\theta}$ and $\mathbf{H}$. AReS and MaRS are compared against the Gaussian process-based VGPA by Vrettas et al. (2015) and NPSDE by Yildiz et al. (2018) as well as the classic Kalman filter-based ESGF recommended by Särkkä et al. (2015).



(a) Ornstein Uhlenbeck (b) Lotka Volterra

(c) Double Well

Figure 3. Sample trajectories for three different benchmark systems. While Ornstein Uhlenbeck and Lotka Volterra are rather tame, the Double Well potential clearly exhibits a bifurcation effect.

4.3. Locally Linear Systems

As mentioned in Section 4.1, the functional form of the drift functions of both the Ornstein-Uhlenbeck process and the Lorenz '63 system satisfies a local linearity assumption, while their diffusion is kept diagonal. Thus, they serve as excellent benchmarks for parameter inference algorithms. The empirical results are shown in Table 1a for the Ornstein-Uhlenbeck. Unfortunately, VGPA turns out to be rather unstable if both diffusion and parameters are unknown, despite on average roughly 54 hours are needed to observe convergence. We then provide it with the true $\mathbf{G}$ and show only its empirical parameter estimates. Since both AReS and MaRS use Equation (22) to determine $\mathbf{G}$, they share

Table 1. Inferred parameters over 100 independent realizations of respectively the Ornstein-Uhlenbeck, Ginzburg-Landau Double-Well and Lotka-Volterra dynamics. For every algorithm, we show the median $\pm$ one standard deviation.

Ground truth	NPSDE	VGPA	ESGF	AReS	MaRS
$\theta_0 = 0.5$	0.41 ± 0.11	0.53 ± 0.08	0.49 ± 0.07	$\mathbf{0.50 \pm 0.21}$	0.46 ± 0.06
$\theta_1 = 1$	0.71 ± 1.34	0.96 ± 0.31	0.96 ± 0.24	1.06 ± 0.93	$\mathbf{0.99 \pm 0.25}$
$H = 0.25$	0.00 ± 0.01	/	0.19 ± 0.06	$\mathbf{0.24 \pm 0.09}$	

(a) Ornstein-Uhlenbeck process

Ground truth	NPSDE	ESGF	AReS	MaRS
$\theta_0 = 2$	1.58 ± 0.71	2.04 ± 0.09	2.36 ± 0.18	$\mathbf{2.00 \pm 0.09}$
$\theta_1 = 1$	0.74 ± 0.31	1.02 ± 0.05	1.18 ± 0.9	$\mathbf{1.00 \pm 0.04}$
$\theta_2 = 4$	2.26 ± 1.51	3.87 ± 0.59	$\mathbf{3.97 \pm 0.63}$	3.70 ± 0.51
$\theta_3 = 1$	0.49 ± 0.35	0.96 ± 0.14	$\mathbf{0.98 \pm 0.18}$	0.91 ± 0.14
$\mathbf{H}_{1,1} = 0.05$	/	0.01 ± 0.03	$\mathbf{0.03 \pm 0.004}$	
$\mathbf{H}_{1,2} = 0.03$	/	0.01 ± 0.01	$\mathbf{0.02 \pm 0.01}$	
$\mathbf{H}_{2,1} = 0.03$	/	0.01 ± 0.01	$\mathbf{0.02 \pm 0.01}$	
$\mathbf{H}_{2,2} = 0.09$	/	0.03 ± 0.02	$\mathbf{0.09 \pm 0.03}$	

(b) Lotka-Volterra

Ground truth	NPSDE	VGPA	ESGF	AReS	MaRS
$\theta_0 = 0.1$	0.09 ± 7.00	0.05 ± 0.04	0.01 ± 0.03	0.09 ± 0.04	$\mathbf{0.10 \pm 0.05}$
$\theta_1 = 4$	3.36 ± 248.82	1.11 ± 0.66	0.11 ± 0.16	3.68 ± 1.34	$\mathbf{3.85 \pm 1.10}$
$H = 0.25$	0.00 ± 0.02	/	0.20 ± 0.05	$\mathbf{0.21 \pm 0.09}$	

(c) Double-Well potential

the same values. Due to space restrictions, the results for Lorenz '63 can be found in Table 2 of the appendix. As demonstrated by this experiment, AReS and MaRS can deal with locally linear systems, outperforming their competitors, especially in their estimates of the diffusion terms.

4.4. Non-Diagonal Diffusion

To investigate the effect of off-diagonal entries in $\mathbf{G}$, we use the Lotka-Volterra dynamics. Since NPSDE is unable to model non-diagonal diffusions, we provide it with the true $\mathbf{G}$ and only compare parameter estimates. As VGPA is already struggling in the lower dimensional cases, we omit it from this comparison due to limited computational resources. The results are shown in Table 1b. AReS and MaRS clearly outperform the other methods in terms of diffusion estimation, while ESGF is the only algorithm that yields drift parameter estimates of comparable quality.

4.5. Dealing with Multi-Modality

As a final challenge, we investigate the Ginzburg-Landau double well potential. Despite one-dimensional, its state distribution is multi-modal even if all parameters are known. As shown in Table 1c, this is definitely a challenge for all classical approaches. While the number of data-points is probably not enough for the non-parametric proxy for the drift function in NPSDE, the time-dependent Gaussianity assumptions in both VGPA and ESGF are problematic in this case. In our gradient matching framework, no such assumption is made. Thus, both AReS and MaRS are able to deal with the multimodality of the problem.

5. Conclusion

Parameter and diffusion estimations in stochastic systems arise in quantitative sciences and many fields of engineering. Current techniques based on Kalman filtering or Gaussian processes approximate the state distribution conditioned on the parameters and iteratively optimize the data likelihood. In this work, we propose to turn this procedure on its head by leveraging key ideas from gradient matching algorithms, originally designed for deterministic ODEs. By introducing a novel noise model for Gaussian process regression that leverages the Doss-Sussmann transformation, we are able to reliably estimate the parameters in the drift and the diffusion processes. Our algorithm can keep up with and occasionally outperform the state-of-the-art on the simpler benchmark systems, while it is also accurately estimating parameters for systems that exhibit multi-modal state densities, a case where traditional methods fail. While our approach is currently restricted to systems with a constant diffusion matrix $\mathbf{G}$, it would be interesting to see how it generalizes to other settings, perhaps using alternative or approximate bridge constructs. Unfortunately, this is outside of the scope of this work. We hope nevertheless that the publicly available code will facilitate future research in that direction.

Acknowledgements

This research was supported by the Max Planck ETH Center for Learning Systems. GA acknowledges funding from Google DeepMind and University of Oxford. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme grant agreement No 815943.

References

Aït-Sahalia, Y. Maximum likelihood estimation of discretely sampled diffusions: a closed-form approximation approach. *Econometrica*, 70(1):223–262, 2002.

Archambeau, C., Cornford, D., Opper, M., and Shawe-Taylor, J. Gaussian process approximations of stochastic differential equations. *Journal of machine learning research*, 1:1–16, 2007.

Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pp. 214–223, 2017.

Bauer, S., Gorbach, N. S., Miladinovic, D., and Buhmann, J. M. Efficient and flexible inference for stochastic systems. In *Advances in Neural Information Processing Systems*, pp. 6988–6998, 2017.

Bhat, H. S., Madushani, R., and Rawat, S. Parameter inference for stochastic differential equations with density tracking by quadrature. In *International Workshop on Simulation*, pp. 99–113. Springer, 2015.

Calderhead, B., Girolami, M., and Lawrence, N. D. Accelerating bayesian inference over nonlinear differential equations with gaussian processes. In *Advances in neural information processing systems*, pp. 217–224, 2009.

Donnet, S. and Samson, A. A review on estimation of stochastic differential equations for pharmacokinetic/pharmacodynamic models. *Advanced Drug Delivery Reviews*, 65(7):929–939, 2013.

Doss, H. Liens entre équations différentielles stochastiques et ordinaires. 1977.

Dziugaite, G. K., Roy, D. M., and Ghahramani, Z. Training generative neural networks via maximum mean discrepancy optimization. *arXiv preprint arXiv:1505.03906*, 2015.

Elerian, O., Chib, S., and Shephard, N. Likelihood inference for discretely observed nonlinear diffusions. *Econometrica*, 69(4):959–993, 2001.

Eraker, B. Mcmc analysis of diffusion models with application to finance. *Journal of Business & Economic Statistics*, 19(2):177–191, 2001.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Gorbach, N. S., Bauer, S., and Buhmann, J. M. Scalable variational inference for dynamical systems. In *Advances in Neural Information Processing Systems*, pp. 4806–4815, 2017.

Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. A kernel two-sample test. *Journal of Machine Learning Research*, 13(Mar):723–773, 2012.

Hurn, A. and Lindsay, K. Estimating the parameters of stochastic differential equations. *Mathematics and computers in simulation*, 48(4-6):373–384, 1999.

Hurn, A. S., Jeisman, J., and Lindsay, K. A. Seeing the wood for the trees: A critical evaluation of methods to estimate the parameters of stochastic differential equations. *Journal of Financial Econometrics*, 5(3):390–455, 2007.

Jiménez, M., Madsen, H., Bloem, J., and Dammann, B. Estimation of non-linear continuous time models for the heat exchange dynamics of building integrated photovoltaic modules. *Energy and Buildings*, 40(2):157–167, 2008.

Karimi, H. and McAuley, K. B. Bayesian objective functions for estimating parameters in nonlinear stochastic differential equation models with limited data. *Industrial & Engineering Chemistry Research*, 57(27):8946–8961, 2018.

Li, Y., Swersky, K., and Zemel, R. Generative moment matching networks. In *International Conference on Machine Learning*, pp. 1718–1727, 2015.

Lotka, A. J. The growth of mixed populations: two species competing for a common food supply. *Journal of the Washington Academy of Sciences*, 22(16/17):461–469, 1932.

Nielsen, J. N., Madsen, H., and Young, P. C. Parameter estimation in stochastic differential equations: an overview. *Annual Reviews in Control*, 24:83–94, 2000.

Picchini, U. Sde toolbox: Simulation and estimation of stochastic differential equations with matlab. 2007.

Pieschner, S. and Fuchs, C. Bayesian inference for diffusion processes: Using higher-order approximations for transition densities. *arXiv preprint arXiv:1806.02429*, 2018.

Rasmussen, C. E. Gaussian processes in machine learning. In *Advanced lectures on machine learning*, pp. 63–71. Springer, 2004.

Riesinger, C., Neckel, T., and Rupp, F. Solving random ordinary differential equations on gpu clusters using multiple levels of parallelism. *SIAM Journal on Scientific Computing*, 38(4):C372–C402, 2016.

Ruttor, A., Batz, P., and Opper, M. Approximate gaussian process inference for the drift function in stochastic differential equations. In *Advances in Neural Information Processing Systems*, pp. 2040–2048, 2013.

Ryder, T., Golightly, A., McGough, A. S., and Prangle, D. Black-box variational inference for stochastic differential equations. *International Conference on Machine Learning*, 2018.

Särkkä, S., Hartikainen, J., Mbalawata, I. S., and Haario, H. Posterior inference on parameters of stochastic differential equations via non-linear gaussian filtering and adaptive mcmc. *Statistics and Computing*, 25(2):427–437, 2015.

Sørensen, H. Parametric inference for diffusion processes observed at discrete points in time: a survey. *International Statistical Review*, 72(3):337–354, 2004.

Sussmann, H. J. On the gap between deterministic and stochastic ordinary differential equations. *The Annals of Probability*, pp. 19–41, 1978.

Tronarp, F. and Särkkä, S. Iterative statistical linear regression for gaussian smoothing in continuous-time non-linear stochastic dynamic systems. *arXiv preprint arXiv:1805.11258*, 2018.

van der Meulen, F., Schauer, M., et al. Bayesian estimation of discretely observed multi-dimensional diffusion processes using guided proposals. *Electronic Journal of Statistics*, 11(1):2358–2396, 2017.

Vrettas, M. D., Opper, M., and Cornford, D. Variational mean-field algorithm for efficient inference in large systems of stochastic differential equations. *Physical Review E*, 91(1):012148, 2015.

Wenk, P., Gotovos, A., Bauer, S., Gorbach, N., Krause, A., and Buhmann, J. M. Fast gaussian process based gradient matching for parameter identification in systems of nonlinear odes. *arXiv preprint arXiv:1804.04378*, 2018.

Yang, L., Zhang, D., and Karniadakis, G. E. Physics-informed generative adversarial networks for stochastic differential equations. *arXiv preprint arXiv:1811.02033*, 2018.

Yildiz, C., Heinonen, M., Intosalmi, J., Mannerström, H., and Lähdesmäki, H. Learning stochastic differential equations with gaussian processes without gradient matching. *arXiv preprint arXiv:1807.05748*, 2018.

A. Supplementary Material

A.1. Parameter Estimation Lorenz '63

Ground truth	NPSDE	ESGF	AReS	MaRS
$\theta_0 = 10$	1.28 ± 2.32	$\mathbf{9.97 \pm 0.33}$	7.24 ± 1.08	9.82 ± 0.56
$\theta_1 = 28$	20.69 ± 5.73	$\mathbf{28.00 \pm 0.17}$	28.16 ± 1.08	27.96 ± 0.21
$\theta_0 = 2.667$	1.86 ± 1.08	$\mathbf{2.65 \pm 0.06}$	2.55 ± 0.10	2.64 ± 0.07
$G = \sqrt{10}$	6.51 ± 1.31	$\mathbf{3.03 \pm 0.2}$	3.54 ± 2.45	

Table 2. Median and standard deviation of the 65 best runs of each algorithm. As ESGF crashed in roughly one third of all experiments, we compare only the best 65 runs, where a crash is treated as a complete failure. While this provides somehow a fair comparison, it should be noted that this significantly overestimates the performance of all algorithms.

A.2. Training Times

	NPSDE	VGPA	ESGF	AReS	MaRS
OU Process	48.8 ± 0.9	$\sim (54 \pm 8)$hours	32.2 ± 0.3	321.3 ± 0.8	17.3 ± 0.3
DW Potential	406.9 ± 147.9	$\sim (12 \pm 6)$hours	35.2 ± 0.1	326.0 ± 1.8	17.7 ± 2.0
Lotka-Volterra	1421.8 ± 1.0	/	244.7 ± 1.2	47.6 ± 1.3	19.3 ± 1.1
Lorenz '63	39273.5 ± 8.9	/	670.7 ± 10.7	26274.0 ± 2529.8	721.1 ± 10.7

Table 3. Computational times (in seconds) required for training the different algorithms.

A.3. Densities for Ancestral Sampling of the SDE-Based Model

Given the graphical model in Figure 1a, it is straightforward to compute the densities used in the ancestral sampling scheme in Algorithm 1. After marginalizing out $\dot{\mathbf{z}}$, the joint density described by the graphical model can be written as

$$p(\mathbf{o}, \mathbf{z}, \mathbf{y}|\boldsymbol{\phi}, \mathbf{G}, \boldsymbol{\sigma}) = p(\mathbf{o}|\mathbf{G})p(\mathbf{z}|\boldsymbol{\phi})p(\mathbf{y}|\mathbf{z}, \mathbf{o}, \boldsymbol{\sigma}) \tag{34}$$

Substituting the densities given by Equations (10), (11), (13) and (21) yields

$$p(\mathbf{o}, \mathbf{z}, \mathbf{y}|\boldsymbol{\phi}, \mathbf{G}, \boldsymbol{\sigma}) = \mathcal{N}(\mathbf{o}|\mathbf{0}, \mathbf{B}\boldsymbol{\Omega}\mathbf{B}^T)\mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{C}_{\boldsymbol{\phi}})\mathcal{N}(\mathbf{y}|\mathbf{z} + \mathbf{o}, \mathbf{T}). \tag{35}$$

Using a change of variables to simplify notation, we write

$$p(\mathbf{o}, \mathbf{z}, \mathbf{y}|\boldsymbol{\phi}, \mathbf{G}, \boldsymbol{\sigma}) = \mathcal{N}(\mathbf{o}|\mathbf{0}, \tilde{\boldsymbol{\Omega}})\mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{C}_{\boldsymbol{\phi}})\mathcal{N}(\mathbf{y}|\mathbf{z} + \mathbf{o}, \mathbf{T}). \tag{36}$$

This equation is now subsequently modified by observing that the product of two Gaussian densities in the same random variable is again a Gaussian density:

$$\begin{aligned} p(\mathbf{o}, \mathbf{z}, \mathbf{y}|\boldsymbol{\phi}, \mathbf{G}, \boldsymbol{\sigma}) &= \mathcal{N}(\mathbf{o}|\mathbf{0}, \tilde{\boldsymbol{\Omega}})\mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{C}_{\boldsymbol{\phi}})\mathcal{N}(\mathbf{y}|\mathbf{z} + \mathbf{o}, \mathbf{T}) \\ &= \mathcal{N}(\mathbf{o}|\mathbf{0}, \tilde{\boldsymbol{\Omega}})\mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{C}_{\boldsymbol{\phi}})\mathcal{N}(\mathbf{z}|\mathbf{y} - \mathbf{o}, \mathbf{T}) \\ &= \mathcal{N}(\mathbf{o}|\mathbf{0}, \tilde{\boldsymbol{\Omega}})\mathcal{N}(\mathbf{y} - \mathbf{o}|\mathbf{0}, \mathbf{C}_{\boldsymbol{\phi}} + \mathbf{T})\mathcal{N}(\mathbf{z}|\mathbf{m}_z, \mathbf{C}_z) \\ &= \mathcal{N}(\mathbf{o}|\mathbf{0}, \tilde{\boldsymbol{\Omega}})\mathcal{N}(\mathbf{o}|\mathbf{y}, \mathbf{C}_{\boldsymbol{\phi}} + \mathbf{T})\mathcal{N}(\mathbf{z}|\mathbf{m}_z, \mathbf{C}_z) \\ &= \mathcal{N}(\mathbf{y}|\mathbf{0}, \tilde{\boldsymbol{\Omega}} + \mathbf{C}_{\boldsymbol{\phi}} + \mathbf{T})\mathcal{N}(\mathbf{o}|\mathbf{m}_o, \mathbf{C}_o)\mathcal{N}(\mathbf{z}|\mathbf{m}_z, \mathbf{C}_z) \end{aligned} \tag{37}$$

where

$$\mathbf{m}_z = \mathbf{C}_z(\mathbf{T}^{-1}(\mathbf{y} - \mathbf{o})) \tag{38}$$

$$\mathbf{C}_z = (\mathbf{C}_{\boldsymbol{\phi}}^{-1} + \mathbf{T}^{-1})^{-1} \tag{39}$$

$$\mathbf{m}_o = \mathbf{C}_o(\mathbf{C}_{\boldsymbol{\phi}} + \mathbf{T})^{-1}\mathbf{y} \tag{40}$$

$$\mathbf{C}_o = (\tilde{\boldsymbol{\Omega}}^{-1} + (\mathbf{C}_{\boldsymbol{\phi}} + \mathbf{T})^{-1})^{-1} \tag{41}$$

This formula can be further refined with the Woodbury identity, i.e.

$$\begin{aligned}\mathbf{C}_z &= (\mathbf{C}_{\boldsymbol{\phi}}^{-1} + \mathbf{T}^{-1})^{-1} \\ &= \mathbf{C}_{\boldsymbol{\phi}} - \mathbf{C}_{\boldsymbol{\phi}}(\mathbf{C}_{\boldsymbol{\phi}} + \mathbf{T})^{-1}\mathbf{C}_{\boldsymbol{\phi}} \\ &= \mathbf{C}_{\boldsymbol{\phi}}(\mathbf{C}_{\boldsymbol{\phi}} + \mathbf{T})^{-1}\mathbf{T} \end{aligned} \tag{42}$$

which leads to

$$\mathbf{m}_z = \mathbf{C}_{\boldsymbol{\phi}}(\mathbf{C}_{\boldsymbol{\phi}} + \mathbf{T})^{-1}(\mathbf{y} - \mathbf{o}) \tag{43}$$

and

$$\begin{aligned}\mathbf{C}_o &= (\tilde{\boldsymbol{\Omega}}^{-1} + (\mathbf{C}_{\boldsymbol{\phi}} + \mathbf{T})^{-1})^{-1} \\ &= \tilde{\boldsymbol{\Omega}} - \tilde{\boldsymbol{\Omega}}(\tilde{\boldsymbol{\Omega}} + \mathbf{C}_{\boldsymbol{\phi}} + \mathbf{T})^{-1}\tilde{\boldsymbol{\Omega}} \\ &= \tilde{\boldsymbol{\Omega}}(\tilde{\boldsymbol{\Omega}} + \mathbf{C}_{\boldsymbol{\phi}} + \mathbf{T})^{-1}(\mathbf{C}_{\boldsymbol{\phi}} + \mathbf{T}) \end{aligned} \tag{44}$$

which leads to

$$\mathbf{m}_o = \tilde{\boldsymbol{\Omega}}(\tilde{\boldsymbol{\Omega}} + \mathbf{C}_{\boldsymbol{\phi}} + \mathbf{T})^{-1}\mathbf{y} \tag{45}$$

Since we observe $\mathbf{y}$, we are interested in calculating the conditional distribution

$$p(\mathbf{o}, \mathbf{z}|\mathbf{y}, \boldsymbol{\phi}, \mathbf{G}, \boldsymbol{\sigma}) = \frac{p(\mathbf{o}, \mathbf{z}, \mathbf{y}|\boldsymbol{\phi}, \mathbf{G}, \boldsymbol{\sigma})}{p(\mathbf{y}|\boldsymbol{\phi}, \mathbf{G}.\boldsymbol{\sigma})} \tag{46}$$

Conveniently enough, the marginal density of $\mathbf{y}$ is already factorized out in Equation (37) (compare Equation (22)). Thus, we have

$$p(\mathbf{o}, \mathbf{z}|\mathbf{y}, \boldsymbol{\phi}, \mathbf{G}, \boldsymbol{\sigma}) = \mathcal{N}(\mathbf{o}|\mathbf{m}_o, \mathbf{C}_o)\mathcal{N}(\mathbf{z}|\mathbf{m}_z, \mathbf{C}_z) \tag{47}$$

As $\mathcal{N}(\mathbf{o}|\mathbf{m}_o, \mathbf{C}_o)$ is independent of $\mathbf{z}$, we can employ ancestral sampling by first obtaining a sample of $\mathbf{o}$ through $\mathcal{N}(\mathbf{o}|\mathbf{m}_o, \mathbf{C}_o)$, and then utilizing such sample to get $\mathbf{z}$ through $\mathcal{N}(\mathbf{z}|\mathbf{m}_z, \mathbf{C}_z)$.

A.4. Calculating the GP Posterior for Data-Based Ancestral Sampling

Given the graphical model in Figure 1b, we can calculate the densities used in the ancestral sampling scheme in Algorithm 1. After marginalizing out $\dot{\mathbf{z}}$ and using the variable substitutions introduced in Equation (36), the joint density described by the graphical model can be written as

$$\begin{aligned}p(\mathbf{o}, \mathbf{z}, \mathbf{y}|\boldsymbol{\phi}, \mathbf{G}, \boldsymbol{\sigma}) &= p(\mathbf{o}|\mathbf{G})p(\mathbf{y}|\boldsymbol{\sigma}, \mathbf{o}, \mathbf{z})p(\mathbf{z}|\boldsymbol{\phi}) \\ &= \mathcal{N}(\mathbf{o}|\mathbf{0}, \tilde{\boldsymbol{\Omega}})\mathcal{N}(\mathbf{y}|\mathbf{z} + \mathbf{o}, \mathbf{T})\mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{C}_{\boldsymbol{\phi}}) \\ &= \mathcal{N}(\mathbf{o}|\mathbf{0}, \tilde{\boldsymbol{\Omega}})\mathcal{N}(\mathbf{o}|\mathbf{y} - \mathbf{z}, \mathbf{T})\mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{C}_{\boldsymbol{\phi}}) \\ &= \mathcal{N}(\mathbf{o}|\mathbf{m}, \mathbf{C})\mathcal{N}(\mathbf{y} - \mathbf{z}|\mathbf{0}, \tilde{\boldsymbol{\Omega}} + \mathbf{T})\mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{C}_{\boldsymbol{\phi}}) \\ &= \mathcal{N}(\mathbf{o}|\mathbf{m}, \mathbf{C})\mathcal{N}(\mathbf{z}|\mathbf{y}, \tilde{\boldsymbol{\Omega}} + \mathbf{T})\mathcal{N}(\mathbf{z}|\mathbf{0}, \mathbf{C}_{\boldsymbol{\phi}}) \\ &= \mathcal{N}(\mathbf{o}|\mathbf{m}, \mathbf{C})\mathcal{N}(\mathbf{y}|\mathbf{0}, \tilde{\boldsymbol{\Omega}} + \mathbf{T} + \mathbf{C}_{\boldsymbol{\phi}})\mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z), \end{aligned} \tag{48}$$

where

$$\boldsymbol{\mu}_z = \boldsymbol{\Sigma}_z(\tilde{\boldsymbol{\Omega}} + \mathbf{T})^{-1}\mathbf{y} \tag{49}$$

$$\begin{aligned}\boldsymbol{\Sigma}_z &= ((\tilde{\boldsymbol{\Omega}} + \mathbf{T})^{-1} + \mathbf{C}_{\boldsymbol{\phi}}^{-1})^{-1} \\ &= \mathbf{C}_{\boldsymbol{\phi}} - \mathbf{C}_{\boldsymbol{\phi}}(\tilde{\boldsymbol{\Omega}} + \mathbf{T} + \mathbf{C}_{\boldsymbol{\phi}})^{-1}\mathbf{C}_{\boldsymbol{\phi}} \\ &= (\tilde{\boldsymbol{\Omega}} + \mathbf{T})(\tilde{\boldsymbol{\Omega}} + \mathbf{T} + \mathbf{C}_{\boldsymbol{\phi}})^{-1}\mathbf{C}_{\boldsymbol{\phi}} \\ &= \mathbf{C}_{\boldsymbol{\phi}}(\tilde{\boldsymbol{\Omega}} + \mathbf{T} + \mathbf{C}_{\boldsymbol{\phi}})^{-1}(\tilde{\boldsymbol{\Omega}} + \mathbf{T}). \end{aligned} \tag{50}$$

After marginalizing out $\mathbf{o}$ and dividing by the marginal of $\mathbf{y}$, we get the conditional distribution

$$p(\mathbf{z}|\mathbf{y}, \boldsymbol{\phi}, \mathbf{G}, \boldsymbol{\sigma}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z). \tag{51}$$

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	AReS and MaRS - Adversarial and MMD-Minimizing Regression for SDEs
Publication Status	☒ Published ☐ Accepted for Publication ☐ Submitted for Publication ☐ Unpublished and unsubmitted work written in a manuscript style
Publication Details	Gabriele Abbati*, Philippe Wenk*, Michael A Osborne, Andreas Krause, Bernhard Schölkopf and Stefan Bauer. AReS and MaRS - Adversarial and MMD-Minimizing Regression for SDEs. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*. Long Beach, California (2019). (* denotes equal contribution)

Student Confirmation

Student Name:	Gabriele Abbati		
Contribution to the Paper	• Conceived the original idea • Co-developed the algorithmic procedure • Coded the software package and released it on github • Performed all experiments • Wrote the paper together with co-first author		
Signature	Gabriele Abbati	Date	23/01/2020

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title:	Mike Osborne, Professor		
Supervisor comments	I endorse this description.		
Signature	MOsborne	Date	24 JAN 2020

5

AdaGeo: Adaptive Geometric Learning for Optimization and Sampling

AdaGeo: Adaptive Geometric Learning for Optimization and Sampling

Gabriele Abbati
Dept. of Engineering Science
University of Oxford
gabb@robots.ox.ac.uk

Alessandra Tosi
Mind Foundry Ltd.
Oxford

Michael A Osborne
Dept. of Engineering Science
University of Oxford

Seth Flaxman
Dept. of Mathematics &
Data Science Institute
Imperial College London

Abstract

Gradient-based optimization and Markov Chain Monte Carlo sampling can be found at the heart of several machine learning methods. In high-dimensional settings, well-known issues such as slow-mixing, non-convexity and correlations can hinder the algorithms' efficiency. In order to overcome these difficulties, we propose AdaGeo, a preconditioning framework for **ada**ptively learning the **geo**metry of the parameter space during optimization or sampling. In particular, we use the Gaussian process latent variable model (GP-LVM) to represent a lower-dimensional embedding of the parameters, identifying the underlying Riemannian manifold on which the optimization or sampling is taking place. Samples or optimization steps are consequently proposed based on the geometry of the manifold. We apply our framework to stochastic gradient descent, stochastic gradient Langevin dynamics, and stochastic gradient Riemannian Langevin dynamics, and show performance improvements for both optimization and sampling.

1 INTRODUCTION

The performances of a large number of machine learning methods rely heavily on the deployment of two main algorithms: gradient-based optimization (e.g. deep neural networks [Goodfellow, Bengio, and Courville, 2016], kernel methods [Schölkopf and Smola, 2002], variational methods [Blei, Kucukelbir, and McAuliffe, 2017]) and Markov Chain Monte Carlo sampling-based learning [e.g. Robert, 2004; Gelman et al., 2013].

The advantages and results that these methods produce are well-known and established, but when the geometry of the parameter space becomes hard to explore (e.g. because of non-convexity, strong correlations between parameters, or multimodality), then a number of practical issues arise and convergence is not reachable within a reasonable amount of time. In this sense, preconditioning can help: by preconditioning, we mean the application of a transformation (the so-called preconditioner) to the parameter space, aimed at helping numerical solvers in their tasks. High-dimensional preconditioning can be crucial in this sense: by adaptively correcting the directions in which the algorithm explores the space, it is possible to obtain significant improvements in efficiency when trying to sample the relevant domain areas of e.g. a posterior distribution over the parameters. This yields a better characterization. In this context, preconditioning refers to the practice of scaling each coordinate differently.

In this paper, we propose a new preconditioning algorithm that exploits the advantages of probabilistic dimensionality reduction through Gaussian process latent variable models (GP-LVM) [Lawrence, 2005]. The proposed concept operates as follows: after t iterations of sampling or optimization, a set of samples of parameter vectors $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_t\}$ is obtained. By defining a latent variable model, $\boldsymbol{\Theta}$ can be described through a lower-dimensional (latent) set $\boldsymbol{\Omega}$. Assuming the existence of a mapping $\mathbf{f}$ between the two sets,

$$\boldsymbol{\theta} = \mathbf{f}(\boldsymbol{\omega}) + \boldsymbol{\eta}, \tag{1}$$

where $\boldsymbol{\omega} \in \boldsymbol{\Omega}$ and $\boldsymbol{\eta}$ denotes a noise term. In particular, GP-LVMs assume a Gaussian process form for $\mathbf{f}$: this generative model can capture the non-linearities hidden in the parameters and can describe them effectively. By making some assumptions of smoothness over the GP mapping $\mathbf{f}$, we can access the full distribution of the derivative process (also a Gaussian Process), and we are not limited to a point estimation of the natural gradient.

Under the assumption of a smooth mapping, the intrinsic structure of the topological space that contains the parameters is learned as a lower-dimensional Riemannian manifold, equipped with a locally varying metric tensor which

Proceedings of the 21^{st} International Conference on Artificial Intelligence and Statistics (AISTATS) 2018, Lanzarote, Spain. PMLR: Volume 84. Copyright 2018 by the author(s).

encapsulates the geometrical properties of the space. In this work, we suggest performing sampling and optimization by exploiting the geometrical insights of the parameter space learned through GP-LVM. Finally, we show analytically how to take advantage of the distribution of the Riemannian metric tensor which such models yield [Tosi et al., 2014]. Code for our method is available online at `https://github.com/gabb7/AdaGeo`.

To the best of our knowledge, no generic approach based on dimensionality reduction techniques was previously applied to improve optimization and sampling methods. The contributions of this paper are listed as follows:

- we develop a generic framework for combining dimensionality reduction techniques with sampling and optimization methods;
- we contribute to gradient-based optimization methods by coupling them with appropriate dimensionality reduction techniques. In particular, we improve the performances of gradient descent and stochastic gradient descent, when training respectively a Gaussian Process and a neural network;
- we contribute to gradient-based Markov Chain Monte Carlo by developing an AdaGeo version of stochastic gradient Langevin dynamics; the information gathered through the latent space is employed to compute the steps of the Markov chain; and
- we extend the approach to stochastic gradient Riemannian Langevin dynamics, thanks to the geometric tensor naturally recovered by the GP-LVM model.

The paper is structured as follows: in section 2 we summarize the previous work that go into the same direction; section 3 contains a short review of GP-LVMs; in section 4 the novel methods and algorithms this work proposes are introduced and explained; in section 5 the results of some experiments are presented; finally in 6 we sum up the contributions of this paper and draw some conclusions.

2 RELATED WORK

Markov Chain Monte Carlo [Robert, 2004] methods are often the preferred choice for sampling-based Bayesian posterior inference. Recent developments include, among others, Hamiltonian Monte Carlo [Neal, 2011] and Particle MCMC [Andrieu, Doucet, and Holenstein, 2010]: both are at the core of probabilistic programming languages (respectively Stan [Carpenter et al., 2016] and Anglican [Tolpin et al., 2016]) and are deployed to perform Bayesian probabilistic inference. Recently, stochastic optimization algorithms [Robbins and Monro, 1951] were combined with Langevin dynamics [Neal, 2011] to develop a class of methods based on stochastic gradient descent capable of sampling from posterior distributions. The resulting algorithm is called stochastic gradient Langevin dynamics [Welling and Teh, 2011].

In an attempt to overcome the sampling issues introduced in the previous section, successful approaches include the results of Riemannian geometry [Do Carmo and Flaherty Francis, 1992], as in Riemannian Hamiltonian Monte Carlo [Girolami and Calderhead, 2011] and stochastic gradient Riemannian Langevin dynamics [Patterson and Teh, 2013].

On the optimization side, we focus our attention on gradient-based optimization algorithms, such as stochastic gradient descent (SGD). SGD processes only a small batch of data, chosen randomly within the total dataset, to approximate the gradients and propose computationally cheap updates. Recent developments mainly address how learning rates can be dynamically modified through time (e.g. AdaGrad [Duchi, Hazan, and Singer, 2011], Adadelta [Zeiler, 2012], Adam [Kingma and Ba, 2014] and RMSProp).

3 A BRIEF REVIEW OF GP-LVM

The aim of this work is to identify meaningful lower-dimensional representations of the parameter space in order to boost the performances of sampling and optimization algorithms. First of all, we introduce the idea of probabilistic dimensionality reduction with latent variable models. Later, we incorporate Gaussian processes into the framework and illustrate the formulation of Gaussian process latent variable models.

3.1 Probabilistic Dimensionality Reduction with Latent Variable Models

We would like to relate a set of observed variables $\mathbf{\Theta} \subset \mathbb{R}^D$ to a set of lower-dimensional latent (unobserved) variables $\mathbf{\Omega} \subset \mathbb{R}^Q$ (with $Q < D$) and define a joint distribution over them. By finding such representation we perform dimensionality reduction on $\mathbf{\Theta}$: such a model is known as a latent variable model.

First, a prior distribution $p(\mathbf{\Omega})$ is introduced over the latent space, which induces a distribution over $\mathbf{\Theta}$. This relation is described by the probabilistic mapping

$$\theta_{i,j} = f_j(\boldsymbol{\omega}_i) + \eta_j, \tag{2}$$

where $\boldsymbol{\omega}_i$ is the latent point associated with the ith observation $\boldsymbol{\theta}_i$, j is the index of the features of $\mathbf{\Theta}$ and $\boldsymbol{\eta}$ is a noise term (typically $\boldsymbol{\eta} \sim \mathcal{N}(\mathbf{0}, \beta^{-1}\mathbf{I})$).

The classic approach would suppose the latent variables to be marginalized out and the parameters to be optimized by maximizing the model likelihood. An alternative method proposes instead to marginalize the parameters and optimize the latent variables: this procedure (together with

other considerations and assumptions) would lead to the formulation of Gaussian process latent variable model (GP-LVM) [Lawrence, 2005].

3.2 Gaussian process latent variable model

Gaussian processes (GP) describe distributions over functions. They are defined as a collection of random variables, any finite number of which have a joint Gaussian distribution [Rasmussen and Williams, 2006]. A Gaussian process is fully specified by a mean function $m(\cdot)$ and a covariance function $k(\cdot,\cdot)$. If a real-valued stochastic process $f(\cdot)$ is a Gaussian process, it will be denoted as

$$f(\cdot) \sim \mathcal{GP}(m(\cdot), k(\cdot,\cdot)), \tag{3}$$

where $m(\cdot)$ and $k(\cdot,\cdot)$ are respectively defined as

$$m(\boldsymbol{\omega}) = \mathbb{E}\big[f(\boldsymbol{\omega})\big], \tag{4}$$

$$k(\boldsymbol{\omega},\boldsymbol{\omega}') = \mathbb{E}\big[\big(f(\boldsymbol{\omega}) - m(\boldsymbol{\omega})\big)\big(f(\boldsymbol{\omega}') - m(\boldsymbol{\omega}')\big)\big]. \tag{5}$$

From a GP, it is possible to instantiate a Gaussian distributed vector which has covariance matrix $\mathbf{K}$, where $K_{ij} = k(\boldsymbol{\omega}_i, \boldsymbol{\omega}_j)$. For simplicity of notation, the mean function is usually taken to be equal to zero, without loss of generality.

In the context of probabilistic dimensionality reduction, Gaussian process latent variable models use GPs to define a prior over the mapping $\mathbf{f}$ (appearing in eq. 2). As suggested above, the likelihood of the data $\boldsymbol{\Theta}$ given the latent $\boldsymbol{\Omega}$ is computed by marginalizing the mapping and optimizing the latent variables. The resulting likelihood is given by

$$\begin{aligned} p(\boldsymbol{\Theta} \mid \boldsymbol{\Omega}, \beta) &= \prod_{j=1}^{D} \mathcal{N}\big(\boldsymbol{\theta}_{:,j} \mid \mathbf{0}, \mathbf{K} + \beta^{-1}\mathbf{I}\big) \\ &= \prod_{j=1}^{D} \mathcal{N}\big(\boldsymbol{\theta}_{:,j} \mid \mathbf{0}, \tilde{\mathbf{K}}\big). \end{aligned} \tag{6}$$

Thus eq. 2 becomes

$$\theta_{i,j} = \tilde{\mathbf{K}}_{(\boldsymbol{\omega}_i, \boldsymbol{\Omega})} \tilde{\mathbf{K}}^{-1} \boldsymbol{\Theta}_{:,j} + \eta_j. \tag{7}$$

For differentiable kernels, it is possible to compute the Jacobian of the mapping $\mathbf{f}$ analytically. Let $\mathbf{J}$ be such Jacobian, defined as

$$J_{ij} = \frac{\partial f_i}{\partial \omega_j}. \tag{8}$$

In the context of GP-LVMs, the Jacobian follows a Gaussian distribution. We can see this if the rows of $\mathbf{J}$ are assumed to be independent:

$$p(\mathbf{J} \mid \boldsymbol{\Omega}, \beta) = \prod_{i=1}^{D} \mathcal{N}\big(\mathbf{J}_{i,:} \mid \boldsymbol{\mu}_{\mathbf{J}_{i,:}}, \boldsymbol{\Sigma}_{\mathbf{J}}\big), \tag{9}$$

and in the case of GP-LVM, for every latent point $\boldsymbol{\omega}_*$,

$$\begin{aligned} p(\mathbf{J} \mid \boldsymbol{\Theta}, \boldsymbol{\Omega}, \beta) &= \prod_{i=1}^{D} \mathcal{N}\left(\mathbf{J}_{i,:} \mid \boldsymbol{\mu}_{\mathbf{J}_{i,:}}, \boldsymbol{\Sigma}_{\mathbf{J}}\right) \\ &= \prod_{i=1}^{D} \mathcal{N}\Big(\mathbf{J}_{i,:} \,\Big|\, \partial\tilde{\mathbf{K}}_{\boldsymbol{\omega},*}^{\top} \tilde{\mathbf{K}}_{\boldsymbol{\omega},\boldsymbol{\omega}}^{-1} \boldsymbol{\Theta}_{:,j}, \\ &\qquad \partial^2\tilde{\mathbf{K}}_{*,*} - \partial\tilde{\mathbf{K}}_{\boldsymbol{\omega},*}^{\top} \tilde{\mathbf{K}}_{\boldsymbol{\omega},\boldsymbol{\omega}}^{-1} \tilde{\mathbf{K}}_{\boldsymbol{\omega},*}\Big) \end{aligned} \tag{10}$$

where:

$$\left(\partial\tilde{\mathbf{K}}_{\boldsymbol{\omega},*}\right)_{nl} = \frac{\partial k(\boldsymbol{\omega}_n, \boldsymbol{\omega}_*)}{\partial \boldsymbol{\omega}_*^{(l)}} \qquad \begin{matrix} n = 1,\dots,N \\ l = 1,\dots,Q \end{matrix} \tag{11}$$

$$\left(\partial^2\tilde{\mathbf{K}}_{*,*}\right)_{il} = \frac{\partial^2 k(\boldsymbol{\omega}_*, \boldsymbol{\omega}_*)}{\partial \boldsymbol{\omega}_*^{(i)} \partial \boldsymbol{\omega}_*^{(l)}} \qquad \begin{matrix} i = 1,\dots,Q \\ l = 1,\dots,Q. \end{matrix} \tag{12}$$

4 OUR APPROACH

In the next sections we show how our GP-LVM-based technique can be applied to optimization and sampling algorithms. First, in section 4.1 we take a look at gradient-based optimization. Second, we consider the framework of Bayesian sampling (section 4.2). Third, the theoretical framework for Riemannian extensions is described in section 4.3.

4.1 AdaGeo Gradient-based Optimization

Optimization represents the most straightforward application of our approach. Consider the problem of finding the minimum of an objective function $g(\boldsymbol{\theta})$:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} g(\boldsymbol{\theta}). \tag{13}$$

This could be tackled with an iterative gradient-based method of the form:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \Delta\boldsymbol{\theta}_t \tag{14}$$

where $\Delta\boldsymbol{\theta}_t = \Delta\boldsymbol{\theta}_t(\nabla_{\boldsymbol{\theta}} g)$. We propose, after t steps of optimization, to: firstly, train the GP-LVM on the samples $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_t\}$; secondly, build the supporting latent space $\boldsymbol{\Omega}$ and; finally, move the computation to this latent space to exploit its meaniningful topological structures,

$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t - \Delta\boldsymbol{\omega}_t. \tag{15}$$

The gradients $\nabla_{\boldsymbol{\theta}} g(\boldsymbol{\theta})$, computed in the observed space, can be mapped into the latent space through the expected Jacobian $\mathbf{J}_{\mathbf{f}}$ of the transformation $\mathbf{f}$:

$$\nabla_{\boldsymbol{\omega}} g\big(\mathbf{f}(\boldsymbol{\omega})\big) = \mathbb{E}\big[\mathbf{J}_{\mathbf{f}}\big] \nabla_{\boldsymbol{\theta}} g(\boldsymbol{\theta}). \tag{16}$$

The mapping in eq. 2 can yield the actual update in the observed parameter space. It is straightforward to

plug this framework into gradient-based optimization algorithms and formulate an AdaGeo version of the previously cited stochastic gradient descent and relative developments (e.g. AdaGrad [Duchi, Hazan, and Singer, 2011], Adadelta [Zeiler, 2012] and Adam [Kingma and Ba, 2014]).

Empirically, we find that the best strategy is to alternate phases of classic optimization updates to phases of latent updates. This is to overcome the exploration issues GP-LVMs suffer from: these models tend to map those portions of the space that data did not cover onto a single point [Duvenaud et al., 2014]. Alternating the updates means being able to explore the space through SGD (or one of its variants), codify it with GP-LVM and exploit the newly acquired topological information until exhaustion; the process is then repeated. This scheme becomes necessary as the GP-LVM optimizer starts to saturate and gets stuck somewhere along the way if the optimum is far from the starting point. The algorithm can be written as shown in algorithm 1. Here $T_{\boldsymbol{\theta}}$ and $T_{\boldsymbol{\omega}}$ refer to the number of iterations performed respectively in the parameter and latent spaces.

Algorithm 1 AdaGeo gradient-based optimization

1: **while** convergence is not reached **do**
2: Perform $T_{\boldsymbol{\theta}}$ iterations with classic updates:

$$\Delta\boldsymbol{\theta}_t = \Delta\boldsymbol{\theta}_t(\nabla_{\boldsymbol{\theta}} g(\boldsymbol{\theta}))$$

3: Train the GP-LVM model on the samples got by optimization
4: Continue performing $T_{\boldsymbol{\omega}}$ using the AdaGeo optimizer:

$$\Delta\boldsymbol{\omega}_t = \Delta\boldsymbol{\omega}_t(\nabla_{\boldsymbol{\omega}} g(\mathbf{f}(\boldsymbol{\omega})))$$
$$\boldsymbol{\omega}_{t+1} = \boldsymbol{\omega}_t - \Delta\boldsymbol{\omega}_t$$

and moving back to the parameter space with

$$\boldsymbol{\theta}_{t+1} = \mathbf{f}(\boldsymbol{\omega}_{t+1}).$$

5: **end while**.

4.2 AdaGeo Bayesian Sampling

Consider a dataset $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, modeled with a generative model whose likelihood is $p(\mathbf{X}, \boldsymbol{\theta}) = \prod_{i=1}^{N} p(\mathbf{x}_i, \boldsymbol{\theta})$ and parameterized by $\boldsymbol{\theta} \in \mathbb{R}^D$, on which we impose some prior $p(\boldsymbol{\theta})$. To perform statistical inference, we would like to be able to sample from the posterior

$$p(\boldsymbol{\theta} \mid \mathbf{X}) = \frac{p(\mathbf{X} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(\mathbf{X})}. \tag{17}$$

However, the denominator (known as model evidence, or marginal likelihood) is often intractable and hence doesn't allow the posterior distribution to be expressed in a closed-form. An approximation is consequently required: Monte Carlo-based sampling is one of the most popular approaches used to estimate the posterior by drawing a set of iid samples $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N$ from $p(\boldsymbol{\theta} \mid \mathbf{X})$.

We specifically focus on Metropolis-adjusted Langevin algorithms (MALA). Part of the family of Markov Chain Monte Carlo, these methods rely on two main mechanisms:

- Langevin Dynamics [Neal, 2011], where opportunely scaled Gaussian noise is used alongside the gradients of the target distribution;
- Metropolis-Hastings acceptance/rejection steps.

While Langevin dynamics push the system towards regions of high probability, the acceptance/rejection steps help mixing and convergence.

4.2.1 Stochastic Gradient Langevin Dynamics

Stochastic gradient Langevin dynamics (SGLD) [Welling and Teh, 2011] belongs to the class of samplers previously described and extends stochastic optimization [Robbins and Monro, 1951] results. Typically, the iterative algorithms belonging to this category behave as follows. At each step t, a mini-batch $\mathbf{X}_t = \{\mathbf{x}_{t1}, \ldots, \mathbf{x}_{tn}\}$ is stochastically extracted from the dataset and employed to estimate the gradients in the update rule

$$\Delta\boldsymbol{\theta}_t = \frac{\epsilon_t}{2}\left(\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}_t) + \frac{N}{n}\sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}_{ti} \mid \boldsymbol{\theta}_t)\right), \tag{18}$$

where ϵ_t is a time-evolving learning rate. The mini-batch strategy allows a faster (albeit less precise) approximation of the gradient that doesn't need a pass over the whole dataset: this gives improved scalability.

Unfortunately MAP estimations would not capture the effects of uncertainty and lead to overfitting (and possibly to other troubles with e.g. multimodality). A classic Bayesian way to deal with uncertainty consists of using Markov Chain Monte Carlo [Metropolis et al., 1953] (MCMC). As mentioned above, by combining stochastic optimization updates with Langevin dynamics we get the SGLD algorithm:

$$\Delta\boldsymbol{\theta}_t = \frac{\epsilon_t}{2}\left(\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}_t) + \frac{N}{n}\sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}_i \mid \boldsymbol{\theta}_t)\right) + \boldsymbol{\eta}_t$$
$$\boldsymbol{\eta}_t \sim \mathcal{N}(\mathbf{0}, \epsilon_t \mathbf{I}), \tag{19}$$

where the step sizes decreases towards 0, with the learning rate satisfying

$$\sum_{t=1}^{\infty} \epsilon_t = \infty, \qquad \sum_{t=1}^{\infty} \epsilon_t^2 < \infty. \tag{20}$$

As $t \to \infty$ the samples will approach the posterior distribution. In particular, for large enough t the system will transition into Langevin dynamics and converge to the true posterior distribution. When Langevin dynamics occur, the algorithm does not need acceptance/rejection steps [Welling and Teh, 2011].

4.2.2 AdaGeo Stochastic Gradient Langevin Dynamics

Next, we show how our framework can be integrated into SGLD. After t steps of sampling, we obtain a set of parameter vectors $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_t\}$. With GP-LVM, it is possible to construct the Q-dimensional latent space $\boldsymbol{\Omega}$ (where $Q < D$), which would condense the relevant features of $\boldsymbol{\Theta}$ in a simpler configuration. The update can be now performed in the latent space; subsequently the GP-LVM mapping $\mathbf{f}(\boldsymbol{\omega})$ in eq. 2 would be used to move back onto the original space and obtain the new observation.

In the case of the SGLD sampler, the latent update can be written as:

$$\Delta\boldsymbol{\omega}_t = \frac{\epsilon_t}{2}\left(\nabla_{\boldsymbol{\omega}} \log p\big(\mathbf{f}(\boldsymbol{\omega}_t)\big) + \frac{N}{n}\sum_{i=1}^{n} \nabla_{\boldsymbol{\omega}} \log p\big(\mathbf{x}_{ti}|\mathbf{f}(\boldsymbol{\omega}_t)\big)\right) + \boldsymbol{\eta}_t, \tag{21}$$

$\boldsymbol{\eta}_t \sim \mathcal{N}(\mathbf{0}, \epsilon_t \mathbf{I})$.

As before, we can compute the "latent" gradients of the likelihood with the Jacobian of the mapping $\mathbf{f}$ (eq. 16). The new algorithm we built, AdaGeo Stochastic Gradient Langevin Dynamics, as we call it, is described as algorithm 2.

Algorithm 2 AdaGeo Stochastic Gradient Langevin dynamics

1: Sample t values in the original space using classic SGLD and construct the set $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_t\}$
2: Train the GP-LVM model on $\boldsymbol{\Theta}$, obtaining in this way a mapping $\mathbf{f} : \boldsymbol{\Omega} \to \boldsymbol{\Theta}$
3: Choose $\boldsymbol{\omega}_0$ as the latent point corresponding to the last item in the dataset $\boldsymbol{\theta}_t$ and use it as next starting point
4: **for** $t_{\boldsymbol{\omega}} = 1 : N$ **do**
5: Compute the update in the latent space as described in eq. 21, which yields $\boldsymbol{\omega}_{t_{\boldsymbol{\omega}}+1}$
6: Compute $\boldsymbol{\theta}_{t_{\boldsymbol{\omega}}+1}$ through: $\boldsymbol{\theta}_{t_{\boldsymbol{\omega}}+1} = \mathbf{f}(\boldsymbol{\omega}_{t_{\boldsymbol{\omega}}+1})$
7: **end for**

4.3 Riemannian Extensions

In the following sections the theoretical framework necessary to plug GP-LVM into the Riemannian development of the stochastic gradient Langevin dynamics is shown and explained.

4.3.1 GP-LVM Local Metric

A trained GP-LVM can grant access to relevant information about the intrinsic geometry of the latent space, i.e. the metric tensor.

If the covariance function of a Gaussian process is differentiable (as it is the case, for example, for the widely-used squared exponential), it is straightforward to show that the mapping $\mathbf{f}$ in eq. 2 is also differentiable. Under this assumption, the pull-back metric acting from the latent space $\boldsymbol{\Omega}$ to the observed space $\boldsymbol{\Theta}$ can be computed [Do Carmo and Flaherty Francis, 1992]. Thus, we are able to define the Riemannian metric tensor over the manifold. The Riemannian metric $\mathbf{G}$ is defined as follows.

Definition (Riemannian metric) *A Riemannian metric $\mathbf{G}$ on a manifold $\mathcal{M}$ is a symmetric and positive definite matrix which defines a smoothly varying inner product:*

$$\langle \mathbf{a}, \mathbf{b} \rangle_x = \mathbf{a}^\top \mathbf{J}^\top \mathbf{J} \mathbf{b} = \mathbf{a}^\top \mathbf{G}(x) \mathbf{b} \tag{22}$$

in the tangent space $T_x\mathcal{M}$, for each point $x \in \mathcal{M}$, $\mathbf{a}, \mathbf{b} \in T_x\mathcal{M}$, where $\langle \cdot, \cdot \rangle_x$ denotes the inner product. The matrix $\mathbf{G}$ is called the metric tensor.

Intuitively, the metric tensor $\mathbf{G}$ gives information about the geometry (e.g. related to distances and geodetic lines) of the latent space. Let $\mathbf{J}$ be the Jacobian of $\mathbf{f}$. Then the tensor

$$\mathbf{G} = \mathbf{J}^\top \mathbf{J} \tag{23}$$

defines a local inner product over the latent space. Eq. 10, together with eq. 23, yields a distribution over the metric tensor $\mathbf{G}$ [Tosi et al., 2014]

$$\mathbf{G} \sim \mathcal{W}_Q\Big(D, \boldsymbol{\Sigma}_{\mathbf{J}}, \mathbb{E}\left[\mathbf{J}^\top\right] \mathbb{E}\big[\mathbf{J}\big]\Big), \tag{24}$$

where $\mathcal{W}$ denotes the non-central Wishart distribution [Anderson, 1946], which is a generalization to multiple dimensions of the chi-squared distribution.

The expected metric tensor can be computed as

$$\mathbb{E}\left[\mathbf{J}^\top \mathbf{J}\right] = \mathbb{E}\left[\mathbf{J}^\top\right] \mathbb{E}\big[\mathbf{J}\big] + D\boldsymbol{\Sigma}_{\mathbf{J}}. \tag{25}$$

Note that the second term in the last equation, depending on the covariance of the Jacobian, makes the metric tensor expand as the uncertainty over the mapping increases.

4.3.2 Stochastic Gradient Riemannian Langevin Dynamics

As mentioned above, issues in MCMC schemes may arise if the components of $\boldsymbol{\theta}$ have different scales or large correlations, as in these cases the isotropic proposal distribution of the stochastic gradient Langevin dynamics will lead to slow mixing. We propose to gather information about the geometry of the space though GP-LVM, but an alternative

approach employing preconditioning tools consists in applying the Riemannian metric tensor $\mathbf{G}$ to the updates. In this way the issues encountered in the parameter space (e.g. different scales or large correlations) could be overcome and sampling improved.

The stochastic gradient Riemannian Langevin dynamics (SGRLD) sampler [Patterson and Teh, 2013] puts together the advantages of exploiting a known Riemannian geometry with the scalability of the stochastic optimization approaches. The update rule can be written as:

$$\begin{aligned} \Delta\boldsymbol{\theta}_t &= \frac{\epsilon_t}{2}\boldsymbol{\mu}(\boldsymbol{\theta}_t) + \mathbf{G}^{-\frac{1}{2}}(\boldsymbol{\theta}_t)\boldsymbol{\eta}_t \\ \boldsymbol{\eta}_t &\sim \mathcal{N}(\mathbf{0}, \epsilon_t \mathbf{I}), \end{aligned} \tag{26}$$

where the jth component of $\boldsymbol{\mu}(\boldsymbol{\theta})$ is given by

$$\begin{aligned} &\boldsymbol{\mu}(\boldsymbol{\theta})_j = \\ &\left(\mathbf{G}^{-1}(\boldsymbol{\theta})\Big(\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}) + \frac{N}{n}\sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}_{ti} \mid \boldsymbol{\theta})\Big)\right)_j \\ &\quad - 2\sum_{k=1}^{D}\left(\mathbf{G}^{-1}(\boldsymbol{\theta})\frac{\partial \mathbf{G}(\boldsymbol{\theta})}{\partial \theta_k}\mathbf{G}^{-1}(\boldsymbol{\theta})\right)_{jk} \\ &\quad + \sum_{k=1}^{D}\big(\mathbf{G}^{-1}(\boldsymbol{\theta})\big)_{jk}\mathrm{Tr}\left(\mathbf{G}^{-1}(\boldsymbol{\theta})\frac{\partial \mathbf{G}(\boldsymbol{\theta})}{\partial \theta_k}\right). \end{aligned} \tag{27}$$

With respect to the non-Riemannian version of the algorithm, it is possible to make some changes: the first term in $\boldsymbol{\mu}(\boldsymbol{\theta})$ is the gradient of the log-likelihood, already seen before, preconditioned with the matrix $\mathbf{G}(\boldsymbol{\theta})$ and hence taking into account the geometry of the space; the second and third term describe the derivatives of the curvature of the manifold. Notice the noise term in (26) is appropriately preconditioned as well. Typically the Fisher information matrix is used as $\mathbf{G}$, but of course an analytical form is necessary (and rarely obtainable in the majority of problems).

4.3.3 AdaGeo Stochastic Gradient Riemannian Langevin Dynamics

Beyond the classic formulation of SGRLD, we now describe our proposal for incorporating dimensionality reduction into its framework. Analogous to the SGLD case, the update in the observed space in eq. 26 can be converted into the one acting in the latent space:

$$\begin{aligned} \Delta\boldsymbol{\omega}_t &= \frac{\epsilon_t}{2}\boldsymbol{\mu}(\boldsymbol{\omega}_t) + \mathbf{G}_{\boldsymbol{\omega}}^{-\frac{1}{2}}(\boldsymbol{\omega}_t)\boldsymbol{\eta}_t \\ \boldsymbol{\eta}_t &\sim \mathcal{N}(\mathbf{0}, \epsilon_t \mathbf{I}), \end{aligned} \tag{28}$$

where the jth component of $\boldsymbol{\mu}(\boldsymbol{\omega})$ is given by

$$\begin{aligned} &\boldsymbol{\mu}(\boldsymbol{\omega})_j = \\ &\left(\mathbf{G}_{\boldsymbol{\omega}}^{-1}(\boldsymbol{\omega})\Big(\nabla_{\boldsymbol{\omega}} \log p\big(\mathbf{f}(\boldsymbol{\omega})\big) + \frac{N}{n}\sum_{i=1}^{n} \nabla_{\boldsymbol{\omega}} \log p\big(\mathbf{x}_{ti} \mid \mathbf{f}(\boldsymbol{\omega})\big)\Big)\right)_j \\ &\quad - 2\sum_{k=1}^{Q}\left(\mathbf{G}_{\boldsymbol{\omega}}^{-1}(\boldsymbol{\omega})\frac{\partial \mathbf{G}_{\boldsymbol{\omega}}(\boldsymbol{\omega})}{\partial \omega_k}\mathbf{G}_{\boldsymbol{\omega}}^{-1}(\boldsymbol{\omega})\right)_{jk} \\ &\quad + \sum_{k=1}^{Q}\big(\mathbf{G}_{\boldsymbol{\omega}}^{-1}(\boldsymbol{\omega})\big)_{jk}\mathrm{Tr}\left(\mathbf{G}_{\boldsymbol{\omega}}^{-1}(\boldsymbol{\omega})\frac{\partial \mathbf{G}_{\boldsymbol{\omega}}(\boldsymbol{\omega})}{\partial \omega_k}\right). \end{aligned} \tag{29}$$

The matrix $G_{\boldsymbol{\omega}}$ is computed by the GP-LVM according to eq. 24. This is an important theoretical contribution of our paper: if the update is performed in the latent space, the Riemannian metric tensor can be accessed and exploited to hopefully increase the quality of the updates and the exploration of the (original) observed space. The proposed algorithm (AdaGeo Stochastic Gradient Riemannian Langevin Dynamics) is described in algorithm 2, with AdaGeo-SGRLD updates instead of AdaGeo-SGLD ones.

5 RESULTS

This section contains the experimental results: in section 5.1 AdaGeo-SGLD is used to sample from a "banana" distribution; section 5.2 shows how a AdaGeo-powered version of SGD can be used to speed up training of a small neural network; in section 5.3 it is possible to find an application of AdaGeo to gradient descent to train a Gaussian Process.

5.1 Sampling High-dimensional Banana Distributions

We employed the AdaGeo version of the SGLD sampler to sample from a probability density, the so-called "banana" distribution. In D dimensions this particular PDF $p(\boldsymbol{\theta})$ assumes the form

$$p(\boldsymbol{\theta}) \propto \exp\left(-\frac{\theta_1^2}{200} - \frac{(\theta_2 - b\theta_1^2 + 100b)^2}{2} - \sum_{j=3}^{D}\theta_j^2\right), \tag{30}$$

which is a particularly interesting density to sample from: the only observable interaction occurs between the first two components of $\boldsymbol{\theta}$ (visible in figure 1), masked by the remaining components which act as Gaussian noise. First, a vanilla Metropolis-Hastings sampler is run for 35,000 steps, with 10,000 samples discarded as burn-in and a thinning factor of 250, yielding 100 samples. Due to the high dimensionality of the problem, samplers relying on Langevin dynamics would require prohibitive amounts of time to reach convergence, hence the choice of the simple Metropolis-Hastings. Afterwards, a GP-LVM model is trained on the 100 samples, using squared exponential kernel and latent dimension $Q = 5$. In this experiment

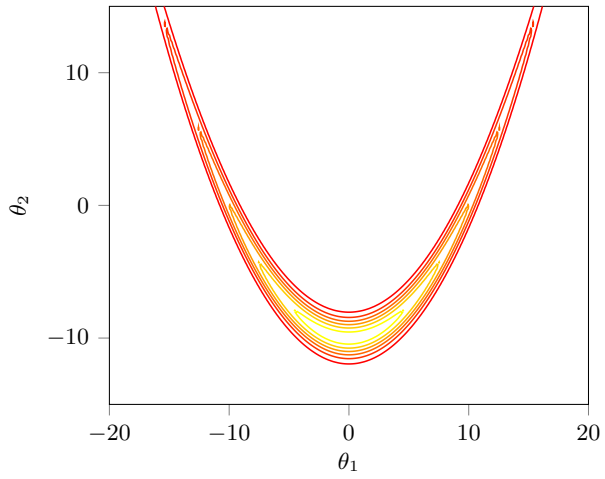


Figure 1: Contour plot showing the level curves of the density generated by a 50-dimensional banana distribution. The figure displays the interaction between the first and second components of $\boldsymbol{\theta}$.

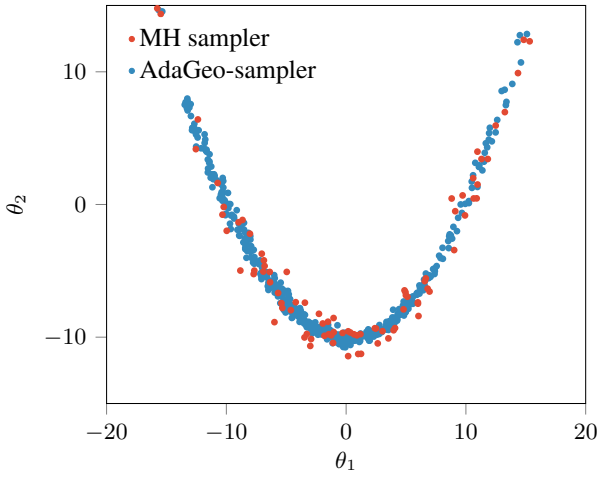


Figure 2: First, our AdaGeo-SGLD sampler trains on the samples returned by a first run of a Metropolis-Hastings algorithm (samples in red). Thanks to those, a lower-dimensional latent space is identified and then sampled from using a SGLD method. The samples obtained using a 5-dimensional latent space are plotted in blue.

and in the following ones, Q is chosen based on the best empirical results (an alternative to explore would be using an ARD kernel to identify the meaningful latent components). At this point it is possible to sample directly from the 5-dimensional latent space, using then the probabilistic mapping to move back to the real "observed" space. The update in the latent space is computed using the SGLD rule (section 4.2.1), whose functioning is made possible by the reduced dimensionality. 1000 burn in iterations are allowed before starting to sample 250 samples, with a thinning factor of 100 (here the total number of steps is 26,000). Notice that in this case we don't alternate classic and latent updates. Results are shown in figure 2 and 3: it is possible to see that the relevant areas of the distributions are well covered by the samples, and that the main characteristics are represented properly (e.g. the banana-shaped structure in fig. 2 and the noise in fig. 3). Together with a better coverage of the distribution, it is important to note that a relevant speed-up is obtained, as the sampler acts on a 5-dimensional latent space. Autocorrelation plots can be found in the supplementary material (appendix A) and show quantitatively the improvement in samples quality.

5.2 Logistic Regression on MNIST with Neural Networks

The second experiment aims at speeding up the training algorithm for a neural network used for classification on the MNIST dataset, while showing the advantages that our method could bring to optimization. Logistic regression on MNIST is implemented through a simple fully connected neural network, consisting of a single layer of 784 nodes featuring a sigmoid activation function. The total num-

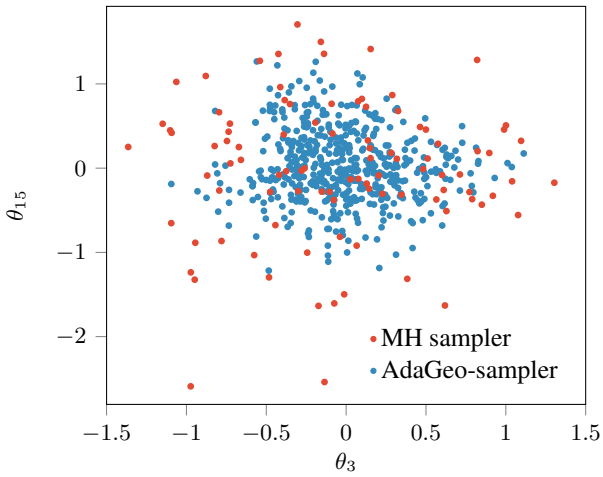


Figure 3: Confounding noise happening between the components of $\boldsymbol{\theta}$. As an example, we show the samples of θ_3 against θ_{15}, but this kind of plot would be similar for every pair of components.

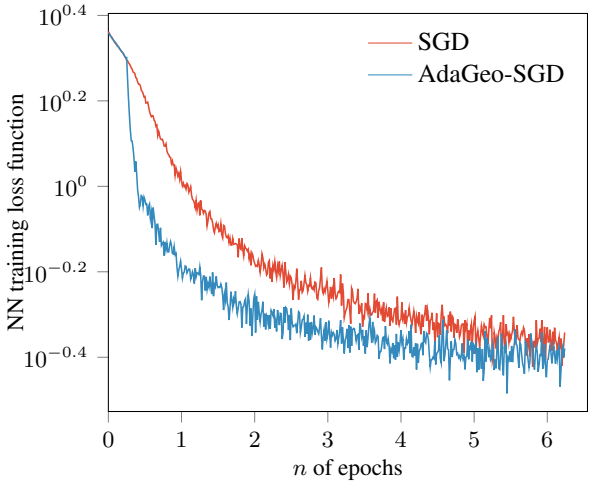


Figure 4: Neural network loss function during training, obtained with vanilla SGD (red) and AdaGeo-SGD (blue), as described in section 5.2.

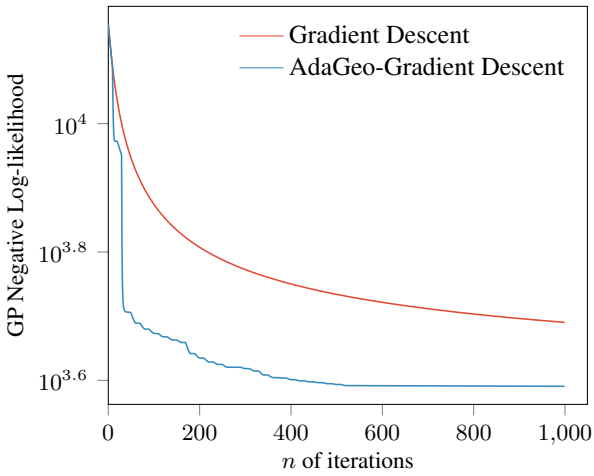


Figure 5: Negative log-likelihood during training of a Gaussian Process, as described in section 5.3. A latent space of dimension 3 is used here to optimize a function whose parameters lie in a 9-dimensional space.

ber of weights that such a network has is 7,850. Out of the 60,000 MNIST images, 50,000 are used for training and 10,000 for testing. During training, cross-entropy loss is minimized. As a fair comparison, we choose to train the network using vanilla stochastic gradient descent and the corresponding AdaGeo version, described as algorithm 1. For the MNIST experiment we have chosen $T_{\boldsymbol{\theta}} = 20$ and $T_{\boldsymbol{\omega}} = 30$, while we have used a surprisingly low-dimensional latent space with $Q = 9$. Results are shown in figure 4: with AdaGeo-SGD updates we are able to outperform vanilla SGD and reach the local optimum almost 2 epochs in advance.

5.3 Gaussian Process Training

The third and last experiment demonstrates further the advantages that AdaGeo could bring to optimization. Here we make use of the Concrete Compressive Strength Data Set [Yeh, 1998]: the model trains on 8 quantitative variables in order to predict a scalar real quantity. The idea is to use a simpler kernel (the GP-LVM one, squared exponential) to fit the hyperparameters of a more complex one. The latter is composed by a linear composition of different kernels [Rasmussen and Williams, 2006]: radial basis function, Matérn kernel with parameter $\nu = 5/2$, another Matérn kernel with parameter $\nu = 3/2$, a linear kernel and some bias, yielding a total of 9 parameters.

We run both gradient descent with Nesterov momentum and AdaGeo-gradient descent for 1000 iterations, with $T_{\boldsymbol{\theta}} = 15$, $T_{\boldsymbol{\omega}} = 15$, and learning rates respectively $\epsilon_{\text{latent}} = 10^{-3}$, $\epsilon_{\text{observed}} = 10^{-5}$. The dimension of the latent space is equal to 3. As we can see from figure 5, AdaGeo successfully improves the performances of gradient descent and significantly speeds up training.

6 CONCLUSION

We propose a novel method that introduces dimensionality reduction techniques, namely Gaussian process latent variable models, into optimization and sampling algorithms, in order to boost their performances and overcome the issues deriving from high dimensionality (non-convexity, correlations, different scales etc.). In particular, we show how to include our approach in existing (gradient-based) optimization and sampling (stochastic gradient Langevin dynamics, stochastic gradient Riemannian Langevin dynamics) algorithms. We prove that in this way, by identifying the underlying topological structure of the variable space, sampling and optimization performances are significantly improved. Furthermore, the methods that this work illustrates help sampling even in the simpler cases that don't require gradients (Metropolis-Hastings, Gibbs, etc.). Indeed, through dimensionality reduction the problems of sampling a high-dimensional space are efficiently tackled.

Due to its modular nature, the approach presented in this paper is suitable for a large number of potential applications (more sophisticated gradient-based optimization algorithms, variational inference, Hamiltonian Monte Carlo, Riemannian stochastic gradient descent). The discussion of the particular formulations of those fall beyond the scope of this paper and will be the subject of future research efforts.

ACKNOWLEDGMENTS

The project was funded by the NCSML PDRA grant 2016/2017. GA was supportd by University of Oxford, Google DeepMind. SRF was supported by the ERC (FP7/617071).

References

Anderson, T. W. (1946). "The Non-Central Wishart Distribution and Certain Problems of Multivariate Statistics". In: *The Annals of Mathematical Statistics* 17.4, pp. 409–431.

Andrieu, Christophe, Arnaud Doucet, and Roman Holenstein (2010). "Particle markov chain monte carlo methods". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 72.3, pp. 269–342.

Blei, David M, Alp Kucukelbir, and Jon D McAuliffe (2017). "Variational inference: A review for statisticians". In: *Journal of the American Statistical Association* just-accepted.

Carpenter, Bob et al. (2016). "Stan: A probabilistic programming language". In: *Journal of Statistical Software* 20, pp. 1–37.

Do Carmo, Manfredo Perdigao and J Flaherty Francis (1992). *Riemannian geometry*. Vol. 115. Birkhäuser Boston.

Duchi, John, Elad Hazan, and Yoram Singer (2011). "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* 12.Jul, pp. 2121–2159.

Duvenaud, David et al. (2014). "Avoiding pathologies in very deep networks". In: *Artificial Intelligence and Statistics*, pp. 202–210.

Gelman, Andrew et al. (2013). *Bayesian data analysis*. CRC press.

Girolami, Mark and Ben Calderhead (2011). "Riemann manifold langevin and hamiltonian monte carlo methods". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 73.2, pp. 123–214.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press.

Kingma, Diederik and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Lawrence, Neil (2005). "Probabilistic non-linear principal component analysis with Gaussian process latent variable models". In: *Journal of machine learning research* 6.Nov, pp. 1783–1816.

Metropolis, Nicholas et al. (1953). "Equation of state calculations by fast computing machines". In: *The journal of chemical physics* 21.6, pp. 1087–1092.

Neal, Radford M et al. (2011). "MCMC using Hamiltonian dynamics". In: *Handbook of Markov Chain Monte Carlo* 2.11.

Patterson, Sam and Yee Whye Teh (2013). "Stochastic gradient Riemannian Langevin dynamics on the probability simplex". In: *Advances in Neural Information Processing Systems*, pp. 3102–3110.

Rasmussen, Carl Edward and Christopher KI Williams (2006). *Gaussian processes for machine learning*. Vol. 1. MIT press Cambridge.

Robbins, Herbert and Sutton Monro (1951). "A stochastic approximation method". In: *The annals of mathematical statistics*, pp. 400–407.

Robert, Christian P (2004). *Monte carlo methods*. Wiley Online Library.

Schölkopf, Bernhard and Alexander J Smola (2002). *Learning with kernels: support vector machines, regularization, optimization and beyond*. the MIT Press.

Tolpin, David et al. (2016). "Design and Implementation of Probabilistic Programming Language Anglican". In: *Proceedings of the 28th Symposium on the Implementation and Application of Functional Programming Languages*. ACM, p. 6.

Tosi, Alessandra et al. (2014). "Metrics for probabilistic geometries". In: *Uncertainty in Artificial Intelligence*, p. 800.

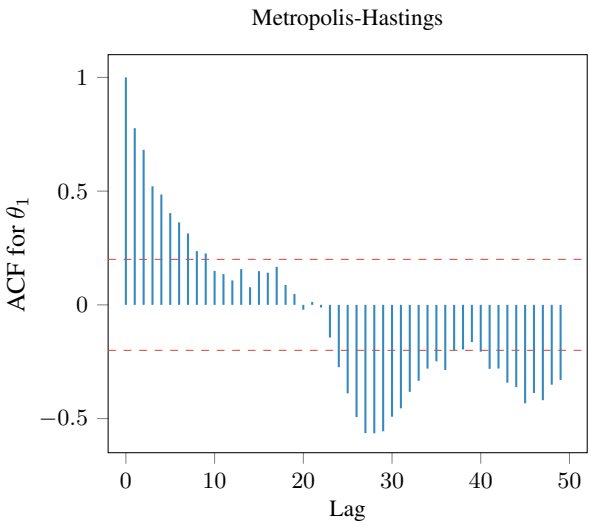
Welling, Max and Yee W Teh (2011). "Bayesian learning via stochastic gradient Langevin dynamics". In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 681–688.

Yeh, I-C (1998). "Modeling of strength of high-performance concrete using artificial neural networks". In: *Cement and Concrete research* 28.12, pp. 1797–1808.

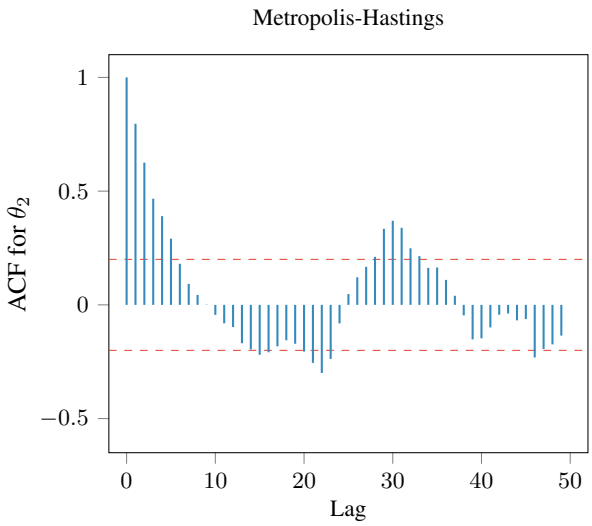
Zeiler, Matthew D (2012). "ADADELTA: an adaptive learning rate method". In: *arXiv preprint arXiv:1212.5701*.

A Autocorrelation Plots

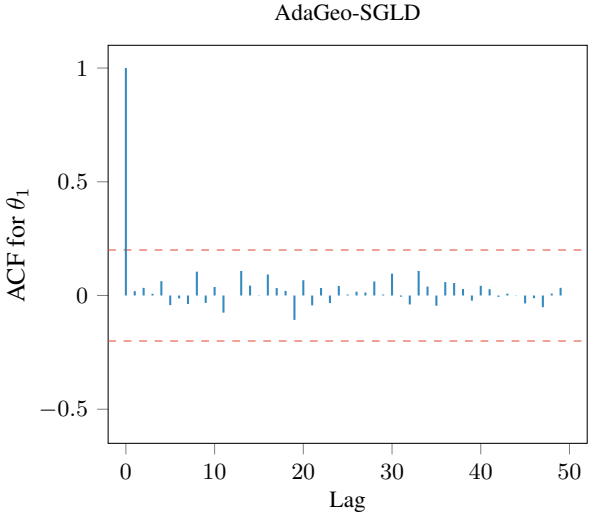
Here we report the autocorrelation plots for the experiment described in section 5.1, respectively for θ_1 and θ_2. Standard Metropolis-Hastings can be found on the left while our method, SGLD with AdaGeo, on the right. While Metropolis-Hastings suffers from high autocorrelation despite a long burn-in and large thinning factor, samples from SGLD-AdaGeo does not show any autocorrelation issues.



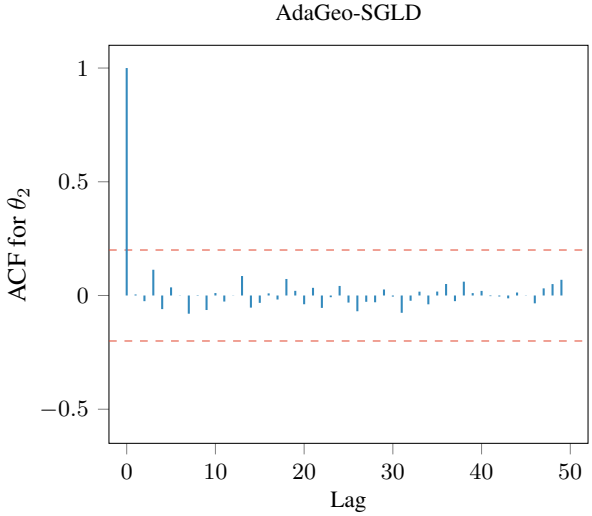
(a) Metropolis-Hastings: Autocorrelation function for θ_1



(b) Metropolis-Hastings: Autocorrelation function for θ_2



(c) AdaGeo-SGLD: Autocorrelation function for θ_1



(d) AdaGeo-SGLD: Autocorrelation function for θ_2

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	AdaGeo: Adaptive Geometric Learning for Optimization and Sampling
Publication Status	☒ Published ☐ Accepted for Publication ☐ Submitted for Publication ☐ Unpublished and unsubmitted work written in a manuscript style
Publication Details	Gabriele Abbati, Alessandra Tosi, Michael A Osborne and Seth Flaxman. AdaGeo: Adaptive Geometric Learning for Optimization and Sampling. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS)*. Lanzarote, Spain (2018).

Student Confirmation

Student Name:	Gabriele Abbati		
Contribution to the Paper	• Develop the original idea by co-author • Conceived the algorithmic procedure • Coded the software package and released it on github • Performed all experiments • Wrote the paper		
Signature	Gabriele Abbati	Date	23/01/2020

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title:	Mike Osborne, Professor		
Supervisor comments	I endorse this description.		
Signature	M Osborne	Date	24 JAN 2020

6

Conclusions

This dissertation presents three original pieces of work, published in as many conference papers. The common theme shared by every article is the investigation of how generative models based on latent variable spaces, together with model-based probabilistic modeling, can be useful for inference, training and interpretability in Machine Learning. In this chapter, we first recap the original contributions collected in the thesis. Then, we conclude by indicating some of the possible future lines of work that extend the current state of the papers.

6.1 Achievements

First, in Chapter 3, we addressed the important problem of parameter and state inference in ordinary differential equations, modeling tools that are ubiquitous in many disciplines. Building upon the gradient matching literature, we introduced a novel generative probabilistic framework based on constrained Gaussian Processes. Then, we proposed a robust, computationally and data efficient algorithm (named ODIN as in ODE-Informed GP regression) for state and parameter inference, which requires little to none hyper-parameter tuning. In an extensive set of experiments,

we showed that our method improves the current state of the art in terms of both accuracy and time to solution. Moreover, we empirically proved linear scaling with respect to data dimensionality. Finally, we presented the promising results that our method shows when dealing with the challenging problem of model selection.

In Chapter 4, we extended the gradient matching framework for parameter inference in ordinary differential equations, as presented in the previous chapter, to the setting of stochastic differential equations. In order to do so, we proposed a novel probabilistic noise model for Gaussian Processes that relies on the Doss-Sussmann transformation to separate the dynamics noise from the observational one. Then, we employed state-of-the-art moment matching and adversarial techniques to build loss functions that, once minimized, would yield good estimates for the parameters of interest. With the gradient matching framework, we circumvent the need for the expensive discretization schemes of more classical approaches. We proved experimentally that the two algorithms proposed in Chapter 4, respectively the adversarial AReS and the moment matching MaRS, show good performances and robustness when compared to state-of-the-art solutions based on extended Kalman filtering and Gaussian Processes.

Chapter 5 saw our attention shifting to the supporting optimization and sampling algorithms that can be found at the heart of several Machine Learning methods. When operating in high-dimensional settings, well-known issues such as slow-mixing, non-convexity and correlations between variables can substantially hinder inference. We proposed to address these issues by assuming that optimization and sampling methods act on a lower dimensional manifold embedded in the real parameter space, which we learned through Gaussian Process latent variable models during the inference process. The resulting algorithm, AdaGeo, adaptively learns the geometry of the parameter space to improve the performances of optimizers and samplers. Despite being generally applicable to any optimization and sampling algorithm, we showed how this framework can be included in any flavour of stochastic gradient descent, stochastic gradient Langevin dynamics and stochastic gradient Riemannian Langevin dynamics. Efficiency improvements

against the vanilla versions of the respective methods were proved in three different experiments.

6.2 Discussion and Future Work

In the very last section of this dissertation, we would like to spend a few words on how the works presented in the previous chapters can be extended and improved.

Alternative Inference Schemes for ODEs In Chapter 3 we presented a novel probabilistic model, based on Gaussian Processes, to address the problem of parameter and state inference in systems of ordinary differential equations, in the context of the gradient matching framework. Because of the inherent nature of the algorithm, there are two unavoidable downsides: first, the optimal parameters $\boldsymbol{\theta}$ are found through optimization, which allows for access to a single point estimate, without uncertainty quantification. Second, since inference relies on Gaussian Processes, the convenient analytical properties are balanced out in the trade-off by the expensive training procedures, which scale cubically in the number of data points N. The ODIN algorithm thus shows two possible directions for improvement.

On one hand, it would be interesting to investigate alternative probabilistic models, in particular exploring the advantages that variational inference [Jordan et al., 1999; Blei et al., 2017] could bring to this framework. The challenge, in this case, consists in formulating an appropriate graphical model that would allow for (approximate) full posterior inference: in this way, following a more Bayesian practice, we could impose prior distributions on the parameter space that would translate to a full posterior distribution on $\boldsymbol{\theta}$ as a consequence of the inference process. In a data-scarce setting, having access to uncertainty quantification and multiple posterior modes can be extremely beneficial.

On the other side, we can address the scalability shortcomings. Despite the fact that in many industrial settings measurements are still expensive and cumbersome (for example when quantifying chemical concentrations in a large composite

bio-reactor), the recent progress in monitoring technologies contributed to increasing the quantity of the available data available. As we mentioned above, one of the main drawbacks of Gaussian Processes is the bottleneck represented by the poor scalability of the training procedure. For this reason, it could be interesting to investigate how sparse GPs [Snelson and Ghahramani, 2006; Titsias, 2009] would perform when inserted in the ODIN framework. Using M latent inducing points (with $M \ll N$), sparse GPs promise faster inference and improved scalability; differentiability and stability with respect to the inputs need nevertheless to be accounted for. Alternatively, one could think about employing deep learning approaches based on neural networks, on which nonetheless smoothness and differentiability with respect to the inputs need to be imposed.

Support for More General SDEs In Chapter 4, we proposed a novel probabilistic noise model for Gaussian Processes that effectively separates the dynamics noise from the observational one in order to perform parameter inference on the diffusion term of the SDE. In more detail, consider the general form for the SDE

$$\mathrm{d}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), \boldsymbol{\theta})\mathrm{d}t + \mathbf{G}\mathrm{d}\mathbf{w}(t), \tag{6.1}$$

which is linear with respect to the diffusion term $\mathrm{d}\mathbf{w}(t)$. In order to infer the matrix $\mathbf{G}$, we apply here the Doss-Sussmann transformation

$$\mathbf{z}(t) = \mathbf{x}(t) - \mathbf{o}(t), \tag{6.2}$$

where $\mathbf{o}(t)$ is the solution of the SDE

$$\mathrm{d}\mathbf{o}(t) = -\mathbf{o}(t) + \mathbf{G}\mathrm{d}\mathbf{w}(t). \tag{6.3}$$

Using Itô's formula, we obtain the following random ODE for $\mathbf{z}$:

$$\mathrm{d}\mathbf{z}(t) = \{\mathbf{f}(\mathbf{z}(t) + \mathbf{o}(t), \boldsymbol{\theta}) + \mathbf{o}(t)\}\,\mathrm{d}t. \tag{6.4}$$

For the framework to work, it is necessary for the SDE in Eq. 6.3 to be independent from the states $\mathbf{x}(t)$, which puts strong limitations on the models our

method can be applied to. Naturally, a clear way to extend this work is to relax the linearity and independence assumptions by enabling support for general diffusion functions $\mathbf{g}(\mathbf{x}(t), \boldsymbol{\theta})$ as opposed to a real-valued constant matrix $\mathbf{G}$. This can be done with more complicated and general bridge construct that would replace the Doss-Sussmann transformation. Of course the ancestral sampling scheme will need to be adjusted as well, in a way that still does not require the user to numerically integrate the stochastic process $\mathbf{o}(t)$.

AdaGeo Extensions In Chapter 5 we showed how Gaussian Process latent variable models can be used in a meta-algorithm fashion to improve efficiency of many optimization and sampling routines. While this particular work is more conceptual than of practical use, it would be interesting to see how it can be applied to more modern sampling algorithms, such as Hamiltonian Monte Carlo [Duane et al., 1987; Neal et al., 2011], but also to the widely used framework of variational inference. Furthermore, one can think about incorporating the ideas of Bayesian model selection in order to avoid some of the necessary hyper-parameter tuning (e.g. kernel selection, latent dimension, etc.).

Finally, on a more speculative side, it would be interesting to consider how AdaGeo can be inserted in the context of reinforcement learning, namely within a policy gradient framework. As the Fisher information matrix of the latent space can be analytically computed as the Hessian matrix of the mapping, the natural gradient could thus be easily obtained. With many efforts that attempt to approximate such gradient in the policy space, it would be interesting to explore the effects of optimizing the policy in the latent space, following the gradient adjusted according to the information matrix.

A

Code Implementation

This section aims at giving a technical and user-friendly overview of the code provided for the three contributions presented in this thesis. For each paper, first we outline the code workflow; then, we proceed to show and comment some examples that clarify how the software can be executed and, most importantly, extended.

The main objectives of this appendix is to promote result transparency in the Machine Learning community and to facilitate code usage and extension for future research.

A.1 ODIN: ODE-Informed Regression for Parameter and State Inference in Time-Continuous Dynamical Systems

First, we look at the code implementation for the ODIN paper. At its core, the algorithm presented in this paper is fundamentally an optimization routine that minimizes the risk formulation presented in Eq. 16 of the paper.

In order to be able to perform a robust and efficient optimization scheme, we implemented a fully differentiable pipeline based on the `TensorFlow` library. This

routine has two main steps:

- First, a Gaussian Process is fit to the system observations by identifying the kernel hyper-parameters that maximize the model likelihood.
- Second, the estimated states $\mathbf{x}$ and derivatives $\dot{\mathbf{x}}$ are inserted in the risk formula and the whole expression is minimized in order to find the system parameters $\boldsymbol{\theta}$ and the derivative uncertainty $\boldsymbol{\gamma}$.

To better understand how the provided implementation can be extended, let us look at a specific example. In the following, we present the code that generated the results relative to the Lotka-Volterra experiment, whose corresponding `python` script can be found in `ODIN/odin/examples/lotka_volterra.py`.

First, if no data is provided or if we wish to fit a synthetic system for diagnostic purposes, we need to generate some noisy observations. In case of the Lotka-Volterra, we provide the necessary API:

```
from odin import LotkaVolterra

lotka_volterra_simulator = LotkaVolterra(
    true_param=(2.0, 1.0, 4.0, 1.0),
    noise_variance=0.1**2,
)

system_obs, t_obs = lotka_volterra_simulator.observe(
    initial_state=(5.0, 3.0),
    initial_time=0.0,
    final_time=2.0,
    t_delta_integration=0.01,
    t_delta_observation=0.1,
)

n_states, n_points = system_obs.shape
```

The second and crucial step consists in instantiating an object that implements the abstract class `TrainableModel`. This class defines the family of systems that we want to fit to the observations and can be passed to the optimizer for the training procedure. There are two abstract methods that need to be implemented:

- `_initialize_parameter_variables()`: initialize the `TensorFlow Variable` `self.theta`, which will contain the optimization parameters $\boldsymbol{\theta}$.
- `compute_gradients(x)`: take the input states $\mathbf{x}$ and compute $\mathbf{f}(\mathbf{x}, \boldsymbol{\theta})$ using the field `self.theta` initialized above.

For Lotka-Volterra, everything is already implemented:

```
import numpy as np
from odin import TrainableLotkaVolterra

theta_bounds = np.array(
    [[0.0, 100.0], [0.0, 100.0], [0.0, 100.0], [0.0, 100.0]]
)

trainable_lotka_volterra = TrainableLotkaVolterra(
    n_states,
    n_points,
    bounds=theta_bounds
)
```

The variable `theta_bounds` represents the hard bounds for $\boldsymbol{\theta}$ when minimizing the risk formulation.

Finally, we are ready to instantiate the `ODIN` optimizer and perform the actual optimization routine:

```
odin_optimizer = ODIN(
    trainable_lotka_volterra,
    system_obs,
    t_obs,
    gp_kernel='RBF',  # For LV we use the RBF kernel
    optimizer='L-BFGS-B',  # L-BFGS-B optimizer for the bounds
    initial_gamma=1.0,  # initial gamma value
    train_gamma=True,  # gamma will be trained as well
    state_bounds=state_bounds,  # Pass the state bounds
    single_gp=True,  # we use one set of HP for both states
    basinhopping=False,  # we don't use the basinhopping here
    time_normalization=True,  # time normalization on
    state_normalization=True,  # states normalization on
)

```

```
# Build the model
odin_optimizer.build_model()

# Fit the model
final_theta, final_gamma, final_x = odin_optimizer.fit()
```

A.2 AReS and MaRS - Adversarial and MMD-Minimizing Regression for SDEs

The implementation code for the AReS and MaRS paper is very similar to the one presented above. Once again, the algorithm is implemented from scratch and leverages the automatic differentiation capabilities of `TensorFlow`. The optimization works in two main steps:

- First, a Gaussian Process (equipped with the novel noise model described by Eq. 22 of the paper) is fit to the system observations by identifying the kernel hyper-parameters and the diffusion matrix $\mathbf{G}$ that maximize the model likelihood.
- Second, the estimated states $\mathbf{z}$ and derivatives $\dot{\mathbf{z}}$ are used in the optimization routine that aims at closing the gap between between model and data, by identifying the system parameters $\boldsymbol{\theta}$ that minimize an adversarial (AReS) or MMD (MaRS) loss function.

Once again, let us look at a specific example. In this case, we are going to discuss an SDE extension of the Lotka-Volterra system presented above and used in the experimental section of the paper. The corresponding `python` scripts can be found in `AReS-MaRS/aresmars/examples/ares/lotka_volterra.py` and `AReS-MaRS/aresmars/examples/mars/lotka_volterra.py`. Without loss of generality, we will focus on the MaRS implementation.

As before, if no data is provided or if we desire to fit a synthetic system, we need to generate some noisy observations. In case of the Lotka-Volterra, the necessary API is provided:

```python
import numpy as np
from aresmars import LotkaVolterraSDE

g_matrix = np.array([[0.2, 0.0], [0.1, 0.3]])

lotka_volterra_simulator = LotkaVolterraSDE(
    true_param=(2.0, 1.0, 4.0, 1.0),
    obs_variance=0.1**2,
    wiener_std_dev=g_matrix
)

system_obs, t_obs = lotka_volterra_simulator.observe(
    initial_state=[5.0, 3.0],
    final_time=2.0,
    t_delta_integration=1e-4,
    t_delta_observation=0.04,
    n_rollouts=1
)
system_obs = np.squeeze(system_obs)
```

Analogously to ODIN, the next step is the most important one. The user needs to instantiate an implementation of the abstract class `TrainableModelSDE`. This object defines the family of systems (this time of stochastic differential equations) that we want to fit to the observations and can be passed to the optimizer for the training procedure. Again, there are two abstract methods to implement:

- `_initialize_parameter_variables()`: same functionality as ODIN, initialize the `TensorFlow Variable` `self.theta`, which will contain the optimization parameters $\boldsymbol{\theta}$.
- `compute_gradients(x)`: take the input states $\mathbf{x}$ and compute $\mathbf{f}(\mathbf{x}, \boldsymbol{\theta})$ using the field `self.theta` initialized above. Notice that this represents only the deterministic part of the SDE system, since the stochasticity in the derivatives is captured by the noise model of the GP.

For Lotka-Volterra we can just import the provided classes:

```python
from aresmars import TrainableLotkaVolterra

trainable_lv = TrainableLotkaVolterra()
```

Last, we instantiate the optimizer and call the appropriate methods:

```python
mars_solver = MARS(
    trainable_lv,
    system_obs,
    t_obs,
    gp_kernel='RBF',
    mmd_kernel='RationalQuadratic',
    optimizer='Adam',
    starter_learning_rate=1e-2,
    batch_size=128,
    use_single_gp=False,
    diagonal_wiener_process=False,
    state_normalization=True,
    time_normalization=True
)

mars_solver.build_model()

# Train the model
final_theta, final_g = mars_solver.train(n_batches=1000)
```

If the AReS optimizer is preferred, then it is to be instantiated instead of MaRS. Apart from this last step, the procedure stays the same. An example can be found in `AReS-MaRS/aresmars/examples/ares/lotka_volterra.py`.

A.3 AdaGeo: Adaptive Geometric Learning for Optimization and Sampling

Finally, we discuss here the `python` implementation for the algorithms illustrated in the AdaGeo paper.

In general, with the proposed methods we aim to analyze the behaviour of sampling and optimization algorithms through Gaussian Process Latent Variable Models: using these probabilistic models we wish to learn a lower-dimensional

representation of the space on which sampling and optimization is effectively taking place. In this way we can speed up computation with geometry-informed proposals of samples or optimization steps.

The AdaGeo API provides implementations for optimizers:

- `AdaGeoGradientDescent`;
- `AdaGeoMomentumDescent`;
- `AdaGeoAdaGrad`;
- `AdaGeoNaturalGradientDescent`,

and samplers:

- `AdaGeoSGLD`;
- `AdaGeoSGRLD`.

Despite the several execution possibilities, the workflow is extremely similar for every option (both for optimization and sampling). Many usage details can be found in the documentation and comments within the code. For these reasons, we will showcase a specific example of how AdaGeo can be used to train a small neural network. Looking at the code, variations are straightforward. The script here shown can be found at `AdaGeo/tests/test_optimizer_nn.py`.

First, we need to instantiate the `LogRegressionMNIST` object, a simple single-layer neural network that performs logistic regression on the MNIST dataset:

```
from tests.optimization_tests import LogRegressionMNIST

test_network = LogRegressionMNIST(n_batch=750)
test_network.build_model()
```

Then, we initialize the vanilla optimizer which AdaGeo will leverage to learn the latent space:

```
from adageo.optimizers import GradientDescent

observed_optimizer = GradientDescent(
    test_network,
    test_network.dim_observed,
    learning_rate=1e-1,
    rate_decay=0.001
)
```

Finally, we can create the AdaGeo optimizer `AdaGeoMomentumDescent` and use it to train the network, together with the vanilla gradient descent:

```
from adageo.optimizers import AdaGeoMomentumDescent

adageo_optimizer = AdaGeoMomentumDescent(
    test_network,
    observed_optimizer,
    learning_rate=5e-1,
    rate_decay=0.01,
    momentum_factor=0.6,^^I
    t_observed=20,
    t_latent=20
)

adageo_optimizer.optimize(dim_latent=9, n_iterations=2, ard=True)
```

As previously mentioned, alternative optimizers (and samplers) can be employed while maintaining the same execution structure.

Bibliography

Hirotugu Akaike. A new look at the statistical model identification. *IEEE transactions on automatic control*, 19(6):716–723, 1974.

Christophe Andrieu and Johannes Thoms. A tutorial on adaptive MCMC. *Statistics and computing*, 18(4):343–373, 2008.

Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to MCMC for machine learning. *Machine learning*, 50(1-2):5–43, 2003.

Christophe Andrieu, Arnaud Doucet, and Roman Holenstein. Particle Markov Chain Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72 (3):269–342, 2010.

Cedric Archambeau, Dan Cornford, Manfred Opper, and John Shawe-Taylor. Gaussian Process approximations of stochastic differential equations. *Journal of machine learning research*, 1:1–16, 2007.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pages 214–223, 2017.

Nachman Aronszajn. Theory of reproducing kernels. *Transactions of the American mathematical society*, 68(3):337–404, 1950.

Yonathan Bard. Nonlinear parameter estimation. 1974.

Richard E Bellman. *Adaptive control processes: a guided tour*, volume 2045. Princeton university press, 2015.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: a review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8): 1798–1828, 2013.

M Benson. Parameter fitting in dynamic models. *Ecological Modelling*, 6(2):97–115, 1979.

Harish S Bhat, RWMA Madushani, and Shagun Rawat. Parameter inference for stochastic differential equations with density tracking by quadrature. In *International Workshop on Simulation*, pages 99–113. Springer, 2015.

Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

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

Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of political economy*, 81(3):637–654, 1973.

David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: a review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

Leo Breiman et al. Statistical modeling: the two cultures. *Statistical science*, 16(3):199–231, 2001.

Ben Calderhead, Mark Girolami, and Neil D Lawrence. Accelerating Bayesian inference over nonlinear differential equations with Gaussian Processes. In *Advances in neural information processing systems*, pages 217–224, 2009.

Olivier Cappé, Randal Douc, Arnaud Guillin, Jean-Michel Marin, and Christian P Robert. Adaptive importance sampling in general mixture classes. *Statistics and Computing*, 18 (4):447–459, 2008.

Ting Chen, Hongyu L He, and George M Church. Modeling gene expression with differential equations. In *Biocomputing'99*, pages 29–40. World Scientific, 1999.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3): 273–297, 1995.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

Frank Dondelinger, Dirk Husmeier, Simon Rogers, and Maurizio Filippone. ODE parameter inference using adaptive gradient matching with Gaussian Processes. In *Artificial Intelligence and Statistics*, pages 216–228, 2013.

Halim Doss. Liens entre équations différentielles stochastiques et ordinaires. 1977.

Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid Monte Carlo. *Physics letters B*, 195(2):216–222, 1987.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

Gintare Karolina Dziugaite, Daniel M Roy, and Zoubin Ghahramani. Training generative neural networks via Maximum Mean Discrepancy optimization. *arXiv preprint arXiv:1505.03906*, 2015.

Ola Elerian, Siddhartha Chib, and Neil Shephard. Likelihood inference for discretely observed nonlinear diffusions. *Econometrica*, 69(4):959–993, 2001.

Bjørn Eraker. MCMC analysis of diffusion models with application to finance. *Journal of Business & Economic Statistics*, 19(2):177–191, 2001.

Dirk Fey, Melinda Halasz, Daniel Dreidax, Sean P Kennedy, Jordan F Hastings, Nora Rauch, Amaya Garcia Munoz, Ruth Pilkington, Matthias Fischer, Frank Westermann, et al. Signaling pathway models as biomarkers: patient-specific simulations of JNK activity predict the survival of neuroblastoma patients. *Sci. Signal.*, 8(408):ra130–ra130, 2015.

Christiane Fuchs. *Inference for diffusion processes with applications in life sciences*. Springer Science & Business Media, 2013.

Andrew Gelman, John B Carlin, Hal S Stern, David B Dunson, Aki Vehtari, and Donald B Rubin. *Bayesian data analysis*. Chapman and Hall/CRC, 2013.

Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521 (7553):452, 2015.

Walter R Gilks and Carlo Berzuini. Following a moving target Monte Carlo inference for dynamic Bayesian models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(1):127–146, 2001.

Walter R Gilks and Pascal Wild. Adaptive rejection sampling for Gibbs sampling. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 41(2):337–348, 1992.

Walter R Gilks, Sylvia Richardson, and David Spiegelhalter. *Markov Chain Monte Carlo in practice*. Chapman and Hall/CRC, 1995.

Daniel T Gillespie. The chemical Langevin equation. *The Journal of Chemical Physics*, 113(1): 297–306, 2000.

Mark Girolami and Ben Calderhead. Riemann manifold Langevin and Hamiltonian Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73 (2):123–214, 2011.

Andrew Golightly and Darren J Wilkinson. Bayesian parameter inference for stochastic biochemical network models using particle Markov Chain Monte Carlo. *Interface focus*, 1 (6):807–820, 2011.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Nico S Gorbach, Stefan Bauer, and Joachim M Buhmann. Scalable variational inference for dynamical systems. In *Advances in Neural Information Processing Systems*, pages 4806–4815, 2017.

Arthur Gretton, Karsten Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex J Smola. A kernel method for the two-sample-problem. In *Advances in neural information processing systems*, pages 513–520, 2007.

Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(Mar):723–773, 2012.

Helge Hass, Kristina Masson, Sibylle Wohlgemuth, Violette Paragas, John E Allen, Mark Sevecka, Emily Pace, Jens Timmer, Joerg Stelling, Gavin MacBeath, et al. Predicting ligand-dependent tumors from multi-dimensional signaling features. *NPJ systems biology and applications*, 3(1):27, 2017.

W Keith Hastings. Monte Carlo sampling methods using Markov Chains and their applications. *Biometrika*, 57(1):97–109, 1970.

David M Higdon. Auxiliary variable methods for Markov Chain Monte Carlo with applications. *Journal of the American statistical Association*, 93(442):585–595, 1998.

Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning, lecture 6a overview of mini-batch gradient descent. *Cited on*, 14:8, 2012.

Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. *The annals of statistics*, pages 1171–1220, 2008.

AS Hurn and KA Lindsay. Estimating the parameters of stochastic differential equations. *Mathematics and computers in simulation*, 48(4-6):373–384, 1999.

Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

Diederik P Kingma and Jimmy Ba. Adam: a method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

Neil Lawrence. Probabilistic non-linear principal component analysis with Gaussian Process latent variable models. *Journal of machine learning research*, 6(Nov):1783–1816, 2005.

Neil D Lawrence. Gaussian Process latent variable models for visualisation of high dimensional data. In *Advances in neural information processing systems*, pages 329–336, 2004.

Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabás Póczos. MMD GAN: Towards deeper understanding of moment matching network. In *Advances in Neural Information Processing Systems*, pages 2203–2213, 2017.

Yujia Li, Kevin Swersky, and Rich Zemel. Generative moment matching networks. In *International Conference on Machine Learning*, pages 1718–1727, 2015.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of the IEEE international conference on computer vision*, pages 3730–3738, 2015.

Alfred J Lotka. The growth of mixed populations: two species competing for a common food supply. *Journal of the Washington Academy of Sciences*, 22(16/17):461–469, 1932.

David JC MacKay. Probable networks and plausible predictions a review of practical Bayesian methods for supervised neural networks. *Network: computation in neural systems*, 6(3):469–505, 1995.

Nicholas Metropolis and Stanislaw Ulam. The Monte Carlo method. *Journal of the American statistical association*, 44(247):335–341, 1949.

Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

Thomas P Minka. Expectation propagation for approximate Bayesian inference. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 362–369. Morgan Kaufmann Publishers Inc., 2001.

Alfred Müller. Integral probability metrics and their generating classes of functions. *Advances in Applied Probability*, 29(2):429–443, 1997.

Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

Iain Murray and Zoubin Ghahramani. A note on the evidence and Bayesian Occams razor. 2005.

Richard M Murray. *A mathematical introduction to robotic manipulation*. CRC press, 2017.

Radford M Neal et al. MCMC using Hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

Yurii E Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547, 1983.

Andrew Y Ng and Michael I Jordan. On discriminative vs. generative classifiers: a comparison of logistic regression and naive Bayes. In *Advances in neural information processing systems*, pages 841–848, 2002.

Jan Nygaard Nielsen, Henrik Madsen, and Peter C Young. Parameter estimation in stochastic differential equations: an overview. *Annual Reviews in Control*, 24:83–94, 2000.

Babatunde Ayodeji Ogunnaike and Willis Harmon Ray. *Process dynamics, modeling, and control*, volume 1. Oxford University Press New York, 1994.

Man-Suk Oh and James O Berger. Adaptive importance sampling in Monte Carlo integration. *Journal of Statistical Computation and Simulation*, 41(3-4):143–168, 1992.

Manfred Opper and Ole Winther. Expectation consistent approximate inference. *Journal of Machine Learning Research*, 6(Dec):2177–2204, 2005.

Sam Patterson and Yee Whye Teh. Stochastic gradient Riemannian Langevin dynamics on the probability simplex. In *Advances in neural information processing systems*, pages 3102–3110, 2013.

Umberto Picchini. SDE toolbox: simulation and estimation of stochastic differential equations with MATLAB. 2007.

Susanne Pieschner and Christiane Fuchs. Bayesian inference for diffusion processes: using higher-order approximations for transition densities. *arXiv preprint arXiv:1806.02429*, 2018.

Carl Edward Rasmussen and Christopher KI Williams. *Gaussian Processes for machine learning*, volume 1. MIT press Cambridge, 2006.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

Gareth O Roberts and Jeffrey S Rosenthal. Optimal scaling of discrete approximations to Langevin diffusions. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 60(1):255–268, 1998.

Gareth O Roberts and Osnat Stramer. Langevin diffusions and Metropolis-Hastings algorithms. *Methodology and computing in applied probability*, 4(4):337–357, 2002.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Walter Rudin. *Real and complex analysis*. Tata McGraw-hill education, 2006.

David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

Andreas Ruttor, Philipp Batz, and Manfred Opper. Approximate Gaussian Process inference for the drift function in stochastic differential equations. In *Advances in Neural Information Processing Systems*, pages 2040–2048, 2013.

Thomas Ryder, Andrew Golightly, A Stephen McGough, and Dennis Prangle. Black-box variational inference for stochastic differential equations. *International Conference on Machine Learning*, 2018.

Simo Särkkä, Jouni Hartikainen, Isambi Sailon Mbalawata, and Heikki Haario. Posterior inference on parameters of stochastic differential equations via non-linear Gaussian filtering and adaptive MCMC. *Statistics and Computing*, 25(2):427–437, 2015.

Bernhard Scholkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

Gideon Schwarz et al. Estimating the dimension of a model. *The annals of statistics*, 6(2): 461–464, 1978.

Edward Snelson and Zoubin Ghahramani. Sparse Gaussian Processes using pseudo-inputs. In *Advances in neural information processing systems*, pages 1257–1264, 2006.

Helle Sørensen. Parametric inference for diffusion processes observed at discrete points in time: a survey. *International Statistical Review*, 72(3):337–354, 2004.

Héctor J Sussmann. On the gap between deterministic and stochastic ordinary differential equations. *The Annals of Probability*, pages 19–41, 1978.

Richard Sutton. Two problems with back propagation and other steepest descent learning procedures for networks. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society, 1986*, pages 823–832, 1986.

Michael E Tipping and Christopher M Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622, 1999.

Michalis Titsias. Variational learning of inducing variables in sparse Gaussian Processes. In *Artificial Intelligence and Statistics*, pages 567–574, 2009.

Filip Tronarp and Simo Särkkä. Iterative statistical linear regression for Gaussian smoothing in continuous-time non-linear stochastic dynamic systems. *arXiv preprint arXiv:1805.11258*, 2018.

Frank van der Meulen, Moritz Schauer, et al. Bayesian estimation of discretely observed multi-dimensional diffusion processes using guided proposals. *Electronic Journal of Statistics*, 11(1):2358–2396, 2017.

Nicolaas Godfried Van Kampen. *Stochastic processes in physics and chemistry*, volume 1. Elsevier, 1992.

Vladimir Vapnik et al. Statistical learning theory, 1998.

James M Varah. A spline least squares method for numerical parameter estimation in differential equations. *SIAM Journal on Scientific and Statistical Computing*, 3(1):28–46, 1982.

Cédric Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

Michail D Vrettas, Manfred Opper, and Dan Cornford. Variational mean-field algorithm for efficient inference in large systems of stochastic differential equations. *Physical Review E*, 91(1):012148, 2015.

Sumio Watanabe. *Algebraic geometry and statistical learning theory*, volume 25. Cambridge university press, 2009.

Sumio Watanabe. Equations of states in singular statistical estimation. *Neural Networks*, 23 (1):20–34, 2010.

Sumio Watanabe. A widely applicable bayesian information criterion. *Journal of Machine Learning Research*, 14(Mar):867–897, 2013.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688, 2011.

Philippe Wenk, Alkis Gotovos, Stefan Bauer, Nico Gorbach, Andreas Krause, and Joachim M Buhmann. Fast Gaussian Process based gradient matching for parameter identification in systems of nonlinear ODEs. *arXiv preprint arXiv:1804.04378*, 2018.

Cagatay Yildiz, Markus Heinonen, Jukka Intosalmi, Henrik Mannerström, and Harri Lähdesmäki. Learning stochastic differential equations with Gaussian Processes without gradient matching. *arXiv preprint arXiv:1807.05748*, 2018.

Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. LSUN: construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

Matthew D Zeiler. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Wei-Bin Zhang. *Differential equations, bifurcations, and chaos in economics*, volume 68. World Scientific Publishing Company, 2005.