Non-Stationary Surrogate Modeling with Deep Gaussian Processes

Vincent Dutordoir

Supervisors: Prof. dr. ir. Tom Dhaene, Dr. ir. Ivo Couckuyt
Counsellors: Joachim van der Herten, Nicolas Knudde

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Department of Information Technology
Chair: Prof. dr. ir. Bart Dhoedt
Faculty of Engineering and Architecture
Academic year 2016-2017

GHENT
UNIVERSITY

This page is intentionally
left blank.

Non-Stationary Surrogate Modeling with Deep Gaussian Processes

Vincent Dutordoir

Supervisors: Prof. dr. ir. Tom Dhaene, Dr. ir. Ivo Couckuyt
Counsellors: Joachim van der Herten, Nicolas Knudde

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Department of Information Technology
Chair: Prof. dr. ir. Bart Dhoedt
Faculty of Engineering and Architecture
Academic year 2016-2017

GHENT
UNIVERSITY

Foreword

With this dissertation, I conclude my current studies. During this five-year journey I have learned the value of education, the power of perseverance and the merits of collaboration. Along the way I have had help from many people.

I want to express my sincere gratitude to my supervisor, Prof. Tom Dhaene, for providing a thriving intellectual environment, giving me the opportunity of researching this very interesting subject and for his invaluable guidance.

I would like to thank my counsellors Nicolas Knudde, Dr. Joachim van der Herten and Dr. Ivo Couckuyt, for their close support of this dissertation. They were always ready with wise advice and eagerness to answer my questions, whether I was in the continent or not. They were patient during the first semester, when I was trying to wrap my head around Gaussian Processes, and trusted me with their many great ideas once I was ready. This work would not have been possible without them. Nicolas, Joachim and Ivo, thank you.

A big thanks to my fellow students and friends, Dorian, Jonas, Lars, Loïc, Timon and Ward. We battled shoulder to shoulder during the long days spent in the lab, shared the hope for good results and the sadness with each failed attempt.

Finally, on a personal note, I would like to thank my amazing partner, Lien, I can achieve more with one hand, provided that you are holding the other. I am also eternally grateful for my family, particularly my parents, who gave me every opportunity.

Vincent Dutordoir, June 2017

> *One, remember to look up at the stars and not down at your feet. Two, never give up work. Work gives you meaning and purpose and life is empty without it. Three, if you are lucky enough to find love, remember it is there and don't throw it away.*
>
> *Stephen Hawking*

Permission of use on loan

"The author gives permission to make this master dissertation available for consultation and to copy parts of this master dissertation for personal use. In all cases of other use, the copyright terms have to be respected, in particular with regard to the obligation to state explicitly the source when quoting results from this master dissertation."

Vincent Dutordoir, June 2017

Non-Stationary Surrogate Modeling with Deep Gaussian Processes

Vincent Dutordoir

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Academic year 2016-2017

Supervisor: prof. dr. ir. T. Dhaene and dr. ir. I. Couckuyt
Counsellors: dr. ir. J. van der Herten and ir. N. Knudde

Faculty of Engineering and Architecture
Ghent University

Department of Information Technology
Chair: Prof. dr. ir. B. Dhoedt

Abstract

Gaussian Processes (GPs) are non-parametric, probabilistic models. They provide a flexible prior over latent functions and tractable posteriors for hyperparameter learning as well as inference. However, this flexibility and regularization power come at a price and some considerations should be made when using GP models. Firstly, GPs configured with most of today's popular kernels only succeed in intelligent smoothing and interpolation. Secondly, the computation and memory requirements during training scales cubically in the number of data. This work focuses on both shortcomings and proposes viable solutions for them. The latter problem, the computational bottleneck, is reduced by speeding up the most expensive operation, from an implementation point-of-view. The first problem, the lack of extrapolation capacities, is tackled from two different directions. On the one hand, Deep Gaussian Processes (DGPs), which are multi-layer generalizations of GPs, are leveraged as they are able to overcome the typical limitations while retaining the advantages. On the other hand, the usage of the Spectral Mixture (SM) kernel is advocated, as this kernel can successfully be used for time series extrapolation and pattern discovery. The applicability and advantages of the suggested approaches are verified by a series of real-world experiments, with a focus on metamodeling and uncertain predictions.

Keywords

Gaussian Processes, Deep Gaussian Processes, Metamodeling, Spectral Mixture Kernel, Uncertainty Propagation

Non-Stationary Surrogate Modeling with Deep Gaussian Processes

Vincent Dutordoir
Supervisors: prof. dr. ir. Tom Dhaene, dr. ir. Ivo Couckuyt
Counsellors: dr. ir. Joachim van der Herten, ir. Nicolas Knudde

***Abstract*—Gaussian Processes (GPs) are non-parametric, probabilistic models. They provide a flexible prior over latent functions and tractable posteriors for hyperparameter learning as well as inference. However, this flexibility and regularization power come at a price and some considerations should be made when using GP models. Firstly, GPs configured with most of today's popular kernels only succeed in intelligent smoothing and interpolation. Secondly, the computation and memory requirements during training scales cubically in the number of data. This work focuses on both shortcomings and proposes viable solutions for them. The latter problem, the computational bottleneck, is reduced by speeding up the most expensive operation, from an implementation point-of-view. The first problem, the lack of extrapolation capacities, is tackled from two different directions. On the one hand, Deep Gaussian Processes (DGPs), which are multi-layer generalizations of GPs, are leveraged as they are able to overcome the typical limitations while retaining the advantages. On the other hand, the usage of the Spectral Mixture (SM) kernel is advocated, as this kernel can successfully be used for time series extrapolation and pattern discovery. The applicability and advantages of the suggested approaches are verified by a series of real-world experiments, with a focus on metamodeling and uncertain predictions.**

***Keywords*—Gaussian Processes, Deep Gaussian Processes, Metamodeling, Spectral Mixture Kernel, Uncertainty Propagation**

I. Background

Gaussian Processes (GPs) became widely used as a non-parametric model after Radford Neal reintroduced them to the probabilistic machine learning community in the mid-nineties. In his work he showed that a one hidden layer Neural Network (NN), given suitable priors for the weights, converges towards a GP as the number of hidden neurons tends to infinity [1]. This led researchers, like Carl Rasmussen, to carry out a series of experiments in which they compared GPs with other frequently used models, such as NNs and regression splines, and show that GPs outperform them [2]. However, in 1997 David MacKay questioned: *"Are Gaussian Processes a replacement for supervised neural networks? Or have we thrown the baby out with the bath water?"* [3]. MacKay's rhetorical question illustrated perfectly the frustration researchers had grown about GPs being nothing more than intelligent smoothing devices, unable to perform difficult tasks, such as feature discovery for example. However, in his work he counter-argues that we do not have to underestimate the power of smoothing methods and that they can be a valuable alternative to the hyped neural networks, when used correctly. Since then, numerous GP-based models have been successfully developed and have shown excellence in a range of different fields, including supervised learning, manifold learning, reinforcement learning and Bayesian optimization [4, 5, 6, 7].

This paper is organized as follows. Section II starts with a brief overview of the GP model. Section III focuses on the computational problem, from a model and an algorithmic point-of-view. Section IV studies the possibilities of using a GP for extrapolation, rather than only interpolation. For this reason, Section IV-A focuses on the Deep Gaussian Process (DGP) model and Section IV-C on the Spectral Mixture (SM) kernel.

II. Gaussian Processes

A GP is a set of random variables $\{\mathrm{f}(\mathbf{x}_1), \mathrm{f}(\mathbf{x}_2), \mathrm{f}(\mathbf{x}_3), \ldots\}$ for which any finite subset follows a multivariate Gaussian distribution [4]. It is fully specified by its mean $m(\mathbf{x})$ and its covariance or kernel function $k(\mathbf{x}, \mathbf{x}')$. The model can impose a prior over a real-valued unknown function $\mathrm{f}(\mathbf{x})$, which can be combined with observed data to give a posterior over this function. As for most supervised learning tasks, the observations are bundled in a dataset $\mathcal{D} = \{\mathbf{x}_n, \mathrm{y}_n\}_{n=1}^N$, where $\mathrm{y}_n = \mathrm{f}(\mathbf{x}_n) + \epsilon_n$, with $\epsilon_n \sim \mathcal{N}(0, \sigma^2)$ uncorrelated white noise. The N inputs $\mathbf{x}_n$ are collected as rows of the matrix $\mathrm{X} \in \mathbb{R}^{N \times D}$, where D represents the number of features per sample.

Inference in the GP model is relatively straightforward. A GP prior on training latent values $\mathbf{f}$ and test latent value $\mathrm{f}(\mathbf{x}_*)$ is placed, and combined with a Gaussian likelihood $p(\mathbf{y}|\mathbf{f})$ using Bayes rule, to obtain the joint posterior $p(\mathbf{y}, \mathrm{f}(\mathbf{x}_*))$. After marginalizing out the training data variables, the desired posterior predictive distribution is obtained $p(\mathrm{f}(\mathbf{x}_*)|\mathbf{x}_*, \mathcal{D}) = \mathcal{N}(\mu(\mathbf{x}_*), \sigma^2(\mathbf{x}_*))$, with

$$\mu(\mathbf{x}_*) = k(\mathrm{X}, \mathbf{x}_*)^\top \tilde{\mathrm{K}}_{\mathbf{ff}}^{-1} \mathbf{y}, \tag{1}$$

$$\sigma^2(\mathbf{x}_*) = k(\mathbf{x}_*, \mathbf{x}_*) - k(\mathrm{X}, \mathbf{x}_*)^\top \tilde{\mathrm{K}}_{\mathbf{ff}}^{-1} k(\mathrm{X}, \mathbf{x}_*), \tag{2}$$

where $\tilde{\mathrm{K}}_{\mathbf{ff}} \triangleq k(\mathrm{X}, \mathrm{X}) + \sigma^2 \mathrm{I}$ and $k(\mathrm{X}, \mathrm{X})$ is a $N \times N$ covariance matrix holding the correlation between every pair of input vectors. The GP's posterior is heavily characterized by the kernel function k used. Its specific form (e.g., Exponential, Linear, Periodic, Matérn, etc.) is chosen upfront, but typically depends on a limited set of kernel hyperparameters $\boldsymbol{\theta}_k$. These hyperparameters, as well as the noise parameter σ^2, can be found by optimizing the log marginal likelihood

$$\log p(\mathbf{y}|\mathrm{X}, \boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^\top \tilde{\mathrm{K}}_{\mathbf{ff}}^{-1} \mathbf{y} - \frac{1}{2}\log|\tilde{\mathrm{K}}_{\mathbf{ff}}| - \frac{N}{2}\log 2\pi, \tag{3}$$

where $\boldsymbol{\theta} = \{\boldsymbol{\theta}_k, \sigma^2\}$. This objective function has a dual purpose: the first term of the equation can be interpreted as a data-fit term, while the second is a regularization term, which guards against overfitting and penalizes unnecessary complex models.

III. Efficient Gaussian Process Modeling

The full predictive distribution, characterized by Eqs. (1) and (2), is one of the GP's main advantage. The mean $\mu(\mathbf{x}_*)$

can serve as a point estimate of the function output $\mathrm{f}(\mathbf{x}_*)$, while the variance $\sigma^2(\mathbf{x}_*)$ can be used to quantify the model's uncertainty of this estimation. In a sense, *the model knows what it does not know*, which is a favorable property to have, for example, in autonomous decision-making systems. However, the problem with the mean, variance and marginal likelihood expressions, is that they require inversion of a $N \times N$ matrix. The most computationally stable and efficient way to obtain this is via the Cholesky decomposition, but still requires $\mathcal{O}(N^3)$ operations. This inevitably restricts the standard GP implementation to smaller datasets, with not more than a few thousands samples.

A. Approximate Inference

The main line of work in the literature attempting to overcome the computational bottleneck of standard GPs is related to sparse (variational) approximations [8, 9, 10]. Those models expand the standard GP with M additional inducing output variables $\mathbf{u}$ and their corresponding inputs Z. They further introduce a GP prior on $\mathbf{u}$ and assume that they are jointly Gaussian with the latent function values $\mathbf{f}$. The expanded model then complies with the following joint probability

$$p(\mathbf{y}, \mathbf{f}, \mathbf{u}|\mathrm{Z}, \mathrm{X}) = p(\mathbf{u}|\mathrm{Z})\, p(\mathbf{f}|\mathbf{u}, \mathrm{Z}, \mathrm{X})\, p(\mathbf{y}|\mathbf{f}).$$

Titsias [10] shows that this equation can be further derived, by applying Variational Inference (VI) using the approximate distribution $q(\mathbf{f}, \mathbf{u}) = q(\mathbf{u})p(\mathbf{f}|\mathbf{u})$, to lower bound the log marginal likelihood as

$$\log p(\mathbf{y}|\mathbf{u}) \geq \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathrm{Q}_{\mathbf{ff}} + \sigma^2\mathrm{I}) - \frac{1}{2\sigma^2}\,\mathrm{tr}\,(\mathrm{K}_{\mathbf{ff}} - \mathrm{Q}_{\mathbf{ff}}), \quad (4)$$

where the matrix $\mathrm{Q}_{\mathbf{ff}} \triangleq \mathrm{K}_{\mathbf{fu}}\mathrm{K}_{\mathbf{uu}}^{-1}\mathrm{K}_{\mathbf{uf}}$ is called the *Nyström* approximation. The chief attribute of sparse approximations is to approximate the marginal likelihood $p(\mathbf{y}|\mathbf{u})$ by replacing the $N \times N$ covariance matrix $\mathrm{K}_{\mathbf{ff}}$ in (3) by a low-rank matrix $\mathrm{Q}_{\mathbf{ff}}$. As the approximation is based on a set of M inducing variables, with $M \ll N$, the computational complexity of the model is lowered to $\mathcal{O}(M^2N)$. However, the quality of the sparse model now highly depends on the representational capacity of the inducing dataset. As will be shown in the experiments of Section IV-B, the approximate inference models are able to handle large datasets more easily but have to comprise on accuracy to achieve this. Nonetheless, they are also a building block for the Deep Gaussian Process (DGP), discussed in Section IV-A.

B. Exact Inference

The approaches of the previous section seek to alleviate the computational problem by introducing an approximate model. On the contrary, in this section the focus lies on simply speeding up the calculations of $\mathbf{\Sigma}^{-1}$ for a general (positive-definite) matrix $\mathbf{\Sigma}$. This enhanced algorithm/implementation can then be used where necessary, for example in Eqs. (1), (2) and (3).

In practice, calculating $\mathbf{\Sigma}^{-1}$ is done using the Cholesky decomposition as it is numerically stable and fast. TensorFlow[1] provides such a Cholesky implementation, but it is in fact just a Python wrapper around a C++ library function. Therefore, even though of using a highly optimized C++ routine, it still imposes two major limitations. (1) The C++ implementation can not be executed on GPUs. (2) When combining different parts, where some are able to exploit GPU hardware and others are not, large transfer times arise when copying data from GPU to CPU and vice versa. Considering both limitations, the GP model could be notably enhanced if a new, more efficient and hybrid (i.e. GPU and CPU compatible) Cholesky implementation could be used.

Table I: Overview of the time required to execute 5 optimization rounds, on CPU and GPU. The *original* is the GPflow GP model, which uses the standard Cholesky decomposition from TensorFlow. The *blocked* model is the GPflow GP model, modified in order to use the new Cholesky implementation.

	N ($\times 1E3$)	2	3	4	5	6	7
CPU	original (TF) [s]	16.2	54.0	124.7	253.1	479.5	821.3
	blocked (new) [s]	32.5	63.2	96.9	143.1	197.6	313.3
	speedup	0.5	0.85	1.29	1.77	2.34	2.62
GPU	original (TF) [s]	19.2	55.4	118.1	223.1	-	-
	blocked (new) [s]	20.8	39.1	61.1	80.2	-	-
	speedup	0.92	1.42	1.93	2.78	-	-

In this dissertation, the desired Cholesky implementation was developed. It solely constitutes out of *basic* TensorFlow operations (e.g., matrix addition, multiplication, etc.), as these automatically execute on both GPU and CPU. In order to achieve great speedups and attain high performance the standard Cholesky algorithm was also reformulated in terms of matrix-matrix operations (i.e. BLAS level 3). These operations are highly efficient as they are cache-friendly and have extremely optimized assembly implementations. The developed algorithm is straightforwardly called the blocked-partitioned Cholesky decomposition. Table I shows the effects on the execution time when the block-partitioned Cholesky implementation is integrated into GPflow[2] and compares it with the original Cholesky implementation from TensorFlow. For small matrices the new algorithm is not faster, but a speedup of a factor 1 to 3 is obtained for larger matrices on CPU and GPU.

IV. Beyond Gaussian Processes

The discovery of patterns and underlying structures in data is one of the core aspirations of machine learning. Ideally, the ambition is to construct intelligent systems that are able to learn from past observations (i.e. the training data) and generalize this information in order to make future predictions. Regrettably, GPs are generally not used for this kind of complicated tasks. The reason is that a vast majority of kernels currently only succeed in interpolation rather than the desired extrapolation. An example of such a kernel is the Squared Exponential (SE), which is the de-facto standard covariance function. It is defined as follows

$$k_{SE}(\boldsymbol{\tau}) = \exp\left(-\sum_{d=1}^{D} \frac{{\tau^{(d)}}^2}{2\,\ell_d^2}\right),$$

where $\boldsymbol{\tau} = \mathbf{x} - \mathbf{x}'$ and ℓ_d is the d^{th} lengthscale. This type of stationary kernels has only a single lengthscale per dimension, which makes it, for example, not applicable for forecasting time-series or extrapolating data outside the training domain.

[1] TensorFlow (1.0) is an open source software library for numerical computation, often used for machine learning [11].

[2] A GP library that uses TensorFlow for its core computations [12].

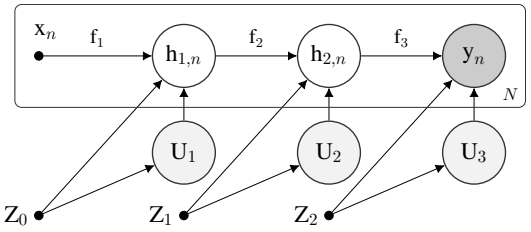


Fig. 1: 4-layer DGP's graphical model. The plate is added to show the independence of the N datapoints when they are conditioned on the inducing points.

Even more, the stationary kernels have also issues dealing with non-uniform input distributions, which are often encountered in sequential metamodeling.

This work proposes two approaches to remedy these limitations. In Section IV-C the Spectral Mixture (SM) kernel is introduced as it empowers GP-based extrapolation, and the next section focuses on DGPs.

A. Deep Gaussian Processes

A DGP [13, 14] is a probabilistic model based on the composition of processes. It is a multi-layer generalization of a GP, able to overcome its limitations while retaining the advantages. Besides, they are robust to over-fitting and provide well-calibrated uncertainty estimations. In DGPs, rather than assuming that an output $\mathbf{y}$ is drawn directly from a single GP, it is assumed that it is drawn from a functional composition of GPs and corrupted by noise $\boldsymbol{\epsilon}$, yielding

$$\mathbf{y} = \mathbf{f}_L(\mathbf{f}_{L-1}(\dots \mathbf{f}_2(\mathbf{f}_1(\mathbf{x}))) + \boldsymbol{\epsilon},$$

where every function $\mathbf{f}_\ell$ is itself a GP and $\boldsymbol{\epsilon}$ is an error term which holds the propagation of white Gaussian noise terms added to every layer. Denoting $\mathbf{h}_\ell = \mathbf{f}_\ell(\mathbf{h}_{\ell-1}) + \boldsymbol{\epsilon}_\ell$ it is possible to write the complete joint probability

$$p(\mathbf{y}, \{\mathbf{h}_\ell\}_{\ell=1}^{L-1} | \mathrm{X}) = \prod_{\ell=1}^{L} p(\mathbf{h}_\ell | \mathbf{h}_{\ell-1}), \tag{5}$$

where X and $\mathbf{y}$ are defined as $\mathbf{h}_0$ and $\mathbf{h}_L$, respectively. Every probability in Eq. (5) can be written as a traditional GP mapping with mean zero and covariance matrix $\mathrm{K}_{\mathbf{h}_\ell \mathbf{h}_\ell}$, i.e. $p(\mathbf{h}_\ell | \mathbf{h}_{\ell-1}) = \mathcal{N}(\mathbf{h}_\ell | \mathbf{0}, \mathrm{K}_{\mathbf{h}_\ell \mathbf{h}_\ell} + \sigma_\ell^2 \mathbf{I})$. However, inference in this *simple* stacked GP model is extremely challenging as the hidden layer variables $\mathbf{h}$ are not independent between layers nor within a layer, turning analytical marginalization practically impossible. Also, if every layer would be trained independently the DGP would scale with the usual restricting cubic complexity.

For these reasons, Hensman et al. propose to first expand each layer of the model with inducing inputs Z_ℓ and outputs U_ℓ and then define a variational distribution over the inducing outputs $q(\mathrm{U}_\ell) = \mathcal{N}(\mathrm{U}_\ell | \mathbf{m}_\ell, \mathrm{S}_\ell)$. This setup can be seen in Fig. 1 for a 4-layer deep DGP. The inducing variables enable the usage of Eq. (4) to lower bound each layer of the network. By defining a second variational distribution $q(\mathbf{h}_\ell)$, it is possible to propagate the hidden layers through the network. This feed-forward movement needs to be continued until the last layer is reached, effectively marginalizing every $\mathbf{h}_\ell$. Thus, by recursively propagating distribution $q(\mathbf{h}_\ell)$ through the $(\ell+1)^{\text{th}}$ layer, it is possible to obtain a lower bound on the marginal likelihood of an arbitrarily deep network [14]

$$\begin{aligned}
\log p(\mathbf{y}|\mathrm{X}) \geq \log \mathcal{N}\left(\mathbf{y} | \boldsymbol{\Omega}_L \mathrm{K}_{\mathbf{u}_L \mathbf{u}_L}^{-1} \mathbf{m}_L, \sigma_L^2\right) \\
- \sum_{\ell=2}^{L} \frac{1}{2\sigma_\ell^2} \operatorname{tr}\left(\mathbf{m}_\ell^\top \mathrm{K}_{\mathbf{u}_\ell \mathbf{u}_\ell}^{-1} (\boldsymbol{\Phi}_\ell - \boldsymbol{\Omega}_\ell^\top \boldsymbol{\Omega}_\ell)^\top \mathrm{K}_{\mathbf{u}_\ell \mathbf{u}_\ell}^{-1} \mathbf{m}_\ell\right) \\
- \sum_{\ell=1}^{L} \mathrm{KL}\left[q(\mathrm{U}_\ell) \| p(\mathrm{U}_\ell)\right] \\
- \sum_{\ell=2}^{L} \frac{1}{2\,\sigma_\ell^2} \left(\xi_\ell - \operatorname{tr}\left(\boldsymbol{\Phi}_\ell \mathrm{K}_{\mathbf{u}_\ell \mathbf{u}_\ell}^{-1}\right)\right) - \frac{1}{2\,\sigma_1^2} \operatorname{tr}\left(\boldsymbol{\Sigma}_1\right).
\end{aligned} \tag{6}$$

This bound is parameterized by the kernel's hyperparameters, the variational distributions and the inducing input locations Z_ℓ. The variables corresponding to the hidden layers $\mathbf{h}_\ell$ are effectively marginalized, but influence the bound through the kernel expectations:

$$\xi_\ell = \left\langle \operatorname{tr}\left(\mathrm{K}_{\mathbf{h}_\ell \mathbf{h}_\ell}\right) \right\rangle_{q(\mathbf{h}_{\ell-1})}, \tag{7}$$

$$\boldsymbol{\Omega}_\ell = \left\langle \mathrm{K}_{\mathbf{h}_\ell \mathbf{u}_\ell} \right\rangle_{q(\mathbf{h}_{\ell-1})}, \tag{8}$$

$$\boldsymbol{\Phi}_\ell = \left\langle \mathrm{K}_{\mathbf{h}_\ell \mathbf{u}_\ell}^\top \mathrm{K}_{\mathbf{h}_\ell \mathbf{u}_\ell} \right\rangle_{q(\mathbf{h}_{\ell-1})}. \tag{9}$$

The three kernel statistics are thus defined as continuous convolutions of the kernel function with respect to a Gaussian density. They can be computed analytically for the linear and SE kernel, but are for most other kernels not obtainable in closed form [15]. However, for this dissertation, the kernel statistics of the SM kernel are also derived, see Section IV-D.

A computationally important advantage of Eq. (6) is that each part can be written as a sum over every individual datapoint, enabling parallelization, mini-batch as well as stochastic training methods. Therefore, the DGP model can also be seen as an approach trying to make GPs much more computationally efficient.

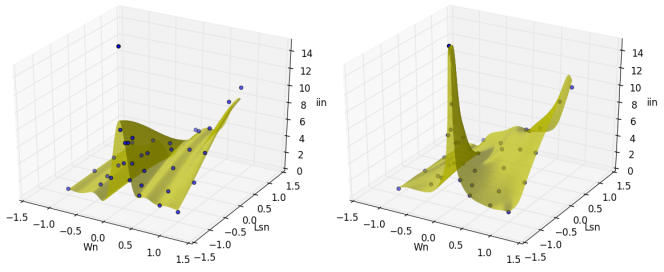
In the next section the usefulness of DGPs for metamodeling is shown.

B. DGPs in Metamodeling

Simulations are often used for the design of complex systems as they allow to explore the design space without the need to build several prototypes. However, over the years, the simulation accuracy, as well as the associated computational cost have increased significantly, limiting the overall number of simulations during the design process. Therefore, metamodeling aims to approximate the simulation response with a cheap-to-evaluate mathematical approximation, learned from a limited set of simulator evaluations (henceforward called the training dataset).

When using GPs as metamodel, the designer is inevitably confronted with selecting an appropriate kernel for the simulation at hand. Despite the fact that there are numerous available kernels, stationary kernels are most often chosen, due their ease-of-use and the limited set of hyperparameters. However, using stationary kernels for metamodeling non-stationary surfaces can be inappropriate and results in poor models. Definitely when the training dataset is constructed using sequential design methodologies.

In sequential design, rather than selecting the dataset upfront through a one-shot approach, the dataset is constructed via an iterative process, where data acquired from previous iterations



(a) GP (b) DGP

Fig. 2: The GP and DGP metamodel, optimized on the same sequential LNA dataset, consisting of 36 points. The training datapoints are the blue dots and the optimized metamodel is plotted in yellow.

is analyzed in order to select locations of new data points. A sequential algorithm will thus identify highly irregular regions of the response surface and will accordingly increase the number of samples there. In addition to preventing under/over sampling, the iterative procedure can be used to halt the construction of the metamodel once the stopping criteria are met, avoiding unnecessary simulator evaluations. However, the downside of this approach is that the resulting dataset is typically non-uniformly distributed over the input space. This can lead to poor GP metamodels as they have difficulty handling the non-uniform dataset.

This is best explained by means of an example. Fig. 2a shows a sequentially constructed dataset of the LNA[3] surface and the resulting GP metamodel. The metamodel is not able to model the high peak at $(0, 0)$ and shows fluctuating behavior in the smoother regions, which are both undesirable features. This is a result of a trade-off during the optimization. On the one hand, the lengthscales ℓ_d of the stationary kernel are still too broad causing the sample on the peak to be ignored. While on the other hand, the lengthscales are still too short in the flat regions, which causes the fluctuating behavior of the approximation as the GP is reverting back to its mean between the datapoints.

Therefore, DGPs are suggested as they provide a computationally efficient learning procedure and are significantly more flexible. In Fig. 2b the result is shown of applying a 5-layer DGP metamodel for the sequentially sampled, non-stationary LNA response surface. Note that the DGP and GP model are trained on the same dataset, but that the DGP is able to model the surface much more accurately. This is due to the composition of the multiple stacked GPs, each having their own kernel with appropriate lengthscales. Every layer of the model is able to learn a specific part of the pattern which results in a model that is able to precisely approximate the highly irregular peak, while simultaneously modeling the smoother regions.

This observation is also confirmed by Table II, which shows that the traditional GP models have indeed difficulty capturing the sequentially sampled, non-stationary LNA response surface. The Sparse Gaussian Process (SGP) and Sparse Variational Gaussian Process (SVGP), reported in the table, are two flavors of the approximating inference models, discussed in Section III-A. The DGP is undeniably outperforming the other models. It is also interesting to note that when the size of

[3]An Low Noise Amplifier (LNA) is an electronic amplifier used to amplify very weak signals (for example, captured by an antenna).

Table II: Test error metrices for the different GP models on the sequentially sampled LNA dataset.

	$N = 30$			$N = 50$		
	RRSE	RMSE	MSE	RRSE	RMSE	MSE
GP	0.38	0.84	0.70	0.73	1.63	2.65
SGP	0.38	0.84	0.70	0.73	1.62	2.63
SVGP	0.37	0.83	0.69	0.57	1.27	1.61
DGP	**0.25**	**0.55**	**0.30**	**0.19**	**0.41**	**0.17**

the sequential dataset increases ($N = 30 \rightarrow N = 50$), the performance of the GP models decreases while the performance of the DGP ameliorates.

C. Spectral Mixture Kernel

In the previous two sections the traditional GP model was enriched by considering a deep hierarchy of GPs. Another approach is to keep using the traditional model (as discussed in Section II), but combine it with a more powerful kernel function. In this light, the SM kernel is studied.

The SM kernel is, compared to the others, developed from a different perspective as it originates from its spectral density, which is the kernel's Fourier transform [16]. Resulting from Bochner's theorem [17] a stationary kernel k has a bijective relation with its spectral density S, which complies

$$k(\boldsymbol{\tau}) = \int S(\boldsymbol{s}) \exp\left(2\pi j \boldsymbol{s}^\top \boldsymbol{\tau}\right) \, d\boldsymbol{s}, \tag{10}$$

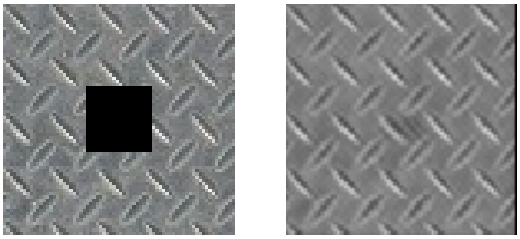
$$S(\boldsymbol{s}) = \int k(\boldsymbol{\tau}) \exp\left(-2\pi j \boldsymbol{s}^\top \boldsymbol{\tau}\right) \, d\boldsymbol{\tau}. \tag{11}$$

To see the usefulness of the SM kernel, a small detour is made. The spectral density of the popular SE kernel is calculated using Eq. (11) and given by $S_{SE}(s) = \sqrt{2\pi}\,\ell\,\exp(-2\pi^2\ell^2 s^2)$, which corresponds to a Gaussian-like density, centered around the origin. This means that in the dual (spectral) space the SE kernel only covers a small portion of the set of possible functions. Therefore, when using the SE kernel (or any linear combination of SE kernels), one has the false impression of using a powerful and broad class of kernels.

This observation motivated Wilson et al. to develop a novel kernel, one which covers a much broader set of functions in the spectral space. More particularly, using a Mixture of Gaussians (MoG) it is possible to construct a significantly broader class of spectral densities. As there is the one-on-one relation between the spectral space and the kernel space, the class of corresponding kernels will also be much broader. Computing Eq. (10) for $S(\boldsymbol{s})$ equal to a MoG with Q Gaussians, having means $\boldsymbol{\mu}_q$, covariance matrices $\boldsymbol{\Lambda}_q$ and relative contributions w_q, yields to the definition of the SM kernel [18]

$$k_{\text{SM}}(\boldsymbol{\tau}) = \sum_{q=1}^{Q} w_q \cos\left(2\pi \boldsymbol{\tau}^\top \boldsymbol{\mu_q}\right) \exp\left(-2\pi^2 \boldsymbol{\tau}^\top \boldsymbol{\Lambda}_q \boldsymbol{\tau}\right).$$

The expressiveness of the SM kernel is best illustrated using Fig. 3. The left plot shows a tread plate, where the center 20×20 pixels are missing. In this experiment, a GP model with a SM kernel is trained on the visible surrounding part. The inputs of the model are 2D locations $\mathbf{x} \in \mathbb{N}^2$ and the targets are pixel intensities $y \in \mathbb{R}$. As shown in the right plot, the objective is to extrapolate the pattern in the missing region. It is clearly visible that the SM kernel is able to learn the particular pattern



(a) Training region (b) Prediction

Fig. 3: Tread plate extrapolation with GP, using a SM kernel.

and extrapolate in the missing center accordingly. If the same experiment is repeated with a SE kernel it simply predicts a completely flat region.

Another experiment, with the SM kernel as protagonist, is conducted in Section IV-E. To be able to fully understand it, a bit more theory is first required regarding predictions under uncertainty.

D. Uncertain Predictions with Gaussian Processes

The predictive distribution of a GP for a deterministic input $\mathbf{x}_*$ is given by Eqs. (1) and (2). However, in many real-world scenarios test inputs $\mathbf{x}_*$ are not deterministic but rather uncertain. If such an uncertain test input is considered to be Gaussian, i.e. $\mathbf{x}_* \sim \mathcal{N}(\mathbf{x}_*|\mathbf{w}, \mathrm{S})$, the predictive distribution becomes the expectation of the deterministic predictive distribution with respect to the distribution of $\mathbf{x}_*$

$$p(\mathrm{f}(\mathbf{x}_*)|\mathbf{w}, \mathrm{S}) = \int \mathcal{N}\big(\mathrm{f}(\mathbf{x}_*)|\mu(\mathbf{x}_*), \sigma^2(\mathbf{x}_*)\big)\, \mathcal{N}(\mathbf{x}_*|\mathbf{w}, \mathrm{S})\, d\mathbf{x}_*$$

Obtaining a closed form for this equation is analytically not possible. Nonetheless, using the law of iterated expectations and conditional variance, its mean and variance can be derived [19]

$$\begin{aligned} m(\mathbf{w}, \mathrm{S}) &= \mathbb{E}_{\mathbf{x}_*}\left[\mu(\mathbf{x}_*)\right] \\ &= \boldsymbol{\Omega}^\top \boldsymbol{\beta}, \\ v(\mathbf{w}, \mathrm{S}) &= \mathbb{E}_{\mathbf{x}_*}\left[\sigma^2(\mathbf{x}_*)\right] + \mathbb{E}_{\mathbf{x}_*}\left[\mu^2(\mathbf{x}_*)\right] - \mathbb{E}^2_{\mathbf{x}_*}\left[\mu(\mathbf{x}_*)\right] \\ &= \xi - \mathrm{tr}\left(\left(\tilde{\mathrm{K}}_{\mathbf{ff}}^{-1} - \boldsymbol{\beta}\boldsymbol{\beta}^\top\right)\boldsymbol{\Phi}\right) - \mathrm{tr}\left(\boldsymbol{\Omega}\boldsymbol{\Omega}^\top\boldsymbol{\beta}\boldsymbol{\beta}^\top\right), \end{aligned}$$

where $\boldsymbol{\beta} \triangleq \tilde{\mathrm{K}}_{\mathbf{ff}}^{-1}\mathbf{y}$ and ξ, $\boldsymbol{\Omega}$ and $\boldsymbol{\Phi}$ are the kernel expectation, defined in Eqs. (7), (8) and (9), over $\mathcal{N}(\mathbf{x}_*|\mathbf{w}, \mathrm{S})$ instead of $q(\mathbf{h}_{\ell-1})$. Using the above equations, the predictive distribution for an uncertain input $\mathbf{x}_*$ can now be approximated by a Gaussian with mean $m(\mathbf{w}, \mathrm{S})$ and variance $v(\mathbf{w}, \mathrm{S})$.

Consequently, if one wishes to predict the outcome of an uncertain input using a GP with a SM kernel, the $\{\xi, \boldsymbol{\Omega}, \boldsymbol{\Phi}\}$-statistics of the SM kernel are required. Therefore, as an objective of this dissertation are the SM kernel expectations derived, implemented in TensorFlow[4] and verified on real-world experiments. One of these experiments is discussed in the next section.

[4] https://github.com/vdutor/GPflow/blob/sm-kernel/GPflow/ekernels.py

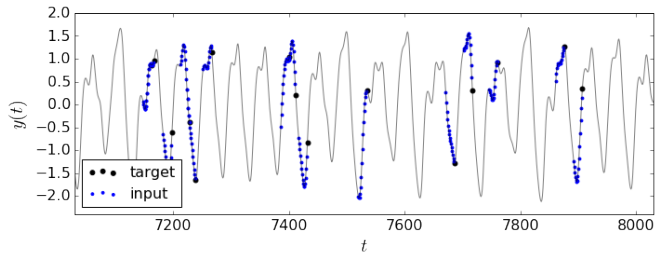
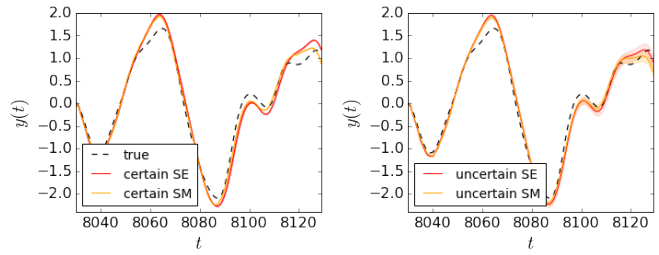


Fig. 4: Extract from the MacKey-Glass time series.



(a) Naive approach (b) Uncertainty propagation

Fig. 5: 100-step-ahead prediction of the MacKey-Glass series.

E. Forecasting the Mackey-Glass time series

The Mackey-Glass chaotic time series is a continuous differential equation [20]: $\frac{\mathrm{d}\,z(t)}{\mathrm{d}\,t} = \frac{a\,z(t-\tau)}{1+z(t-\tau)^{10}} - b\,z(t)$, where $a = 0.2$, $b = 0.1$ and $\tau = 17$. For this experiment, it is numerically solved and then sampled 8030 times with frequency 1 to obtain an output vector $\mathbf{z} \in \mathbb{R}^{8030}$. This 1D output vector is then transformed into a state-space model, where the state at time t equals $\mathbf{x}_t = [\mathrm{z}_{t-1}, \mathrm{z}_{t-2}, \ldots, \mathrm{z}_{t-16}]^\top$ and the corresponding target $\mathrm{y}_t = \mathrm{z}_t$. A GP is trained, on 100 randomly picked samples from the 8030, to model the transition function $\mathrm{f}(\cdot)$ between two consecutive states, i.e. $\mathrm{y}_t = \mathrm{f}(\mathbf{x}_t)$. In Fig. 4 this setup is illustrated: the black dots are the targets, picked at random timestamps between 0 and 8030 and the blue dots, preceding a black dot, are the corresponding 16D inputs.

The objective is to extrapolate this time series 100-steps ahead, starting from 8030, to obtain $\mathrm{y}_{T+1}, \ldots, \mathrm{y}_{T+100}$, with $T = 8030$. To achieve this, two cases are considered: a *naive* and a more *intelligent* approach. Both assume that an optimized GP is already constructed for $\mathrm{f}(\cdot)$.

The simplest, but most naive approach, starts with predicting y_{T+1} with the standard predictive equations to obtain $\mathcal{N}(\mu_{T+1}, \sigma^2_{T+1})$. It will then ignore the variance over this prediction and only use the mean to perform the next prediction of y_{T+2}. Similarly, will it neglect the variance over y_{T+2} and use only its predicted mean to obtain y_{T+3} and so on, up to y_{T+100}. Hence, this simple approach uses only the output estimates and does not take into account the uncertainty induced by each successive prediction.

This approach is followed to obtain Fig. 5a, once for a GP using a SM kernel (yellow) and once for a GP using a SE kernel (red). The uncertainty over the predictions, i.e. $\pm 2\sigma$, is also plotted. However, as σ is almost 0 these uncertainty regions are barely visible. This is also the most important problem with the naive approach: it consistently throws away the uncertainty

over its inputs which leads to severely underestimating the variance over its predictions.

However, when using the results derived in the previous section, a more intelligent approach can be designed. That is, using the uncertain input prediction framework [19], a model can account for the uncertainty on y_{T+1} to calculate the predictive distribution of y_{T+2}. Likewise can it incorporate the uncertainty over y_{T+2} to obtain the predictive distribution of y_{T+3}, and so on until the horizon is reached. Hence, in the intelligent approach, an input is now considered as an uncertain variable, with a mean formed by the predicted means and a covariance matrix holding the predicted variances.

Fig. 5b shows the obtained extrapolation when using this latter approach. As in Fig. 5a the experiment is conducted for a GP model with a SE (red) and a SM (yellow) kernel. It can be observed that the predicted means are relatively similar. However, in the case of uncertainty propagation (b) the variance over the predictions increases with k and the true curve is more often within the uncertainty interval $\pm 2\sigma$, which is definitely more realistic. When assessing the predictions numerically (using MSE and RRSE) the lowest error is found for the intelligent approach in combination with the SM kernel, which highlights the usefulness of the SM kernel expectation derivation.

V. Conclusion

Even when taking recent advances into consideration, GPs are still not fully applicable for modeling big data. In this work multiple approaches are suggested to remedy this.

Firstly, as every GP software-framework requires the Cholesky decomposition for the inversion of a positive-definite matrix, it is important to have a fast implementation available. Therefore, an algorithm was developed and implemented with as goal to significantly speed up the required computation. When integrating the developed implementation in a standard GP it achieves a speedup of factor 3 on large datasets.

Further, DGPs were studied as they address both problems of standard GPs. They are computational much more efficient while simultaneously being more flexible. In this work, their usefulness was shown in the context of metamodeling non-stationary surfaces, where they outperformed the other traditional GP models. Extensive work was done to improve current initialization and training procedures.

Finally, the usage of the SM kernel, instead of the popular SE, is advocated through different experiments. However, more importantly, the kernel expectations of the SM kernel were derived in closed analytical form. To the best of our knowledge this was not available in literature. Having the kernel expectations allows one to use the SM kernel for tackling real-world problems with more advanced and efficient GP models, like input uncertainty propagation.

References

[1] R. M. Neal. *Bayesian Learning for Neural Networks*. Ph.D. thesis, Dept. of Computer Science, University of Toronto, 1994.

[2] Carl Edward Rasmussen. *Evaluation of Gaussian processes and other methods for non-linear regression*. PhD thesis, Citeseer, 1996.

[3] David JC MacKay. Gaussian processes-a replacement for supervised neural networks? 1997.

[4] Rasmussen and Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[5] Neil Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. In *In NIPS*, page 2004, 2003.

[6] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

[7] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[8] Matthias Seeger, Christopher K. I. Williams, and Neil D. Lawrence. Fast forward selection to speed up sparse Gaussian process regression. In *workshop on AI and Statistics 09*, 2003.

[9] Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. In Y. Weiss, B. Schölkopf, and J. C. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 1257–1264. MIT Press, 2006.

[10] Michalis K. Titsias. Variational learning of inducing variables in sparse Gaussian processes. In *In Artificial Intelligence and Statistics 12*, pages 567–574, 2009.

[11] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. visited on 06/04/2017, 2016, 2016.

[12] Alexander G. G. MatthewsG. Matthews, Mark van der Wilk, Tom Nickson, Keisuke Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. Gpflow: A gaussian process library using tensorflow. *Journal of Machine Learning Research*, 18(40):1–6, 2017.

[13] Andreas C. Damianou and Neil D. Lawrence. Deep Gaussian processes. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2013, Scottsdale, AZ, USA, April 29 - May 1, 2013*, pages 207–215, 2013.

[14] James Hensman and Neil D. Lawrence. Nested variational compression in deep Gaussian processes. visited on 12/03/2016, 2014, 2014.

[15] Andreas Damianou. *Deep Gaussian Processes and Variational Propagation of Uncertainty*. PhD thesis, University of Sheffield, 2015.

[16] C. Chatfield. *Time Series Analysis: An Introduction*. Chapman and Hall, 1989.

[17] Salomon Bochner. *Lectures on Fourier Integrals.(AM-42)*, volume 42. Princeton University Press, 1959.

[18] Andrew Gordon Wilson and Ryan Prescott Adams. Gaussian process kernels for pattern discovery and extrapolation. In *ICML*, pages 1067–1075, 2013.

[19] Agathe Girard, Carl Edward Rasmussen, Joaquin Quinonero Candela, and Roderick Murray-Smith. Gaussian process priors with uncertain inputs-application to multiple-step ahead time series forecasting. *Advances in neural information processing systems*, pages 545–552, 2003.

[20] Michael C Mackey, Leon Glass, et al. Oscillation and chaos in physiological control systems. *Science*, 197(4300):287–289, 1977.

Contents

1 Introduction **1**
1.1 Probabilistic Machine Learning . . . 1
1.2 The Gaussian Distribution . . . 5
1.3 Outline . . . 6

2 Gaussian Processes **7**
2.1 Background . . . 7
2.2 Preliminary . . . 8
2.3 Gaussian Process Regression . . . 9
2.3.1 Prior and Hyperparameter Optimization . . . 9
2.3.2 Inference . . . 10
2.4 Gaussian Process Classification . . . 13
2.4.1 Inference and Decision Theory . . . 13
2.4.2 Generative vs. Discriminative . . . 15
2.4.3 Gaussian Process Classification . . . 15
2.5 Covariance Functions . . . 18
2.5.1 The Squared Exponential covariance . . . 19
2.5.2 The Matérn class covariance . . . 20
2.5.3 The Spectral Mixture covariance . . . 21
2.5.4 Other non-stationary covariance functions . . . 21

3 Efficient Gaussian Processes **23**
3.1 Approximations . . . 24
3.1.1 Sparse Pseudo-input Gaussian Process . . . 25
3.1.2 Projected Process Approximation . . . 28
3.1.3 Sparse Variational Gaussian Process . . . 29
3.1.4 Experiments . . . 33
3.2 Cholesky decomposition . . . 35
3.2.1 Background . . . 35
3.2.2 The Blocked-Partitioned Cholesky Algorithm . . . 37
3.2.3 Performance . . . 39

4 Finding Uncertain Patterns with Gaussian Processes **42**
4.1 Uncertain Input Propagation . . . 42
4.1.1 Background . . . 42
4.1.2 Prediction at an Uncertain Input . . . 43
4.2 Gaussian Process Manifold Learning . . . 45
4.2.1 (Probabilistic) Principal Component Analysis . . . 46
4.2.2 GP Latent Variable Model . . . 47
4.2.3 Bayesian GP-LVM . . . 48
4.3 Deep Gaussian Processes . . . 49
4.4 Kernel Expectations . . . 54

5 Metamodeling of Sequentially Sampled Non-Stationary Surfaces **55**
5.1 Metamodeling . . . 56
5.2 Sequential Design . . . 57
5.3 Gaussian Processes for Metamodeling . . . 58
5.4 Experimental Setup . . . 61
5.5 Results . . . 62
5.5.1 The Langley Glide-Back Booster Simulation . . . 62
5.5.2 Low Noise Amplifier . . . 64
5.6 Conclusion . . . 65

6 Spectral Mixture Kernels **67**
6.1 Derivation . . . 68
6.2 Experiments . . . 71
6.2.1 Airline data . . . 72
6.2.2 Tread plate . . . 73
6.3 Kernel Expectations of the SM kernel . . . 74
6.3.1 Derivation . . . 75
6.4 Experiments . . . 79
6.4.1 Visualization Task . . . 79
6.4.2 State-Space Model Extrapolation . . . 81

7 Conclusion **85**

A Mathematical Identities **93**
A.1 Matrix Identities . . . 93
A.2 Gaussian Identities . . . 94
A.3 KL Divergence . . . 94

B Proof Unblocked Cholesky Algorithm **95**

List of Figures

1.1 Bayesian Occam's razor. . . . 3
1.2 The PDF of the 1D and 2D Gaussian distribution. . . . 5

2.1 Draws from a GP prior. . . . 10
2.2 Illustration of the GP conditional predictive distribution. . . . 12
2.3 The process of squeezing a latent function f through a sigmoid function. . . . 16
2.4 GP prior samples from different kernels. . . . 20

3.1 Comparison of the full GP, SPGP and PP sparse approximations. . . . 33
3.2 Test metrices of the SPGP, PP and SVGP on SARCOS dataset 34
3.3 GPU-CPU execution scheduling during GP optimization. . . . 37
3.4 Execution time of the Cholesky implementations. . . . 41
3.5 Execution time of the Cholesky gradient implementations. . . . 41
3.6 Execution time of 5 optimization rounds of the GPR model. . . . 41

4.1 Graphical models GP-based manifold learning. . . . 47
4.2 Graphical model of a generic Deep Gaussian Process. . . . 51

5.1 The sequential design metamodeling paradigm. . . . 58
5.2 Impact of different sampling strategies on the GP metamodel of the LNA surface. 60
5.3 A GP and DGP metamodel, optimized on the same sequential LNA dataset. . . . 61
5.4 A GP and DGP metamodel of the LGBB surface. . . . 63
5.5 Summarizing figure LNA metamodeling experiment. . . . 66

6.1 Initialization of the SM kernel illustrated on a synthetic dataset. . . . 70
6.2 Dominant frequency extraction using Fast Fourier Transform. . . . 71
6.3 Forecasting airline passengers based on observations between 1949 and 1964. . . . 72
6.4 Extrapolation of a metal treat plate image. . . . 73
6.5 Boxplot of the squared error of the SM kernel expectations implementation. . . . 79
6.6 Visualization three phase oil dataset. . . . 80
6.7 PCA and Bayesian GP-LVM reduction on the three phase oil dataset. . . . 80
6.8 Extract from the MacKey-Glass time series. . . . 82
6.9 100-steps ahead predictions of the MacKey-Glass time series. . . . 82

List of Tables

3.1 Optimized hyperparameters on sparse toy example. 33
3.2 Comparison between the SPGP, PP and SVGP approximations on SARCOS. . . 34

5.1 Test error metrices of the GP metamodels on the LGBB dataset. 64
5.2 Test error metrices of the GP metamodels on the sequentially sampled LNA dataset. 65

6.1 Test metrices for 100-steps ahead predictions of the MacKey-Glass time series. . 84

List of Abbreviations

ANN	Artificial Neural Network
ARD	Automatic Relevance Determination
BC GP-LVM	Back Constraint GP-LVM
BEEQ	Bayesian Estimation Error Quotient
BGP-LVM	Bayesian GP-LVM
DGP	Deep Gaussian Process
DKL	Deep Kernel Learning
DoF	Degree of Freedom
DTC	Deterministic Training Conditional
ELBO	Evidence Lower Bound
EP	Expectation Propagation
EQ	Exponentiated Quadratic
FFT	Fast Fourier Transform
FITC	Fully Independent Training Conditional
GAN	Generative Adversarial Network
GP	Gaussian Process
GP-LVM	GP Latent Variable Model
GP-SSM	GP State-Space Model
GPC	Gaussian Process Classification
GPR	Gaussian Process Regression
KL	Kullback-Leibler
LS-SVM	Least Squares - Support Vector Machine
MAP	Maximum A Posteriori
MCMC	Monte Carlo Markov Chain
MoG	Mixture of Gaussians
MSE	Mean Square Error
NN	Neural Network
NNE	Nearest Neighbor Error
PCA	Principal Component Analysis

PDF	Probability Density Function
PP	Projected Process
PPCA	Probabilistic Principal Component Analysis
RBF	Radial Basis Function
RL	Reinforcement learning
RMSE	Root Mean Square Error
RRSE	Root Relative Square Error
SD	Subset of Data
SE	Squared Exponential
SGP	Sparse Gaussian Process
SM	Spectral Mixture
SPGP	Sparse Pseudo-input Gaussian Process
SVGP	Sparse Variational Gaussian Process
VI	Variational Inference

Notation

Scalars, vectors and matrices

x	scalars are usually represented by a lower-case character
$\boldsymbol{x}$	vectors are usually represented by a bold lower-case character
$\boldsymbol{X}$	vectors are usually represented by a bold upper-case character

Common terms

$\boldsymbol{x}$	a D-dimensional column vector, usually the input
$\boldsymbol{X}$	a collection of N input variables $\boldsymbol{x}$, arranged into a $N \times D$ matrix
$\boldsymbol{y}$	a N-dimensional column vector of observations
$\boldsymbol{Z}$	a collection of M inducing input variables, arranged into a $M \times D$ matrix
$\boldsymbol{u}$	a M-dimensional column vector of inducing outputs
$\boldsymbol{\theta}$	vector holding the hyperparameters
$f(\cdot)$	unknown (latent) function
$k(\cdot)$	kernel or covariance function

Probability

$p(\boldsymbol{x})$	the probability density function (PDF) on $\boldsymbol{x}$
$q(\boldsymbol{x})$	usually implies an approximate (variational) probability density function on $\boldsymbol{x}$
$\mathbb{E}_{\boldsymbol{x}}[g(\boldsymbol{x})]$	the expectation of $g(\boldsymbol{x})$ over the distribution of $\boldsymbol{x}$
$\langle g(\boldsymbol{x}) \rangle_{p(\boldsymbol{x})}$	the expectation of $g(\boldsymbol{x})$ over the distribution of $\boldsymbol{x}$
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	the Gaussian (normal) distribution, with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

Chapter 1

Introduction

1.1 Probabilistic Machine Learning

We live in an era of abundant data. From all kinds of different angles and for all kinds of different reasons, mankind is generating and storing vast amounts of data. This data comes from large-scale scientific experiments, digital messages, pictures, and videos from social networks and businesses such as e-commerce, electronic trading and online advertising. Cisco, a leading company in networking and IT, calculated that in 2016 the global IP traffic payload reached 1.1 ZB, which roughly corresponds to 88.7 EB per month (Cisco 2016). This amount is not expected to stay constant in the years to come. By 2020 Cisco predicts that the global IP traffic will reach 2.3 ZB per year. Emerging new intelligent technologies such as personal assistants, self-driving cars or autonomous decision-making systems are mainly accountable for this. They are connected with each other and with the backbone of the internet in order to perform a variety of daily tasks, which always require some kind of transfer of information. Self-driving cars, for example, will be able to pay for gas or parking over cellular networks and will require constantly updated maps to perform lane control or avoid road hazards. We observe that these intelligent systems have one particular characteristic in common: they adapt to their environment and learn from it by constantly collecting and studying data.

This observation leads to the evident conclusion that we require tools for the modeling, the visualization and the understanding of extremely large datasets (millions of datapoints). If we wish to deploy these systems in real-life scenario's, the tools need to be resilient to noise and uncertainty, exhibit robustness and adaptiveness and scale well to large datasets. Probabilistic modeling is an approach towards this, as it provides a probabilistic framework for the manipulation and the representation of data. During the last decades, it has emerged as one of the fundamental building blocks in the design of adaptive and versatile intelligent systems (Ghahramani 2015). The framework, which tries to understand what learning is, plays a crucial role in artificial intelligence (AI), machine learning (ML), robotics, cognitive science and data-analysis.

The main rationale behind probabilistic modeling for machine learning is that learning is possible through inferring plausible models that explain the observed data. These models can then be used to predict certain outcomes in the future or to make sensible decisions. Within this framework, uncertainty has a central role, as multiple predictions about future data can be simultaneously plausible. Similarly, observed data can be consistent with many models. Hence, the *correct* model, given the observed data, is uncertain. The tool used to express all these forms of uncertainty comes from the mathematical domain of probability theory.

Probability theory is relatively simple, there are two basic rules that allow us to perform a range of manipulations such as inferring unobserved quantities or quantifying the likeliness of a certain model. There is the sum rule (also referred to as the marginalization rule)

$$p(x) = \sum_{y \in Y} p(x, y),$$

and the product rule

$$p(x, y) = p(x)\, p(y|x),$$

where x and y are elements of discrete sets X and Y, respectively. The rules in the continuous case are analogous. These probabilities are best explained by means of a simple example. Consider x to be the outcome of rolling a fair dice and y to describe whether the player won or lost the game. Then $X = \{1, 2, 3, 4, 5, 6\}$ and $Y = \{0, 1\}$. $p(x)$ corresponds to the probability of x, which is according to the frequentist interpretation defined as the limit of its relative number of occurrences in a large number of trials. In this example $p(x) = 1/6, \forall x \in X$. $p(x, y)$ is the probability of jointly observing events x and y, and the conditional probability $p(x|y)$, is the probability of observing event x given y.

Bayes' theorem is found by combing both rules

$$p(y|x) = \frac{p(y)\, p(x|y)}{p(x)} = \frac{p(y)\, p(x|y)}{\sum_{y \in Y} p(y)\, p(x|y)}.$$

Bayes' theorem is about incorporating prior knowledge that might be related to the events. For example, if rolling a particular number with the dice is related to winning the game, Bayes' rule can be used to more accurately assess whether a player has won knowing the outcome of roll, compared to the assessment of winning made without this knowledge. It can also be seen as an equation that allows new evidence to update beliefs.

The usage of Bayes' theorem in probabilistic machine learning becomes clear after renaming the above variables. Let $\mathcal{D}$ replace x and stand for the observed data and let $\boldsymbol{\theta}$ replace y and denote the latent parameters. Furthermore we condition on the class of probability models $\mathcal{M}$ we are considering. Then, we get

$$p(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M}) = \frac{p(\boldsymbol{\theta}|\mathcal{M})\, p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{M})}{p(\mathcal{D}|\mathcal{M})}, \tag{1.1}$$

where $p(\boldsymbol{\theta}|\mathcal{M})$ is the prior distribution, $p(\mathcal{D}|\boldsymbol{\theta}, \mathcal{M})$ is the likelihood, $p(\mathcal{D}|\mathcal{M})$ is the marginal likelihood and $p(\boldsymbol{\theta}|\mathcal{D}, \mathcal{M})$ is the posterior distribution. Eq. 1.1 is used for learning and inferring the unknown, latent parameters. However, in a Bayesian setting it is not assumed that there

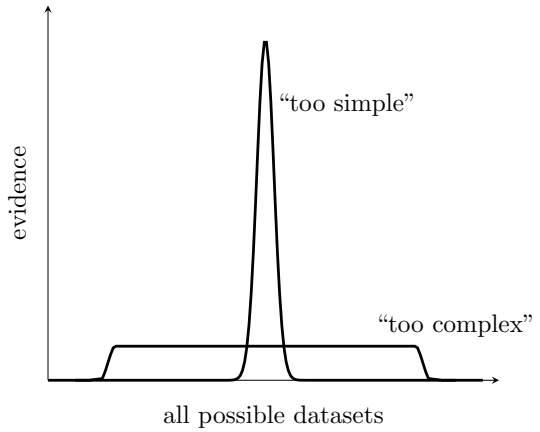


Figure 1.1: Bayesian Occam's razor.

exist an unique and optimal set of parameters $\boldsymbol{\theta}_{\text{opt}}$ that model the observed data perfectly (this approach corresponds to a Maximum A Posteriori (MAP) estimation). Bayesian machine learning rather advocates to use the posterior probability to average over all possible parameter values to make a prediction for future data $\mathcal{D}_*$. By straightforwardly applying the product and sum rule, we find

$$p(\mathcal{D}_*|\mathcal{D},\mathcal{M}) = \int p(\boldsymbol{\theta}|\mathcal{D},\mathcal{M})p(\mathcal{D}_*|\boldsymbol{\theta},\mathcal{M})\,d\boldsymbol{\theta}. \tag{1.2}$$

Different probabilistic model classes are evaluated using Bayes' rule on $\mathcal{M}$, conditioned on $\mathcal{D}$

$$p(\mathcal{M}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathcal{M})p(\mathcal{M})}{p(\mathcal{D})},$$

where $p(\mathcal{D}|\mathcal{M}) = \int p(\mathcal{D}|\boldsymbol{\theta},\mathcal{M})p(\boldsymbol{\theta}|\mathcal{M})\,d\boldsymbol{\theta}$ is considered to be the model evidence and can be interpreted as the probability that parameters, randomly selected from the prior, are able to generate the dataset. The evidence introduces a preference for simple model classes and penalizes unnecessarily complex ones. This effect, illustrated in Figure 1.1, is known as the Bayesian version of Occam's razor (MacKay 1991). A particular model that is able to generate many datasets (i.e. the evidence is relatively large for many $\mathcal{D}$) is highly unlikely to generate the one considered now. Those classes are too complex and are prone to overfitting. On the contrary, probabilistic model classes that are too simple will be able to generate only a small number of datasets and are therefore unlikely to generate the dataset at hand.

With this, all the fundamentals of probabilistic machine learning are covered. It is quite remarkable that this is possible in only a single page. The ground rules are short and clear *but* very powerful considering all the models and methods you can derive from it. To paraphrase Zoubin Ghahramani, professor of Information Engineering at the University of Cambridge and a prominent promoter of Bayesian machine learning,

> *"Actually, that's it!"*

With this catchphrase, which he often uses to present his research to a new audience, he expresses that the ideas, that underlie Bayesian machine learning, are not numerous and are conceptually

simple. Building upon those key concepts is what Bayesian machine learning practitioners have successfully been doing over the last three decades. It is also the main focus of this thesis. Starting from Chapter 2 we thoroughly cover Gaussian Processes (GPs) and numerous variants and show how they perfectly fit into this probabilistic framework. We identify their limitations and provide suitable solutions in order to remedy them. In the later chapters we use GPs and GP-based extensions to model real-world problems. However, before diving into this, we start with highlighting some interesting properties of Bayesian machine learning to emphasize its importance and this thesis' relevance. The final section of this introduction is devoted to the Gaussian distribution as it is a fundamental building block of GPs and Bayesian models in general.

Bayesian machine learning, although being conceptually straightforward, is extremely versatile and has several appealing properties. The prior in the framework $p(\boldsymbol{\theta})$ can be chosen in two different ways. On the one hand it can be objective, which means that it is not particularly informative but has good frequentist properties. A typical objective prior is a Gaussian distribution due to its physical interpretation, wide applicability and analytical tractability (see Section 1.2). On the other hand, subjective priors can be used to capture beliefs and add domain knowledge. In this thesis the former will typically be used, as we strive to develop black-box methods. The goal of developing black-box methods is to create general purpose methods that have good performance regardless of the problem. Therefore, we are not able to incorporate expert domain knowledge as we do not know the problem a priori.

The uncertainty, provided by Eq. 1.2, over predictions is of immense importance for a lot of applications, as it quantifies the model's confidence of a prediction made. In Chapter 5, for example, we use this to intelligently compose better datasets by trying to reduce the overall model's uncertainty over the input space. Another example is to use the uncertainty to reject certain predictions when the confidence of the model is too low. It is clear that such a feature is of great value for autonomous intelligent machines, such as self-driving cars. In the context of an autonomous car, it would be foolish to perform a lane switch when the model is, for example, only 50% certain about the safe outcome of it. Another advantage of Bayesian machine learning is that it is possible to combine many small and simple models in order to compose a larger and complexer structure. This is often realized through the use of graphical models, which can be seen as a marriage between graph and probability theory. Graphical models will mainly be used in Chapter 4 to hierarchically extend the GP model into a Deep Gaussian Process (DGP). Markov model theory and Bayesian networks are also very related to these subjects.

Although being conceptually very simple, the fully probabilistic approach followed in Bayesian machine learning results in a number of computational challenges. The integrals encountered in the framework are often over continuous and high-dimensional spaces which means that they are extremely hard to compute. For most, it is even impossible to calculate them in closed analytical form. This has led to the development of approximate integration methods that are used during learning and inference. Well-established methods are Expectation Propagation (EP), Monte Carlo Markov Chain (MCMC), moment matching and variational Bayes. In fact, the development of these approximating techniques is currently the most active research domain in Bayesian ML. Chapter 3 of this dissertation focuses on these computational challenges.

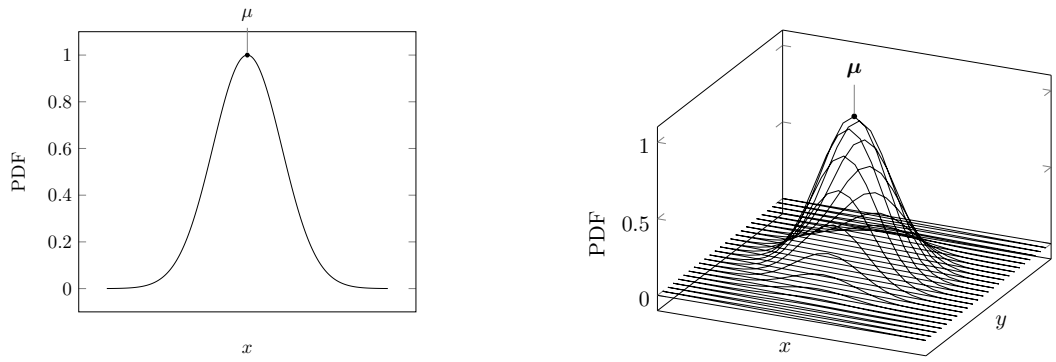


Figure 1.2: The PDF of the 1D and 2D Gaussian distribution.

1.2 The Gaussian Distribution

Gaussian Processes have -not surprisingly- a lot in common with Gaussian (multivariate) distributions. A multivariate Gaussian distribution is a distribution over vectors, whereas a Gaussian Process is a distribution over functions. While a Gaussian distribution is governed by a mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, a GP is fully specified by a mean function $f(\boldsymbol{x})$ and a covariance function $k(\boldsymbol{x}, \boldsymbol{x}')$. Chapter 2 is devoted completely to GPs. However, it is convenient to already introduce the Gaussian distribution here, as it is one of the most important continuous probability distributions and will extensively be used throughout this thesis.

The Gaussian distribution, also known as the normal distribution, of a D-dimensional real-valued vector $\boldsymbol{x}$ is specified by a mean vector $\boldsymbol{\mu}$ and a symmetric, positive-definite covariance matrix $\boldsymbol{\Sigma}$. The Probability Density Function (PDF) has the following analytical form

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = (2\pi)^{-D/2}|\boldsymbol{\Sigma}|^{-1/2} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right)$$
$$\mathbb{E}[\boldsymbol{x}] = \boldsymbol{\mu}$$
$$\text{cov}[\boldsymbol{x}] = \boldsymbol{\Sigma}$$

The inverse covariance matrix $\boldsymbol{\Sigma}^{-1}$, commonly referred to as the precision matrix, is also symmetric and positive-definite. In Figure 1.2 the PDF of a 1D and 2D Gaussian is drawn in the left and right plot, respectively. Note that the mean $\boldsymbol{\mu}$ is also the mode of the distribution. The width of the peak is characterized by the variance (1D case) or the covariance matrix (multi-dimensional case).

The central limit theorem is one of the reasons why Gaussian distributions are so widely used. It states that when independent random variables are added, their sum tends toward a normal distribution, even if the original variables themselves are not normally distributed. This motivates the use of a Gaussian distribution as a universally applicable, objective prior as we commonly expect that events or variables are a combination of multiple factors. Furthermore, Gaussian distributions have a lot of convenient properties, which makes analytical closed-form calculations possible. In the next paragraph we cover the most important one and in Appendix A.2 the others can be found.

Property 1. The marginal distribution of a multivariate Gaussian w.r.t. a subset of its variables is again a Gaussian distribution. To see this, consider the joint Gaussian probability distribution $p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$, where we define the following partitions

$$\boldsymbol{x} = \begin{pmatrix} \boldsymbol{x}_a \\ \boldsymbol{x}_b \end{pmatrix}, \quad \boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{pmatrix}, \quad \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{aa} & \boldsymbol{\Sigma}_{ab} \\ \boldsymbol{\Sigma}_{ab}^\top & \boldsymbol{\Sigma}_{bb} \end{pmatrix},$$

then the conditional distribution $p(\boldsymbol{x}_a|\boldsymbol{x}_b)$ is given by

$$p(\boldsymbol{x}_a|\boldsymbol{x}_b) = \mathcal{N}(\boldsymbol{x}_a|\boldsymbol{\mu}_{a|b}, \boldsymbol{\Sigma}_{a|b}), \tag{1.3}$$

where $\boldsymbol{\mu}_{a|b} = \boldsymbol{\mu}_a + \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}(\boldsymbol{x}_b - \boldsymbol{\mu}_b)$ and $\boldsymbol{\Sigma}_{a|b} = \boldsymbol{\Sigma}_{aa} - \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}\boldsymbol{\Sigma}_{ab}^\top$.

1.3 Outline

Chapter 2 gives an overview of the GP model, including hyperparameter optimization, GPs for regression, GPs for classification and covariance functions. It also covers briefly some key concepts in probabilistic machine learning. Readers familiar with GP modeling can safely skip this chapter.

One of the main problems with applying GPs successfully in machine learning is the computational difficulties for large datasets. Chapter 3 addresses this problem from two different angles. Firstly, the chapter gives a review of sparse (variational) approximation techniques, which reduce the training and prediction costs. There is special attention to explain the models mathematically in-depth and in an unifying framework. Secondly, this chapter presents a new Cholesky implementation, which is the most expensive operation, in order to reduce the computational bottleneck.

Chapter 4 discusses, from a theoretic perspective, three advances GP-based models. These models will be used in real-world modeling problems in the last two chapters. Particularly, the chapter deals with performing predictions of uncertain inputs, unsupervised GP-based dimensionality reduction using Bayesian GP Latent Variable Models (GP-LVMs) and DGPs, a hierarchical stacking of multiple GPs.

Traditional kernel based metamodeling approaches have difficulties with modeling non-stationary surfaces when the training dataset is non-uniformly distributed over the input space. For this reason, Chapter 5 proposes the use of DGPs for the metamodeling of non-stationary response surfaces.

Chapter 6 focuses on the Spectral Mixture (SM) kernel, which is a covariance function able to extrapolate data instead of the usual GP interpolation. This chapter's main contribution is the derivation of the SM kernel expectations, which are three integrals necessary to implement the models discussed in Chapter 4. The chapter highlights the usefulness of the SM kernel expectations through two experiments.

Chapter 7 summarizes the key contributions of the thesis and discusses ideas for future work.

Chapter 2

Gaussian Processes

2.1 Background

Gaussian Processes (GPs) became widely used as a non-parametric model after Radford Neal reintroduced them to the probabilistic machine learning community in the mid-nineties. In his work he showed that a one hidden layer Neural Network (NN), given suitable priors for the weights, converges towards a GP with a particular kernel as the number of hidden neurons tends to infinity (Neal 1994). This led researchers, like Carl Rasmussen, to carry out a series of experiments in which they compared GPs with other frequently used models, such as NNs and regression splines, and showed that GPs outperform them (Rasmussen 1996). However, in 1997 David MacKay questioned: *"Are Gaussian Processes a replacement for supervised neural networks? Or have we thrown the baby out with the bath water?"* (MacKay 1997). MacKay's rhetorical question illustrated perfectly the frustration researchers had grown about GPs being nothing more than intelligent smoothing devices, unable to perform difficult tasks, such as feature discovery for example. However, in his work he counter-argues that we do not have to underestimate the power of smoothing methods and that they can be a valuable alternative to the hyped neural networks, if used correctly. Since then, numerous GP-based models have been successfully developed and have shown excellence in a range of different fields, including supervised learning, dimensionality reduction and reinforcement learning. The most important ones for supervised learning and dimensionality reduction will be covered throughout this thesis.

Although the GP formalism is applicable to almost all machine learning tasks, they are mostly used for regression problems. In a typical regression task, there is a dataset consisting of input vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$ and corresponding continuous outputs $y_1, y_2, \ldots, y_N$. Generally, we assume that the continuous outputs are formed by noisy observation of an underlying mapping function. The noise is in most practical applications modeled by a Gaussian distribution. This means that the observations are formed by an unknown latent function f and Gaussian noise, i.e. $y = f(\boldsymbol{x}) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. The objective of the task is to determine the underlying mapping function f from the given dataset. A function is a good fit when it is able to predict outputs, given new inputs, that correspond with reality.

In order to perform inference it is required to make some general assumptions about the dataset at hand. The assumptions can be described by a GP prior and are formed by a particular choice of covariance and mean function. The prior distribution can then be optimized, resulting in a posterior distribution, often called the marginal likelihood or evidence of the observed data. The Bayesian properties of the GP formalism give automatic complexity control and will penalize unnecessarily complex solutions at a cost of increased computational complexity, as assured by the Bayesian Occam's razor (see Section 1.1).

This chapter is organized as follows. First, we recall the definitions and key equations of GP modeling. We cover GPs for regression as well as for classification in Section 2.3 and 2.4. Subsequently, the usage and importance of covariance functions is discussed in Section 2.5. This chapter's main resources are MacKay (2002), Bishop (2006), Rasmussen and Williams (2006), which we also refer the interested reader to for a thorough explanation of GPs. Note that readers familiar with GP modeling can safely skip this chapter.

2.2 Preliminary

Definition 1. A Gaussian Process is any distribution over functions such that any finite set of function values $f(\boldsymbol{x}_1), f(\boldsymbol{x}_2), ..., f(\boldsymbol{x}_N)$ have a joint multivariate Gaussian distribution.

This is formally written as

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

where,

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

A GP is thus fully specified by its mean and covariance function. The mean function, defined as the expectation over f at an input $\boldsymbol{x}$, is often set to zero, i.e. $m(\boldsymbol{x}) = 0, \forall\, \boldsymbol{x}$. This can be done without losing generality because normalized data can approximately impose a zero mean. Therefore, through this thesis we will choose $m(\boldsymbol{x}) = 0$, unless stated otherwise.

The covariance or kernel k is a positive definite function chosen upfront and defines the correlation between the input vectors by assuming some properties such as smoothness, periodicity, trends or bias. It is usually parametrized by a limited set of kernel hyperparameters $\boldsymbol{\theta}$. The covariance function maps inputs from the input domain to values in $\mathbb{R}$. Over the past decades, a variety of covariance functions has been created, these will be discussed in more detail in Section 2.5.

2.3 Gaussian Process Regression

2.3.1 Prior and Hyperparameter Optimization

We assume a dataset, $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_{i=1}^N$ where $y_i = f(\boldsymbol{x}_i) + \epsilon_i$, with ϵ_i additive uncorrelated white Gaussian noise with variance σ^2, $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$. We further assume that the mean of the targets is zero. The N inputs $\boldsymbol{x}_i$ are collected as rows of the matrix $\boldsymbol{X} \in \mathbb{R}^{N \times D}$, where D represents the number of features per sample. The N single-dimensional outputs y_i are gathered in a vector $\boldsymbol{y} \in \mathbb{R}^N$. The objective is to infer the predictive distribution of $f(\boldsymbol{x}_*)$ at a new input $\boldsymbol{x}_*$.

The Gaussian Process Regression (GPR) procedure starts by creating a prior, called the GP prior, for the latent function $\boldsymbol{f} = [f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_N)]^\top$. The marginal likelihood or model evidence is found by marginalizing out that latent function $\boldsymbol{f}$

$$p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) = \int p(\boldsymbol{y}|\boldsymbol{f})\, p(\boldsymbol{f}|\boldsymbol{X}, \boldsymbol{\theta})\; d\boldsymbol{f}, \tag{2.1}$$

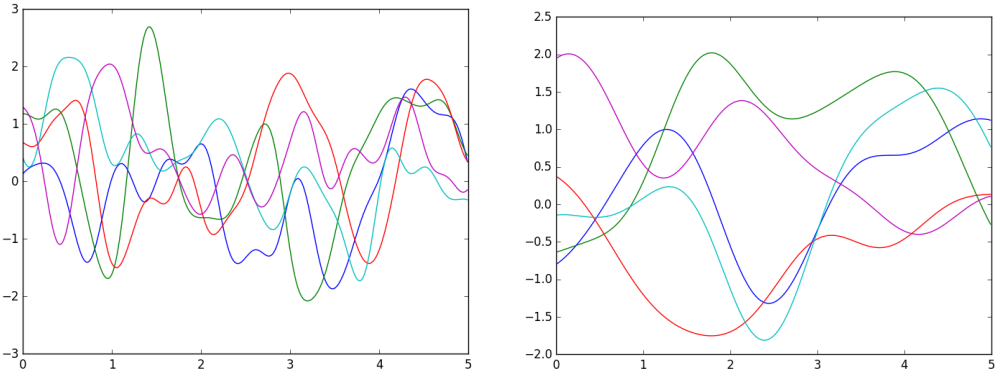
where

$$\begin{aligned} p(\boldsymbol{f}|\boldsymbol{X}, \boldsymbol{\theta}) &= \mathcal{N}(\boldsymbol{f}|\boldsymbol{0}, \boldsymbol{K_{ff}}) \quad \text{(GP prior)}, \\ p(\boldsymbol{y}|\boldsymbol{f}) &= \mathcal{N}(\boldsymbol{y}|\boldsymbol{f}, \sigma^2 \boldsymbol{I}) \quad \text{(likelihood)}. \end{aligned} \tag{2.2}$$

The GP prior is, by assumption, a multivariate Gaussian distribution with a covariance matrix $\boldsymbol{K_{ff}}$, depending on a particular covariance function. The elements of $\boldsymbol{K_{ff}}$ hold the correlation between every two input vectors, $(\boldsymbol{K_{ff}})_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j|\boldsymbol{\theta})$. As the covariance function is only evaluated at a finite number of inputs (the training dataset) the intrinsically infinite model is transformed to a finite one. Thanks to the Gaussian form of both factors in Eq. 2.1, the model evidence is analytically computable. The result is a Gaussian multivariate distribution with mean zero and covariance matrix $\tilde{\boldsymbol{K}} \triangleq \boldsymbol{K_{ff}} + \sigma^2 \boldsymbol{I}$.

$$\begin{aligned} p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) &= \mathcal{N}(\boldsymbol{y}\,|\,\boldsymbol{0}, \tilde{\boldsymbol{K}}) \\ &= \mathcal{N}\left(\boldsymbol{y} \;\middle|\; \boldsymbol{0}, \begin{bmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}_1) & k(\boldsymbol{x}_1, \boldsymbol{x}_2) & \ldots & k(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ \vdots & \vdots & & \vdots \\ k(\boldsymbol{x}_N, \boldsymbol{x}_1) & k(\boldsymbol{x}_N, \boldsymbol{x}_2) & \ldots & k(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{bmatrix} + \sigma^2 \boldsymbol{I}\right) \end{aligned} \tag{2.3}$$

The values of $\boldsymbol{K_{ff}}$ are defined by two factors: (1) the choice of the covariance function and (2) the hyperparameter values defining the parametric covariance function. In Figure 2.1 different samples from a GP prior are shown, where (a) and (b) differ in the kernel's hyperparameters. We observe that the prior of (a) advocates rougher functions while the prior of (b) prefers smoother ones. The figure clearly shows the influence of the hyperparameters on the latent function samples. For this reason, the training phase consists of optimizing the marginal likelihood (Eq. 2.3) w.r.t. the hyperparameters $\boldsymbol{\theta}$ in order to find the set of optimal hyperparameters that represent the given data, in term of smoothness, periodicity, bias, etc. In practice, instead of



(a) hyperparameters $\boldsymbol{\theta}_1$ (b) hyperparameters $\boldsymbol{\theta}_2$

Figure 2.1: Draws from a GP prior. The covariance function used for both plots is the same, but the hyperparameters differ. Hence, the difference in the samples from the prior.

directly optimizing the marginal likelihood, the log marginal likelihood

$$\mathcal{L}(\boldsymbol{\theta}) = \log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) = -\frac{1}{2}\boldsymbol{y}^\top \tilde{\boldsymbol{K}}^{-1}\boldsymbol{y} - \frac{1}{2}\log|\tilde{\boldsymbol{K}}| - \frac{N}{2}\log 2\pi \tag{2.4}$$

is optimized. During the optimization we encounter the standard bias-variance trade-off, which is present in many machine learning models. However, the GPR model possesses an automatic complexity control mechanism due to the twofold purpose of the objective function (Eq. 2.4). The first term of the equation can be interpreted as a data-fit term (lowering the bias), while the second term is a regularizer, which guards against overfitting as it will penalize unnecessary complex models (lowering the variance).

To maximize the log marginal likelihood the derivative of $\mathcal{L}(\boldsymbol{\theta})$ with respect to each hyperparameter θ_j is required

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \theta_j} = -\frac{1}{2}\operatorname{tr}\left(\tilde{\boldsymbol{K}}^{-1}\frac{\partial \tilde{\boldsymbol{K}}}{\partial \theta_j}\right) + \frac{1}{2}\boldsymbol{y}^\top \tilde{\boldsymbol{K}}^{-1}\frac{\partial \tilde{\boldsymbol{K}}}{\partial \theta_j}\tilde{\boldsymbol{K}}^{-1}\boldsymbol{y}.$$

These derivatives allow us to use any gradient based optimization algorithm, like the popular Limited memory BFGS (Zhu et al. 1997) method, to obtain the optimal hyperparameters.

2.3.2 Inference

Drawing random samples from the (optimized) prior, as was done to generate Figure 2.1, is usually not the primary goal. We are more interested in using knowledge, obtained during the training phase, to predict the function outcome at a new point. This can be achieved by assuming that the test dataset $\boldsymbol{X}_*$ is correlated in the same way as the training dataset $\boldsymbol{X}$. We further assume that the joint distribution over the training targets $\boldsymbol{y}$ and test outputs $\boldsymbol{f}_*$ is distributed according to a multivariate Gaussian

$$p(\boldsymbol{y}, \boldsymbol{f}_*|\boldsymbol{X}, \boldsymbol{X}_*) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{y}\\ \boldsymbol{f}_*\end{bmatrix}\middle|\middle|\begin{bmatrix}\boldsymbol{0}\\ \boldsymbol{0}\end{bmatrix}, \begin{bmatrix}K(\boldsymbol{X},\boldsymbol{X}) + \sigma^2\boldsymbol{I} & K(\boldsymbol{X},\boldsymbol{X}_*)\\ K(\boldsymbol{X}_*,\boldsymbol{X}) & K(\boldsymbol{X}_*,\boldsymbol{X}_*)\end{bmatrix}\right).$$

Using Gaussian identities (Eq. 1.3) we can derive the conditioned distribution, also known as the key predictive distribution

$$\begin{aligned} p(\boldsymbol{f}_*|\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{X}_*) &= \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{f}_*}, \boldsymbol{\Sigma}_{\boldsymbol{f}_*}), \quad \text{with} \\ \boldsymbol{\mu}_{\boldsymbol{f}_*} &= \boldsymbol{K}_{*\boldsymbol{f}}\tilde{\boldsymbol{K}}^{-1}\boldsymbol{y}, \\ \boldsymbol{\Sigma}_{\boldsymbol{f}_*} &= \boldsymbol{K}_{**} - \boldsymbol{K}_{*\boldsymbol{f}}\tilde{\boldsymbol{K}}^{-1}\boldsymbol{K}_{\boldsymbol{f}*}. \end{aligned} \tag{2.5}$$

For notational simplicity a compact notation is used: matrices $K(\boldsymbol{X}, \boldsymbol{X})$, $K(\boldsymbol{X}_*, \boldsymbol{X})$ and $K(\boldsymbol{X}_*, \boldsymbol{X}_*)$ are replaced by $\boldsymbol{K}_{\boldsymbol{ff}}, \boldsymbol{K}_{*\boldsymbol{f}}$ and $\boldsymbol{K}_{**}$, respectively. In Figure 2.2 the predictive distribution is illustrated on a toy dataset.

The full predictive distribution is one of the main advantage of GPs. The mean $\boldsymbol{\mu}_{\boldsymbol{f}_*}$ serves as a point estimation of the function outputs $f(\boldsymbol{X})$, while the covariance can be used to quantify the model's uncertainty of the estimation. Some remarks about the covariance matrix can be made:

1. The covariance matrix only depends on the observed inputs and not the observed outputs.
2. The second term of the covariance matrix, which is always positive, can be seen as the amount of information the training points provide about the latent function.
3. Eq. 2.5 does not only provide pointwise variances but holds the complete covariance matrix (see bottom plots in Figure 2.2).
4. The predictive distribution of test targets $\boldsymbol{y}_*$ (instead of $\boldsymbol{f}_*$) can simply be computed by adding $\sigma^2\boldsymbol{I}$ to the covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{f}_*}$.

Rewriting the predictive distribution (Eq. 2.5) for a single test point $\boldsymbol{x}_*$ reveals interesting properties of the GPR model. First we write $\boldsymbol{k}_* = k(\boldsymbol{X}, \boldsymbol{x}_*)$ to stand for the vector $(N \times 1)$ of covariance evaluations between the single input point $\boldsymbol{x}_*$ and all the train input points and k_{**} to denote $k(\boldsymbol{x}_*, \boldsymbol{x}_*)$. Rewriting Eq. 2.5 with this compact notation and for a single test point gives

$$\begin{aligned} p(f_*|\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{X}_*) &= \mathcal{N}(\mu_{f_*}, \sigma^2_{f_*}), \quad \text{with} \\ \mu_{f_*} &= \boldsymbol{k}_*^\top\tilde{\boldsymbol{K}}^{-1}\boldsymbol{y}, \qquad (2.6) \\ \sigma^2_{f_*} &= k_{**} - \boldsymbol{k}_*^\top\tilde{\boldsymbol{K}}^{-1}\boldsymbol{k}_*. \qquad (2.7) \end{aligned}$$

Examining Eq. 2.6 shows that the mean is in fact a linear combination of observations $\boldsymbol{y}$

$$\mu_{f_*} = \boldsymbol{w}_1^\top\boldsymbol{y}, \quad \text{with } \boldsymbol{w}_1^\top = \boldsymbol{k}_*^\top\tilde{\boldsymbol{K}}^{-1},$$

where $\boldsymbol{w}_1$ is called the smoothing or effective kernel. Alternatively, we can write the mean as a linear combination of $\boldsymbol{k}_*$, to find

$$\mu_{f_*} = \boldsymbol{k}_*^\top\boldsymbol{w}_2, \quad \text{with } \boldsymbol{w}_2 = \tilde{\boldsymbol{K}}^{-1}\boldsymbol{y}.$$

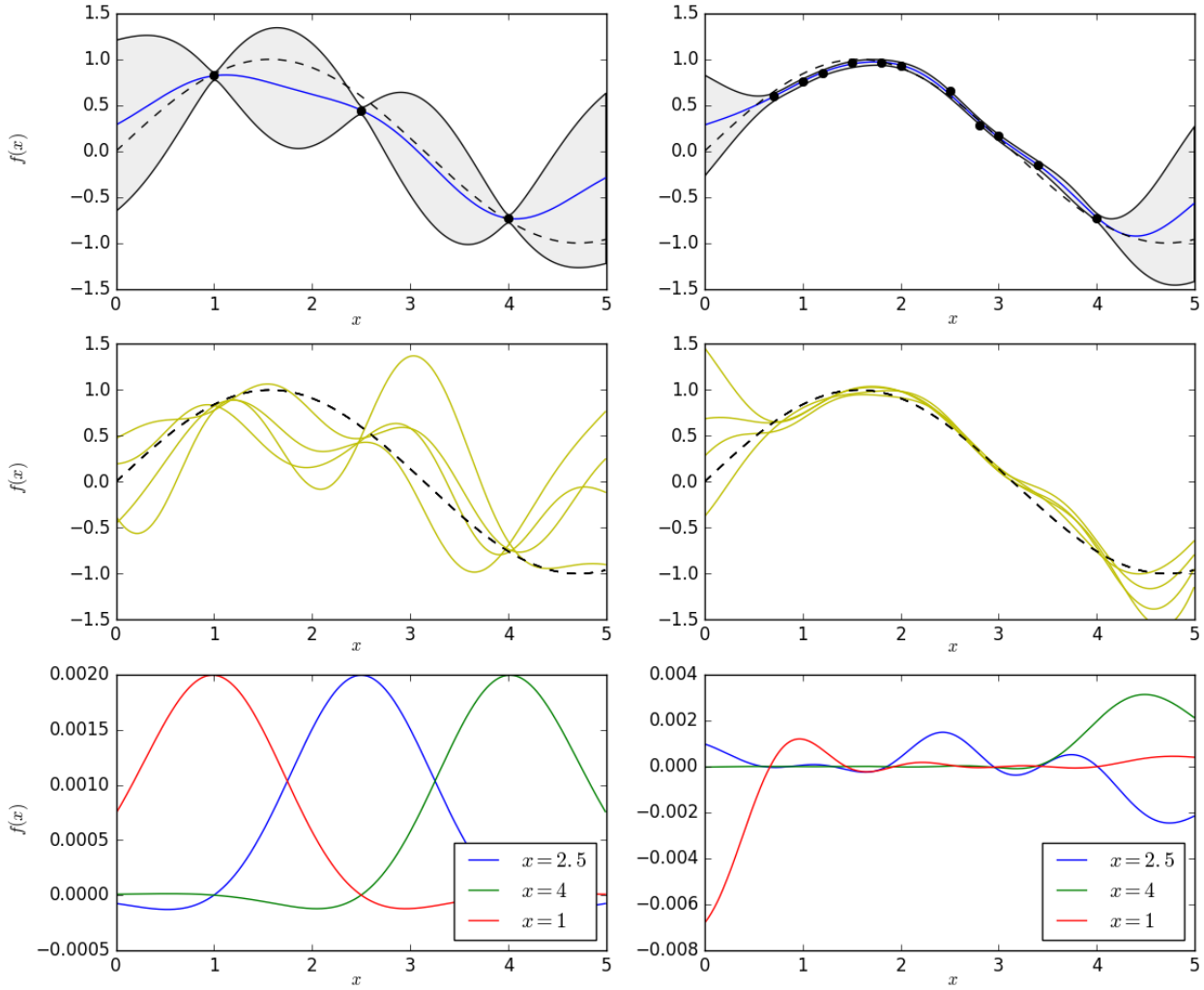


Figure 2.2: Pane illustrating the properties of the conditional predictive distribution. The objective (dashed line) is a $sin(x)$ and the training points are plotted as black dots. The model is composed with the squared exponential covariance function. The optimized hyperparameters are the following: noise variance $\sigma^2 = 1.04E-6$, kernel variance $\sigma^2_{SE} = 0.67$ and lengthscale $\ell = 1.19$. The top figures show the mean (blue line) and the pointwise std. dev. (gray area) of $\boldsymbol{f}_*|\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{X}_*$. In regions where the data is scarce the model will have high uncertainty on the mean which will result in increasing std. dev. The middle figures are samples from the predictive distribution. The bottom plots show the covariance between a fixed point and the complete input domain, which illustrates the availability of the full covariance matrix of $\boldsymbol{f}_*|\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{X}_*$.

In both cases, either the effective kernels or the linear combination of $\boldsymbol{k}_*$, the mean significantly depends on the density of the input points, added by the inverse of $\tilde{\boldsymbol{K}}$ which is present in both equations. Consequently, the performance of the prediction generally benefits from trainings data covering the complete input domain, rather than closely packed data points.

2.4 Gaussian Process Classification

Before covering GPs for classification, a brief and general introduction is given regarding probabilistic classification. Readers familiar with these basic concepts can directly proceed to Section 2.4.3.

2.4.1 Inference and Decision Theory

In the previous section we studied GPR, which is a model for solving regression problems. In a regression problem, the targets are real valued, while in classification problems the targets are discrete class labels and the goal is to assign each input $\boldsymbol{x}$ with its corresponding class $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_K$, assuming that there are K non-overlapping classes.

A well-known classification task is: given an image of a handwritten digit, predict the most likely digit in the image. The model should predict the class (between 0 and 9) to which an input image belongs. This examples deals with 10 distinct classes but many real-world tasks are just binary. For example, predicting whether or not a person will pay back his or her loan. In this case, there are only 2 classes: $\mathcal{C}_1$, people who paid back there loan and $\mathcal{C}_2$, people who will not pay it back. This type of classification is called binary classification, while the other examples belong to the multi-class classification problems. In the following section we will primarily focus on the binary case.

Probabilistic classification is a particular form of classification where we rather predict the probability that a certain input $\boldsymbol{x}$ belongs to a class than the class label itself. Instead of returning a class label, the output of the model becomes a probability and the goal is to approximate the function $p(y = \mathcal{C}_i|\boldsymbol{x})$. Therefore, probabilistic classification can be seen as a form of regression where the targets are real values in $[0, 1]$.

Once the predictive probabilities are calculated (i.e. $p(y = \mathcal{C}_i|\boldsymbol{x}),\ \forall i$) we can turn to the decision problem. In the decision problem, the predictive probabilities are used in combination with consequences in order to make an optimal class assignment. A consequence can be interpreted as the impact of a misclassification for an input belonging to a certain class. In the simplest case, the consequences are independent of the classes, but, as we will see below, there exists examples where consequences are class dependent. Regardless of whether or not consequences are class dependent or not, in order to make classifications we need a rule that assigns an input $\boldsymbol{x}$ to a certain class $\mathcal{C}_j$. Such a rule will divide the input space in decision regions $\mathcal{R}_j$. Within $\mathcal{R}_j$ every input is assigned to class $\mathcal{C}_j$. The boundaries of such regions are called decision surfaces (input dimension ≥ 3) or decision boundaries (input dimensions < 3). A mistake will take place when an input $\boldsymbol{x}$, belonging to class $\mathcal{C}_i$, is located in decision region $\mathcal{R}_j$ and assigned to class $\mathcal{C}_j$,

where $i \neq j$. The probability of a mistake is thus

$$p(\text{mistake}) = \int_{\mathcal{R}_1} p(\boldsymbol{x}) p(\boldsymbol{x} \in \mathcal{C}_2 | \boldsymbol{x}) d\boldsymbol{x} + \int_{\mathcal{R}_2} p(\boldsymbol{x}) p(\boldsymbol{x} \in \mathcal{C}_1 | \boldsymbol{x}) d\boldsymbol{x}.$$

In order to minimize the probability of making a mistake it is clear that assigning the class $\mathcal{C}_i$ to input $\boldsymbol{x}$ with the largest value of $p(\boldsymbol{x} \in \mathcal{C}_i | \boldsymbol{x})$ is optimal. In other words, when $p(\boldsymbol{x} \in \mathcal{C}_1 | \boldsymbol{x}) > p(\boldsymbol{x} \in \mathcal{C}_2 | \boldsymbol{x})$, we should assign $\boldsymbol{x}$ to class $\mathcal{C}_1$, and vice versa. In the case of multi-class classification we can again minimize the probability of making a mistake by selecting class $\mathcal{C}_i$ for input $\boldsymbol{x}$ when $i = \text{argmax}_{k \in [1,K]} \; p(\boldsymbol{x} \in \mathcal{C}_k | \boldsymbol{x})$.

In the previous paragraph we have simply tried to minimize the probability of making a mistake. Note that this is analogous to maximizing the probability of being correct $p(\text{correct})$. However, for many applications the objective will be a lot more complex and the consequences of making a mistake will be class dependent. Consider, for example, a medical diagnosis case, where we are given a chest's X-ray and the goal is to classify the image as originating from a healthy person ($\mathcal{C}_1$), or from a person with cancer ($\mathcal{C}_2$). The cost or consequence of assigning an X-ray image, originating from a patient with cancer, to class $\mathcal{C}_1$ (healthy people) is much more problematic than the inverse. In the inverse case we classify an X-ray from a healthy patient as being from a patient with cancer. When the first mistake occurs, the possibility exists that the patient dies due to a lack of treatment as his/her cancer was not diagnosed. The consequences of the second type of mistake are obviously less dramatic. In medical statistics, these types of misclassifications are called false negative and false positive errors, respectively.

It is possible to formalize these different misclassification consequences in the form of a loss function (or utility function). The goal is then to select decision regions in order to minimize the loss function (or maximize the utility function). The expected loss function is defined as

$$\mathbb{E}[L] = \sum_k \sum_j \int_{\mathcal{R}_j} L_{kj} \, p(\boldsymbol{x}) \, p(\boldsymbol{x} \in \mathcal{C}_k | \boldsymbol{x}) d\boldsymbol{x},$$

where L_{kj} is the cost or consequence of assigning input vector $\boldsymbol{x}$ to class $\mathcal{C}_j$ when it actually belongs to class $\mathcal{C}_k$. For the above example, we could assign L_{kj} with the following values

$$L_{11} = 0, \quad L_{12} = 1000, \quad L_{21} = 1, \quad L_{22} = 0.$$

These values indicate that there is no cost associated with assigning the correct class to an input and that making a false negative mistake (i.e. when the patient has cancer $\mathcal{C}_2$ but is assigned to the healthy class $\mathcal{C}_1$) is far more dramatic than a false positive one. We proceed by selecting the decision regions $\mathcal{R}_j$ in order to minimize the expected loss $\mathbb{E}[L]$. This is achieved by assigning a new input $\boldsymbol{x}$ to the class $\mathcal{C}_j$ which minimizes the quantity

$$\sum_k L_{kj} \, p(\boldsymbol{x} \in \mathcal{C}_k | \boldsymbol{x}).$$

In conclusion, it is possible to separate the classification problem in two different tasks. In the first task, called the inference problem, a model is learned based on the labeled trainings data

to model the predictive distribution $p(\mathcal{C}_k|\boldsymbol{x})$. The subsequent task is the decision problem, in which the predictive distribution is used to assign a single class label to a new input. In the above we discussed the decision problem and in the next sections we will focus on the inference task.

2.4.2 Generative vs. Discriminative

In general, there are two different approaches to solving the inference task. The first, called the generative approach, models the joint probability $p(\mathcal{C}_k, \boldsymbol{x})$ by calculating the prior probability $p(\mathcal{C}_k)$ and learning the class conditioned probability $p(\boldsymbol{x}|\mathcal{C}_k)$ for each class. The posterior distribution is then obtained by applying Bayes' rule

$$p(\mathcal{C}_k|\boldsymbol{x}) = \frac{p(\mathcal{C}_k)p(\boldsymbol{x}|\mathcal{C}_k)}{p(\boldsymbol{x})}.$$

The second approach, called the discriminative approach, focuses directly on modeling the posterior class probabilities $p(\mathcal{C}_k|\boldsymbol{x})$. Both methods have their weaknesses and strengths, so choosing the correct one can significantly influence the final accuracy and usefulness of the model. The generative approach gives access to the complete joint probability $p(\mathcal{C}_k, \boldsymbol{x}) = p(\mathcal{C}_k)p(\boldsymbol{x}|\mathcal{C}_k)$ which makes it possible to deal with missing data, unlabeled data or outliers. Furthermore, by sampling from the complete joint probability it is possible to generate synthetic data, hence the name of the approach. The priors $p(\mathcal{C}_k)$ are in practice easily to obtain by calculating the fraction of each class in the trainings dataset. On the contrary, obtaining the class conditioned distributions $p(\boldsymbol{x}|\mathcal{C}_k)$ can be computationally very challenging, especially when the input dimension is high. Therefore, when the application allows to predict only the posterior distribution, it can be computationally much more efficient to apply the discriminative approach. The Gaussian Process Classification (GPC) model, considered in the next section, is an example of the discriminative approach.

2.4.3 Gaussian Process Classification

In this section we consider the case of binary classification with GPs. We assume that $\boldsymbol{y}$ stores the class labels and that an element $y_n = +1$ when $\boldsymbol{x}_n$ belongs to class $\mathcal{C}_1$ and equals -1 when $\boldsymbol{x}_n$ belongs to class $\mathcal{C}_2$. The goal is to model the posterior probability $p(y = +1|\boldsymbol{x})$. As there are only two classes we can directly infer the probability of the other class by calculating $1 - p(y = +1|\boldsymbol{x}) = p(y = -1|\boldsymbol{x})$.

The GPC model is quite analogous to the GPR model. We start by defining a prior over the latent function $f(\boldsymbol{x})$. Then, this latent function is directly squeezed through a sigmoid function $\sigma(f(\boldsymbol{x}))$ to obtain a prior over the probability $p(y = +1|\boldsymbol{x})$, as illustrated in Figure 2.3. A sigmoid function is a monotonically increasing function mapping from $\mathbb{R}$ to $[0, 1]$. Examples are the logistic $(1+\exp(-x))^{-1}$ or the cumulative distribution of a normal PDF $\int_{-\infty}^{x} \mathcal{N}(z|0, 1)\, dz$.

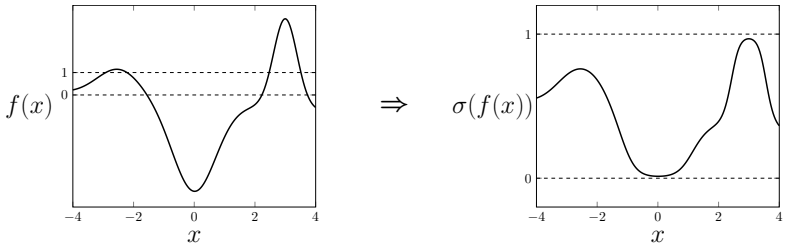


Figure 2.3: The process of squeezing a latent function f through a sigmoid function.

If we now consider a new test point $\boldsymbol{x}_*$ and we wish to obtain $p(y_* = +1|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}_*)$ we need to calculate the integral

$$p(y_* = +1|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}_*) = \int \sigma(f_*) p(f_*|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}_*) df_*. \tag{2.8}$$

The second factor of the integrand, the probability of the latent function, is given by

$$p(f_*|\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}_*) = \int p(f_*|\boldsymbol{X}, \boldsymbol{x}_*, \boldsymbol{f}) p(\boldsymbol{f}|\boldsymbol{X}, \boldsymbol{y}) d\boldsymbol{f}, \tag{2.9}$$

where due to Bayes' rule,

$$p(\boldsymbol{f}|\boldsymbol{X}, \boldsymbol{y}) = \frac{p(\boldsymbol{f}|\boldsymbol{X}) p(\boldsymbol{y}|\boldsymbol{f})}{p(\boldsymbol{y}|\boldsymbol{X})}. \tag{2.10}$$

The first factor of the nominator in Eq. 2.10, $p(\boldsymbol{f}|\boldsymbol{X})$, is the traditional GP prior, which is defined as a Gaussian distribution $\mathcal{N}(0, K(\boldsymbol{X}, \boldsymbol{X}))$. The second factor, the likelihood, is in the case of regression also defined as a Gaussian distribution $\mathcal{N}(0, \sigma^2 \boldsymbol{I})$, which makes the GPR model analytically tractable (see Eq. 2.1) as two Gaussians can be multiplied in a analytical closed form. In the case of binary GPC, the targets are either -1 or $+1$ which makes using a Gaussian distribution as likelihood a poor choice. A more suitable choice would be to use a sigmoid function, as follows

$$p(\boldsymbol{y}|\boldsymbol{f}) = \prod_{n=1}^{N} \sigma(f_n)^{0.5\,(1+y_n)} (1 - \sigma(f_n))^{0.5\,(1-y_n)} = \prod_{n=1}^{N} \sigma(y_n f_n), \tag{2.11}$$

assuming symmetry, $\sigma(-x) = 1 - \sigma(x)$. Unfortunately, this likelihood multiplied with the Gaussian prior in Eq. 2.10 is no longer analytically tractable. Hence, the resulting Eq. 2.8 is for most sigmoid functions, such as the logistic one, not analytically computable.

From the above explanation it becomes apparent that the predictive distribution can not be computed analytically. Therefore, some approximation must be employed to obtain a different version. Over the years, several techniques have been proposed but only a couple withstood the test of time: (1) the Laplace approximation (Williams and Barber 1998), (2) Variational methods (Gibbs and MacKay 2000), (3) Expectation-Propagation (Minka 2001) and (4) MCMC sampling (Neal 1997). In the next paragraphs we will discuss the Laplace approximation due to its general applicability and good overall performance. We refer the interested reader to Kuss and Rasmussen (2005) and Nickisch and Rasmussen (2008) for an in-depth discussion and comparison of the other techniques.

The Laplace Approximation

The key idea in Laplace's method is to replace the intractable posterior $p(\boldsymbol{f}|\boldsymbol{X},\boldsymbol{y})$ in Eq. 2.9 by an approximation $q(\boldsymbol{f}|\boldsymbol{X},\boldsymbol{y})$, which is given a Gaussian form $\mathcal{N}(\boldsymbol{f}|\boldsymbol{m},\boldsymbol{A})$. First, we will show how this substitution is able to remedy the encountered intractabilities. Subsequently, we derive the center $\boldsymbol{m}$ and covariance matrix $\boldsymbol{A}$ of the Gaussian approximation used. As already mentioned, replacing $p(\boldsymbol{f}|\boldsymbol{X},\boldsymbol{y})$ with $q(\boldsymbol{f}|\boldsymbol{X},\boldsymbol{y})$ in Eq. 2.9 will enable the exact calculation of the approximation

$$\begin{aligned} p(f_*|\boldsymbol{X},\boldsymbol{y},\boldsymbol{x}_*) &\approx \int p(f_*|\boldsymbol{X},\boldsymbol{x}_*,\boldsymbol{f})\, q(\boldsymbol{f}|\boldsymbol{X},\boldsymbol{y}) d\boldsymbol{f} \\ &= \int \mathcal{N}\left(f_*|\boldsymbol{k}_*^\top \boldsymbol{K}^{-1}\boldsymbol{f}, k_{**} - \boldsymbol{k}_*^\top \boldsymbol{K}^{-1}\boldsymbol{k}_*\right) \mathcal{N}\left(\boldsymbol{f}|\boldsymbol{m},\boldsymbol{A}\right) d\boldsymbol{f} \\ &= \mathcal{N}\left(f_*|\boldsymbol{k}_*^\top \boldsymbol{K}^{-1}\boldsymbol{m}, k_{**} - \boldsymbol{k}_*^\top (\boldsymbol{K}^{-1} - \boldsymbol{K}^{-1}\boldsymbol{A}\boldsymbol{K}^{-1})\boldsymbol{k}_*\right). \end{aligned} \tag{2.12}$$

In order to make predictions, computing $\int \sigma(f_*) p(f_*|\boldsymbol{X},\boldsymbol{y},\boldsymbol{x}_*) df_*$ (Eq. 2.8) is still required. If we substitute the result of Eq. 2.12 into this integral, we are faced with a product of a sigmoid and a normal distribution. For most sigmoid functions, this can not be computed analytically. However, if $\sigma(f_*)$ is a cumulative Gaussian it is possible due to the fact that $\int \Phi(x)\mathcal{N}(x|\mu,\sigma^2)dx = \Phi(\mu/\sqrt{1+\sigma^2})$, where $\Phi(x) = \int_{-\infty}^{x} \mathcal{N}(z|0,1)\, dz$. Filling in the correct variables for μ and σ^2 we obtain

$$p(y_* = +1|\boldsymbol{X},\boldsymbol{y},\boldsymbol{x}_*) = \Phi\left(\frac{\boldsymbol{k}_*^\top \boldsymbol{K}^{-1}\boldsymbol{m}}{\left(1 + k_{**} - \boldsymbol{k}_*^\top (\boldsymbol{K}^{-1} - \boldsymbol{K}^{-1}\boldsymbol{A}\boldsymbol{K}^{-1})\boldsymbol{k}_*\right)^{1/2}}\right).$$

For different choices of $\sigma(x)$, such as the logistic function, Eq. 2.8 is not analytically computable anymore and we typically need to resort to sampling techniques. However, as the integral is only over a 1-dimensional variable, good approximations are obtained with Monte Carlo or importance sampling in a limited number of iterations.

We now focus on deriving the optimal center $\boldsymbol{m}$ and covariance matrix $\boldsymbol{A}$ of the approximation. We start by taking the log of the unnormalized posterior (Eq. 2.10)

$$\begin{aligned} \log \mathcal{Q}(\boldsymbol{f}|\boldsymbol{X},\boldsymbol{y}) &= \log p(\boldsymbol{f}|\boldsymbol{X}) + \log p(\boldsymbol{y}|\boldsymbol{f}) \\ &= -\frac{1}{2}\log|\boldsymbol{K}| - \frac{N}{2}\log 2\pi - \frac{1}{2}\boldsymbol{f}^\top \boldsymbol{K}\boldsymbol{f} + \log \mathcal{L}(\boldsymbol{f}), \end{aligned}$$

where $\log \mathcal{L}(\boldsymbol{f}) = \log p(\boldsymbol{y}|\boldsymbol{f})$. The Laplace approximation is obtained by expanding $\log \mathcal{Q}(\boldsymbol{f}|\boldsymbol{X},\boldsymbol{y})$ by a second order Taylor polynomial around the mode (maximum) of the function. In general, the second order Taylor series for a function $f : \mathbb{R}^N \mapsto \mathbb{R}$ around $\boldsymbol{x}_0$ is given by

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}_0) + (\boldsymbol{x} - \boldsymbol{x}_0)^\top \nabla f(\boldsymbol{x}_0) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}_0)^\top \nabla\nabla f(\boldsymbol{x}_0)(\boldsymbol{x} - \boldsymbol{x}_0),$$

where $\nabla\nabla f(\boldsymbol{x}_0)$ is the Hessian matrix of f. However, here we approximate $\log \mathcal{Q}(\boldsymbol{f}|\boldsymbol{X},\boldsymbol{y})$ around

its maximum which turns the first derivate into the zero vector. Yielding

$$\log \mathcal{Q}(\boldsymbol{f}|\boldsymbol{X},\boldsymbol{y}) \approx \log \mathcal{Q}(\boldsymbol{m}|\boldsymbol{X},\boldsymbol{y}) - \frac{1}{2}(\boldsymbol{m}-\boldsymbol{f})^\top \boldsymbol{A}^{-1}(\boldsymbol{m}-\boldsymbol{f}), \tag{2.13}$$

where

$$\begin{aligned}\boldsymbol{m} &= \operatorname*{argmax}_{\boldsymbol{f}} \log \mathcal{Q}(\boldsymbol{f}|\boldsymbol{X},\boldsymbol{y}) \\ \boldsymbol{A} &= -(\nabla\nabla \log \mathcal{Q}(\boldsymbol{m}|\boldsymbol{X},\boldsymbol{y}))^{-1} = (\boldsymbol{K}^{-1} + \nabla\nabla \log \mathcal{L}(\boldsymbol{m}))^{-1}.\end{aligned}$$

Note that when $\log p(\boldsymbol{y}|\boldsymbol{f})$ is log concave, the Hessian in Eq. 2.13 is negative definite and $\log \mathcal{Q}(\boldsymbol{f}|\boldsymbol{X},\boldsymbol{y})$ becomes concave as well. This means that $\boldsymbol{m}$ is an unique maximum, which can be found conveniently using Newton's method.

Up until now we have not discussed how to learn the hyperparameters $\boldsymbol{\theta}$ for the GPC model. Although the approach is very similar to the one followed in the GPR model, yet again the computations are complicated by non-Gaussian intractabilities. Fortunately, Laplace's method also accounts for this. The optimal hyperparameters are found by maximizing – as always – the log marginal likelihood

$$\begin{aligned}p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta}) &= \int p(\boldsymbol{f}|\boldsymbol{X},\boldsymbol{\theta}) p(\boldsymbol{y}|\boldsymbol{f}) d\boldsymbol{f} \\ &= \int \exp(\log \mathcal{Q}(\boldsymbol{f}|\boldsymbol{X},\boldsymbol{y})) d\boldsymbol{f}\end{aligned}$$

w.r.t. hyperparameters $\boldsymbol{\theta}$. Substituting $\log \mathcal{Q}(\boldsymbol{f}|\boldsymbol{X},\boldsymbol{y})$ by its second order Taylor series (Eq. 2.13) the integral can be solved. We obtain

$$\log p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta}) \approx \log \mathcal{Q}(\boldsymbol{m}|\boldsymbol{X},\boldsymbol{y}) + \frac{1}{2}\log|\boldsymbol{A}| + \frac{N}{2}\log 2\pi.$$

This approximate marginal log likelihood can be analytically derived w.r.t. $\boldsymbol{\theta}$, which can then be used for gradient based optimization of the hyperparameters.

2.5 Covariance Functions

Covariance functions or kernels form a crucial part of the framework as they are one of the two key components defining a GP (recall Definition 1). In fact, as the mean function is often chosen to be zero, the covariance function is essentially the only one of importance. A kernel is used to construct covariance matrices, as follows

$$\boldsymbol{K}_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j),$$

where $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \mathbb{E}[(f(\boldsymbol{x}_i) - m(\boldsymbol{x}_i))(f(\boldsymbol{x}_j) - m(\boldsymbol{x}_j))]$.

A covariance function k can formally be described as a positive semi-definite function ($\boldsymbol{v}^\top \boldsymbol{K}\boldsymbol{v}$, $\forall\, \boldsymbol{v} \in \mathbb{R}^D$) over the input space which maps every pair of inputs $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ to a number in

$\mathbb{R}$. The kernel should represent the particular structure of the problem at hand, such as the smoothness, linearity, stationarity, etc. However, this task is ill-posed as most of these properties are usually unknown at the start of the process. Therefore, historically, the Squared Exponential (SE) function is often used. However, in this thesis (more particularly in Chapter 5 and 6) we will see that other kernels, such as the Matérn or the Spectral Mixture (SM) kernels, are often more realistic for engineering problems, although they suffer from other issues.

Before studying some practical kernels, we mention a few important properties, which are often used to classify them. Most of the covariance functions used in practice belong to the class of stationary covariance functions. A covariance function is stationary when the output of the function depends only on the relative position of its two inputs, and not on their absolute location. This is formally written as $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = k(\boldsymbol{x}_i - \boldsymbol{x}_j) = k(\boldsymbol{\tau})$, which also implies that the function output is invariant to translations in the input space. Typical stationary covariance functions are the SE, Matérn, Periodic and Linear kernel. Further, a covariance function is considered isotropic when the outcome is a function of the Euclidean distance $r = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|$ only. The SE is also an example of an isotropic kernel.

2.5.1 The Squared Exponential covariance

De de-facto standard stationary kernel for GPs (as well as for SVMs) is the SE, also known as the Exponentiated Quadratic (EQ) or the Radial Basis Function (RBF)

$$k_{SE}(r) = \sigma_k \exp\left(-\frac{r^2}{2\,\ell^2}\right), \tag{2.14}$$

where $r = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|$. Samples from this kernel are shown in the upper-left plot of Figure 2.4. The SE kernel has only two parameters: the signal or output variance σ_k and the lengthscale ℓ. The latter determines the extrapolation capacity or the smoothness of the kernel. A small lengthscale typically results in highly oscillating behavior while a large lengthscale leads to a smoother function. This difference is clearly visible in Figure 2.1, showing samples from $\mathcal{N}(0, \boldsymbol{K})$ using the SE, with (a) and (b) using a small and large lengthscale, respectively. The signal variance specifies the average distance from the signal to its mean. Note that every kernel has such a parameter as it is just a scaling factor, denoting the amplitude. However, when the training data is normalized, i.e. zero mean and unit variance, and the noise level is relatively small, the optimized signal variance should ideally be close to unit.

The SE kernel uses a single lengthscale ℓ for all its dimensions, which can lead to undesirable results in a lot of real-world modeling problems. For example, when we have an irrelevant feature in our dataset the standard SE kernel will give it the exact same importance as it would give a very important one. The Automatic Relevance Determination (ARD) SE kernel resolves this problem. Instead of using a single lengthscale ℓ for all dimensions, a characteristic lengthscale ℓ_d is used per dimension.

$$k_{SE}(\boldsymbol{\tau}) = \sigma_k \exp\left(-\sum_{d=1}^{D} \frac{{\tau^{(d)}}^2}{2\,\ell_d^2}\right),$$

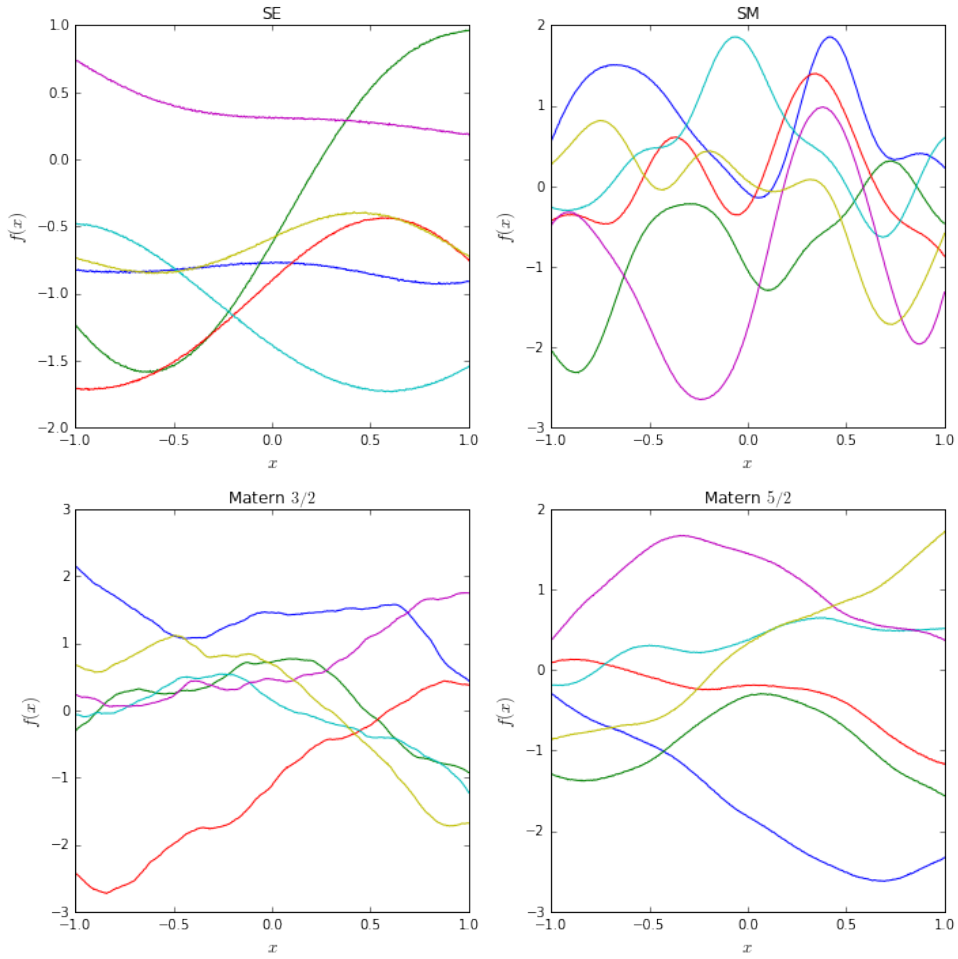


Figure 2.4: GP prior (i.e. $\mathcal{N}(0, \boldsymbol{K})$) samples from different kernels.

where $\tau^{(d)}$ is the value in the d^{th} dimension of $\boldsymbol{\tau}$. Note that the ARD-SE is not isotropic anymore, but is still stationary. The ARD-SE kernel has $D + 1$ parameters, compared to only 2 for the SE. Although this adds a lot of flexibility it complicates the optimization process.

The downside of both, the SE and ARD-SE kernel, is that they are in a sense nothing more than intelligent smoothing devices. In general, they are not able to extrapolate data further than a single lengthscale away from training points. This makes them, for example, not applicable for forecasting time-series or extrapolating data outside the domain of the training data. Besides this, the SE kernels have issues dealing with non-uniform input distributions, which is often encountered in a metamodeling setting. This issue is discussed in greater detail in Chapter 5.

2.5.2 The Matérn class covariance

Kernel functions of the Matérn class have the following form

$$k_{\text{MAT}}(r) = \sigma_k \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}\, r}{\ell} \right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu}\, r}{\ell} \right), \tag{2.15}$$

where $\Gamma(\cdot)$ is the Gamma function, $K_\nu(\cdot)$ is a modified Bessel function and ν, σ_k and ℓ are hyperparameters. Matérn kernels are stationary and isotropic as they only depend on the Euclidean distance between the two input vectors. Furthermore, σ_k and ℓ are again the scaling factor and the lengthscale, respectively.

In fact, Eq. 2.15 describes a complete set of kernel functions depending on the value of ν, which regulates the smoothness of typical samples as they are $\nu - 1$ times differentiable. Indeed, for ν going to infinity we obtain the SE kernel, which is, thanks to the exponential function, infinitely differentiable. For smaller ν's the samples become less differentiable and are much rougher. This behavior is clearly visible in Figure 2.4.

In practice, ν is frequently chosen to be either $1/2$, $3/2$ or $5/2$ as it suits most real-world modeling problems. Remarkably in the case of $\nu = 1/2$, the Matérn function reduces to the isotropic exponential kernel $k_{\text{EXP}}(r) = \sigma_k \exp(-r/\ell)$.

2.5.3 The Spectral Mixture covariance

The SM kernel is the main subject of Chapter 6 but we include it already here for completeness and to directly compare it with the other kernels. The SM kernel is a linear combination of Q terms, where each term is composed out of a Periodic and SE similar-looking kernel. This gives

$$k_{SM}(\boldsymbol{\tau}) = \sum_{q=1}^{Q} w_q \cos\left(2\pi \boldsymbol{\tau}^\top \boldsymbol{\mu_q}\right) \exp\left(-2\pi^2 \|\boldsymbol{\tau} \odot \boldsymbol{v}_q\|^2\right), \tag{2.16}$$

where each mixture component is parameterized by a vector $\boldsymbol{v}_q$ of lengthscales, a vector $\boldsymbol{\mu}_q$ of frequencies and weight w_q. The weights w_q specify the relative contribution of each term, the lengthscales determine how smooth the functions are or how quickly a component varies with the inputs and the inverse of the frequency parameters encode the components period. In Chapter 6 we will derive Eq. 2.16 starting from its spectral density, which is the Fourier transform of a kernel.

The SM kernel is, compared to the other kernels, particularly powerful in the extrapolation of time-series due to its large number of hyperparameters and its periodic components. However, this flexibility imposes a critical challenge: the optimization routine becomes extremely prone to local minima and highly sensitive to hyperparameter initialization. In Chapter 6 we further analyze this problem and propose suitable solutions.

2.5.4 Other non-stationary covariance functions

Although non-stationary kernels are not used in this thesis, they are briefly covered for completeness. Moreover, the Linear non-stationary kernel perfectly illustrates the tight link between GPs and Bayesian parametric models. If the covariances are expected to not depend on the relative distance between the input points but rather on the values the inputs take, one should consider non-stationary covariance functions. Non-stationary kernels allow the GP to adapt to functions who have varying smoothness over the input space.

The Linear (non-stationary) kernel is given by

$$k_{\text{LIN}}(\boldsymbol{x}_i, \boldsymbol{x}_i) = \sum_{d=1}^{D} \sigma_d^2 x_i^{(d)} x_j^{(d)}, \tag{2.17}$$

where $x^{(d)}$ is the d^{th} component of a vector $\boldsymbol{x} \in \mathbb{R}^D$. This kernels is interestingly also obtained by considering a Bayesian linear model, i.e. $y = \boldsymbol{x}^\top \boldsymbol{w}$, where $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma})$ and $\boldsymbol{\Sigma} = \text{diag}(\sigma_1^2, \ldots, \sigma_D^2)$. Then

$$m(\boldsymbol{x}) = \mathbb{E}[f(\boldsymbol{x})] = \mathbb{E}[\boldsymbol{x}^\top \boldsymbol{w}] = \boldsymbol{x}^\top \mathbb{E}[\boldsymbol{w}] = 0.$$

Therefore, we find for the kernel

$$\begin{aligned} k(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \mathbb{E}[(f(\boldsymbol{x}_i) - m(\boldsymbol{x}_i))(f(\boldsymbol{x}_j) - m(\boldsymbol{x}_j))] \\ &= \boldsymbol{x}_i^\top \mathbb{E}[\boldsymbol{w}\boldsymbol{w}^\top] \boldsymbol{x}_j \\ &= \boldsymbol{x}_i^\top \boldsymbol{\Sigma} \boldsymbol{x}_j \\ &= \sum_{d=1}^{D} \sigma_d^2 x_i^{(d)} x_j^{(d)}. \end{aligned}$$

□

This final expression corresponds to the definition of the Linear kernel (Eq. 2.17), which shows a clear duality between Bayesian parametric models (here: a linear model with parameters $\boldsymbol{w}$, which have a Gaussian prior) and GPs with a particular kernel.

There exist numerous other non-stationary covariance function, such as the neural network and the dot product kernel. We refer the reader to Rasmussen and Williams (2006) for a detailed overview.

Chapter 3

Efficient Gaussian Processes

Gaussian Process (GP) models provide a flexible, probabilistic approach to numerous machine learning tasks thanks to their Bayesian, non-parametric nature. The non-parametricness is responsible for its flexibility as the model's expressiveness grows with the amount of available data. From a traditional machine learning perspective, one would expect that this probably leads to a model which is overfitting the training data, i.e. overreacting on minor fluctuations (e.g., noise), rather than learning a generalized structure that can be extended to the test data. Fortunately, the probabilistic approach used in GPs averages over all the latent functions (and sometimes parameters), which makes the framework less prone to overfitting. However, this flexibility and regularization power comes at a price and some considerations should be made when using GP models:

1. The approximation of the (marginal) posterior when the likelihood is non-Gaussian is troublesome. In Section 2.4 we emphasized the complexity of performing inference in GP classification due to the sigmoidal likelihood (Eq. 2.11). We turned to the Laplace approximation, as it is one of the most straightforward solutions, but many other techniques have been developed for the efficient handling of non-Gaussian likelihoods (Nickisch and Rasmussen 2008). Most of these circumvent the problem by approximating a term in one of the intractable equations. Although these approximations enable the analytical calculation or speedup the process, they often result in a lower performance and accuracy. Numerical approximations, such as Monte Carlo, also exist and can result in desirable results when the dimensionality is limited.

2. Computation and storage during training and inference scales poorly in the number of data. Standard GPs require the evaluation of $\left(\boldsymbol{K_{ff}} + \sigma^2 \boldsymbol{I}\right)^{-1} \boldsymbol{y}$ and $|\boldsymbol{K_{ff}} + \sigma^2 \boldsymbol{I}|$, for a $N \times N$ covariance matrix $\boldsymbol{K_{ff}}$ and a column-vector $\boldsymbol{y} \in \mathbb{R}^N$, during the optimization of the model's hyperparameters (Eq. 2.4). The most computationally stable and efficient way to obtain these forms is via the Cholesky decomposition, which requires $\mathcal{O}(N^2)$ memory and $\mathcal{O}(N^3)$ computations. This polynomial scaling of the complexity restricts standard GPs in modeling more than a few thousand data points and it is one of the main reasons why GPs are currently not being applied to very large datasets.

3. GPs using standard covariance functions, such as the Squared Exponential (SE) or the Matérn kernel, can typically only be used successfully for smoothing or interpolation. Where other advanced techniques, such as Neural Networks (NNs), are intrinsically better able to automatically discover meaningful patterns, GPs have difficulties with this. As GPs are mainly characterized by the kernel function used, most of the successful solutions constitute of incorporating more expressiveness and extrapolation power in the kernel function. (Wilson and Adams 2013, Yang et al. 2014, Wilson et al. 2016). Chapters 5 and 6 address this challenge.

The second issue (i.e. the computational scalability problems during training and inference) is the main focus of this chapter. We approach it as follows. In Section 3.2 we try to speed up the Cholesky decomposition from an implementation point of view. The underlying idea of this approach is to simply keep everything as is, but implementation-wise improve the algebraic parts causing the bottleneck. In Section 3.1 we investigate approximations of the standard model that enable more efficient inference techniques.

3.1 Approximations

Over the past decades, a lot of research has been done that attempts to alleviate the computational and storage complexity of GP learning and inference. Early strategies are based on selecting a subset of the training points, the active set, upon which further computations are based on. A typical way of choosing such a subset is through a heuristic information criterion. Subset of Data (SD) (Csató and Opper 2002) – arguably the most straightforward approach – reduces computational complexity by simply dropping a large part of the training data. In most practical scenario's this approach yields a considerable loss of accuracy. However, when the dataset is highly redundant and extra data points provide little to none additional information, this technique is able to lower the complexity by significantly reducing the dimensions of the covariance matrix. More advanced methods (Smola and Bartlett 2001, Williams and Seeger 2001) concentrate on constructing rigorous heuristics that enable selecting a qualitative subset out of the original dataset.

Newer approaches develop efficient approximations to the full GP. For example, Sparse Gaussian Processes (SGPs) expand the model with M additional pseudo-inputs $\boldsymbol{Z}$ and pseudo-outputs $\boldsymbol{u}$. The pseudo-inputs and outputs are also often referred to as inducing points. The general idea is to transform the inducing points into additional hyperparameters, characterizing the original dataset. Their values are estimated by jointly maximizing the log marginal likelihood. Adding M new datapoints as hyperparameters, significantly increases the flexibility of the model, but can lead to a form of overfitting, due to the absence of regularization. Nonetheless, SGPs reduce the complexity of the training phase to $\mathcal{O}(M^2N)$. The test phase requires, per datapoint, $\mathcal{O}(M)$ and $\mathcal{O}(M^2)$ for the prediction mean and variance, respectively. SGP models proposed in the literature are legion, a throughout overview can be found in Quiñonero-Candela and Rasmussen (2005). Although these approaches have been derived from various different angles, many are closely related to each other. In the coming sections we cover the three most important ones, as

they are widely used and provide us with the necessary material for Chapter 4. Where possible, a link between the different SGP models will be shown.

3.1.1 Sparse Pseudo-input Gaussian Process

Snelson and Ghahramani (2006) proposed the Sparse Pseudo-input Gaussian Process (SPGP) model after examining the predictive mean and variance equations of a standard GP for a single test point (Eq. 2.6 and 2.7):

$$\mu_{f_*} = \boldsymbol{k}_*^T \left(\boldsymbol{K_{ff}} + \sigma^2 \boldsymbol{I}\right)^{-1} \boldsymbol{y},$$
$$\sigma_{f_*}^2 = k_{**} - \boldsymbol{k}_*^T \left(\boldsymbol{K_{ff}} + \sigma^2 \boldsymbol{I}\right)^{-1} \boldsymbol{k}_*.$$

They noted that the values of the mean and variance are primarily characterized by the original dataset $\boldsymbol{X}$ and $\boldsymbol{y}$, each consisting of N samples. This led to the idea of replacing the original dataset with a pseudo dataset of a size M. This pseudo dataset can then be used to effectively parameterize the predictive distribution and the marginal likelihood. The computational speedup will follow because we select M to be much smaller than N, i.e. $M \ll N$. The pseudo-inputs $\boldsymbol{z}$ lie in the same domain as original inputs $\boldsymbol{x} \in \mathbb{R}^D$ and we collect them in a $M \times D$ matrix $\boldsymbol{Z} = \{\boldsymbol{z}_m\}_{m=1}^M$. The (scalar) pseudo-outputs are collected in column-vector $\boldsymbol{u}$ of size M. The pseudo-outputs are not real observation and therefore it does not make sense to include white noise on them.

With these assumptions, the pseudo dataset can be used to predict a single point of our original dataset. This comes down to using Eq. 2.6 and Eq. 2.7 and substituting the test point with a single point from the original dataset and the original dataset $\{\boldsymbol{X}, \boldsymbol{y}\}$ with the pseudo dataset $\{\boldsymbol{Z}, \boldsymbol{u}\}$

$$p(y|\boldsymbol{x}, \boldsymbol{Z}, \boldsymbol{u}) = \mathcal{N}\left(\boldsymbol{k_{fu}}\boldsymbol{K_{uu}^{-1}}\boldsymbol{u}, k_{ff} - \boldsymbol{k_{fu}}\boldsymbol{K_{uu}^{-1}}\boldsymbol{k_{fu}^\top} + \sigma^2\right),$$

where $\boldsymbol{k_{fu}}$ is the $1 \times M$ column vector storing the covariance between $\boldsymbol{x}$ and every pseudo-input from $\boldsymbol{Z}$, $\boldsymbol{K_{uu}}$ is the covariance matrix between every pair of pseudo-inputs, and $k_{ff} = k(\boldsymbol{x}, \boldsymbol{x})$ is a scalar with the self-covariance of $\boldsymbol{x}$. The key assumption made in SPGPs is that the observed outputs $\boldsymbol{y}$ become i.i.d. when they are conditioned on the pseudo dataset. This gives:

$$\begin{aligned} p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{u}) &= \prod_{n=1}^{N} p(y_n|\boldsymbol{x}_n, \boldsymbol{Z}, \boldsymbol{u}) \\ &= \mathcal{N}\left(\boldsymbol{y}|\boldsymbol{K_{fu}}\boldsymbol{K_{uu}^{-1}}\boldsymbol{u}, \operatorname{diag}(\boldsymbol{K_{ff}} - \boldsymbol{Q_{ff}}) + \sigma^2 \boldsymbol{I}_N\right), \end{aligned} \tag{3.1}$$

where we adopted a common notation to denote the low-rank *Nyström* approximation covariance matrix $\boldsymbol{Q_{ff}} \triangleq \boldsymbol{K_{fu}}\boldsymbol{K_{uu}^{-1}}\boldsymbol{K_{fu}^\top}$, which arises naturally in this derivation.

The SPGP model has hyperparameters which need to be optimized for optimal performance as well. While in the standard model there are only a few hyperparameters $\boldsymbol{\theta}$, the SPGP has in addition to the standard hyperparameters $\boldsymbol{\theta}$ also the pseudo dataset $\{\boldsymbol{Z}, \boldsymbol{u}\}$ to learn. However, instead of jointly optimizing the marginal likelihood $p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{u})$ for the complete hyperparameter set $\{\boldsymbol{\theta}, \boldsymbol{Z}, \boldsymbol{u}\}$, we choose to first marginalize out the pseudo-outputs $\boldsymbol{u}$. Therefore, we

place a prior on the pseudo-outputs,

$$p(\boldsymbol{u}) = \mathcal{N}(\boldsymbol{u}|\boldsymbol{0}, \boldsymbol{K_{uu}}). \tag{3.2}$$

This prior corresponds exactly to the original GP prior of Eq. 2.2, which is reasonable as we expect the pseudo data to be distributed in a similar fashion as the observed data. Combining Eq. 3.1 with Eq. 3.2 and using Eq. A.2 we can marginalize out the pseudo-outputs and find the marginal likelihood

$$\begin{aligned} p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{Z}) &= \int p(\boldsymbol{u}) p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{Z}, \boldsymbol{u}) \; d\boldsymbol{u} \\ &= \mathcal{N}(\boldsymbol{y}|\boldsymbol{0}, \boldsymbol{Q_{ff}} + \operatorname{diag}(\boldsymbol{K_{ff}} - \boldsymbol{Q_{ff}}) + \sigma^2 \boldsymbol{I}_N). \end{aligned} \tag{3.3}$$

This equation closely reassembles the marginal likelihood of the standard GP. The most notable difference is that the SPGP model uses a corrected *Nyström* approximation $\boldsymbol{Q_{ff}}$, where the diagonal elements of $\boldsymbol{Q_{ff}}$ are corrected with the exact covariance matrix elements from $\boldsymbol{K_{ff}}$, as covariance matrix.

The predictive distribution for a single test point $\boldsymbol{x}_*$ is obtained by considering the joint probability $p(\boldsymbol{y}, y_*)$ and then conditioning on the observed data $\boldsymbol{y}$. Again, this procedure is identical to the one followed for standard, full GPs. We get

$$y_*|\boldsymbol{x}_*, \boldsymbol{y}, \boldsymbol{X} \sim \mathcal{N}(\mu_*, \sigma_*^2), \tag{3.4}$$

with,

$$\mu_* = \boldsymbol{Q}_{*\boldsymbol{f}} \left(\boldsymbol{Q_{ff}} + \operatorname{diag}(\boldsymbol{K_{ff}} - \boldsymbol{Q_{ff}}) + \sigma^2 \boldsymbol{I}_N\right)^{-1} \boldsymbol{y}, \tag{3.5}$$

$$\sigma_*^2 = k_{**} - \boldsymbol{Q}_{*\boldsymbol{f}} \left(\boldsymbol{Q_{ff}} + \operatorname{diag}(\boldsymbol{K_{ff}} - \boldsymbol{Q_{ff}}) + \sigma^2 \boldsymbol{I}_N\right)^{-1} \boldsymbol{Q}_{*\boldsymbol{f}}^\top + \sigma^2, \tag{3.6}$$

where $\boldsymbol{Q}_{*\boldsymbol{f}} = \boldsymbol{K}_{*\boldsymbol{u}} \boldsymbol{K}_{\boldsymbol{uu}}^{-1} \boldsymbol{K}_{\boldsymbol{fu}}^\top$.

The computationally most expensive operation in Eq. 3.5 and Eq. 3.6 is the inversion of the covariance matrix of the marginal likelihood, i.e. $\boldsymbol{Q_{ff}} + \operatorname{diag}(\boldsymbol{K_{ff}} - \boldsymbol{Q_{ff}}) + \sigma^2 \boldsymbol{I}_N$. We observe that this matrix is still a $N \times N$ matrix, which requires $\mathcal{O}(N^3)$ operations for inversion. Fortunately, the matrix has a particular structure which allows us to use the Woodbury matrix identity. Rewriting $\left(\boldsymbol{Q_{ff}} + \operatorname{diag}(\boldsymbol{K_{ff}} - \boldsymbol{Q_{ff}}) + \sigma^2 \boldsymbol{I}_N\right)$ as $\left(\boldsymbol{K_{fu}} \boldsymbol{K_{uu}^{-1}} \boldsymbol{K_{fu}^\top} + \boldsymbol{A}\right)$, where $\boldsymbol{A} = \boldsymbol{\Lambda} + \sigma^2 \boldsymbol{I}_N$ and $\boldsymbol{\Lambda} = \operatorname{diag}(\boldsymbol{K_{ff}} - \boldsymbol{Q_{ff}})$, and using the Woodbury matrix identity (Eq. A.1), we get:

$$\begin{aligned} \left(\boldsymbol{K_{fu}} \boldsymbol{K_{uu}^{-1}} \boldsymbol{K_{fu}^\top} + \boldsymbol{A}\right)^{-1} &= \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1} \boldsymbol{K_{fu}} (\boldsymbol{K_{uu}} + \boldsymbol{K_{fu}^\top} \boldsymbol{A}^{-1} \boldsymbol{K_{fu}})^{-1} \boldsymbol{K_{fu}^\top} \boldsymbol{A}^{-1} \\ &= \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1} \boldsymbol{K_{fu}} \boldsymbol{B}^{-1} \boldsymbol{K_{fu}^\top} \boldsymbol{A}^{-1}, \end{aligned} \tag{3.7}$$

where $\boldsymbol{B} = \boldsymbol{K_{uu}} + \boldsymbol{K_{fu}^\top} \boldsymbol{A}^{-1} \boldsymbol{K_{fu}}$. The Woodbury identity allows us to transform the inversion of a $N \times N$ matrix to an equation where two different matrices need to be inverted, $\boldsymbol{A}$ and $\boldsymbol{B}$. At first sight this does not look desirable, but $\boldsymbol{A}$ is a $N \times N$ diagonal matrix and $\boldsymbol{B}$ is a $M \times M$ matrix, requiring $\mathcal{O}(N)$ and $\mathcal{O}(M^3)$ operations for inversion. As we assume that $M \ll N$ this significantly reduces the number of computations necessary.

The final form of the predictive distribution is found by substituting Eq. 3.7 in the first predictive distribution Eq. 3.4. Although the substitution and the subsequent simplification is quite tedious we derive it here for the completeness of this section and because it is not found (at the best of our knowledge) in the literature. We start with the predictive mean

$$\begin{aligned}\mu_* &= \boldsymbol{Q}_{*\boldsymbol{f}}\left(\boldsymbol{Q}_{\boldsymbol{ff}} + \operatorname{diag}(\boldsymbol{K}_{\boldsymbol{ff}} - \boldsymbol{Q}_{\boldsymbol{ff}}) + \sigma^2\boldsymbol{I}_N\right)^{-1}\boldsymbol{y} \\ &= \left(\boldsymbol{K}_{*\boldsymbol{u}}\boldsymbol{K}_{\boldsymbol{uu}}^{-1}\boldsymbol{K}_{\boldsymbol{fu}}^{\top}\right)\left(\boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{K}_{\boldsymbol{fu}}\boldsymbol{B}^{-1}\boldsymbol{K}_{\boldsymbol{fu}}^{\top}\boldsymbol{A}^{-1}\right)\boldsymbol{y} \\ &= \left(\boldsymbol{K}_{*\boldsymbol{u}}\boldsymbol{K}_{\boldsymbol{uu}}^{-1}\boldsymbol{K}_{\boldsymbol{fu}}^{\top}\boldsymbol{A}^{-1} - \boldsymbol{K}_{*\boldsymbol{u}}\boldsymbol{K}_{\boldsymbol{uu}}^{-1}\boldsymbol{K}_{\boldsymbol{fu}}^{\top}\boldsymbol{A}^{-1}\boldsymbol{K}_{\boldsymbol{fu}}\boldsymbol{B}^{-1}\boldsymbol{K}_{\boldsymbol{fu}}^{\top}\boldsymbol{A}^{-1}\right)\boldsymbol{y}.\end{aligned}$$

As $\boldsymbol{K}_{\boldsymbol{fu}}^{\top}\boldsymbol{A}^{-1}\boldsymbol{K}_{\boldsymbol{fu}} = \boldsymbol{B} - \boldsymbol{K}_{\boldsymbol{uu}}$, we find

$$\begin{aligned}\mu_* &= \left(\boldsymbol{K}_{*\boldsymbol{u}}\boldsymbol{K}_{\boldsymbol{uu}}^{-1}\boldsymbol{K}_{\boldsymbol{fu}}^{\top}\boldsymbol{A}^{-1} - \boldsymbol{K}_{*\boldsymbol{u}}\boldsymbol{K}_{\boldsymbol{uu}}^{-1}(\boldsymbol{B} - \boldsymbol{K}_{\boldsymbol{uu}})\boldsymbol{B}^{-1}\boldsymbol{K}_{\boldsymbol{fu}}^{\top}\boldsymbol{A}^{-1}\right)\boldsymbol{y} \\ &= \left(\cancel{\boldsymbol{K}_{*\boldsymbol{u}}\boldsymbol{K}_{\boldsymbol{uu}}^{-1}\boldsymbol{K}_{\boldsymbol{fu}}^{\top}\boldsymbol{A}^{-1}} - \cancel{\boldsymbol{K}_{*\boldsymbol{u}}\boldsymbol{K}_{\boldsymbol{uu}}^{-1}\boldsymbol{B}\boldsymbol{B}^{-1}\boldsymbol{K}_{\boldsymbol{fu}}^{\top}\boldsymbol{A}^{-1}} + \boldsymbol{K}_{*\boldsymbol{u}}\boldsymbol{K}_{\boldsymbol{uu}}{}^{-1}\boldsymbol{K}_{\boldsymbol{uu}}\boldsymbol{B}^{-1}\boldsymbol{K}_{\boldsymbol{fu}}^{\top}\boldsymbol{A}^{-1}\right)\boldsymbol{y} \\ &= \boldsymbol{K}_{*\boldsymbol{u}}\boldsymbol{B}^{-1}\boldsymbol{K}_{\boldsymbol{fu}}^{\top}\boldsymbol{A}^{-1}\boldsymbol{y}.\end{aligned}$$

A similar approach can be followed to derive the variance of the predictive distribution σ_*^2,

$$\begin{aligned}\sigma_*^2 &= k_{**} - \boldsymbol{Q}_{*\boldsymbol{f}}\left(\boldsymbol{Q}_{\boldsymbol{ff}} + \operatorname{diag}(\boldsymbol{K}_{\boldsymbol{ff}} - \boldsymbol{Q}_{\boldsymbol{ff}}) + \sigma^2\boldsymbol{I}_N\right)^{-1}\boldsymbol{Q}_{*\boldsymbol{f}}^{\top} + \sigma^2 \\ &= k_{**} - \boldsymbol{K}_{*\boldsymbol{u}}(\boldsymbol{K}_{\boldsymbol{uu}}^{-1} - \boldsymbol{B}^{-1})\boldsymbol{K}_{*\boldsymbol{u}}^{\top} + \sigma^2\end{aligned}$$

The computationally dominant operation of the predictive distribution is the matrix multiplication $\boldsymbol{K}_{\boldsymbol{fu}}^{\top}\boldsymbol{A}^{-1}\boldsymbol{K}_{\boldsymbol{fu}}$, which has a complexity of $\mathcal{O}(N + 2NM^2) = \mathcal{O}(NM^2)$. Once the required precomputations are done, the mean and the variance of a test point can be calculated in $\mathcal{O}(M)$ and $\mathcal{O}(M^2)$, respectively.

The number of hyperparameters that require optimization in the SPGP model equals $MD + |\boldsymbol{\theta}|$. Comparing this to the number in the full GP model, which equals $|\boldsymbol{\theta}|$, one is compelled to consider overfitting. However, Snelson and Ghahramani (2006) state that the pseudo points will not cause the model to overfit. They motivate this statement by considering the special case where the size of the pseudo datasets equals the size of the original dataset, i.e. $M = N$, and the pseudo dataset coincide with the original one. Then, the marginal likelihood of the SPGP will be equal to the full GP's marginal likelihood. This is because the covariance matrices will become identical $\boldsymbol{K}_{\boldsymbol{fu}} = \boldsymbol{K}_{\boldsymbol{ff}} = \boldsymbol{K}_{\boldsymbol{uu}}$, which leads to $\boldsymbol{Q}_{\boldsymbol{ff}} = \boldsymbol{K}_{\boldsymbol{ff}}$. Similarly, the predictive distribution of the SPGP will also collapse to the full GP's predictive distribution. Further, adding more pseudo points than actual data points, i.e. $M > N$, will not have any effect on the marginal likelihood nor the predictive distribution as the approximation is already equivalent to the full GP. Therefore, using a pseudo dataset larger than the original training dataset is contra-advised as this will only result in a waste of computational resources.

One of the advantages of the model is that the locations of the pseudo-inputs can be optimized together with the other hyperparameters. However, due to the large flexibility of the model, the joint optimization is a complicated case. When either the input dimension D or number of

pseudo points M is very large the optimization surface becomes highly multimodal and local optima will occur. To partially circumvent this problem, intelligent initialization of the pseudo-inputs is mandatory, e.g., k-means (Hensman et al. 2015).

Although Snelson and Ghahramani (2006) argue that overfitting does not occur easily by jointly optimizing the hyperparameters and the MD pseudo-input locations, Titsias (2009) disagrees and points out that this complex optimization problem can still lead to a form of overfitting. Therefore, in Sparse Variational Gaussian Processes (SVGPs) he further explores the sparse approximation idea by introducing a variational lower bound on the exact marginal likelihood. The pseudo-inputs and the hyperparameters are jointly selected by maximizing this variational lower bound. In this formulation, the maximization of the lower bound corresponds to the minimization of the Kullback-Leibler (KL) divergence between the variational GP and the true posterior GP. This allows to better avoid overfitting and to approximate the exact GP model by minimizing a distance between the sparse model and the exact one. In the variational model, the pseudo dataset is transformed from a set of hyperparameters to a variational distribution. This techniques significantly lowers the concerns about overfitting. A more in-depth explanation of the variational sparse approximation is given in Section 3.1.3.

Before visiting the variational approximations we briefly discuss another sparse model, called the Projected Process (PP) approximation (Seeger et al. 2003). The PP method is historically developed before the SPGP model and is derived from a different angle and in a different framework. In the next section we reformulate the model in a consistent way and derive the original marginal likelihood.

3.1.2 Projected Process Approximation

In the work of Quiñonero-Candela and Rasmussen (2005) the SPGP model is renamed to the equivalent name Fully Independent Training Conditional (FITC) approximation based on the i.i.d. assumption made in Eq. 3.1. The SPGP model is closely related to the PP sparse approximation. However, in the PP approximation the assumption is that there exists a deterministic relation between the pseudo and the observed dataset. The marginal likelihood of the latent function $\boldsymbol{f}$, given the pseudo dataset is then

$$p(\boldsymbol{f}|\boldsymbol{X}, \boldsymbol{u}, \boldsymbol{Z}) = \mathcal{N}(\boldsymbol{f}|\boldsymbol{K}_{\boldsymbol{fu}}\boldsymbol{K}_{\boldsymbol{uu}}^{-1}\boldsymbol{u}, \boldsymbol{0}).$$

As the covariance matrix is zero, this relation is deterministic. Due to this deterministic character, Seeger's PP model is systematically renamed to the Deterministic Training Conditional (DTC) approximation in Quiñonero-Candela and Rasmussen (2005).

As always, the values $\boldsymbol{y}$ are a noisy observation of the latent function values $\boldsymbol{f}$, i.e. $y_n = f_n + \epsilon_n$ and $\epsilon_n \sim \mathcal{N}(0, \sigma^2)$, which translates into the following relation with the pseudo dataset

$$p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{u}, \boldsymbol{Z}) = \mathcal{N}(\boldsymbol{y}|\boldsymbol{K}_{\boldsymbol{fu}}\boldsymbol{K}_{\boldsymbol{uu}}^{-1}\boldsymbol{u}, \sigma^2\boldsymbol{I}_N).$$

As mentioned earlier, the original derivation of the PP model differentiates from the one out-

lined here. Nonetheless, the same marginal likelihood is obtained through both ways. In this derivation we use the same approach as followed for the SPGP model for consistency. We start by marginalizing out the pseudo-outputs $\boldsymbol{u}$ by introducing the prior $p(\boldsymbol{u}) = \mathcal{N}(\boldsymbol{u}|\boldsymbol{0}, \boldsymbol{K_{uu}})$ and using Eq. A.2

$$\begin{aligned} p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{Z}) &= \int p(\boldsymbol{u}) p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{u}, \boldsymbol{Z}) d\boldsymbol{u} \\ &= \int \mathcal{N}(\boldsymbol{u}|\boldsymbol{0}, \boldsymbol{K_{uu}}) \mathcal{N}(\boldsymbol{y}|\boldsymbol{K_{fu}} \boldsymbol{K_{uu}^{-1}} \boldsymbol{u}, \sigma^2 \boldsymbol{I}_N) d\boldsymbol{u} \\ &= \mathcal{N}(\boldsymbol{y}|\boldsymbol{0}, \boldsymbol{Q_{ff}} + \sigma^2 \boldsymbol{I}_N), \end{aligned} \tag{3.8}$$

where we recall the definition of $\boldsymbol{Q_{ff}} \triangleq \boldsymbol{K_{fu}} \boldsymbol{K_{uu}^{-1}} \boldsymbol{K_{fu}^{\top}}$.

When we compare the marginal likelihoods of the SPGP and the PP method (Eq. 3.3 and Eq. 3.8) we observe that they are very similar. When removing the diagonal matrix $\text{diag}(\boldsymbol{K_{ff}} - \boldsymbol{Q_{ff}})$ from the SPGP model one immediately obtains the marginal likelihood of the PP model.

However, an important difference between the PP and SPGP method is that in the latter the hyperparameters and the pseudo-input locations are optimized together, by maximizing the marginal likelihood of Eq. 3.3. On the contrary, Seeger et al. (2003) originally developed the PP method to not use the marginal likelihood of Eq. 3.8 to learn the locations of pseudo dataset inputs. He suggests to interleave pseudo dataset selection with hyperparameter learning. In other words, in the PP model the pseudo-inputs are based on an information criteria and only the kernel hyperparameters are learned through optimization. On one side, this is advantageous as only a limited number of kernel hyperparameters is involved in the gradient based optimization. On the other side, this means that each time the pseudo dataset is updated, the optimization surface is modified and the kernel hyperparameter learning needs to be restarted. The pseudo dataset in the original PP formulation is called the active set because every point from it is an element of the training dataset. Therefore, the PP method is regarded as a hybrid combination between a sparse method, which uses an approximate likelihood, and a simple Subset of Data (SD) technique, which uses a subset of the training data.

3.1.3 Sparse Variational Gaussian Process

The SVGP model is an extension of the sparse approximation models as it uses exactly the same concepts. However, keeping overfitting in mind, the SVGP model tries to reduce the high number of hyperparameters by transforming the pseudo dataset into variational parameters. Considering the explanation above, we know that the marginal likelihood can be written as

$$\begin{aligned} p\left(\boldsymbol{y}|\boldsymbol{X}\right) &= \frac{p\left(\boldsymbol{y}, \boldsymbol{f}, \boldsymbol{u}|\boldsymbol{X}, \boldsymbol{Z}\right)}{p\left(\boldsymbol{f}, \boldsymbol{u}|\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{Z}\right)} \\ &= \frac{p\left(\boldsymbol{u}|\boldsymbol{Z}\right) p\left(\boldsymbol{f}|\boldsymbol{u}, \boldsymbol{Z}, \boldsymbol{X}\right) p\left(\boldsymbol{y}|\boldsymbol{f}\right)}{p\left(\boldsymbol{f}, \boldsymbol{u}|\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{Z}\right)}, \end{aligned} \tag{3.9}$$

where the variables have the same meaning as in Section 3.1.1. Note that we will drop the conditional variables $\boldsymbol{X}$ and $\boldsymbol{Z}$ from the notation, but implicitly assume their presence.

We can multiply and divide Eq. 3.9 by any arbitrary (variational) distribution $q(\boldsymbol{f}, \boldsymbol{u})$ to obtain

$$\begin{aligned} p\left(\boldsymbol{y}\right) &= \frac{p\left(\boldsymbol{y}, \boldsymbol{f}, \boldsymbol{u}\right)}{q(\boldsymbol{f}, \boldsymbol{u})} \frac{q(\boldsymbol{f}, \boldsymbol{u})}{p\left(\boldsymbol{f}, \boldsymbol{u}|\boldsymbol{y}\right)} \\ \Rightarrow \log p\left(\boldsymbol{y}\right) &= \log \frac{p\left(\boldsymbol{y}, \boldsymbol{f}, \boldsymbol{u}\right)}{q(\boldsymbol{f}, \boldsymbol{u})} + \log \frac{q(\boldsymbol{f}, \boldsymbol{u})}{p\left(\boldsymbol{f}, \boldsymbol{u}|\boldsymbol{y}\right)}. \end{aligned}$$

The next step is to take the expectation of both sides w.r.t. $q(\boldsymbol{f}, \boldsymbol{u})$. However, the left side is independent of the variational distribution and is thus not effected

$$\begin{aligned} \log p\left(\boldsymbol{y}\right) &= \int q(\boldsymbol{f}, \boldsymbol{u}) \log \frac{p\left(\boldsymbol{y}, \boldsymbol{f}, \boldsymbol{u}\right)}{q(\boldsymbol{f}, \boldsymbol{u})} \, d\boldsymbol{u} \, d\boldsymbol{f} + \int q(\boldsymbol{f}, \boldsymbol{u}) \log \frac{q(\boldsymbol{f}, \boldsymbol{u})}{p\left(\boldsymbol{f}, \boldsymbol{u}|\boldsymbol{y}\right)} \, d\boldsymbol{u} \, d\boldsymbol{f} \\ &= \int q(\boldsymbol{f}, \boldsymbol{u}) \log \frac{p\left(\boldsymbol{y}, \boldsymbol{f}, \boldsymbol{u}\right)}{q(\boldsymbol{f}, \boldsymbol{u})} \, d\boldsymbol{u} \, d\boldsymbol{f} + \mathrm{KL}\left[q(\boldsymbol{f}, \boldsymbol{u}) \| p\left(\boldsymbol{f}, \boldsymbol{u}|\boldsymbol{y}\right)\right], \end{aligned}$$

where the second term is the exact definition of the KL divergence between the arbitrary variational distribution $q(\boldsymbol{f}, \boldsymbol{u})$ and $p\left(\boldsymbol{f}, \boldsymbol{u}|\boldsymbol{y}\right)$. As a KL divergence is always positive the log marginal likelihood is lower bounded by the first term

$$\log p(\boldsymbol{y}) \geq \int q(\boldsymbol{f}, \boldsymbol{u}) \log \frac{p(\boldsymbol{y}, \boldsymbol{f}, \boldsymbol{u})}{q(\boldsymbol{f}, \boldsymbol{u})} \, d\boldsymbol{u} \, d\boldsymbol{f} \triangleq \mathcal{F}(q(\boldsymbol{u})). \tag{ELBO}$$

This inequality is called the Evidence Lower Bound (ELBO) and it is valid for any distribution $q(\boldsymbol{f}, \boldsymbol{u})$. Although, if we wish to obtain a tight lower bound, we benefit from minimizing the KL term. In general, the variational distribution $q(\boldsymbol{f}, \boldsymbol{u}) = q(\boldsymbol{f}|\boldsymbol{u})q(\boldsymbol{u})$ but Titsias (2009) made an interesting choice and decomposed $q(\boldsymbol{f}, \boldsymbol{u})$ in two factors $q(\boldsymbol{u})p(\boldsymbol{f}|\boldsymbol{u})$ where $p(\boldsymbol{f}|\boldsymbol{u}) = \mathcal{N}(\boldsymbol{y}|\boldsymbol{K}_{\boldsymbol{fu}}\boldsymbol{K}_{\boldsymbol{uu}}^{-1}\boldsymbol{u}, \boldsymbol{K}_{\boldsymbol{ff}} - \boldsymbol{K}_{\boldsymbol{fu}}\boldsymbol{K}_{\boldsymbol{uu}}^{-1}\boldsymbol{K}_{\boldsymbol{uf}})$ (Eq. 2.5 with renamed variables). This choice will enable the exact analytical calculation of $q(\boldsymbol{u})$. We start by substituting $q(\boldsymbol{u})p(\boldsymbol{f}|\boldsymbol{u})$ into the ELBO

$$\begin{aligned} \log p(\boldsymbol{y}) \geq \mathcal{F}(q(\boldsymbol{u})) &= \int q(\boldsymbol{u})p(\boldsymbol{f}|\boldsymbol{u}) \log \frac{p(\boldsymbol{y}, \boldsymbol{f}, \boldsymbol{u})}{q(\boldsymbol{u})p(\boldsymbol{f}|\boldsymbol{u})} \, d\boldsymbol{u} \, d\boldsymbol{f} \\ &= \int q(\boldsymbol{u})p(\boldsymbol{f}|\boldsymbol{u}) \log \frac{p(\boldsymbol{y}|\boldsymbol{f})\cancel{p(\boldsymbol{f}|\boldsymbol{u})}p(\boldsymbol{u})}{q(\boldsymbol{u})\cancel{p(\boldsymbol{f}|\boldsymbol{u})}} \, d\boldsymbol{u} \, d\boldsymbol{f} \\ &= \int q(\boldsymbol{u})p(\boldsymbol{f}|\boldsymbol{u}) \log \frac{p(\boldsymbol{u})}{q(\boldsymbol{u})} d\boldsymbol{u} d\boldsymbol{f} + \int q(\boldsymbol{u})p(\boldsymbol{f}|\boldsymbol{u}) \log p(\boldsymbol{y}|\boldsymbol{f}) \, d\boldsymbol{u} \, d\boldsymbol{f}. \end{aligned} \tag{3.10}$$

We wish to maximize this equation by intelligently choosing the variational distribution $q(\boldsymbol{u})$. The optimal form for $q(\boldsymbol{u})$ is found by differentiating $\mathcal{F}(q(\boldsymbol{u}))$, extended with a Lagrange multiplier to enforce the normalization, w.r.t. $q(\boldsymbol{u})$

$$\begin{aligned} \frac{d\,\mathcal{L}}{d\,q(\boldsymbol{u})} &= \frac{d}{d\,q(\boldsymbol{u})} \left\{ \mathcal{F}(q(\boldsymbol{u})) + \lambda \left(\int q(\boldsymbol{u}) \, d\boldsymbol{u} - 1 \right) \right\} \\ &= \frac{d\,\mathcal{F}(q(\boldsymbol{u}))}{d\,q(\boldsymbol{u})} + \lambda \\ &= \int p(\boldsymbol{f}|\boldsymbol{u}) \left(\log \frac{p(\boldsymbol{u})}{q(\boldsymbol{u})} - 1 \right) d\boldsymbol{f} + \int p(\boldsymbol{f}|\boldsymbol{u}) \log p(\boldsymbol{y}|\boldsymbol{f}) \, d\boldsymbol{f} + \lambda \\ &= \log p(\boldsymbol{u}) - \log q(\boldsymbol{u}) - 1 + \int p(\boldsymbol{f}|\boldsymbol{u}) \log p(\boldsymbol{y}|\boldsymbol{f}) \, d\boldsymbol{f} + \lambda. \end{aligned}$$

This should be set to 0 to find the optimal form of $q(\boldsymbol{u})$

$$\begin{aligned} q(\boldsymbol{u}) &= \frac{p(\boldsymbol{u}) \exp\left(\int p(\boldsymbol{f}|\boldsymbol{u}) \log p(\boldsymbol{y}|\boldsymbol{f})\, d\boldsymbol{f}\right)}{\exp(1-\lambda)} \\ &= \frac{p(\boldsymbol{u}) H(\boldsymbol{u}, \boldsymbol{y})}{Z}, \end{aligned} \tag{3.11}$$

where $H(\boldsymbol{u}, \boldsymbol{y}) = \exp\left(\int p(\boldsymbol{f}|\boldsymbol{u}) \log p(\boldsymbol{y}|\boldsymbol{f})\, d\boldsymbol{f}\right)$ and $Z = \exp(1-\lambda)$ takes the role of a normalization constant, to fulfill the Lagrangian constrain.

Substituting the optimal form of $q(\boldsymbol{u})$ in the ELBO (Eq. 3.10) gives

$$\begin{aligned} \mathcal{F}(q(\boldsymbol{u})) &= \int q(\boldsymbol{u}) p(\boldsymbol{f}|\boldsymbol{u}) \log \frac{p(\boldsymbol{y}|\boldsymbol{f}) p(\boldsymbol{u}) Z}{p(\boldsymbol{u}) H(\boldsymbol{u}, \boldsymbol{y})}\, d\boldsymbol{u}\, d\boldsymbol{f} \\ &= \int q(\boldsymbol{u}) p(\boldsymbol{f}|\boldsymbol{u}) \left(\log Z + \log p(\boldsymbol{y}|\boldsymbol{f}) - \log H(\boldsymbol{u}, \boldsymbol{y})\right)\, d\boldsymbol{u}\, d\boldsymbol{f} \\ &= \int q(\boldsymbol{u}) p(\boldsymbol{f}|\boldsymbol{u}) \left(\log Z + \log p(\boldsymbol{y}|\boldsymbol{f}) - \int p(\boldsymbol{f}|\boldsymbol{u}) \log p(\boldsymbol{y}|\boldsymbol{f}) d\boldsymbol{f}\right)\, d\boldsymbol{u}\, d\boldsymbol{f} \\ &= \int q(\boldsymbol{u}) \left(\log Z + \int p(\boldsymbol{f}|\boldsymbol{u}) \log p(\boldsymbol{y}|\boldsymbol{f})\, d\boldsymbol{f} - \int p(\boldsymbol{f}|\boldsymbol{u}) \log p(\boldsymbol{y}|\boldsymbol{f})\, d\boldsymbol{f}\right)\, d\boldsymbol{u} \\ &= \log Z. \end{aligned}$$

We derived that the ELBO equals the log of the normalization constant Z of the variational distribution $q(\boldsymbol{u})$. In order to find an analytical form for $\log Z$ we derive $q(\boldsymbol{u})$ (Eq. 3.11) and then use the normalization constrain.

We begin the calculation of $q(\boldsymbol{u})$ with $\log H(\boldsymbol{u}, \boldsymbol{y})$. For notational simplicity we write $p(\boldsymbol{f}|\boldsymbol{u})$ as $\mathcal{N}(\boldsymbol{f}|\boldsymbol{M}\boldsymbol{u}, \boldsymbol{\Sigma}_{\boldsymbol{f}})$

$$\begin{aligned} \log H(\boldsymbol{u}, \boldsymbol{y}) &= \int p(\boldsymbol{f}|\boldsymbol{u}) \log p(\boldsymbol{y}|\boldsymbol{f})\, d\boldsymbol{f} \\ &= \int \mathcal{N}(\boldsymbol{f}|\boldsymbol{M}\boldsymbol{u}, \boldsymbol{\Sigma}_{\boldsymbol{f}}) \log \mathcal{N}(\boldsymbol{y}|\boldsymbol{f}, \sigma^2 \boldsymbol{I})\, d\boldsymbol{f} \\ &= \int \mathcal{N}(\boldsymbol{f}|\boldsymbol{M}\boldsymbol{u}, \boldsymbol{\Sigma}_{\boldsymbol{f}}) \left(-\frac{N}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} (\boldsymbol{y} - \boldsymbol{f})^\top \boldsymbol{I} (\boldsymbol{y} - \boldsymbol{f})\right) d\boldsymbol{f} \\ &= -\frac{N}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \int \mathcal{N}(\boldsymbol{f}|\boldsymbol{M}\boldsymbol{u}, \boldsymbol{\Sigma}_{\boldsymbol{f}})\, \mathrm{tr}\left(\boldsymbol{y}\boldsymbol{y}^\top - 2\boldsymbol{y}\boldsymbol{f}^\top + \boldsymbol{f}\boldsymbol{f}^\top\right)\, d\boldsymbol{f} \\ &= -\frac{N}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \mathrm{tr}\left(\boldsymbol{y}\boldsymbol{y}^\top - 2\boldsymbol{y}(\boldsymbol{M}\boldsymbol{u})^\top + (\boldsymbol{M}\boldsymbol{u})(\boldsymbol{M}\boldsymbol{u})^\top + \boldsymbol{\Sigma}_{\boldsymbol{f}}\right) \\ &= \frac{1}{2\sigma^2} \mathrm{tr}\left(\boldsymbol{\Sigma}_{\boldsymbol{f}}\right) + \log \mathcal{N}(\boldsymbol{y}|\boldsymbol{M}\boldsymbol{u}, \sigma^2 \boldsymbol{I}). \end{aligned}$$

Using this result, we can derive a closed form for $q(\boldsymbol{u})$,

$$\begin{aligned} q(\boldsymbol{u}) &= \frac{1}{Z} p(\boldsymbol{u}) \exp\left(\frac{1}{2\sigma^2} \mathrm{tr}\left(\boldsymbol{\Sigma}_{\boldsymbol{f}}\right) + \log \mathcal{N}(\boldsymbol{y}|\boldsymbol{M}\boldsymbol{u}, \sigma^2 \boldsymbol{I})\right) \\ &= \frac{1}{Z} \exp\left(\frac{1}{2\sigma^2} \mathrm{tr}\left(\boldsymbol{\Sigma}_{\boldsymbol{f}}\right)\right)\, p(\boldsymbol{u})\, \mathcal{N}(\boldsymbol{y}|\boldsymbol{M}\boldsymbol{u}, \sigma^2 \boldsymbol{I}). \end{aligned}$$

Due to the symmetry of the Gaussian distribution $\mathcal{N}(\boldsymbol{y}|\boldsymbol{M}\boldsymbol{u}, \sigma^2 \boldsymbol{I})$ in $\boldsymbol{u}$ and $\boldsymbol{y}$ it is possible to

swap the variables if a correcting term is added. Further, $\frac{1}{Z}\exp\left(\frac{1}{2\sigma^2}\mathrm{tr}\left(\boldsymbol{\Sigma_f}\right)\right)$ is renamed to C and the definition of $p(\boldsymbol{u}) = \mathcal{N}(\boldsymbol{u}|\boldsymbol{0}, \boldsymbol{K_{uu}})$ is used

$$\begin{aligned} q(\boldsymbol{u}) &= C\sqrt{\frac{|2\pi\sigma^2\boldsymbol{M}^\top\boldsymbol{M}|}{|2\pi\sigma^2\boldsymbol{I}|}}\mathcal{N}(\boldsymbol{u}|\boldsymbol{0}, \boldsymbol{K_{uu}})\,\mathcal{N}(\boldsymbol{u}|\boldsymbol{M}^{-1}\boldsymbol{y}, \sigma^2\boldsymbol{M}^{-1}\boldsymbol{M}^{-\top}) \\ &= C\sqrt{\frac{|2\pi\sigma^2\boldsymbol{M}^\top\boldsymbol{M}|}{|2\pi\sigma^2\boldsymbol{I}|}}\mathcal{N}(\boldsymbol{M}^{-1}\boldsymbol{y}|\boldsymbol{0}, \boldsymbol{K_{uu}}+\boldsymbol{B})\,\mathcal{N}(\boldsymbol{u}|\boldsymbol{L}\boldsymbol{B}^{-1}\boldsymbol{M}^{-1}\boldsymbol{y}, \boldsymbol{L}), \end{aligned}$$

where $\boldsymbol{B} = \sigma^2\boldsymbol{M}^{-1}\boldsymbol{M}^{-\top}$ and $\boldsymbol{L} = (\boldsymbol{K_{uu}^{-1}} + \boldsymbol{B}^{-1})^{-1}$. We used the convenient identity of Eq. A.3 to analytically calculate the product of both normals.

As $q(\boldsymbol{u})$ is a distribution we know that

$$C\sqrt{\frac{|2\pi\sigma^2\boldsymbol{M}^\top\boldsymbol{M}|}{|2\pi\sigma^2\boldsymbol{I}|}}\mathcal{N}(\boldsymbol{M}^{-1}\boldsymbol{y}|\boldsymbol{0}, \boldsymbol{K_{uu}}+\boldsymbol{B}) = 1, \tag{3.12}$$

which completes the derivation of the variational distribution $q(\boldsymbol{u})$

$$\begin{aligned} q(\boldsymbol{u}) &= \mathcal{N}(\boldsymbol{u}|\boldsymbol{L}\boldsymbol{B}^{-1}\boldsymbol{M}^{-1}\boldsymbol{y}, \boldsymbol{L}) \\ &= \mathcal{N}(\boldsymbol{u}|\boldsymbol{K_{uu}}(\boldsymbol{K_{uu}} + \sigma^{-2}\boldsymbol{K_{uf}}\boldsymbol{K_{fu}})^{-1}\boldsymbol{K_{uf}}\boldsymbol{y}, \boldsymbol{K_{uu}}(\boldsymbol{K_{uu}} + \sigma^{-2}\boldsymbol{K_{uf}}\boldsymbol{K_{fu}})^{-1}\boldsymbol{K_{uu}}). \end{aligned}$$

We obtain Z by using the equality of Eq. 3.12 and substituting the definition of C

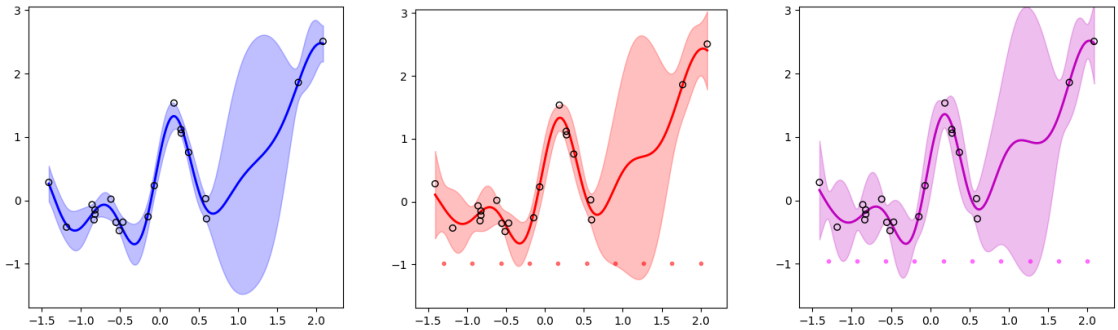
$$\begin{aligned} Z &= \exp\left(\frac{1}{2\sigma^2}\mathrm{tr}\left(\boldsymbol{\Sigma_f}\right)\right)\sqrt{\frac{|2\pi\sigma^2\boldsymbol{M}^\top\boldsymbol{M}|}{|2\pi\sigma^2\boldsymbol{I}|}}\mathcal{N}(\boldsymbol{M}^{-1}\boldsymbol{y}|\boldsymbol{0}, \boldsymbol{K_{uu}}+\boldsymbol{B}) \\ &= \exp\left(\frac{1}{2\sigma^2}\mathrm{tr}\left(\boldsymbol{\Sigma_f}\right)\right)\sqrt{\frac{|2\pi\sigma^2\boldsymbol{M}^\top\boldsymbol{M}|}{|2\pi\sigma^2\boldsymbol{I}|}}\sqrt{\frac{|2\pi(\boldsymbol{K_{uu}}+\boldsymbol{B})|}{|2\pi\boldsymbol{M}(\boldsymbol{K_{uu}}+\boldsymbol{B})\boldsymbol{M}^\top|}}\mathcal{N}(\boldsymbol{y}|\boldsymbol{0}, \boldsymbol{M}(\boldsymbol{K_{uu}}+\boldsymbol{B})\boldsymbol{M}^\top) \\ &= \exp\left(\frac{1}{2\sigma^2}\mathrm{tr}\left(\boldsymbol{\Sigma_f}\right)\right)\mathcal{N}(\boldsymbol{y}|\boldsymbol{0}, \boldsymbol{M}(\boldsymbol{K_{uu}}+\boldsymbol{B})\boldsymbol{M}^\top). \end{aligned}$$

The ELBO $\mathcal{F}(q(\boldsymbol{u}))$ equals $\log Z$, therefore

$$\begin{aligned} \mathcal{F}(q(\boldsymbol{u})) &= -\frac{1}{2\sigma^2}\mathrm{tr}\left(\boldsymbol{\Sigma_f}\right) + \log\mathcal{N}(\boldsymbol{y}|\boldsymbol{0}, \boldsymbol{M}(\boldsymbol{K_{uu}}+\boldsymbol{B})\boldsymbol{M}^\top) \\ &= -\frac{1}{2\sigma^2}\mathrm{tr}\left(\boldsymbol{K_{ff}} - \boldsymbol{K_{fu}}\boldsymbol{K_{uu}^{-1}}\boldsymbol{K_{uf}}\right) + \log\mathcal{N}(\boldsymbol{y}|\boldsymbol{0}, \sigma^2\boldsymbol{I} + \boldsymbol{K_{fu}}\boldsymbol{K_{uu}^{-1}}\boldsymbol{K_{uf}}). \end{aligned} \tag{3.13}$$

This completes the tedious derivation of the evidence lower bound of the SVGP model. To the best of our knowledge, a more detailed and in-depth mathematical derivation is not found in the literature. Some final remarks:

1. The obtained bound is very similar to the marginal likelihood of the PP approximation (Eq. 3.8) except for the trace, which acts as a penalty term.



(a) Full GP (b) SPGP (c) PP

Figure 3.1: Comparison of the full GP, SPGP and PP sparse approximations. The colored line depicts the predictive mean, the shaded part shows the uncertainty region $\pm 2\,\sigma$, the circle markers show the position of the training data, and in the sparse approximations are the pseudo-input locations displayed with dots on the x-axis.

Table 3.1: The numerical values of the hyperparameters after optimization. The hyperparameters σ_{SE} and ℓ are respectively the signal variance and lengthscale, associated with the SE kernel.

Method	σ_{noise}	σ_{SE}	ℓ
Full GP	0.14	1.01	0.30
SPGP	0.09	1.03	0.32
PP	0.15	1.08	0.27

2. An important difference is that the marginal likelihoods of the PP and SPGP models is an approximation, based on a certain assumption, while the ELBO of the SVGP model is a lower bound of the marginal likelihood.

3. As we were able to analytically calculate the optimal form for the variational distribution $q(\boldsymbol{u})$, we can not consider $\boldsymbol{u}$ as truly variational parameters, these variables are *collapsed*.

3.1.4 Experiments

In this section we present the results of two comparison experiments. In the first one we compare the SPGP and PP sparse approximation models from a more theoretical perspective. In the second experiment we benchmark the SPGP, PP and SVGP models on a real-world dataset and evaluate their execution time and predictive performance. The benchmark code makes use of implementations provided by GPy (GPy 2014). The experiments ran on a machine powered by 8 Intel i7-6700HQ CPU @ 2.60GHz and 16 GB RAM. The three models use the SE kernel function.

The first experiment, shown in Figure 3.1, can be seen as a toy example on a synthetic dataset. In the figure, the predictive mean and variance is plotted for the full GP, SPGP and PP models. The sparse approximations are using a pseudo dataset of 10 samples, uniformly distributed over the input space. In this experiment we fix the locations of the pseudo-inputs, which allows us to compare the performance of the different marginal likelihoods for hyperparameter optimization.

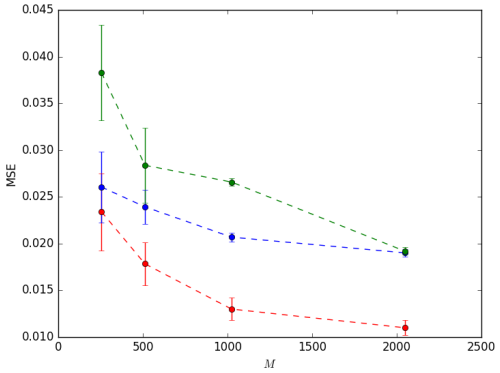


Figure 3.2: The MSE of the SPGP (blue), PP (red) and SVGP (green) approximations in function of the pseudo dataset size M on the SARCOS, inverse dynamics dataset. The error bars show $\pm$ one standard deviation.

Table 3.2: Results of the comparison experiment between the PP, SPGP and SVGP approximations on the SARCOS, inverse dynamics dataset. M stands for the size of the pseudo dataset and MSE is the Mean Squared Error. The MSE and runtime are the averages of running the experiment 5 times. For the MSE the standard deviation is shown as well.

Method	M	MSE	runtime (s)
PP	256	0.023 ± 0.0041	164.38
	512	0.018 ± 0.0023	402.76
	1024	0.013 ± 0.0012	1095.48
	2048	0.011 ± 0.0008	3560.48
SPGP	256	0.026 ± 0.0038	175.73
	512	0.024 ± 0.0018	412.46
	1024	0.021 ± 0.0005	1140.14
	2048	0.019 ± 0.0004	3747.86
SVGP	256	0.038 ± 0.0051	179.35
	512	0.028 ± 0.004	550.93
	1024	0.026 ± 0.0002	1949.11
	2048	0.019 ± 0.0003	6812.61

Table 3.1 shows the the optimized hyperparameter values. We observe that the PP method chooses a lower lengthscale but higher signal and noise variance, which explains the more pronounced fluctuation in the scarce region. In general, the SPGP model is prone to severely underestimating the noise variance. The model is able to shrunk the noise to a smaller value, without forcing the predictive mean to go through every data point. This is possible as the SPGP model has a heteroscedastic (input dependent) noise term $\text{diag}(\boldsymbol{K_{ff}} - \boldsymbol{Q_{ff}})$ in the marginal likelihood. The term will cause an increase of the model's noise for data far from the pseudo-inputs and far from the predictive mean, which advocates a smaller overall noise variance (Bauer et al. 2016).

In the second experiment the models are benchmarked on a real-world dataset, called SARCOS. The dataset holds the inverse dynamics of a seven Degree of Freedom (DoF) robotic arm. The objective is to predict the torque of a single joint (1D), given a 21-dimensional input (7 joint

position, 7 joint velocities and 7 joint accelerations). Not surprisingly, this regression problem is highly non-linear. The dataset consists out of $44,484$ training samples and $4,449$ test samples. The dataset has previously been used in Rasmussen and Williams (2006) and Vijayakumar et al. (2002). Before the experiment, the dataset was normalized to have zero mean and unit variance.

Using a full GP model on this dataset is practically impossible due to its size. As a rule of thumb, one can assume that a full GP can be successfully applied, with a reasonable execution time on a modern workstation, when the dataset has around $10,000$ data points, each having a couple of features. The SARCOS dataset does not meet these restrictions and therefore we need to reside to approximating models. In this experiment we test the PP, SPGP and VSGP models. Table 3.2 and Figure 3.2 report the average Mean Square Error (MSE) and the execution time in relation to the number of pseudo-inputs. We observe an improved performance but a longer execution time with increasing number of pseudo data. Both observations are quite obvious as the training complexity of the models scale quadratically with M, and the predictive accuracy depends on the amount of information captured by the pseudo data, which typically increases when the pseudo dataset gets larger.

In conclusion, the chief attribute of sparse approximations is to approximate the marginal likelihood $p(\boldsymbol{y}|\boldsymbol{u}, \boldsymbol{Z}, \boldsymbol{X})$ by replacing the $N \times N$ covariance matrix $\boldsymbol{K_{ff}}$ in (2.4) by a low-rank matrix $\boldsymbol{K_{uu}}$. As the approximation is based on a set of M pseudo input - output pairs, with $M \ll N$, the computational complexity of the model is lowered. The quality of the sparse model, however, now highly depends on the M pseudo inputs and outputs. As we have seen in the experiments, the SGP and VSGP models are able to handle large datasets more easily by being computational very efficient but have to comprise on accuracy in doing so. These models are also a fundamental building block for Deep Gaussian Processes (DGPs), discussed in Section 4.3.

3.2 Cholesky decomposition

3.2.1 Background

When implementing the full GP or any approximate GP-based model, one is inevitably faced with the inversion of a symmetric positive definite matrix $\boldsymbol{\Sigma}$ (e.g., Eq. 2.4, 3.8 and 3.13), which has a cubic complexity depending on the size of the matrix. This inversion is practically always the computational bottleneck of every GP model. The approaches of the previous section seek to alleviate this problem by introducing an approximation which requires the inversion of a small matrix instead of a big matrix containing all the input samples. On the contrary, in this section we do not consider any particular model but rather focus on speeding up the calculation of $\boldsymbol{\Sigma}^{-1}$ for a general matrix $\boldsymbol{\Sigma}$. This enhanced algorithm/implementation can then be used where necessary (e.g., Eq. 2.4, 3.8 and 3.13).

Remarkably, the inverse $\boldsymbol{\Sigma}^{-1}$ itself is rarely required. More common forms are $\boldsymbol{v}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{v}$ and $\boldsymbol{\Sigma}^{-1} \boldsymbol{v}$. The most computationally stable and efficient way to obtain these is via the Cholesky decomposition. The Cholesky decomposition $\boldsymbol{L}$ of a symmetric positive definite matrix $\boldsymbol{\Sigma}$ is

defined as the unique lower-triangular matrix with positive diagonal elements, satisfying $\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^\top$. The Cholesky decomposition or factorization can intuitively be seen as the square root of a matrix as the matrix equivalent of α^2 is $\boldsymbol{A}\boldsymbol{A}^\top$.

The Cholesky decomposition has many application. In practice it is often used for solving a system of linear equations, although in a GP setting it is most often involved in the following calculations:

1. $\boldsymbol{v}^\top \boldsymbol{\Sigma}^{-1}\boldsymbol{v} = \boldsymbol{v}^\top (\boldsymbol{L}\boldsymbol{L}^\top)^{-1}\boldsymbol{v} = \|\boldsymbol{L}^{-1}\boldsymbol{v}\|^2$
2. $\boldsymbol{\Sigma}^{-1}\boldsymbol{v} = \boldsymbol{L}^{-\top}(\boldsymbol{L}^{-1}\boldsymbol{v})$
3. $|\boldsymbol{\Sigma}| = \prod_{n=1}^{N} \boldsymbol{L}_{n,n}^2$
4. Sampling from a Gaussian multivariate distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is possible by sampling $\boldsymbol{u} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and and transforming it via $\boldsymbol{\mu} + \boldsymbol{L}\boldsymbol{u}$.

The computational complexity of the Cholesky decomposition grows cubically with the size of the matrix, i.e. $\mathcal{O}(N^3)$. While this equals the computational complexity of a matrix inversion, the Cholesky is typically preferred as it is numerically more stable and the implementations are, in practice, faster. Additionally, once the Cholesky decomposition is calculated for a given matrix the same decomposition can be reused for other calculations, which is often the case in GP models (e.g., Eq. 2.4).

TensorFlow provides a Cholesky implementation, which is a lightweight Python wrapper around a C++ Eigen implementation. Although of using a highly optimized C++ routine, choosing this library for the Cholesky computation introduces a major limitation: the Eigen implementation can not be executed on GPU. This causes a significant drop in execution speed and performance. Even more, when combining different algorithms, where some are able to exploit GPU hardware and others are not, the effects accumulate and the consequences become even more problematic. This is due to large data transfer times when copying data from GPU to CPU and vice versa, as shown in Figure 3.3. The figure shows which parts of the code run on the GPU and which parts run on the CPU, during a single optimization round of the GPR model. Both, the Cholesky and CholeskyGrad algorithm, have no GPU support and will therefore always be executed on the CPU. This results in large transfer times and supplementary synchronization between the GPU and CPU execution nodes.

The execution trace can be significantly optimized if both CPU-only parts of the code (i.e. Cholesky and CholeskyGrad algorithms) could also be executed on the GPU. This optimization would speedup the execution in two ways. First, the large transfer times would be reduced as we only use a single execution node and thus limit the necessity of copying data between the nodes. Secondly, for most matrix operation algorithms, GPU implementations are faster than CPU variants as they are able to benefit from the massively parallel architecture consisting of millions of smaller and more efficient cores designed for handling multiple tasks simultaneously. These observations inspired us to implement a TensorFlow compatible Cholesky implementation which can be executed on CPU as well as the GPU. This implementation can then be integrated

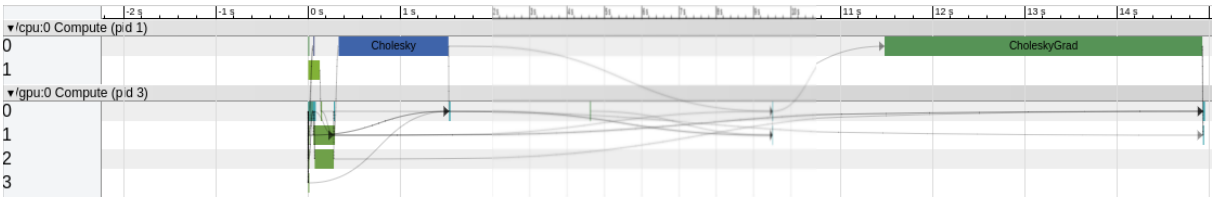


Figure 3.3: Computational trace (generated using TensorFlow visualization tools) of a single optimization round of a GP. Note that the middle of the trace is downscaled. We are able to pinpoint two problems: (1) there is a large overhead due to transferring data from CPU to GPU, and vice versa, (2) the methods that run on the CPU (cholesky and choleskyGrad) would execute faster if they would run on the GPU.

in existing models of GPflow, a GP library that uses TensorFlow for its core computations (Matthews et al. 2017), in order to speedup the current learning and inferences phases, which make extensive use of the Cholesky decomposition.

3.2.2 The Blocked-Partitioned Cholesky Algorithm

The goal of this part is to implement of a Cholesky decomposition algorithm in basic TensorFlow operations, such as slicing, padding, matrix addition and multiplication. TensorFlow provides for each of these basic operations a hybrid CPU and GPU implementation. Therefore, if the new Cholesky implementation constitutes solely out of these operations, it will automatically execute on CPU and GPU.

Our implementation of the Cholesky decomposition is based on a block-partitioned algorithm, which is designed to exploit the efficient, cache-friendly and easy-to-parallelize BLAS 3 operations (matrix-matrix multiplication). In fact, it is a clear reformulation of the basic scalar Cholesky factorization algorithm, but instead of using scalar operations, the blocked algorithm will advocate the use of matrix-matrix operations. The same algorithm is implemented in the ScaLAPACK and the LAPACK library (Choi et al. 1996, Higham 2002, Murray 2016), which were also our main resources.

The remainder of this section is organized as follows. We start with explaining the scalar (unblocked) version of the Cholesky algorithm. We do this for two reasons: first, the unblocked version is internally used in the blocked variant and secondly, explaining the blocked algorithm as an extension of the unblocked eases understanding. Once both algorithms are explained, we report their performance.

Unblocked version

In the coming paragraph, we cover an algorithm that constructs the decomposition of $\boldsymbol{\Sigma}$, a $N \times N$ symmetric positive definite, by $\boldsymbol{L}$, a $N \times N$ lower-triangular matrix, so that $\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^\top$. It is an in-place algorithm, which means that the resulting decomposition $\boldsymbol{L}$ will be stored in the lower-triangular part of the input matrix $\boldsymbol{\Sigma}$. The algorithm can be described by considering

partitioned matrices

$$\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_{11} & \boldsymbol{\sigma}_{21}^\top \\ \boldsymbol{\sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix} \text{ and } \boldsymbol{L} = \begin{pmatrix} \ell_{11} & \mathbf{0}^\top \\ \boldsymbol{\ell}_{21} & \boldsymbol{L}_{22} \end{pmatrix},$$

where σ_{11} and ℓ_{11} are scalars, $\boldsymbol{\sigma}_{21}$ and $\boldsymbol{\ell}_{21}$ are column-vectors of length $N-1$ and $\boldsymbol{\Sigma}_{22}$ and $\boldsymbol{L}_{22}$ are the right-lower submatrices of $\boldsymbol{\Sigma}$ and $\boldsymbol{L}$, respectively, with dimension $N-1 \times N-1$. We wish that

$$\begin{aligned} \boldsymbol{\Sigma} = \begin{pmatrix} \sigma_{11} & \boldsymbol{\sigma}_{21}^\top \\ \boldsymbol{\sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix} &= \boldsymbol{L}\boldsymbol{L}^\top \\ &= \begin{pmatrix} \ell_{11} & \mathbf{0}^\top \\ \boldsymbol{\ell}_{21} & \boldsymbol{L}_{22} \end{pmatrix} \begin{pmatrix} \ell_{11} & \boldsymbol{\ell}_{21}^\top \\ \mathbf{0} & \boldsymbol{L}_{22}^\top \end{pmatrix} \\ &= \begin{pmatrix} \ell_{11}^2 & \ell_{11}\boldsymbol{\ell}_{21}^\top \\ \ell_{11}\boldsymbol{\ell}_{21} & \boldsymbol{\ell}_{21}\boldsymbol{\ell}_{21}^\top + \boldsymbol{L}_{22}\boldsymbol{L}_{22}^\top \end{pmatrix}, \end{aligned}$$

therefore we derive the following update equations

$$\begin{aligned} \sigma_{11} &\leftarrow \ell_{11} = \sqrt{\sigma_{11}}, \\ \boldsymbol{\sigma}_{21} &\leftarrow \boldsymbol{\ell}_{21} = \boldsymbol{\sigma}_{21}/\ell_{11}, \\ \boldsymbol{\Sigma}_{22} &\leftarrow \boldsymbol{L}_{22} = \boldsymbol{\Sigma}_{22} - \boldsymbol{\ell}_{21}\boldsymbol{\ell}_{21}^\top. \end{aligned}$$

Continuing this procedure recursively on $\boldsymbol{\Sigma}_{22}$ leads to the discovery of the Cholesky decomposition $\boldsymbol{L}$ of the matrix $\boldsymbol{\Sigma}$, where $\boldsymbol{L}$ is stored in the lower-triangular part of $\boldsymbol{\Sigma}$. Note that the above derivation is not a closed mathematical proof, more an intuitive approach for implementing the Cholesky factorization. For the sake of completeness we derived and enclosed the proof of this algorithm in Appendix B.

Blocked version

In order to have a greater speedup and to attain high performance, we reformulate the above algorithm in terms of matrix-matrix operations (i.e. BLAS 3). As mentioned earlier, these operation are highly efficient on both CPU and GPU as they are cache-friendly and have extremely parallel and optimized assembly implementations. The algorithm is straightforwardly called the blocked-partitioned Cholesky decomposition algorithm and can be derived by partitioning $\boldsymbol{\Sigma}$ and $\boldsymbol{L}$ in the following block matrices

$$\boldsymbol{\Sigma} = \left(\begin{array}{c|c} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{21}^\top \\ \hline \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{array}\right) \text{ and } \boldsymbol{L} = \left(\begin{array}{c|c} \boldsymbol{L}_{11} & \mathbf{0} \\ \hline \boldsymbol{L}_{21} & \boldsymbol{L}_{22} \end{array}\right),$$

where $\boldsymbol{\Sigma}_{11}$ and $\boldsymbol{L}_{11}$ are in $\mathbb{R}^{N_b \times N_b}$, $\boldsymbol{\Sigma}_{21}$ and $\boldsymbol{L}_{21}$ are in $\mathbb{R}^{N_b \times (N-N_b)}$, $\boldsymbol{\Sigma}_{22}$ and $\boldsymbol{L}_{22}$ are elements of $\mathbb{R}^{(N-N_b)\times(N-N_b)}$ and N_b is a arbitrarily chosen submatrix size.

The blocked-partitioned Cholesky decomposition algorithm is then described by

$$\boldsymbol{\Sigma}_{11} \leftarrow \boldsymbol{L}_{11} = \textsc{Cholesky}\left(\boldsymbol{\Sigma}_{11}\right), \tag{3.14}$$

$$\boldsymbol{\Sigma}_{21} \leftarrow \boldsymbol{L}_{21} = \boldsymbol{\Sigma}_{21}\boldsymbol{L}_{11}^{-T},$$
$$\boldsymbol{\Sigma}_{22} \leftarrow \boldsymbol{L}_{22} = \boldsymbol{\Sigma}_{22} - \boldsymbol{L}_{21}\boldsymbol{L}_{21}^{\top}.$$

Recursively applying these equations on submatrix $\boldsymbol{\Sigma}_{22}$ will result in finding the Cholesky decomposition. The attentive reader will have noticed the remarkable resemblance between the blocked and basic algorithms. In the blocked variant, (1) the $\sqrt{\sigma_{11}}$ is replaced by the Cholesky decomposition $\textsc{Cholesky}\,(\boldsymbol{\Sigma}_{11})$ (which is simplistically spoken the matrix equivalent of a square root), (2) the multiplication with ℓ_{11}^{-1} is replaced by a multiplication with the transposed inverse $\boldsymbol{L}_{11}^{-T}$ and (3) the vector-vector multiplication $\boldsymbol{\ell}_{21}\boldsymbol{\ell}_{21}^{\top}$ is transformed into a matrix-matrix multiplication $\boldsymbol{L}_{21}\boldsymbol{L}_{21}^{\top}$.

In our implementation, we use the unblocked, basic algorithm as the Cholesky-routine for the calculation of $\boldsymbol{L}_{11}$ in Eq. 3.14. In the next section we will show that the unblocked algorithm is reasonably fast for small matrices, which justifies this decision. Another option would be to recursively use the blocked variant with a smaller block size $N_b' < N_b$, but this was found experimentally to be less efficient.

3.2.3 Performance

We provide performance results of the unblocked and blocked-partitioned Cholesky decomposition and compare it with the default implementation of TensorFlow[1]. The performance results are reported for both GPU and CPU execution. Selecting GPU or CPU execution can easily be done with TensorFlow by choosing the correct version of the library before executing the benchmark code. However, care should be taken as selecting TensorFlow's GPU version will not result in executing TensorFlow's Cholesky implementation on the GPU. Operations that are not GPU compatible (e.g., Cholesky and CholeskyGrad), will always be executed on the CPU despite of selecting the GPU version of the TensorFlow library. Therefore, note that in the plots below TensorFlow's Cholesky implementation (blue) has always been executed on the CPU and is added to the GPU plots for illustrative purposes. All experiments were performed using double-precision floating-point arithmetic on a machine powered by 40 Intel Xeon (E5-2640) CPUs @ 2.40GHz, each having a cache size of 25.6 MB. The machine is also equipped with two Nvidia GeForce GTX 1080 GPUs, each having 8112 MB of memory available.

Figure 3.4 plots the execution time of the different Cholesky algorithms in function of the matrix size N. We observe that the unblocked algorithm (magenta) is significantly slower than the blocked variants. Also, the execution time of all algorithms increases with N, as expected due to the cubic scaling of the complexity. As we use the unblocked algorithm (magenta) for the calculation of $\textsc{Cholesky}\,(\boldsymbol{\Sigma}_{11})$ in the blocked algorithm (Eq. 3.14) it is intuitively clear that we should use a relatively small block size N_b. However, choosing N_b too small would result in too much recursive steps and the associated overhead. This trade-off motivates the choice for $N_b \approx 200$.

[1]The experiments use TensorFlow 1.0.

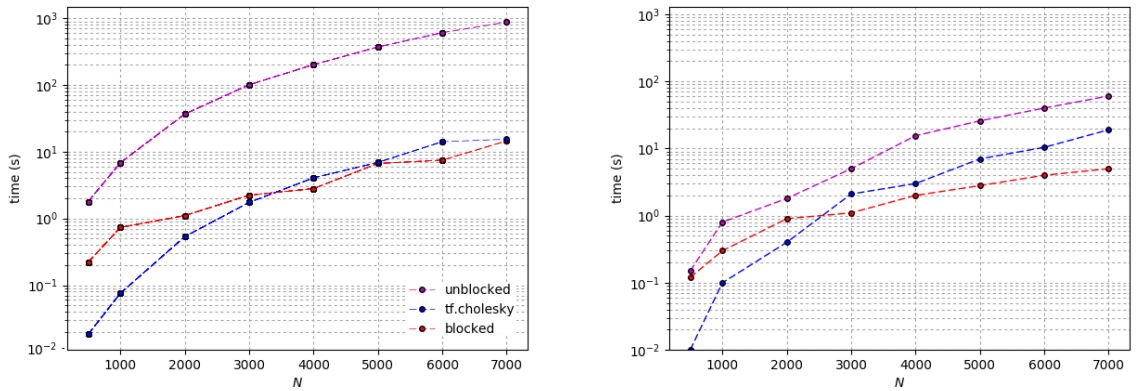
For $N > 2000$ the execution time of our blocked implementation and TensorFlow's default implementation are similar. This outcome is expected as both implementations are based on the same algorithm. For smaller N we see that the recursion implemented by the Eigen library in C++ is faster than our blocked one in Python/TensorFlow.

Benchmarking the algorithms with GPU support enabled, reveals different properties. In this case, our blocked implementation becomes faster the TensorFlow default, with a maximum speedup factor of approximately 3 for $N = 7000$. This is the result of optimally exploiting the GPU hardware by the TensorFlow matrix-matrix operations.

During the optimization of a certain objective function, such as the likelihood of a model, it is often necessary to calculate its gradients (e.g., CholeskyGrad in Figure 3.3). In GPs the objective typically consists out of a number of Cholesky decompositions. For this reason, it is obligatory that our implemented blocked-partitioned algorithm is differentiable and supports the symbolic partial derivatives routine used in TensorFlow. Due to some particular design choices in the implementation this is made possible. The performance is reported in Figure 3.5. We observe a similar trend for both, GPU and CPU execution, as in the previous figure. The unblocked algorithm is not reported as it is unable to support gradient calculations for matrices larger than 2000.

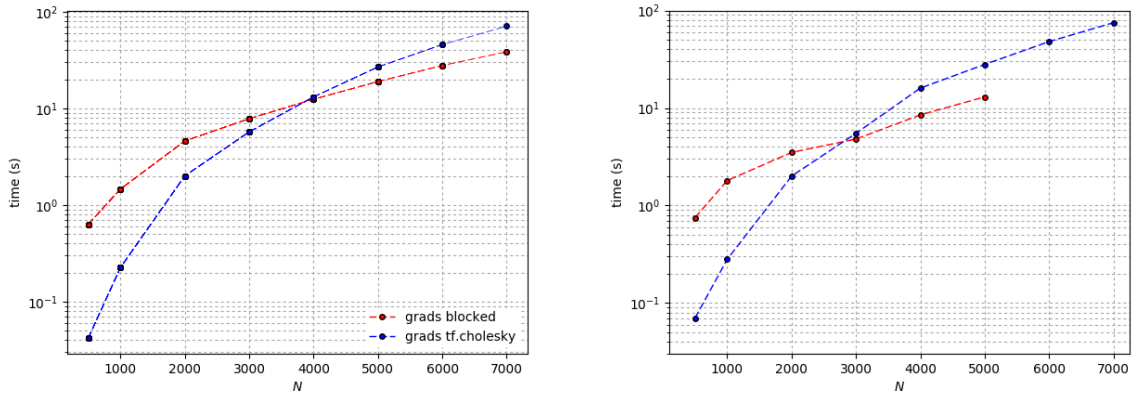
The final step is the integration of our new, block-partitioned Cholesky implementation into GPflow, and compare the effects on the execution time. In Figure 3.6 this is shown. The figure plots the time required to execute 5 optimization rounds of two different models. The first model (blue) is the GPflow GPR model, which uses the standard Cholesky decomposition from TensorFlow. The second model (red) is the GPflow GPR model, modified in order to use the new Cholesky implementation. As an optimization typically constitutes out of multiple Cholesky and Cholesky gradient calls, we expect Figure 3.6 to be a combination of Figure 3.4 and Figure 3.5. Indeed, this is the case. Note that the GPU implementation is faster but can not handle matrices that are larger than 5000 due to memory restrictions.

In conclusion, the plots proof the usefulness of our implementation, which is a bit slower for small datasets but speed ups the optimization for large datasets by approximately a factor 3. This is of significant importance as using the most efficient implementation for each component of the model is one of the first steps towards leveraging GPs for big data.



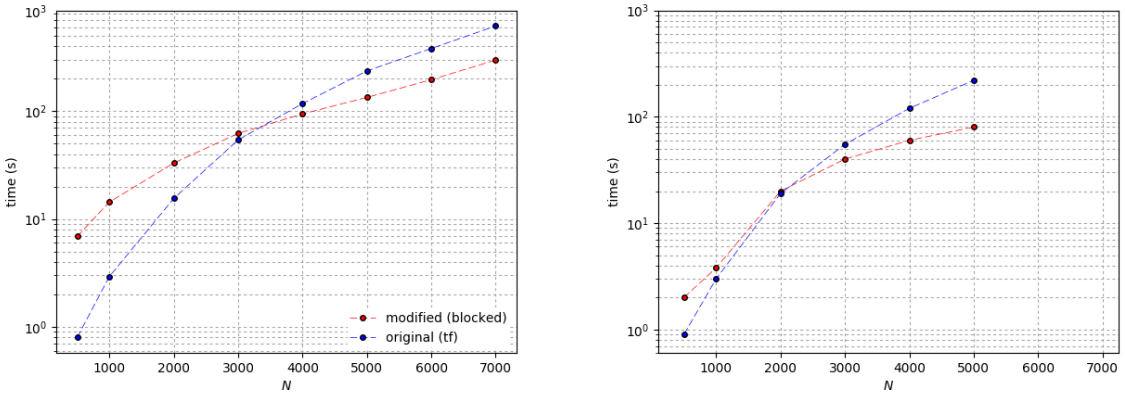
(a) CPU (b) GPU

Figure 3.4: Execution time of the Cholesky implementations in function of the size N.



(a) CPU (b) GPU

Figure 3.5: Execution time of the Cholesky *gradient* implementations in function of the size N.



(a) CPU (b) GPU

Figure 3.6: Execution time of 5 optimization rounds of the GPR model.

Chapter 4

Finding Uncertain Patterns with Gaussian Processes

The focus of this chapter lies on theoretical aspects in advanced Gaussian Process (GP) models. More precisely, Section 4.1 deals with performing predictions of uncertain inputs. Section 4.2 focuses on unsupervised GP-based dimensionality reduction and Section 4.3 covers the Deep Gaussian Process (DGP) model, a hierarchical stacking of multiple GPs. In Chapters 5 and 6 these models will be put into practice and excessively used to tackle real-world engineering problems.

4.1 Uncertain Input Propagation

4.1.1 Background

In the previous chapters, the inputs of the GP system $\{\boldsymbol{X}, \boldsymbol{X}_*\}$ during learning and inference are always noise-free (i.e. deterministic) values. We derived, under these condition, that the predictive distribution is a Gaussian with an analytically tractable mean and variance. However, in many real-world scenarios test inputs $\boldsymbol{X}_*$ are not deterministic but rather uncertain, noisy or even missing (only noise) variables. Consider for example the extrapolation of time series, let $y_1, y_2, \ldots, y_t$ be the observed values and assume that the next value y_{t+1} is a function of the current y_t and the previous L values, i.e. $y_{t+1} = f(y_t, y_{t-1}, \ldots, y_{t-L})$, where L specifies the number of previous states influencing the next state. If one now wishes predictions for k steps ahead, then y_{t+1}, y_{t+2} up to y_{t+k} are required.

Two different approaches are possible to achieve this. They both start by transforming the time series into input-output pairs. Subsequently, they construct a GP model for $f(\cdot)$ and readily use it to obtain a Gaussian predictive distribution over $y_{t+1} \sim \mathcal{N}(\mu_{t+1}, \sigma^2_{t+1})$. However, for predicting y_{t+2} both approaches diverge. The simplest, but most naive, way is to consider μ_{t+1} as a point estimate of y_{t+1} and to use it as an deterministic input to obtain the predictive distribution over y_{t+2}. In other words, we ignore the uncertainty over y_{t+1} and only use the

mean to perform the next prediction. Likewise, we neglect the variance over y_{t+2} and use only its mean to obtain y_{t+3} and so on, up to y_{t+k}. This approach is ill-advised as it throws away important information (namely the variance over each prediction) and can subsequently lead to over-confident estimations, underestimating the uncertainty. The more intelligent approach account for the uncertainty on y_{t+1} to calculate the predictive distribution of y_{t+2}. It is able to deal with the fuzzy input y_{t+1} to derive the distribution of y_{t+2}. In Section 6.4.2 a detailed example of both approaches is given, which highlights the importance of uncertainty propagation.

For static problems as well, the model's performance can significantly be improved when input uncertainty is incorporated into the predictions. For example, when dealing with applications using high frequency sensor data, it is useful to account for input uncertainty as they are typically corrupted by numerous imperfections (e.g., measuring disturbances). If the model does not accounts for this, it will predict over-confident estimations. In case of decision-making systems, such as self-driving cars, this can have severe consequences as the system is relying on misleading uncertainties to make its decisions.

4.1.2 Prediction at an Uncertain Input

In the following derivation we assume that we have a GPR model optimized for the dataset $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{y}\}$, where $y_n = f(\boldsymbol{x}_n) + \epsilon_n$ and $\epsilon_n \sim \mathcal{N}(0, \sigma^2)$. The GP makes use of a generic kernel function k and is configured with a zero-mean. The objective is to obtain a predictive distribution f_* for input $\boldsymbol{x}_*$, which has a Gaussian distribution $\mathcal{N}(\boldsymbol{w}, \boldsymbol{S})$. The true predictive distribution is then given by

$$p(f_*|\boldsymbol{w}, \boldsymbol{S}, \mathcal{D}) = \int p(f_*|\boldsymbol{x}_*, \mathcal{D})\, \mathcal{N}(\boldsymbol{x}_*|\boldsymbol{w}, \boldsymbol{S})\, d\boldsymbol{x}_*,$$

where $p(f_*|\boldsymbol{x}_*, \mathcal{D})$ is the deterministic predictive distribution (Eq. 2.5), yielding in

$$p(f_*|\boldsymbol{w}, \boldsymbol{S}, \mathcal{D}) = \int \mathcal{N}\left(f_*|\mu(\boldsymbol{x}_*), \sigma^2(\boldsymbol{x}_*)\right)\, \mathcal{N}(\boldsymbol{x}_*|\boldsymbol{w}, \boldsymbol{S})\, d\boldsymbol{x}_*. \tag{4.1}$$

The mean $\mu(\boldsymbol{x}_*)$ and variance $\sigma^2(\boldsymbol{x}_*)$ are according to Eq. 2.6 and 2.7

$$\begin{aligned}
\mu(\boldsymbol{x}_*) &= k(\boldsymbol{X}, \boldsymbol{x}_*)^\top \tilde{\boldsymbol{K}}^{-1} \boldsymbol{y} = k(\boldsymbol{X}, \boldsymbol{x}_*)^\top \boldsymbol{\beta}, \\
\sigma^2(\boldsymbol{x}_*) &= k_{**} - \boldsymbol{k}_*^\top \tilde{\boldsymbol{K}}^{-1} \boldsymbol{k}_* = k(\boldsymbol{x}_*, \boldsymbol{x}_*) - \sum_{n,m} \left(\tilde{\boldsymbol{K}}^{-1}\right)_{n,m} k(\boldsymbol{x}_n, \boldsymbol{x}_*)\, k(\boldsymbol{x}_m, \boldsymbol{x}_*),
\end{aligned}$$

where $\boldsymbol{\beta} \triangleq \tilde{\boldsymbol{K}}^{-1} \boldsymbol{y}$. Eq. 4.1 is a complicated integral as it is necessary to propagate the normal distribution of $\boldsymbol{x}_*$ through a highly non-linear mapping. It can not be solved directly and therefore Quiñonero-Candela et al. (2002) and Girard et al. (2003) suggest three different approaches to compute/approximate it:

1. The most straightforward way is a numerical solution which approximates the integral by using Monte-Carlo: $p(f_*|\boldsymbol{w}, \boldsymbol{S}, \mathcal{D}) \approx \frac{1}{T} \sum_{t=1}^{T} \mathcal{N}\left(f_*|\mu(\boldsymbol{x}_*^t), \sigma^2(\boldsymbol{x}_*^t)\right)$, where every $\boldsymbol{x}_*^t$ is drawn from $\mathcal{N}(\boldsymbol{x}_*|\boldsymbol{w}, \boldsymbol{S})$. This approximation is valuable as we known that it will tend to

the true distribution as the number of samples T grows. However, computationally it is less preferred as T needs to be relatively large to obtain good approximations, especially for high-dimensional spaces.

2. The second approach will approximate the mean $\mu(\boldsymbol{x}_*)$ and variance $\sigma^2(\boldsymbol{x}_*)$ by their first and second order Taylor expansion. These approximations are then used to analytically compute the first and second order central moments of Eq. 4.1. These moments can be computed using the law of iterated expectations and conditional variance

$$
\begin{aligned}
m(\boldsymbol{w}, \boldsymbol{S}) &= \mathbb{E}_{\boldsymbol{x}_*}\left[\mathbb{E}_{f_*}\left[f_* | \boldsymbol{x}_*\right]\right] = \mathbb{E}_{\boldsymbol{x}_*}\left[\mu(\boldsymbol{x}_*)\right], && (4.2)\\
v(\boldsymbol{w}, \boldsymbol{S}) &= \mathbb{E}_{\boldsymbol{x}_*}\left[\mathrm{var}_{f_*}\left[f_* | \boldsymbol{x}_*\right]\right] + \mathrm{var}_{\boldsymbol{x}_*}\left[\mathbb{E}_{f_*}\left[f_* | \boldsymbol{x}_*\right]\right] \\
&= \mathbb{E}_{\boldsymbol{x}_*}\left[\sigma^2(\boldsymbol{x}_*)\right] + \mathrm{var}_{\boldsymbol{x}_*}\left[\mu(\boldsymbol{x}_*)\right] \\
&= \mathbb{E}_{\boldsymbol{x}_*}\left[\sigma^2(\boldsymbol{x}_*)\right] + \mathbb{E}_{\boldsymbol{x}_*}\left[\mu^2(\boldsymbol{x}_*)\right] - \mathbb{E}^2_{\boldsymbol{x}_*}\left[\mu(\boldsymbol{x}_*)\right]. && (4.3)
\end{aligned}
$$

3. Similar as in the second approach, the marginal predictive distribution of Eq. 4.1 will be approximated by its first (mean) and second (variance) order central moment. However, instead of using the Taylor expansion of $\mu(\boldsymbol{x}_*)$ and $\sigma^2(\boldsymbol{x}_*)$ to be able to compute Eq. 4.2 and 4.3, we calculate them exactly.

In the following paragraphs we focus on the third approach, as it is analytically the most exact one and often used in other models. We start by computing the mean (Eq. 4.2)

$$
\begin{aligned}
m(\boldsymbol{w}, \boldsymbol{S}) &= \mathbb{E}_{\boldsymbol{x}_*}\left[\mu(\boldsymbol{x}_*)\right] \\
&= \int \left(k(\boldsymbol{X}, \boldsymbol{x}_*)^\top \boldsymbol{\beta}\right) \mathcal{N}(\boldsymbol{x}_* | \boldsymbol{w}, \boldsymbol{S})\, d\boldsymbol{x}_* \\
&= \boldsymbol{\Omega}^\top \boldsymbol{\beta}, && (4.4)
\end{aligned}
$$

where $\boldsymbol{\Omega} = [\Omega(\boldsymbol{x}_1), \Omega(\boldsymbol{x}_2), \ldots, \Omega(\boldsymbol{x}_n)]^\top$ is a kernel expectation and is defined as an expectation of the kernel under the normal distribution of $\boldsymbol{x}_*$, i.e. $\Omega(\boldsymbol{x}_i) = \int k(\boldsymbol{x}_i, \boldsymbol{x}_*)\, \mathcal{N}(\boldsymbol{x}_* | \boldsymbol{w}, \boldsymbol{S})\, d\boldsymbol{x}_*$. In Section 4.4 we will deal with calculating kernel expectations and we shall see that they can be obtained in closed analytical form for a couple of often-used kernels. For now, we continue with the variance (Eq. 4.3)

$$
\begin{aligned}
v(\boldsymbol{w}, \boldsymbol{S}) &= \mathbb{E}_{\boldsymbol{x}_*}\left[\sigma^2(\boldsymbol{x}_*)\right] + \mathbb{E}_{\boldsymbol{x}_*}\left[\mu^2(\boldsymbol{x}_*)\right] - \left(\mathbb{E}_{\boldsymbol{x}_*}\left[\mu(\boldsymbol{x}_*)\right]\right)^2 \\
&= \mathbb{E}_{\boldsymbol{x}_*}\left[k(\boldsymbol{x}_*, \boldsymbol{x}_*) - \sum_{n,m} \left(\tilde{\boldsymbol{K}}^{-1}\right)_{n,m} k(\boldsymbol{x}_n, \boldsymbol{x}_*)\, k(\boldsymbol{x}_m, \boldsymbol{x}_*)\right] + \\
&\quad \mathbb{E}_{\boldsymbol{x}_*}\left[\sum_{n,m} \beta_n \beta_m\, k(\boldsymbol{x}_n, \boldsymbol{x}_*)\, k(\boldsymbol{x}_m, \boldsymbol{x}_*)\right] - \left(\mathbb{E}_{\boldsymbol{x}_*}\left[\mu(\boldsymbol{x}_*)\right]\right)^2 \\
&= \mathbb{E}_{\boldsymbol{x}_*}\left[k(\boldsymbol{x}_*, \boldsymbol{x}_*)\right] - \sum_{n,m} \left(\left(\tilde{\boldsymbol{K}}^{-1}\right)_{n,m} - \beta_n \beta_m\right) \mathbb{E}_{\boldsymbol{x}_*}\left[k(\boldsymbol{x}_n, \boldsymbol{x}_*) k(\boldsymbol{x}_m, \boldsymbol{x}_*)\right] - \left(\boldsymbol{\Omega}^\top \boldsymbol{\beta}\right)^2 \\
&= \xi - \sum_{n,m} \left(\left(\tilde{\boldsymbol{K}}^{-1}\right)_{n,m} - \beta_n \beta_m\right) \Phi(\boldsymbol{x}_n, \boldsymbol{x}_m) - \left(\boldsymbol{\Omega}^\top \boldsymbol{\beta}\right)^2,
\end{aligned}
$$

where $\xi = \int k(\boldsymbol{x}_*, \boldsymbol{x}_*)\, \mathcal{N}(\boldsymbol{x}_* | \boldsymbol{w}, \boldsymbol{S})\, d\boldsymbol{x}_*$ and $\Phi(\boldsymbol{x}_n, \boldsymbol{x}_m) = \int k(\boldsymbol{x}_n, \boldsymbol{x}_*)\, k(\boldsymbol{x}_m, \boldsymbol{x}_*)\, \mathcal{N}(\boldsymbol{x}_* | \boldsymbol{w}, \boldsymbol{S})\, d\boldsymbol{x}_*$. Both ξ and Φ are other types of kernel expectations, discussed in more detail in Section 4.4. The expression can be further simplified into a sum of traces[1]:

$$v(\boldsymbol{w}, \boldsymbol{S}) = \xi - \operatorname{tr}\left(\left(\tilde{\boldsymbol{K}}^{-1} - \boldsymbol{\beta}\boldsymbol{\beta}^\top\right)\boldsymbol{\Phi}\right) - \operatorname{tr}\left(\boldsymbol{\Omega}\boldsymbol{\Omega}^\top \boldsymbol{\beta}\boldsymbol{\beta}^\top\right), \tag{4.5}$$

where matrix $\boldsymbol{\Phi}_{n,m} = \Phi(\boldsymbol{x}_n, \boldsymbol{x}_m)$.

In conclusion, the predictive distribution of f_*, belonging to a stochastic input $\boldsymbol{x}_* \sim \mathcal{N}(\boldsymbol{w}, \boldsymbol{S})$, is approximated by a Gaussian $\mathcal{N}(m(\boldsymbol{w}, \boldsymbol{S}), v(\boldsymbol{w}, \boldsymbol{S}))$, which has analytically the exact same mean and variance as the true – not analytically computable – predictive distribution (Eq. 4.1). A rigorous experiment, where this concept is successfully applied, is covered in Section 6.4.2.

4.2 Gaussian Process Manifold Learning

All the models and methodologies discussed until now are completely in a supervised setting, even though a well-known difficulty present in all those models is the necessity for labeled data. The models require a well-organized dataset containing inputs and corresponding outputs. Unfortunately, in real-world applications labeled data is extremely scarce, not available or expensive. Consider, for example, data originated from IoT devices or surveillance cameras. The devices are easily capable of generating gigabytes of data each day. It is, however, impractical and almost impossible to label this bulk. Consequently, unsupervised and semi-supervised learning are lately considered as one of the most fundamental and important research directions in machine learning. This is despite the fact that unsupervised learning is actually ill-posed, as the relevant downstream tasks are unknown during training time. A low dimensional representation, one which explicitly represents the latent attributes of a dataset, is required for relevant but unknown tasks. For example, for a faces-dataset, a useful disentangled representation should be low dimensional where each of the following attributes are separable: eye color, presence or absence of eyeglasses, facial expression and the identity of the corresponding person. Thus, to be useful, an unsupervised learning algorithm must in effect correctly guess the likely set of downstream classification tasks without being directly exposed to them, which is extremely hard.

This section is dedicated to GP-based models for unsupervised learning and dimensionality reduction. Rather than creating a model for a dataset $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{y}\}$ consisting of inputs and targets, in an unsupervised learning setting we are only given a dataset $\mathcal{D} = \{\boldsymbol{Y}\}$, with $\boldsymbol{Y} \in \mathbb{R}^{N \times D}$. The objective is to obtain a lower dimensional dataset $\boldsymbol{X} \in \mathbb{R}^{N \times d}$, which is a projection of the high dimensional data onto a lower dimensional manifold or latent space ($d < D$). This lower dimensional representation can then be used for visualization ($d = 2$), pruning irrelevant features, lossy compression, for downstream classification or regression, etc.

[1] $\operatorname{tr}\left(\boldsymbol{A}\boldsymbol{B}^\top\right) = \sum_{i,j} \boldsymbol{A}_{i,j} \boldsymbol{B}_{i,j}$

4.2.1 (Probabilistic) Principal Component Analysis

The most well-known and most widely used dimensionality reduction technique is Principal Component Analysis (PCA) (Jolliffe 2002). The PCA algorithm will orthogonally project the data onto a manifold which maximizes the variances between the datapoints in that latent space, or alternatively, a manifold which minimizes the projection error. Both formulations come down to linearly projecting the data onto the latent space, spanned by the d eigenvectors corresponding to the d largest eigenvalues.

The formulation of PCA does not incorporate any uncertainty in the observed space nor in the latent space. Hence, it is not able to probabilistically asses the likelihood or quality of the dimensionality reduction operation. Probabilistic Principal Component Analysis (PPCA) is a Bayesian reformulation of the PCA procedure which accounts for uncertainty in the latent space (Tipping and Bishop 1999). The most prominent advantage of the probabilistic approach is that it is able to give a likelihood of the performed projection. This empirical value can be used to compare the model with other probabilistic density models and automatically find the optimal dimensionality of the subspace. PPCA can also be seen as a generative approach as we are able to sample from the posterior distribution.

Deriving the PPCA's marginal likelihood is relatively simple. We introduce an isotropic Gaussian prior distribution over the latent variables $p(\boldsymbol{x}_n) = \mathcal{N}(\boldsymbol{x}_n|\mathbf{0}, \boldsymbol{I}_d)$ and projection components $\boldsymbol{W} \in \mathbb{R}^{D\times d}$ and $\boldsymbol{\mu} \in \mathbb{R}^D$ to form the conditional distribution $p(\boldsymbol{y}_n|\boldsymbol{x}_n) = \mathcal{N}(\boldsymbol{y}_n|\boldsymbol{W}\boldsymbol{x}_n+\boldsymbol{\mu}, \sigma^2\boldsymbol{I}_D)$. The marginal likelihood of an observed datapoint is derived by integrating over the latent space

$$\begin{aligned} p(\boldsymbol{y}_n|\boldsymbol{W}, \boldsymbol{\mu}, \sigma^2) &= \int p(\boldsymbol{y}_n|\boldsymbol{x}_n)\, p(\boldsymbol{x}_n)\; d\boldsymbol{x}_n \\ &= \mathcal{N}\left(\boldsymbol{y}_n|\boldsymbol{\mu}, \boldsymbol{C}\right), \end{aligned} \tag{4.6}$$

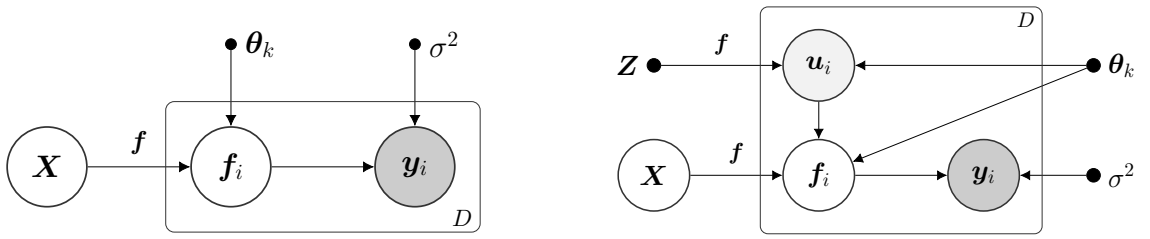
where $\boldsymbol{C} = \boldsymbol{W}\boldsymbol{W}^\top + \sigma^2\boldsymbol{I}$ (using Eq. A.2).

The optimal parameters $\boldsymbol{\Theta} = \{\boldsymbol{W}, \boldsymbol{\mu}, \sigma^2\}$ are obtained by maximizing the complete likelihood. As we assume that the observations are i.i.d. the log complete marginal likelihood is the sum over the N individual likelihoods (Eq. 4.6), we obtain

$$\begin{aligned} \mathcal{F}(\boldsymbol{\Theta}) \triangleq \log p(\boldsymbol{Y}|\boldsymbol{\mu}, \boldsymbol{W}, \sigma^2) &= \sum_{n=1}^{N} \log p(\boldsymbol{y}_n|\boldsymbol{W}, \boldsymbol{\mu}, \sigma^2) \\ &= -\frac{ND}{2}\log 2\pi - \frac{N}{2}\log|\boldsymbol{C}| - \frac{1}{2}\sum_{n=1}^{N}(\boldsymbol{y}_n-\boldsymbol{\mu})^\top\boldsymbol{C}^{-1}(\boldsymbol{y}_n-\boldsymbol{\mu}). \end{aligned}$$

This equations is differentiable w.r.t. all the parameters and has an exact closed-form solution. Particularly, using the chain-rule we obtain an equation to derive the projection matrix $\boldsymbol{W}$: $(\partial\mathcal{F}/\partial\boldsymbol{\Theta})\,(\partial\boldsymbol{\Theta}/\partial\boldsymbol{W}) = \mathbf{0}$. This equation can be solved analytically to obtain the MLE of $\boldsymbol{W}$.

It is interesting to note that the dual formulation, where we integrate over the parameter space $\boldsymbol{W}$ and optimize w.r.t. the latent space $\boldsymbol{X}$, yields in the same MLE solution as the approach followed above (i.e. integrate over the latent space $\boldsymbol{X}$ and optimize w.r.t. the hyperparameters).



(a) GP-LVM (b) Bayesian GP-LVM

Figure 4.1: Graphical models GP-based manifold learning.

4.2.2 GP Latent Variable Model

The approach followed in a GP Latent Variable Model (GP-LVM) is similar to the one in PPCA. However, instead of performing a linear mapping from the latent to the observed space, we apply the GP *kernel trick* (Lawrence 2003). The features (or dimensions) of $\boldsymbol{Y}$ are in GP-LVMs considered to be independent. The forward mapping is then governed by a product of D GPs

$$\begin{aligned} p(\boldsymbol{Y}|\boldsymbol{X}) &= \prod_{i=1}^{D} \int p(\boldsymbol{y}_i|\boldsymbol{f}_i)\, p(\boldsymbol{f}_i|\boldsymbol{X})\; d\boldsymbol{f}_i \\ &= \prod_{i=1}^{D} \mathcal{N}(\boldsymbol{y}_i|K(\boldsymbol{X},\boldsymbol{X}) + \sigma^2\boldsymbol{I}), \end{aligned} \tag{4.7}$$

where $\boldsymbol{y}_i \in \mathbb{R}^N$ is a vector denoting the i^{th} column of $\boldsymbol{Y}$. A peculiar choice of the GP-LVM is to use the same GP across every dimension. In fact, this corresponds to the marginal likelihood of a multi-dimensional target GP regression model (Section 2.3). Figure 4.1a shows the graphical representation of the GP-LVM likelihood construction.

The latent variables $\boldsymbol{X}$ are again given by an isotropic Gaussian distribution $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_d)$ and assumed to be i.i.d.

$$p(\boldsymbol{X}) = \prod_{n=1}^{N} \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{0}, \boldsymbol{I}_d),$$

which completes the full joint probability of the GP-LVM $p(\boldsymbol{Y}, \boldsymbol{X}) = p(\boldsymbol{X})p(\boldsymbol{Y}|\boldsymbol{X})$.

In practice, the optimal values for the (hyper)parameters $\boldsymbol{\Theta} = \{\boldsymbol{X}, \sigma^2, \boldsymbol{\theta}_k\}$ of the model are found by performing a MLE of $\boldsymbol{X}$, while jointly maximizing the likelihood (Eq. 4.7) with respect to the other hyperparameters (Lawrence 2005).

The performance of this non-linear probabilistic model often outperforms PCA (non-probabilistic and linear) and PPCA (probabilistic and linear). The non-linear kernel mapping from latent space to the observed space in the GP-LVM greatly enhances the discovery of complex structures and enables more accurate manifold learning. However, as the GP-LVM model is much more flexible, it requires sensible initialization of the parameters in order to obtain a desirable outcome. Especially, if $\boldsymbol{X}$ is initialized with random values, the gradient based optimization routine will typically only find a *bad* local optimum and the resulting dimensionality reduction will be inferior compared to the one from PCA and PPCA. Contrarily, initializing $\boldsymbol{X}$ with the result

of a (P)PCA routine will help the GP-LVM model in finding a more optimal forward mapping. In other words, the GP-LVM is only able to outperform other models if it is first intelligently initialized with the result of another dimensionality reduction technique (e.g., $\boldsymbol{X}^{\text{init}} = \text{PCA}(\boldsymbol{Y})$). This inconvenience is justified by prof. Lawrence as follows: *"What better way to outperform PCA then by starting from it?"*.

Notwithstanding, the shortcoming has led researchers to build upon the standard GP-LVM model. One of the most notable extension is the Back Constraint GP-LVM (BC GP-LVM) (Lawrence and Quiñonero-Candela 2006). In a BC GP-LVM instead of directly maximizing Eq. 4.7 w.r.t. the manifold representation $\boldsymbol{X}$, each element $\boldsymbol{x}_n$ is replaced by a backward mapping $\boldsymbol{x}_n = \boldsymbol{g}(\boldsymbol{y}_n, \boldsymbol{W}_{\boldsymbol{g}})$. Rather than directly maximizing the likelihood, a constrained likelihood is now maximized. The added parameters $\boldsymbol{W}_{\boldsymbol{g}}$ are in the BC GP-LVM jointly optimized with the original kernel hyperparameters and noise parameter. The backward mapping function $\boldsymbol{g}$ can have different forms. The only restriction is that it needs to be differentiable with respect to its parameters so that the derivatives can be used in gradient based optimization schemes. In practice, a multilayer perceptron is often used as it turns out that this backward mapping is rapidly trained and able to elegantly preserve distances from the observed to the latent space.

4.2.3 Bayesian GP-LVM

Although a GP-LVM is probabilistic it is not completely Bayesian as only the latent function $\boldsymbol{f}$ is treated in a Bayesian way, while the manifold representation $\boldsymbol{X}$ is still found using a MLE approach. Contrarily, in a Bayesian GP-LVM (BGP-LVM), $\boldsymbol{X}$ is along with $\boldsymbol{f}$ considered to be uncertain and propagated through the GP mapping, this results in a fully Bayesian approach (Titsias and Lawrence 2010). Mathematically, this translates into

$$\begin{aligned}
\text{GP-LVM: } & \underset{\boldsymbol{X},\boldsymbol{\Theta}}{\operatorname{argmax}}\, p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\Theta}), \text{where } p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\Theta}) = \int p(\boldsymbol{Y}|\boldsymbol{F}) p(\boldsymbol{F}|\boldsymbol{X},\boldsymbol{\Theta})\, d\boldsymbol{F}, \\
\text{BGP-LVM: } & \underset{\boldsymbol{\Theta}}{\operatorname{argmax}}\, p(\boldsymbol{Y}|\boldsymbol{\Theta}), \text{where } p(\boldsymbol{Y}|\boldsymbol{\Theta}) = \int \int p(\boldsymbol{Y}|\boldsymbol{F}) \underbrace{p(\boldsymbol{F}|\boldsymbol{X},\boldsymbol{\Theta}) p(\boldsymbol{X})}_{\text{intractable}}\, d\boldsymbol{F}\, d\boldsymbol{X}. \qquad (4.8)
\end{aligned}$$

On the one hand, this fully Bayesian treatment looks desirable and leads to several appealing characteristics. The most important ones are a training procedure which is robust to overfitting, the ability to incorporate prior information in $p(\boldsymbol{X})$ (e.g., temporal correlation or dynamics), handle missing or partially observed data and automatically detect the dimensionality of $\boldsymbol{X}$. On the other hand, propagating a prior distribution on $\boldsymbol{X}$ through the GP mapping is intractable due to the non-linear appearance of $\boldsymbol{X}$ in $p(\boldsymbol{f}|\boldsymbol{X},\boldsymbol{\Theta})$.

The proposed solution is to replace the log marginal likelihood (Eq. 4.8) by a closed-form variational lower bound through Jensen's inequality. Furthermore, to achieve a tractable and computationally efficient lower bound, the model is augmented with inducing variables $\{\boldsymbol{Z}, \boldsymbol{u}\}$. The graphical model is shown in Figure 4.1b. This approach is very similar to the one followed in Sparse Variational Gaussian Process (SVGP) (Section 3.1.3), not surprisingly as both models are developed by Michalis Titsias.

In the remaining of this section we sketch the most important mathematics and ideas to obtain the variational lower bound on the marginal likelihood of the Bayesian GP-LVM. The details of this derivation will be postponed until Section 4.3 as every layer of a DGP is in fact a Bayesian GP-LVM. This can be seen by comparing a single layer from Figure 4.2 with Figure 4.1b.

We wish to obtain the model evidence (Eq. 4.8, where $\boldsymbol{F}$ is already marginalized)

$$p(\boldsymbol{Y}|\boldsymbol{\Theta}) = \int p(\boldsymbol{X}|\boldsymbol{\Theta})\; p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{\Theta})\; d\boldsymbol{X}.$$

In the following we omit the dependence on $\boldsymbol{\Theta}$ to lighten the notation. As mentioned above, this integral is not analytically tractable due to the non-linear relation of $\boldsymbol{X}$ in $\boldsymbol{K}^{-1}$ and $|\boldsymbol{K}|$ inside $p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{\Theta})$, where $\boldsymbol{K} = k(\boldsymbol{X}, \boldsymbol{X})$ (see Eq. 2.3). The main idea of the BGP-LVM is to introduce a variational distribution $q(\boldsymbol{X})$ to approximate the true posterior distribution $p(\boldsymbol{X}|\boldsymbol{Y})$. The variational distribution is arbitrarily chosen to be Gaussian and factorisable over the datapoints

$$q(\boldsymbol{X}) = \prod_{n=1}^{N} \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_n, \boldsymbol{S}_n).$$

Using Jensen's inequality we are now able to lower bound the marginal likelihood. It is interesting to note that this *trick* is basically the foundation of every Variational Inference (VI) framework (e.g., Section 3.1.3 and 4.3). We obtain

$$\begin{aligned}
\log p(\boldsymbol{Y}) &= \log \int p(\boldsymbol{X}) p(\boldsymbol{Y}|\boldsymbol{X}) \frac{q(\boldsymbol{X})}{q(\boldsymbol{X})}\; d\boldsymbol{X} \\
&\geq \int q(\boldsymbol{X}) \log \frac{p(\boldsymbol{Y}|\boldsymbol{X})\, q(\boldsymbol{X})}{q(\boldsymbol{X})}\; d\boldsymbol{X} \\
&= \langle \log p(\boldsymbol{Y}|\boldsymbol{X}) \rangle_{q(\boldsymbol{X})} - \mathrm{KL}\left[q(\boldsymbol{X}) \| p(\boldsymbol{X})\right] \triangleq \mathcal{F}(q),
\end{aligned}$$

where $\langle \cdot \rangle_{p(x)} \triangleq \mathbb{E}_x\left[\,\cdot\,\right]$ denotes the expectation under $p(x)$. The second term in $\mathcal{F}(q)$ can be calculated effortlessly as both distributions are Gaussians (see Eq. A.4). The first term is more problematic as $\log p(\boldsymbol{Y}|\boldsymbol{X}) = \sum_{i=1}^{D} \log p(\boldsymbol{y}_i|\boldsymbol{X})$ still contains the non-linearity in $\boldsymbol{X}$. This problem is remedied, as we will see in Section 4.3, by expanding the model with auxiliary inducing datasets $\{\boldsymbol{Z}, \boldsymbol{u}\}$ and defining an additional mean-field variational distribution $q(\boldsymbol{u})$.

The BGP-LVM is an important model as it is the basic building block of a DGP or a multi-view GP-LVM. Furthermore, it turns traditional parameters into variational parameters which assuredly reduces the chances of overfitting. At last, having an approximate posterior over $q(\boldsymbol{X}) \approx p(\boldsymbol{X}|\boldsymbol{Y})$ is very useful in numerous applications (e.g., anomaly detection and generative modeling).

4.3 Deep Gaussian Processes

Deep Gaussian Processes (DGPs) (Damianou and Lawrence 2013, Hensman and Lawrence 2014) are a probabilistic model based on the composition of processes. Rather than assuming that an output $\boldsymbol{y}$ is drawn directly from a single Gaussian Process, we assume that it is drawn from a

functional composition of GPs and is corrupted by noise ϵ, this is written as follows

$$\boldsymbol{y} = \boldsymbol{f}_L(\boldsymbol{f}_{L-1}(\dots \boldsymbol{f}_2(\boldsymbol{f}_1(\boldsymbol{x}))) + \boldsymbol{\epsilon}$$

where every function $\boldsymbol{f}_\ell$ is itself a GP and $\boldsymbol{\epsilon}$ is an error term which holds the propagation of white Gaussian noise terms, $\boldsymbol{\epsilon_\ell} \sim \mathcal{N}(0, \sigma_\ell^2 \boldsymbol{I})$, added to every layer. Denoting $\boldsymbol{h}_\ell = \boldsymbol{f}_\ell(\boldsymbol{h}_{\ell-1}) + \boldsymbol{\epsilon}_\ell$ it is possible to write the complete joint probability

$$p(\boldsymbol{y}, \{\boldsymbol{h_\ell}\}_{\ell=1}^{L-1} | \boldsymbol{X}) = p(\boldsymbol{h}_1|\boldsymbol{X}) \left(\prod_{\ell=2}^{L-1} p(\boldsymbol{h}_\ell|\boldsymbol{h}_{\ell-1}) \right) p(\boldsymbol{y}|\boldsymbol{h}_{L-1}). \tag{4.9}$$

After renaming $\boldsymbol{X}$ and $\boldsymbol{y}$ as $\boldsymbol{h}_0$ and $\boldsymbol{h}_L$, respectively, every probability in Eq. 4.9 can be written as a single GP mapping with mean zero and a covariance matrix $\boldsymbol{K}_{\boldsymbol{h}_\ell \boldsymbol{h}_\ell}$. Note the – at first sight – counterintuitive notation: covariance matrix $\boldsymbol{K}_{\boldsymbol{h}_\ell \boldsymbol{h}_\ell}$ is constructed with the outputs of the previous layer $\boldsymbol{h}_{\ell-1}$ and not $\boldsymbol{h}_\ell$.

$$p(\boldsymbol{h}_\ell|\boldsymbol{h}_{\ell-1}) = \mathcal{N}(\boldsymbol{h}_\ell|\boldsymbol{0}, \boldsymbol{K}_{\boldsymbol{h}_\ell \boldsymbol{h}_\ell} + \sigma_\ell^2 \boldsymbol{I}).$$

Inference in this *simple* stacked GP model is very challenging for two reasons. First, traditional GPs scale cubically with the number of datapoints, similarly will a composition of multiple GPs also scale with the usual $\mathcal{O}(N^3)$. Second, the hidden layer random variables $\boldsymbol{h}$ are not independent between layers nor within a layer, turning analytical marginalization practically impossible. To circumvent these problems we turn to Gaussian approximation techniques.

We focus on the nested variational approach, proposed by Hensman and Lawrence (2014), as it is computationally efficient and scales appropriately to large datasets. The approach starts by augmenting each layer with inducing inputs $\boldsymbol{Z}$ and outputs $\boldsymbol{U}$ and then uses variational compression. Both these concepts were already introduced in Section 3.1.1 and Section 3.1.3, respectively. With the auxiliary datasets in place, it is possible to lower bound the evidence on the hidden variables $\boldsymbol{h}$ within each layer. The next step is to propagate the input $\boldsymbol{X}$ through the first layer. As $\boldsymbol{X}$ is deterministic this is relatively straightforward. However, starting from $\boldsymbol{h}_1$ the hidden variables contain uncertainty and therefore it is required to propagate their probability distribution through the layer. This operation is very similar to the approach followed in Section 4.1. As a result, we obtain a lower bound on the complete model evidence $p(\boldsymbol{y}|\boldsymbol{X})$, which is independent of $\boldsymbol{h}$. The augmented graphical model is presented in Figure 4.2. Note the plate in the figure, which is added to show the independence of the N datapoints when they are conditioned on the inducing points.

Before readily deriving a variational lower bound on Eq. 4.9 we start with the derivation of a lower bound within each layer. Therefore, consider a one-layer DGP with inputs $\boldsymbol{X} \in \mathbb{R}^{N \times D}$, single dimensional outputs $\boldsymbol{y} \in \mathbb{R}^N$, inducing inputs $\boldsymbol{Z} \in \mathbb{R}^{M \times D}$ and inducing outputs $\boldsymbol{u} \in \mathbb{R}^M$. These variables have the same meaning as in Section 3.1, however, in the context of DGPs the term *inducing points* is more adopted then the term *pseudo points*. The variables comply with the following joint probability

$$p(\boldsymbol{y}, \boldsymbol{f}, \boldsymbol{u}|\boldsymbol{X}, \boldsymbol{Z}) = p(\boldsymbol{u}|\boldsymbol{Z}) p(\boldsymbol{f}|\boldsymbol{u}, \boldsymbol{Z}, \boldsymbol{X}) p(\boldsymbol{y}|\boldsymbol{f})$$

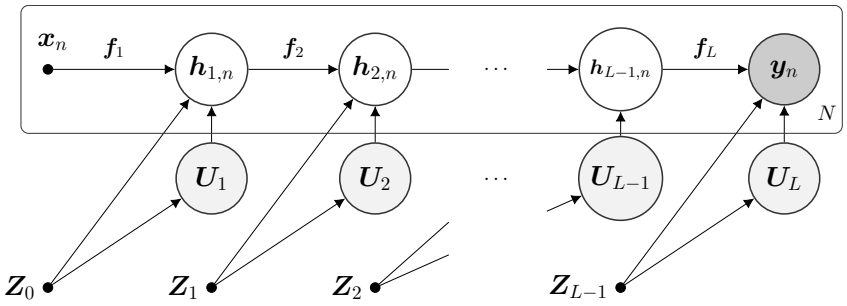


Figure 4.2: Graphical model of a generic DGP with $L+1$ layers: one input layer ($\boldsymbol{x}$), $L-1$ hidden layers ($\boldsymbol{h}$'s) and one observed layer ($\boldsymbol{y}$).

$$\Rightarrow p(\boldsymbol{y}|\boldsymbol{u},\boldsymbol{Z},\boldsymbol{X}) = \int p(\boldsymbol{f}|\boldsymbol{u},\boldsymbol{Z},\boldsymbol{X})p(\boldsymbol{y}|\boldsymbol{f})\ d\boldsymbol{f}, \tag{4.10}$$

where $p(\boldsymbol{y}|\boldsymbol{f})$ is the traditional likelihood and $p(\boldsymbol{f}|\boldsymbol{u},\boldsymbol{Z},\boldsymbol{X})$ is the typical GP predictive distribution (Eq. 2.5). Substituting the correct variables in the equation gives

$$p(\boldsymbol{y}|\boldsymbol{f}) = \prod_{n=1}^{N} p(y_n|f_n) = \prod_{n=1}^{N} \mathcal{N}(y_n|f_n,\sigma^2) = \mathcal{N}(\boldsymbol{y}|\boldsymbol{f},\sigma^2\boldsymbol{I}), \tag{4.11}$$

$$p(\boldsymbol{f}|\boldsymbol{u}) = \mathcal{N}\left(\boldsymbol{f}|\boldsymbol{K}_{\boldsymbol{fu}}\boldsymbol{K}_{\boldsymbol{uu}}^{-1}\boldsymbol{u}, \boldsymbol{K}_{\boldsymbol{ff}} - \boldsymbol{K}_{\boldsymbol{fu}}\boldsymbol{K}_{\boldsymbol{uu}}^{-1}\boldsymbol{K}_{\boldsymbol{uf}}\right) \triangleq \mathcal{N}\left(\boldsymbol{f}|\boldsymbol{\mu}_{\boldsymbol{f}},\boldsymbol{\Sigma}_{\boldsymbol{f}}\right). \tag{4.12}$$

For notational simplicity $\boldsymbol{X}$ and $\boldsymbol{Z}$, the input locations, will be dropped from the list of conditional variables.

The first lower bound is derived by applying Jensen's inequality on Eq. 4.10

$$\begin{aligned}\log p(\boldsymbol{y}|\boldsymbol{u}) &= \log \int p(\boldsymbol{f}|\boldsymbol{u})p(\boldsymbol{y}|\boldsymbol{f})\ d\boldsymbol{f} \\ &\geq \langle \log p(\boldsymbol{y}|\boldsymbol{f})\rangle_{p(\boldsymbol{f}|\boldsymbol{u})} \triangleq \log \tilde{p}(\boldsymbol{y}|\boldsymbol{u}).\end{aligned} \tag{4.13}$$

Substituting Eq. 4.11 and 4.12 into Eq. 4.13 gives

$$\begin{aligned}\log \tilde{p}(\boldsymbol{y}|\boldsymbol{u}) &= \left\langle \log \mathcal{N}(\boldsymbol{y}|\boldsymbol{f},\sigma^2\boldsymbol{I})\right\rangle_{p(\boldsymbol{f}|\boldsymbol{u})} \\ &= \int p(\boldsymbol{f}|\boldsymbol{u})\left\{-\frac{N}{2}\log 2\pi\sigma^2 - \frac{1}{2\sigma^2}(\boldsymbol{y}-\boldsymbol{f})^\top(\boldsymbol{y}-\boldsymbol{f})\right\} d\boldsymbol{f} \\ &= -\frac{N}{2}\log 2\pi\sigma^2 - \frac{1}{2\sigma^2}\int p(\boldsymbol{f}|\boldsymbol{u})(\boldsymbol{y}^\top\boldsymbol{y} - 2\boldsymbol{y}^\top\boldsymbol{f} + \boldsymbol{f}^\top\boldsymbol{f})\ d\boldsymbol{f} \\ &= -\frac{N}{2}\log 2\pi\sigma^2 - \frac{1}{2\sigma^2}\left(\boldsymbol{y}^\top\boldsymbol{y} - 2\boldsymbol{y}^\top\boldsymbol{\mu}_{\boldsymbol{f}} + \boldsymbol{\mu}_{\boldsymbol{f}}^\top\boldsymbol{\mu}_{\boldsymbol{f}} + \mathrm{tr}\left(\boldsymbol{\Sigma}_{\boldsymbol{f}}\right)\right) \\ &= \log \mathcal{N}\left(\boldsymbol{y}|\boldsymbol{\mu}_{\boldsymbol{f}},\sigma^2\boldsymbol{I}\right) - \frac{1}{2\sigma^2}\mathrm{tr}\left(\boldsymbol{\Sigma}_{\boldsymbol{f}}\right).\end{aligned} \tag{4.14}$$

The key idea is now to lower bound the complete joint probability (Eq. 4.9) by replacing every distribution $p(\boldsymbol{h}_\ell|\boldsymbol{h}_{\ell-1})$ by its corresponding lower bound (Eq. 4.14). Before this substitution can be carried out, we define $\boldsymbol{\Sigma}_\ell = \boldsymbol{K}_{\boldsymbol{h}_\ell\boldsymbol{h}_\ell} - \boldsymbol{K}_{\boldsymbol{h}_\ell\boldsymbol{u}_\ell}\boldsymbol{K}_{\boldsymbol{u}_\ell\boldsymbol{u}_\ell}^{-1}\boldsymbol{K}_{\boldsymbol{u}_\ell\boldsymbol{h}_\ell}$ and $\tilde{p}(\boldsymbol{h}_\ell|\boldsymbol{u}_\ell,\boldsymbol{h}_{\ell-1}) = \mathcal{N}\left(\boldsymbol{h}_\ell|\boldsymbol{\mu}_\ell,\sigma_\ell^2\boldsymbol{I}\right)$, where $\boldsymbol{\mu}_\ell = \boldsymbol{K}_{\boldsymbol{h}_\ell\boldsymbol{u}_\ell}\boldsymbol{K}_{\boldsymbol{u}_\ell\boldsymbol{u}_\ell}^{-1}\boldsymbol{u}_\ell$.

This yields in

$$p(\boldsymbol{h}_L, \{\boldsymbol{h}_\ell\}_{\ell=1}^{L-1} | \boldsymbol{h}_0, \{\boldsymbol{u}_\ell\}_{\ell=1}^{L}) \geq \prod_{\ell=1}^{L} \tilde{p}(\boldsymbol{h}_\ell | \boldsymbol{u}_\ell, \boldsymbol{h}_{\ell-1}) \ \exp\left(\sum_{\ell=1}^{L} -\frac{1}{2\sigma_\ell^2} \operatorname{tr}(\boldsymbol{\Sigma}_\ell)\right). \tag{4.15}$$

Analytical tractability is achieved by introducing a variational distribution for the inducing outputs $q(\boldsymbol{u}_\ell) = \mathcal{N}(\boldsymbol{u}_\ell | \boldsymbol{m}_\ell, \boldsymbol{S}_\ell)$, which is then used to obtain another variational distribution over $\boldsymbol{h}_\ell$

$$q(\boldsymbol{h}_\ell) = \int q(\boldsymbol{u}_\ell) \ \tilde{p}(\boldsymbol{h}_\ell | \boldsymbol{u}_\ell) \ d\boldsymbol{u}_\ell. \tag{4.16}$$

For now, we consider only the first two layers of the DGP and variationally marginalize out the variables associated with the first layer $\boldsymbol{h}_1$ and $\boldsymbol{u}_1$. By taking the appropriate factors from Eq. 4.15 and marginalizing $\boldsymbol{h}_1$, we find

$$\log p(\boldsymbol{h}_2 | \boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{X}) \geq \langle \log \tilde{p}(\boldsymbol{h}_2 | \boldsymbol{h}_1, \boldsymbol{u}_1) \rangle_{\tilde{p}(\boldsymbol{h}_1 | \boldsymbol{u}_1)} - \left\langle \frac{1}{2\,\sigma_2^2} \operatorname{tr}(\boldsymbol{\Sigma}_2) \right\rangle_{\tilde{p}(\boldsymbol{h}_1 | \boldsymbol{u}_1)} - \frac{1}{2\,\sigma_1^2} \operatorname{tr}(\boldsymbol{\Sigma}_1) \ .$$

By marginalizing a second time over the variational distribution $q(\boldsymbol{u}_1)$, $\boldsymbol{u}_1$ can be eliminated from the equation as well. We obtain

$$\begin{aligned} \log p(\boldsymbol{h}_2 | \boldsymbol{u}_2, \boldsymbol{X}) \geq \ & \langle \log \tilde{p}(\boldsymbol{h}_2 | \boldsymbol{h}_1, \boldsymbol{u}_2) \rangle_{q(\boldsymbol{h}_1)} - \mathrm{KL}\left[q(\boldsymbol{u}_1) \| p(\boldsymbol{u}_1)\right] \\ & - \frac{1}{2\,\sigma_2^2} \left\langle \operatorname{tr}(\boldsymbol{\Sigma}_2) \right\rangle_{q(\boldsymbol{h}_1)} - \frac{1}{2\,\sigma_1^2} \operatorname{tr}(\boldsymbol{\Sigma}_1) \ . \end{aligned} \tag{4.17}$$

This bound is further simplified by explicitly computing the expectation over $q(\boldsymbol{u}_1)$. We begin with the first term and simply fill in the definitions

$$\begin{aligned} \langle \log \tilde{p}(\boldsymbol{h}_2 | \boldsymbol{h}_1, \boldsymbol{u}_2) \rangle_{q(\boldsymbol{h}_1)} &= \left\langle -\frac{D_2}{2} \log 2\pi\sigma_2^2 - \frac{1}{2\sigma_2^2} (\boldsymbol{\mu}_2 - \boldsymbol{h}_2)^\top (\boldsymbol{\mu}_2 - \boldsymbol{h}_2) \right\rangle_{q(\boldsymbol{h}_1)} \\ &= -\frac{D_2}{2} \log 2\pi\sigma_2^2 - \frac{1}{2\sigma_2^2} \left(\left\langle \boldsymbol{\mu}_2^\top \boldsymbol{\mu}_2 \right\rangle_{q(\boldsymbol{h}_1)} - 2 \langle \boldsymbol{\mu}_2 \rangle_{q(\boldsymbol{h}_1)}^\top \boldsymbol{h}_2 + \boldsymbol{h}_2^\top \boldsymbol{h}_2 \right), \end{aligned}$$

where D_2 is the dimension of the second hidden layer.

Substituting the definition of $\boldsymbol{\mu}_2$, we obtain

$$\begin{aligned} \langle \log \tilde{p}(\boldsymbol{h}_2 | \boldsymbol{h}_1, \boldsymbol{u}_2) \rangle_{q(\boldsymbol{h}_1)} &= -\frac{D_2}{2} \log 2\pi\sigma_2^2 - \frac{1}{2\sigma_2^2} \left\langle \left(\boldsymbol{K}_{\boldsymbol{h}_2\boldsymbol{u}_2} \boldsymbol{K}_{\boldsymbol{u}_2\boldsymbol{u}_2}^{-1} \boldsymbol{u}_2\right)^\top \left(\boldsymbol{K}_{\boldsymbol{h}_2\boldsymbol{u}_2} \boldsymbol{K}_{\boldsymbol{u}_2\boldsymbol{u}_2}^{-1} \boldsymbol{u}_2\right) \right\rangle_{q(\boldsymbol{h}_1)} \\ &\quad + \frac{1}{2\sigma_2^2} 2 \left\langle \boldsymbol{K}_{\boldsymbol{h}_2\boldsymbol{u}_2} \boldsymbol{K}_{\boldsymbol{u}_2\boldsymbol{u}_2}^{-1} \boldsymbol{u}_2 \right\rangle_{q(\boldsymbol{h}_1)}^\top \boldsymbol{h}_2 - \frac{1}{2\sigma_2^2} \boldsymbol{h}_2^\top \boldsymbol{h}_2 \\ &= -\frac{D_2}{2} \log 2\pi\sigma_2^2 - \frac{1}{2\sigma_2^2} \left(\boldsymbol{u}_2^\top \boldsymbol{K}_{\boldsymbol{u}_2\boldsymbol{u}_2}^{-1} \left\langle \boldsymbol{K}_{\boldsymbol{h}_2\boldsymbol{u}_2}^\top \boldsymbol{K}_{\boldsymbol{h}_2\boldsymbol{u}_2} \right\rangle_{q(\boldsymbol{h}_1)} \boldsymbol{K}_{\boldsymbol{u}_2\boldsymbol{u}_2}^{-1} \boldsymbol{u}_2 \right) \\ &\quad + \frac{1}{2\sigma_2^2} 2 \left(\langle \boldsymbol{K}_{\boldsymbol{h}_2\boldsymbol{u}_2} \rangle_{q(\boldsymbol{h}_1)} \boldsymbol{K}_{\boldsymbol{u}_2\boldsymbol{u}_2}^{-1} \boldsymbol{u}_2 \right) \boldsymbol{h}_2 - \frac{1}{2\sigma_1^2} \boldsymbol{h}_2^\top \boldsymbol{h}_2. \end{aligned}$$

In this derivation we are again naturally confronted with kernel expectations $\langle \boldsymbol{K}_{\boldsymbol{h}_2\boldsymbol{u}_2} \rangle_{q(\boldsymbol{h}_1)}$ and $\left\langle \boldsymbol{K}_{\boldsymbol{h}_2\boldsymbol{u}_2}^\top \boldsymbol{K}_{\boldsymbol{h}_2\boldsymbol{u}_2} \right\rangle_{q(\boldsymbol{h}_1)}$. In Section 4.4 we deal with them in more detail, for now we only rename

them to keep the notation compact: $\langle \boldsymbol{K}_{\boldsymbol{h}_2\boldsymbol{u}_2} \rangle_{q(\boldsymbol{h}_1)}$ is renamed to $\boldsymbol{\Omega}_2$ and $\left\langle \boldsymbol{K}_{\boldsymbol{h}_2\boldsymbol{u}_2}^\top \boldsymbol{K}_{\boldsymbol{h}_2\boldsymbol{u}_2} \right\rangle_{q(\boldsymbol{h}_1)}$ to $\boldsymbol{\Phi}_2$. Note that the subscript $\cdot_2$ is indicating the layer.

$$\begin{aligned}\langle \log \tilde{p}(\boldsymbol{h}_2|\boldsymbol{h}_1, \boldsymbol{u}_2) \rangle_{q(\boldsymbol{h}_1)} &= -\frac{D_2}{2} \log 2\pi\sigma_2^2 - \frac{1}{2\sigma_2^2} \left(\boldsymbol{u}_2^\top \boldsymbol{K}_{\boldsymbol{u}_2\boldsymbol{u}_2}^{-1} \boldsymbol{\Phi}_2 \boldsymbol{K}_{\boldsymbol{u}_2\boldsymbol{u}_2}^{-1} \boldsymbol{u}_2 \right) \\ &\quad + \frac{1}{2\sigma_2^2} 2 \left(\boldsymbol{\Omega}_2 \boldsymbol{K}_{\boldsymbol{u}_2\boldsymbol{u}_2}^{-1} \boldsymbol{u}_2 \right)^\top \boldsymbol{h}_2 - \frac{1}{2\sigma_2^2} \boldsymbol{h}_2^\top \boldsymbol{h}_2 \\ &= \log \mathcal{N}\left(\boldsymbol{h}_2 | \boldsymbol{\Omega}_2 \boldsymbol{K}_{\boldsymbol{u}_2\boldsymbol{u}_2}^{-1} \boldsymbol{u}_2, \sigma_2^2 \right) - \frac{1}{2\sigma_2^2} \operatorname{tr}\left(\boldsymbol{u}_2^\top \boldsymbol{K}_{\boldsymbol{u}_2\boldsymbol{u}_2}^{-1} (\boldsymbol{\Phi}_2 - \boldsymbol{\Omega}_2^\top \boldsymbol{\Omega}_2)^\top \boldsymbol{K}_{\boldsymbol{u}_2\boldsymbol{u}_2}^{-1} \boldsymbol{u}_2 \right).\end{aligned}$$

Substituting the above result in Eq. 4.17 and using a third type of kernel expectation $\xi_2 = \langle \operatorname{tr}(\boldsymbol{K}_{\boldsymbol{h}_2\boldsymbol{h}_2}) \rangle_{q(\boldsymbol{h}_1)}$ to simplify $\operatorname{tr}(\boldsymbol{\Sigma}_2)$, we obtain a lower bound for the second layer for which the variables of the first layer are eliminated through marginalization

$$\begin{aligned}\log p(\boldsymbol{h}_2|\boldsymbol{u}_2, \boldsymbol{X}) \geq \log \mathcal{N}\left(\boldsymbol{h}_2 | \boldsymbol{\Omega}_2 \boldsymbol{K}_{\boldsymbol{u}_2\boldsymbol{u}_2}^{-1} \boldsymbol{u}_2, \sigma_2^2 \right) \\ - \frac{1}{2\sigma_2^2} \operatorname{tr}\left(\boldsymbol{u}_2^\top \boldsymbol{K}_{\boldsymbol{u}_2\boldsymbol{u}_2}^{-1} (\boldsymbol{\Phi}_2 - \boldsymbol{\Omega}_2^\top \boldsymbol{\Omega}_2)^\top \boldsymbol{K}_{\boldsymbol{u}_2\boldsymbol{u}_2}^{-1} \boldsymbol{u}_2 \right) \\ - \mathrm{KL}\left[q(\boldsymbol{u}_1) \| p(\boldsymbol{u}_1) \right] \\ - \frac{1}{2\,\sigma_2^2} \left(\xi_2 - \operatorname{tr}\left(\boldsymbol{\Phi}_2 \boldsymbol{K}_{\boldsymbol{u}_2\boldsymbol{u}_2}^{-1} \right) \right) - \frac{1}{2\,\sigma_2^2} \operatorname{tr}(\boldsymbol{\Sigma}_1) \,.\end{aligned}$$

To summarize, this bound was found by first computing $q(\boldsymbol{h}_1)$ for the given dataset $\boldsymbol{X}$ and fixed variational distribution $q(\boldsymbol{u}_1)$, according to Eq. 4.16. Then, using the subsequent variational distribution $q(\boldsymbol{u}_2)$ it is possible to variationally propagate $q(\boldsymbol{h}_1)$ to obtain $q(\boldsymbol{h}_2)$. This feed-forward movement is continued until the last layer is reached. The variational propagation will introduce a regularizing term for each layer, which will affect the final bound on the complete marginal likelihood. Thus, by recursively propagating distribution $q(\boldsymbol{h}_\ell)$ through the $(\ell+1)^{\text{th}}$ layer, we obtain a lower bound on the marginal likelihood of an arbitrarily deep network

$$\begin{aligned}\log p(\boldsymbol{y}|\boldsymbol{X}) \geq \log \mathcal{N}\left(\boldsymbol{y} | \boldsymbol{\Omega}_L \boldsymbol{K}_{\boldsymbol{u}_L\boldsymbol{u}_L}^{-1} \boldsymbol{m}_L, \sigma_L^2 \right) \\ - \sum_{\ell=2}^{L} \frac{1}{2\sigma_\ell^2} \operatorname{tr}\left(\boldsymbol{m}_\ell^\top \boldsymbol{K}_{\boldsymbol{u}_\ell\boldsymbol{u}_\ell}^{-1} (\boldsymbol{\Phi}_\ell - \boldsymbol{\Omega}_\ell^\top \boldsymbol{\Omega}_\ell)^\top \boldsymbol{K}_{\boldsymbol{u}_\ell\boldsymbol{u}_\ell}^{-1} \boldsymbol{m}_\ell \right) \\ - \sum_{\ell=1}^{L} \mathrm{KL}\left[q(\boldsymbol{u}_\ell) \| p(\boldsymbol{u}_\ell) \right] \\ - \sum_{\ell=2}^{L} \frac{1}{2\,\sigma_\ell^2} \left(\xi_\ell - \operatorname{tr}\left(\boldsymbol{\Phi}_\ell \boldsymbol{K}_{\boldsymbol{u}_\ell\boldsymbol{u}_\ell}^{-1} \right) \right) - \frac{1}{2\,\sigma_1^2} \operatorname{tr}(\boldsymbol{\Sigma}_1) \,. \qquad (4.18)\end{aligned}$$

Studying Eq. 4.18 we note that only the first term is similar to a traditional GP likelihood, the other terms will have the task of regularizing the expression. This bound is parameterized by the kernel hyperparameters (different hyperparameters and kernels can be used in every layer), the variational distributions of the inducing outputs $q(\boldsymbol{u}_\ell)$ and the inducing inputs $\boldsymbol{Z}_\ell$. The variables corresponding to the hidden layers $\boldsymbol{h}_\ell$ are marginalized, but influence the bound through the kernel expectations. It should be noted that other bounds exist, the most notable are Damianou and Lawrence (2013), Bui et al. (2016) and Salimbeni and Deisenroth (2017).

However, in Chapter 5 the bound of Eq. 4.18 will be used. A computationally very important advantage of this lower bound, compared to the one derived in Damianou and Lawrence (2013), is that in Eq. 4.18 each part can be written as a sum over every individual datapoint. This enables parallelization, mini-batch as well as stochastic methods during learning and inference. Additionally, it circumvents the need to deal with a large number of hidden variables $\boldsymbol{h}_\ell$ when N increases. The difference between both lower bounds is that in Damianou and Lawrence (2013) the inducing outputs $\boldsymbol{u}_\ell$ are collapsed instead of variational and the hidden variables $\boldsymbol{h}_\ell$ are variational instead of marginalized. This approach results also in a tight lower bound, but has as downside that the latent layers are not factorisable over the datapoints anymore, which makes it less scalable towards big data.

4.4 Kernel Expectations

We complete this chapter by studying the so-called kernel expectations, also referred to as the $\{\xi, \Omega, \Phi\}$-statistics (Girard et al. 2003). Since their introduction, kernel expectations are playing a crucial role in many advanced GP-based methods, such as models dealing with input uncertainty (Section 4.1), Bayesian GP-LVMs (Section 4.2) and DGPs (Section 4.3). Also in other GP frameworks, like GP State-Space Models (GP-SSMs) and Bayesian reinforcement learning, are they very important (Turner et al. 2010, Deisenroth and Rasmussen 2011).

The kernel expectations are calculated by taking the expectation of the kernel (or the product of two kernels) with respect to a normal distribution. In some applications (e.g., input uncertainty propagation) this normal distribution belongs to a stochastic input, while in other models (e.g., Bayesian GP-LVM and DGP) the distribution is variational. Here we assume a general normal distribution $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\gamma}, \boldsymbol{S})$. In practice, the models require three types of kernel expectations:

$$\begin{aligned}
\xi &= \int k(\boldsymbol{x}, \boldsymbol{x})\, \mathcal{N}(\boldsymbol{x}|\boldsymbol{\gamma}, \boldsymbol{S})\, \mathrm{d}\boldsymbol{x}, \\
\Omega(\boldsymbol{y}) &= \int k(\boldsymbol{x}, \boldsymbol{y})\, \mathcal{N}(\boldsymbol{x}|\boldsymbol{\gamma}, \boldsymbol{S})\, \mathrm{d}\boldsymbol{x}, \\
\Phi(\boldsymbol{y}, \boldsymbol{z}) &= \int k(\boldsymbol{x}, \boldsymbol{y})\, k(\boldsymbol{x}, \boldsymbol{z})\, \mathcal{N}(\boldsymbol{x}|\boldsymbol{\gamma}, \boldsymbol{S})\, \mathrm{d}\boldsymbol{x}.
\end{aligned}$$

Note that although these kernel expectations also depend on the mean $\boldsymbol{\gamma}$ and the covariance matrix $\boldsymbol{S}$ of the distribution, these parameters are not added to the list of inputs. The mathematically more correct way would be to write $\xi(\boldsymbol{\gamma}, \boldsymbol{S})$, $\Omega(\boldsymbol{\gamma}, \boldsymbol{S}, \boldsymbol{y})$ and $\Phi(\boldsymbol{\gamma}, \boldsymbol{S}, \boldsymbol{y}, \boldsymbol{z})$, but in order the notation less unwieldy this is often neglected.

For most stationary kernels, calculating ξ is easy as $\boldsymbol{\tau} = \boldsymbol{x} - \boldsymbol{x} = \boldsymbol{0}$, which simplifies the integrand significantly. The other two kernel expectations are often a lot more difficult to calculate analytically. While it is always possible to reside to approximate Monte-Carlo integration or quadrature rules, this can quickly lead to a computational bottleneck. Fortunately, for some popular choices of kernel functions, including the ARD SE (Eq. 2.14) and Linear kernel, they can be computed in closed form (see Damianou (2015)). Section 6.3 is devoted to calculating the $\{\xi, \Omega, \Phi\}$-statistics for the Spectral Mixture kernel.

Chapter 5

Metamodeling of Sequentially Sampled Non-Stationary Surfaces

Simulations are often used for the design of complex systems as they allow to explore the design space without the need to build several prototypes. However, over the years, the simulation accuracy, as well as the associated computational cost has increased significantly, limiting the overall number of simulations during the design process. Therefore, metamodeling aims to approximate the simulation response with a cheap-to-evaluate mathematical approximation, learned from a limited set of simulator evaluations.

Different supervised machine learning algorithms can be used for metamodeling. Among the most popular ones are Artificial Neural Networks (ANNs), Kriging, Gaussian Processes (GPs) and Least Squares - Support Vector Machines (LS-SVMs). The latter three belong to the class of kernel-based algorithms as they make use of a kernel function to transform the input vectors to a different (possibly infinite) space. When using kernel-based methods, the algorithm designer is faced with choosing an appropriate kernel for the problem at hand. Despite the fact that there are numerous available kernels, stationary kernels are nowadays wildly used. This is mainly due their ease-of-use and their limited set of hyperparameters, as was depicted in Section 2.5. However, using stationary kernels for non-stationary responses can be inappropriate and result in poor models when combined with sequential design. In this chapter, we use Deep Gaussian Processes (DGPs), a hierarchical stacking of multiple GPs, to circumvent these shortcomings (Damianou and Lawrence 2013, Hensman and Lawrence 2014). More particularly, we apply DGP configured with stationary kernels to approximate non-uniformly sampled, non-stationary response surfaces.

The chapter is organized as follows. In Section 5.1 we introduce metamodeling. Subsequently, we focus on Design of Experiment (DoE) and sequential design. In Section 5.3 we identify the limitations of GPs using stationary kernels to model non-stationary response surfaces. In Section 5.4 and 5.5 we empirically show the benefits of using DGPs over other metamodeling techniques by conducting a series of experiments.

5.1 Metamodeling

During the design and analysis of complex systems, computer simulations are often used to limit the number of expensive prototypes or real-life measurements. While adding more flexibility to study phenomena under controlled conditions, high-accuracy computer simulations often require a substantial investment of computational resources and time. A single simulation may easily take several hours or even days to complete. This is especially problematic for routine tasks such as optimization, sensitivity analysis and design space exploration. As a result, over the past decade, a lot of research has focused on alleviating these computational issues. An accomplished technique is the construction of a mathematical equation which approximates the behavior of the simulation as closely as possible, while being computationally cheap(er) to evaluate. If the approximation is based on a (limited) set of simulator evaluations, this approximation is known as a metamodel (Kleijnen 2015, Gorissen et al. 2010). The metamodel may then be used instead of the computationally expensive simulator to derive characteristics of the problem at hand or to search for optima. More formally, we sample the expensive simulation N times at different locations $\boldsymbol{x}_i$ in order to get the simulation response $f_i \triangleq f(\boldsymbol{x}_i)$ and then collect these into a (training) dataset $\{\boldsymbol{x}_i, f_i\}_{i=1}^N$. Different than in the other chapters, in this chapter we assume that the simulation is deterministic so that the outputs are actually noise-free (with the exception of quantization errors due to the finite representations of real numbers). This dataset is then used to construct a metamodel, aiming to approximate the real simulation response surface as accurately as possible, so it can replace the simulation for prediction of new points $\boldsymbol{x}_*$ in the design space.

Many different supervised machine learning techniques can be used to obtain the metamodel. Among the most popular ones are ANN (Fonseca et al. 2003), Kriging (Santner, Williams, and Notz 2013, Forrester, Sobester, and Keane 2008), GPs (Rasmussen and Williams 2006, Gorissen et al. 2010) and LS-SVMs (Suykens and Vandewalle 1999). The latter three metamodels are kernel-based regression methods, for which the prediction is a linear combination of kernel evaluations between the test and data points. This can be written as

$$f(\boldsymbol{x}_*) = \sum_{i=1}^{N} w_i \, k(\boldsymbol{x}_*, \boldsymbol{x}_i),$$

where k is the kernel used and N is the number of training data points. In Chapter 2 different type of kernels are discussed, such as the Matérn, Periodic, Linear and Spectral Mixture (SM) kernel. However, from practice we known that in metamodeling communities the Squared Exponential (SE) is definitely the most well-known and very often used. For completeness, we repeat the definition of the Automatic Relevance Determination (ARD) SE kernel here, but we refer the reader to Section 2.5.1 for more details

$$k_{SE}(\boldsymbol{\tau}) = \sigma_k \exp\left(-\sum_{d=1}^{D} \frac{{\tau^{(d)}}^2}{2\,\ell_d^2}\right).$$

It is important to note that the ARD SE kernel has only a single parameters ℓ_d per dimension.

GPs and Kriging models have a number of advantages which make them particularly attractive to use in metamodeling scenarios. The learning process involves optimization of a cost function which automatically guards against overfitting. This typically results in good generalization behavior. The methods have comparatively few hyperparameters (in most cases only a couple of hyperparameters which parametrize the kernel) which makes them favorable to use when the dataset is limited in size. It is also possible to get a full confidence estimate on the predictions, i.e. for every prediction we get the accompanying uncertainty of the model. This greatly supports the design of sequentially constructed datasets, discussed in Section 5.2.

One of the main difficulties of kernel-based methods is the selection of a suitable kernel, which should represent the particular properties of the response under study. Unfortunately, most of the properties of the response surface such as the smoothness, linearity or non-stationarity, are usually unknown at the start of the metamodeling process. If sufficient data is available and uniformly distributed over the input space this problem is solved by using a stationary kernel, such as the SE, as they are universally applicable under these conditions. However, requiring a dataset to be sufficiently large and uniformly distributed is often not possible in metamodeling. In the context of metamodeling, constructing a large dataset is challenging as it involves numerous computationally expensive simulation evaluations. Furthermore, efficient metamodeling procedures frequently make use of sequential sampling methodologies which may result in non-uniform distributions (Crombecq et al. 2009). As will be illustrated in the remainder of this chapter, the use of stationary SE kernel under these conditions can result in poor models as it becomes impossible to approximate the response surface using a single set of lengthscales.

5.2 Sequential Design

A crucial step in the metamodeling process is the selection of the data points, i.e. inputs and corresponding simulation evaluations, to build the metamodel. During the selection procedure, there are two counter-working forces. On the one side, we try to keep the dataset as small as possible in order to limit the number of expensive simulation evaluations. On the other side, it is of paramount importance that the dataset is large and informative enough in order to build an accurate model that covers the complete design space. In traditional Design of Experiments (DoE) the dataset is selected upfront through a one-shot approach, such as factorial designs (Lehmensiek et al. 2002) and optimized Latin hypercubes (Park 1994).

As these selection procedures do not take into account any prior knowledge about the simulation response surface at hand they make an arbitrary choice in the number of samples required. This can lead to a metamodel constructed with too few samples (undersampling) which has a displeasing performance. On the contrary, selecting too many samples (oversampling) results in wasting computational resources and increases the learning time. As a solution, sequential design of the dataset is nowadays often used to improve the metamodeling efficiency and accuracy. In sequential designs the one-shot approach is turned into an iterative process where the data acquired from previous iterations is analyzed in order to intelligently select locations for new data points. In addition to preventing under/over sampling, the iterative procedure can be used to

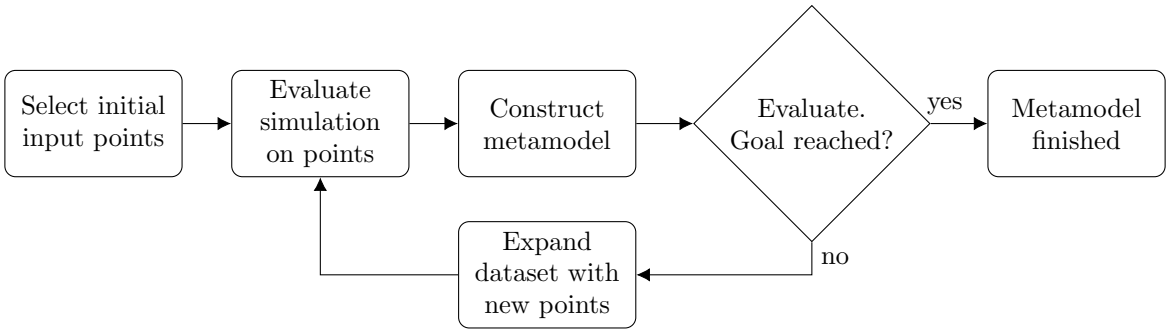


Figure 5.1: The sequential design metamodeling paradigm.

halt the construction of the metamodel once the stopping criteria are met, avoiding unnecessary simulator evaluations. The complete procedure is as follows. The process starts by selecting a small initial dataset through a one-shot approach. Then, the metamodel is constructed and the stopping criteria (e.g., Is the metamodel sufficiently accurate according to a quality criterion? Has the time limit or total number of allowed evaluations been exceeded?) are evaluated. When the criteria are not met, new points are intelligently selected in demanding regions and the procedure restarts. This is illustrated in the flowchart of Figure 5.1.

FLOLA-Voronoi (van der Herten et al. 2015) is a frequently used algorithm for the sequential selection of datapoints. It identifies highly irregular and non-linear regions in the response surface and will accordingly increase the number of samples in these regions. The FLOLA-Voronoi algorithm combines a gradient estimation in the datapoints, based on a locally linear approximation, for exploitation, with a Voronoi space-filling criterion for exploration. FLOLA-Voronoi offers a fuzzy approach for determining the neighbors in the exploitation step, increasing the efficiency in higher dimensions over regular LOLA-Voronoi. By doing this, it assists the metamodel in approximating the more difficult regions while limiting the number of expensive simulator evaluations needed. The algorithm will be used in the reported experiments.

5.3 Gaussian Processes for Metamodeling

Gaussian Processes are a well-established class of metamodels, which are popular due to their analytical tractability, a training procedure which automatically guards against overfitting and their ability to provide a full predictive distribution over the outputs. Even tough the GP model is already discussed in great detail in previous chapters, in this section we focus again on defining GPs, but for metamodeling purposes. Furthermore, we point out the major shortcomings of using traditional GPs as go-to metamodel.

Constructing the GP metamodel (as in the 3$^{\text{rd}}$ step of Figure 5.1) consists of finding the optimal set of hyperparameters $\boldsymbol{\theta}_k$. This is done by optimizing the likelihood function $p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}_k)$ in order to *fit* the data. Denoting the training input data as $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^N$ and collecting the noisy simulation evaluations in $\boldsymbol{y}$, i.e. $y_i = f_i + \epsilon_i$ with $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$, this can be written as the

Probability Density Function (PDF) of a multivariate normal distribution

$$p(\boldsymbol{y}|\boldsymbol{X},\boldsymbol{\theta}_k) = (2\pi)^{-\frac{N}{2}}|\boldsymbol{K_{ff}} + \sigma^2\boldsymbol{I}|^{-\frac{1}{2}}\exp\left(-\frac{1}{2}\boldsymbol{f}^\top(\boldsymbol{K_{ff}} + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{f}\right), \tag{5.1}$$

In metamodeling σ^2 is typically chosen to be zero as we assume that there is no noise on the simulation evaluations.

Once the optimal hyperparameters for the dataset are determined, the model can be used to predict the response at a new point in the design space $\boldsymbol{x}_*$. As explained in Chapter 2, GPs are able to provide a full predictive distribution for the simulation evaluation f_*, which is again characterized by a Gaussian distribution

$$f_*|\boldsymbol{y},\boldsymbol{X},\boldsymbol{x}_* \sim \mathcal{N}(\mu_*,\sigma_*^2), \tag{5.2}$$

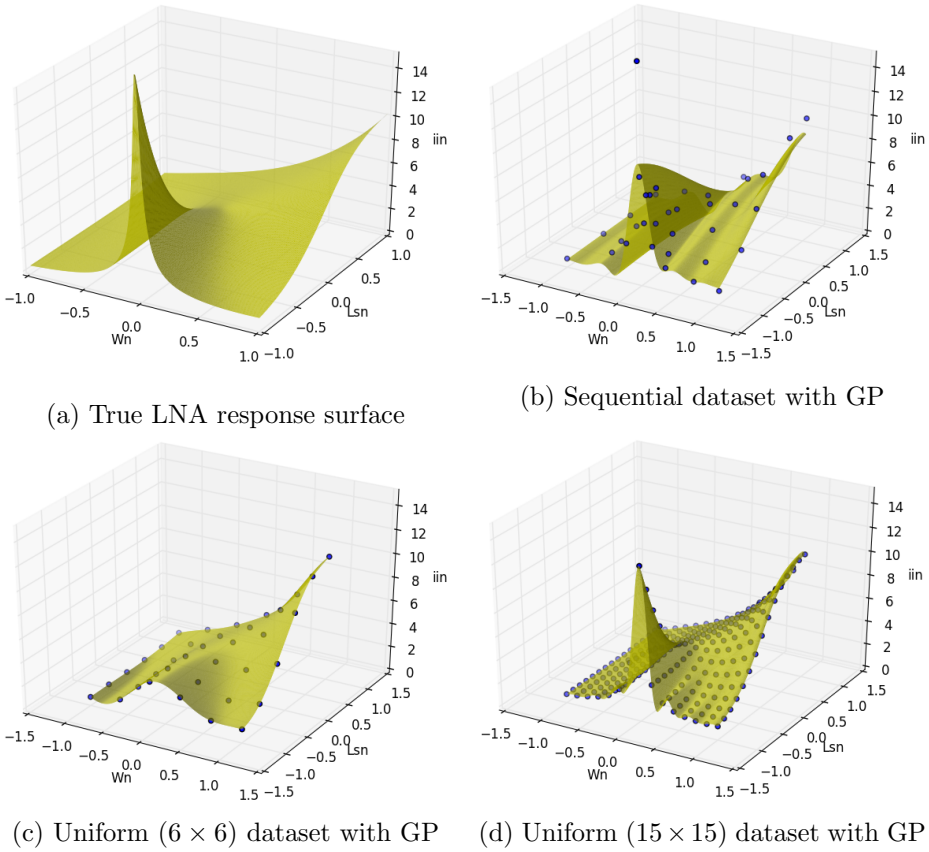
with

$$\mu_* = \boldsymbol{k}_*^\top(\boldsymbol{K_{ff}} + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{f} \quad \text{and} \quad \sigma_*^2 = k_{**} - \boldsymbol{k}_*^\top(\boldsymbol{K_{ff}} + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{k}_*,$$

where $k_{**} = k(\boldsymbol{x}_*,\boldsymbol{x}_*)$ and $\boldsymbol{k}_* = k(\boldsymbol{X},\boldsymbol{x}_*)$ stands for the vector $(N\times 1)$ of covariance evaluations between the single input point $\boldsymbol{x}_*$ and all the training input points.

The advantage of obtaining a complete predictive distribution for new test points is that the mean μ_* can conveniently be used as a point estimation of the simulation response and that the variance σ_* can be used to quantify the uncertainty of the model at that point. This can be extremely valuable for critical application, such as in Section 5.5. However, the downside is that the calculation of the likelihood (5.1), as well as the predictive distribution (5.2) are of the order $\mathcal{O}(N^3)$ due to the need to compute the Cholesky decomposition for inversion of $\boldsymbol{K_{ff}}$. This makes the usage of GPs problematic when the dataset becomes very large.

The choice of kernel is crucial as it encodes the properties of the response surface, which is usually unknown a priori. Frequently, stationary kernels such as the SE are used successfully. However, this type of kernels applied to non-stationary problems can be troublesome in combination with sequential design. Consider, for example, Figure 5.2a, showing the true response surface of a Low Noise Amplifier (LNA) (Gorissen et al. 2008) and Figure 5.2b, showing a GP model with SE kernel, fitted on a sequential dataset (blue dots). The sequential dataset has a much higher sampling density in the irregular regions of the response surface (e.g., near the peak) than in the flatter regions. We observe that the approximating GP has undesirable behavior as it oscillates in the undersampled regions and ignores the sample on the peak. This problem is mainly due to the fact that the GP is using a stationary kernel, which is only able to detect a single lengthscale ℓ_d for each dimension. However, for the given dataset on this LNA-surface, short lengthscales are required near the peak whereas longer lengthscales are needed for the smooth surroundings. As a result, the likelihood optimization results in a trade-off. On the one hand, the lengthscales are still too broad causing the sample on the peak to be ignored. On the other hand, the lengthscales are still too short in the flat regions, which causes the fluctuating behavior of the approximation as the GP is reverting back to its mean between the datapoints.



(a) True LNA response surface

(b) Sequential dataset with GP

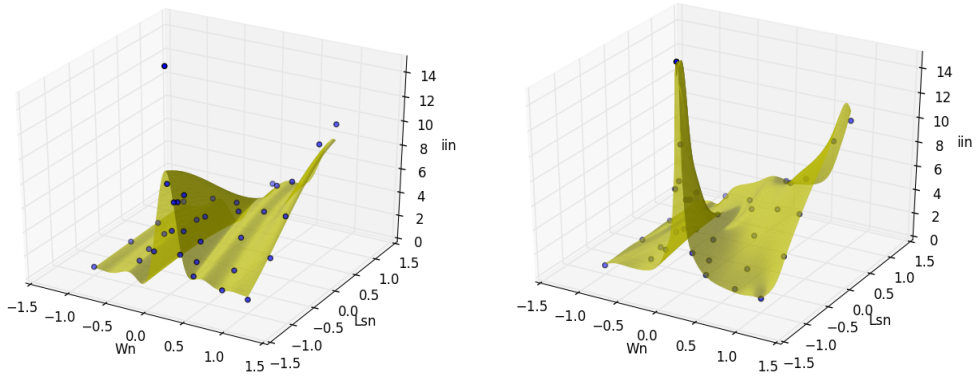
(c) Uniform (6×6) dataset with GP

(d) Uniform (15×15) dataset with GP

Figure 5.2: Impact of different sampling strategies on the GP metamodel of the LNA surface.

The problems introduced by the sequential dataset could be solved by simply not using sequential design and making use of the easier one-shot, uniform sampling approaches. As visible in Figures 5.2c and 5.2d, for these uniform schemes the fitting of the non-stationary LNA response surface is much more satisfactory. However, in order to represent the highly irregular response surface with a uniform grid, a dense training set is required, ensuring us that even sharp non-linear behavior is represented. This is also illustrated in the plots: the model of 5.2c is constructed using a 6-by-6 undersampled grid and the model of 5.2d is constructed using a 15-by-15 oversampled grid. The first dataset is clearly less representative as the high peak at $(0, 0)$ is only slightly present. Although this problem can be solved partially using a denser grid, as is done in 5.2d, gathering such a grid is significantly more expensive as it involves many additional computationally expensive simulations. The problem becomes even more troublesome for high dimensional systems due to the fact that in order to keep the sampling density at the same level, we need to increase the number of samples exponentially as a consequence of the *curse of dimensionality*. Even more, increasing the number of samples in the dataset does not only increase the time required to construct the dataset, it also increases the time required to optimize the hyperparameters, which scales $\mathcal{O}(N^3)$ with N the number of training samples, as discussed in Chapter 3.

In conclusion, neither uniform nor sequential datasets result in desirable GP metamodels for highly non-stationary surfaces. Uniform datasets are often too computationally expensive to compose, while sequential datasets complicate the GP training procedure.



(a) Sequential dataset with GP (b) Sequential dataset with DGP

Figure 5.3: Difference of the GP and DGP metamodel, optimized on the same sequential LNA dataset, consisting of 36 points.

Therefore, we propose the use of DGPs as they provide a computationally efficient learning and inference procedure and are significantly more flexible. As seen in Section 4.3, DGPs are a multi-layer generalization of GP functions, able to overcome the limitations of traditional GPs while retaining the advantages. Besides, they are robust to over-fitting and provide well-calibrated uncertainty estimations.

In Figure 5.3b we see the result of using a 5-layer DGP metamodel for the sequentially sampled, non-stationary LNA response surface. Note that the DGP and GP model are trained on the same dataset, but that the DGP is able to model the surface much more accurately. This is due to the composition of the multiple stacked GPs, each having their own kernel with appropriate lengthscales. Every layer of the model is able to learn a specific part of the pattern which results in a model that is able to accurately approximate the highly irregular region and the flatter regions around the peak simultaneously. Furthermore, we see that by combining DGPs with sequential sampling we are able to construct a regression model that can capture the highest value at the peak. In the uniform sampling case (Figure 5.2c), the training set did not contain this value so the model was not aware of the existence and consequently did not model it. However, in the case of Figure 5.3a, the sample at the top of the peak is present in the training dataset but the standard GP is not able to model it, as it would require different lengthscales to do so.

To conclude, it is clearly visible that the DGP is outperforming the GP on the sequentially sampled LNA surface. In the next sections we numerically asses this observation and conduct a second experiment to double-check it.

5.4 Experimental Setup

The GP models used to conduct the experiments belong to GPflow (Matthews et al. 2017), a Gaussian Process library that uses TensorFlow for its core computations and Python for its front-end. We report the performance of four different models implemented in the package: a standard GP model (GP), a variational sparse GP (SGP) (Titsias 2009), a scalable version of

the variational sparse GP (SVGP) (Hensman et al. 2013) and an implementation of a DGP (Hensman and Lawrence 2014). We configured the models with the same stationary SE kernel with Automatic Relevance Determination (ARD) lengthscales and the variational models all have the same number of inducing points. For the DGP model, the inducing point locations and the lengthscales of the first GP layer are chosen to correspond with the centers of k-means clustering and the median distance between the input points of the training dataset, as suggested by Bui et al. (2016). The DGP is trained in two stages. First, the locations of the inducing points are fixed and only the kernel hyperparameters are optimized. After this first optimization round, all the hyperparameters, i.e. the locations of the inducing points and the kernel hyperparameters, are optimized jointly. This routine was found empirically to be much less prone to local minima as we assist the optimizer by limiting the optimization space in the first stage. The performance of the models is evaluated on a test set using four metrics: Root Mean Square Error (RMSE), Root Relative Square Error (RRSE), Bayesian Estimation Error Quotient (BEEQ) and the Mean Square Error (MSE), which is the square of the RMSE (Li and Zhao 2006):

$$\mathrm{RMSE}(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \tilde{y}_i)^2}{N}},$$

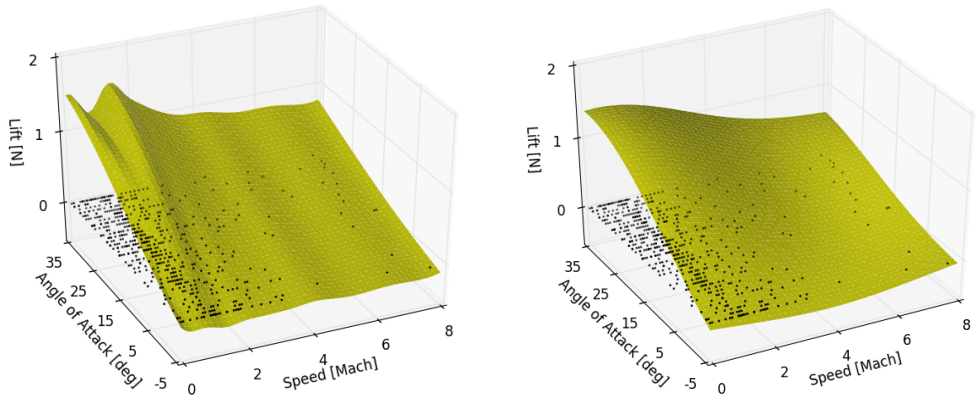
$$\mathrm{RRSE}(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \tilde{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}},$$

$$\mathrm{BEEQ}(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = \sqrt[N]{\prod_{i=1}^{N}\frac{|y_i - \tilde{y}_i|}{|y_i - \bar{y}|}}.$$

5.5 Results

5.5.1 The Langley Glide-Back Booster Simulation

The Langley Glide-Back Booster (LGBB) (Gramacy and Lee 2008) is a proposed rocket booster designed by NASA. A rocket booster is an instrument which assists the space shuttle during the launch process and then parachutes back to Earth once the fuel has been exhausted. The trajectory of a returning rocket booster is carefully planned so that they fall into the ocean, where they can be recovered and reused for subsequent missions. The design of such an LGBB heavily relies on simulators as wind tunnel experiments are still not fully realistic and quite expensive. More particularly, the re-entering of the rocket booster into the Earth's atmosphere is one of the most critical moments of the trajectory. Physically recreating such an environment is almost infeasible and hence simulations are turned to as they form an attractive and cheaper alternative. The particular simulator of a LGBB models the forces applied on the vehicle when it is re-entering the atmosphere. It is based on a computational fluid dynamics system and solves for approximately 1.5 million points the relevant Euler equations. One run of the simulator, for a single set of input parameters, can easily take between 5 to 20 hours on NASA's supercomputers. Therefore, NASA was interested in constructing a metamodeling of the simulation as it allows them to steer the design into promising directions and identify potential problems at an early



(a) DGP (b) GP

Figure 5.4: Two metamodels constructed using the same dataset (scattered dots) of the LGBB response:. (a) DGP's predicted mean of the LGBB's lift response in function of the speed and the angle of attack and (b) the GP's predicted mean. The scattered dots show the non-uniformly distributed, 2D input dataset used to construct the models.

stage. For this kind of critical applications Kriging and GP metamodels can be highly valuable as they are able to provide the uncertainty for each prediction.

The two most dominant input parameters of the simulator are the angle of attack (deg) and the speed at re-entry (Mach). The output of the simulator are the six relevant forces that can be applied on a free body in space: lift, drag, pitch, side-force, yaw, and roll. However, in the case of space-vehicle design, we only focus on the lift force, as that is the most important one for keeping a vehicle aloft. Figure 5.4a shows the lift response plotted as a function of speed and angle of attack with the other parameters fixed at a constant value. The large ridge at 1 Mach separates subsonic from supersonic cases. This transition from subsonic to supersonic is distinctly non-linear and non-continuous while the other parts of the surface are quite smooth. The scattered dots on the plot correspond to the input locations of the given dataset (i.e. the points where the simulation is evaluated) which should be used to construct the metamodels. We observe that this dataset is constructed using a sequential technique as the sampling density in non-stationary regions is significantly higher. Therefore, we expect GPs with stationary kernels to have again issues approximating this response surface.

Table 5.1 summarizes the test error metrics of the different optimized GP models. The DGP outperforms the other models as it is the only one able to accurately approximate both, the non-stationary ridge at the transition from subsonic to supersonic and the smoother regions (Figure 5.4a). Consulting the predicted mean of the GP model (Figure 5.4b), we observe that it ignores the ridge and models the entire surface as completely smooth. This is due to the fact that during the optimization, the GP has to choose between a configuration with large or small lengthscales. The latter one would result in highly oscillating behavior of the predicted mean surface. Although this configuration would be able to model the ridge correctly, it would fail to model the smoother regions. On the contrary, choosing larger lengthscales leads to a completely smooth approximation, which is not able to model the non-stationary ridge. After optimization the GP resulted in the latter, as visible in the figure.

Table 5.1: Test error metrices, calculated on 156 test points, of the GP metamodels on the LGBB dataset.

	RRSE	RMSE	MSE	BEEQ
GP	0.16	0.079	0.61E−2	0.14
SGP	0.22	0.11	1.11E−2	**0.042**
SVGP	0.16	0.077	0.59E−2	0.12
DGP	**0.053**	**0.025**	**0.06E−2**	0.043

5.5.2 Low Noise Amplifier

As a second application we revisit the Low Noise Amplifier (LNA) modeling problem (Gorissen et al. 2008). An LNA is an electronic amplifier used to amplify very weak signals (for example, captured by an antenna). It is usually located very close to the detection device and is frequently used in microwave systems like GPS. An LNA is typically the first stage of a receiver, having the main function of providing the gain needed to suppress the noise of subsequent stages, such as a mixer. It has to give negligible distortion to the signal while adding as little noise as possible. The performance figures of an LNA (gain, input impedance, noise figure and power consumption) can be determined by means of computer simulations where the underlying physics are accurately taken into account. Each simulation typically requires about a minute, which is too long for a circuit designer to work with efficiently. Instead, a designer could use a very accurate metamodel to quickly explore how performance figures of the LNA scale with key circuit-design parameters, such as the dimensions of transistors, passive components, signal properties and bias conditions. The goal of the design process is to figure out one or more sets of design parameters resulting in a circuit which fulfills the specifications. For this example, we now consider the modeling input noise-current (i_{in}) in function of the normalized transistor width (Wn) and inductance (Lsn). The relation to the width and inductance quantities are given by $\mathrm{W} = 100\mathrm{E}{-6} \times 10^{\mathrm{Wn}}$ m and $\mathrm{Ls} = 0.1\mathrm{E}{-9} \times 10^{\mathrm{Lsn}}$ H. The true surface is shown in Figure 5.2a. This surface is sampled by using the state-of-the-art FLOLA-Voronoi algorithm (van der Herten et al. 2015). Its strengths are that it is able to automatically identify non-linear regions in the input domain and sample these more densely. In this way, we limit the number of computationally expensive simulations but introduce a non-uniform dataset. See Figure 5.2b for an example of a sequentially sampled dataset consisting of 36 points.

If we study Table 5.2 it shows that the traditional GP models have again difficulty capturing the non-stationary LNA response surface. This statement was already discussed in the context of Figure 5.3 for $N = 36$, but is also true for $N = 60$, as visible by comparing Figures 5.5a and 5.5b. The DGP is the only model able to behave smooth around the peak, while simultaneously approximating the slope accurately. On the contrary, the GP models the slope reasonably well but starts to fluctuate too heavily in the surrounding region.

It is interesting to note that when the number of training samples N increases, the performance of the GP decreases while the performance of the DGP ameliorates. When a sequential dataset gets larger, more and more samples will be located in the irregular peak region. As a result,

the GP model will shrink its lengthscales to models these samples more accurately. However, this causes heavier oscillating behavior in the smooth regions, which explains the degradation in performance (see Figure 5.5c up to Figure 5.5f). DGPs do not suffer from this and show better performance with increasing N.

Table 5.2: Test error metrices of the GP metamodels on the sequentially sampled LNA dataset.

	$N = 30$				$N = 50$			
	RRSE	RMSE	MSE	BEEQ	RRSE	RMSE	MSE	BEEQ
GP	0.38	0.84	0.70	0.14	0.73	1.63	2.65	0.39
SGP	0.38	0.84	0.70	0.15	0.73	1.62	2.63	0.38
SVGP	0.37	0.83	0.69	0.13	0.57	1.27	1.61	0.32
DGP	**0.25**	**0.55**	**0.30**	**0.10**	**0.19**	**0.41**	**0.17**	**0.095**

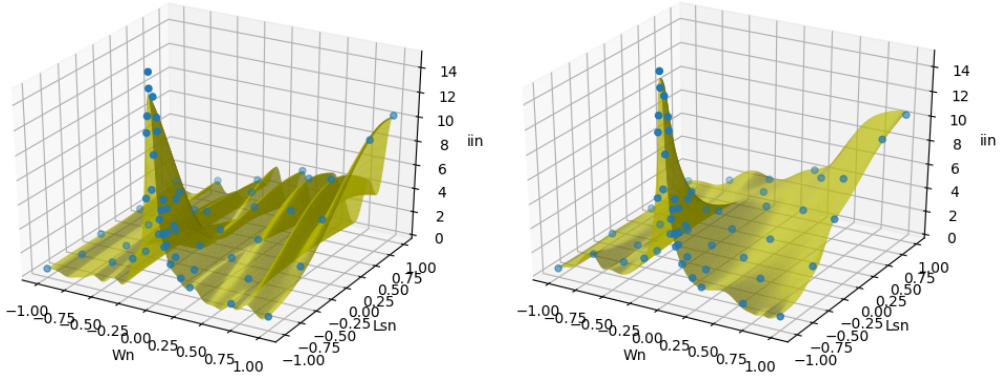
5.6 Conclusion

In this chapter we proposed the use of a recent regression technique, known as Deep Gaussian Processes, for efficient sequential metamodeling of non-stationary response surfaces. Metamodeling of non-stationary surfaces can be troublesome for stationary GP and Kriging models as they require equidistant training samples. This is due to the restriction of having only a single lengthscale over the complete input space, which leads to either models behaving too smooth (thus missing important features) or too non-linear (capturing the features but failing to fit the smooth parts). These problems can partially be remedied by using a uniformly distributed, dense grid of training samples, but this increases the computational complexity significantly.

Our empirical experiments show that DGPs are able to model the non-stationary response surfaces more accurately and do not require an uniformly distributed training dataset. Therefore, we believe that DGPs can be a significant contribution to metamodeling in combination with sequential design as they are able to leverage non-uniform sampling distributions to their advantage and have the appealing characteristic of providing the uncertainty over their predictions.

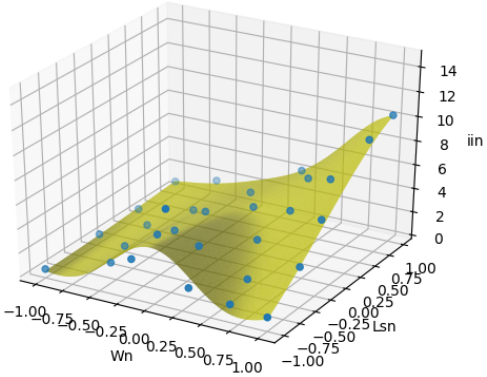
This chapter led to a conference paper:

Dutordoir, V., Knudde, N., van der Herten, J., Couckuyt, I and Dhaene, T. Deep Gaussian Process Metamodeling of Sequentially Sampled Non-Stationary Response Surfaces. Accepted in Proceedings of the 2017 Winter Simulation Conference, 2017.

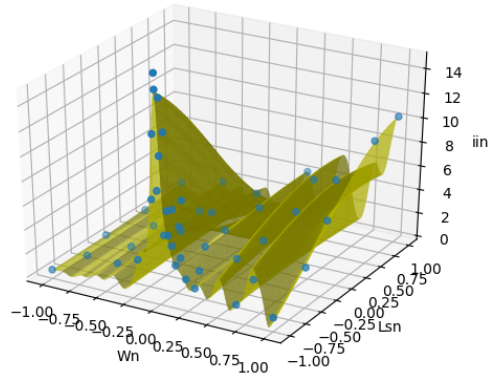


(a) GP – $N = 60$

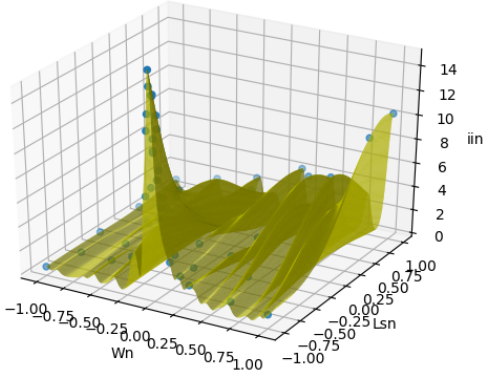
(b) DGP – $N = 60$



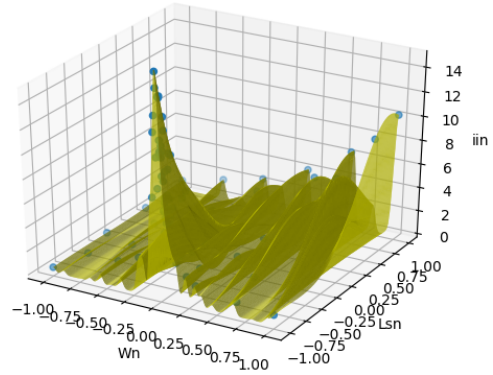
(c) GP – $N = 30$



(d) GP – $N = 50$



(e) GP – $N = 70$



(f) GP – $N = 90$

Figure 5.5: Figure summarizing the LNA metamodeling experiment: plots (a) and (b) illustrate the difference between the GP and DGP metamodel, respectively. Both models are constructed using the same sequential dataset, consisting of 60 samples. Plots (c) until (f) show how the performance of the GP model decreases when the number of samples increases.

Chapter 6

Spectral Mixture Kernels

The discovery of patterns and underlying structure in data is one of the core aspirations of machine learning. Ideally, the ambition is to construct intelligent systems that are able to learn from past observations (i.e. the training dataset) and generalize this information in order to make future predictions. As a result of the recent breakthroughs in reinforcement learning and pattern discovery, such as AlphaGo and GoogLeNet, Artificial Neural Networks (ANNs) are often considered as the go-to machine learning model for difficult tasks. Regrettably, Gaussian Processes (GPs) are generally not used for these complicated tasks. The reason is that a vast majority of kernel learning methods, currently only succeed in intelligent interpolation and smoothing, rather than the desired extrapolation.

Different extensions to the GP framework have been suggested to mitigate this shortcoming. A notable one is the Deep Kernel Learning (DKL) paradigm, which combines the non-parametric flexibility of GPs with deep Neural Networks (NNs) (Wilson et al. 2016). In DKL the inputs of the kernel are first transformed through a deep architecture, which greatly enhances the expressive power of the standard kernel being used. Although they have had successes on a couple of real-world problems, in practice it turns out that the NN is really hard to optimize. Another approach towards leveraging GPs for difficult feature extraction and data extrapolation are the Spectral Mixture (SM) kernels. These kernels are already discussed briefly in Section 2.5.3, but are the main focus of this chapter.

The remainder of this chapter is organized as follows. In Section 6.1 the definition of the SM kernel is derived, starting from its dual Fourier transform. The advantages of using the SM kernel in combination with a standard GP for pattern discovery and data forecasting is shown in Section 6.2. If one wishes to use the SM kernel in advanced GP models, such as a Bayesian GP Latent Variable Model (GP-LVM), a Deep Gaussian Process (DGP) or for predictions with noisy inputs, kernel expectations are required. Therefore, in Section 6.3 we derive those expectations for the SM kernel and in Section 6.4 we illustrate their practicality on a series of experiments.

6.1 Derivation

In this section the definition of the SM kernel is derived. However, compared to the other kernels used in this thesis, the SM kernel is developed from a different angle. That is, instead of directly studying the kernel itself, we investigate the properties of the kernel's spectral density, which is the Fourier transform of the kernel.

Bochner's theorem states that any stationary kernel can be expressed as an integral (Bochner 1959, Stein 1999):

Theorem 1 (Bochner)**.** A complex-valued function k on $\mathbb{R}^P$ is the covariance function of a weakly stationary mean square continuous complex-valued random process on $\mathbb{R}^P$ if and only if it can be represented as

$$k(\boldsymbol{\tau}) = \int_{\mathbb{R}^P} \exp\left(2\pi j \boldsymbol{s}^\top \boldsymbol{\tau}\right) \psi(d\boldsymbol{s}),$$

where ψ is a positive finite measure.

Chatfield (1989) showed that if ψ has a density $S(\boldsymbol{s})$, k and S are Fourier duals where S is called the power spectrum or the spectral density of k. They comply with the following relation:

$$k(\boldsymbol{\tau}) = \int S(\boldsymbol{s}) \exp\left(2\pi j \boldsymbol{s}^\top \boldsymbol{\tau}\right) d\boldsymbol{s}, \tag{6.1}$$

$$S(\boldsymbol{s}) = \int k(\boldsymbol{\tau}) \exp\left(-2\pi j \boldsymbol{s}^\top \boldsymbol{\tau}\right) d\boldsymbol{\tau}. \tag{6.2}$$

Theorem 1 implies that there exists a one-on-one correspondence or relationship between a stationary kernel k and its spectral density S. Therefore, studying the spectral density implicitly reveals properties of the stationary kernel, which can be used to construct novel kernels. To gain more insight in to the theorem, we calculate the density of the well-known single-dimensional Squared Exponential (SE) kernel, $k_{SE}(x, x') = \exp(-\frac{1}{2}(x-x')^2/\ell^2)$. Substituting k_{SE} in Eq. 6.2 gives

$$S_{SE}(s) = \sqrt{2\pi}\ell \exp(-2\pi^2 \ell^2 s^2).$$

The spectral density of the SE kernel corresponds thus to a Gaussian density, centered around the origin. This means that in the dual (spectral) space the SE kernel only covers a small portion of the set of possible functions. Therefore, when using the SE kernel (or any linear combination of SE kernels), one has the false impression of using a powerful and broad class of kernels.

This observation motivated Wilson and Adams (2013) to develop a novel kernel, one which covers a much broader set of functions in the spectral space. More particularly, using a Mixture of Gaussians (MoG) it is possible to construct a significantly broader class of spectral densities. As there is the one-on-one relation between the spectral space and the kernel space, the class of corresponding kernels will also be much broader. Even more, it is possible to approximate any stationary kernel to arbitrary precision, given enough components in the MoG (Kostantinos 2000).

The SM kernel is derived by constructing $S(\boldsymbol{s})$ as a symmetrized rescaled MoG:

$$S(\boldsymbol{s}) = \sum_{q=1}^{Q} w'_q \left(\mathcal{N}(\boldsymbol{s}|\boldsymbol{\mu_q}, \boldsymbol{\Lambda_q}) + \mathcal{N}(-\boldsymbol{s}|\boldsymbol{\mu_q}, \boldsymbol{\Lambda_q})\right),$$

where Q is the number components in the MoG and the q^{th} component has weight w'_q, mean vector $\boldsymbol{\mu}_q = (\mu_q^{(1)}, \mu_q^{(2)}, \ldots, \mu_q^{(P)})$ and diagonal covariance matrix $\boldsymbol{\Lambda}_q = \text{diag}(v_q^{(1)}, v_q^{(2)}, \ldots, v_q^{(P)})^2$. Substituting this spectral density into Eq. 6.1 and writing $\boldsymbol{\tau} = \boldsymbol{x} - \boldsymbol{x'}$ gives

$$k(\boldsymbol{\tau}) = \sum_{q=1}^{Q} w'_q \frac{|\boldsymbol{\Lambda}_q|^{\frac{1}{2}}}{(2\pi)^{\frac{P}{2}}} \cos\left(2\pi \boldsymbol{\tau}^\top \boldsymbol{\mu_q}\right) \exp\left(-\frac{1}{2}\pi^2 \|\boldsymbol{\Lambda}_q^{\frac{1}{2}} \boldsymbol{\tau}_p\|^2\right).$$

The above expression is slightly simplified by replacing the factor $w'_q|\boldsymbol{\Lambda}_q|^{\frac{1}{2}}/(2\pi)^{\frac{P}{2}}$ by a single parameter w_q. The resulting kernel, written in two different forms in the equations below, is henceforth referred to as the Spectral Mixture (SM) kernel.

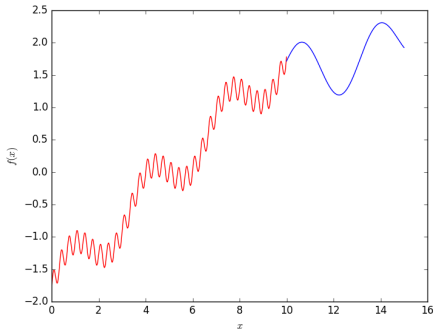
$$\begin{aligned} k_{SM}(\boldsymbol{\tau}) &= \sum_{q=1}^{Q} w_q \cos\left(2\pi \boldsymbol{\tau}^\top \boldsymbol{\mu_q}\right) \exp\left(-2\pi^2 \|\boldsymbol{\tau} \odot \boldsymbol{v}_q\|^2\right) \\ &= \sum_{q=1}^{Q} w_q \cos\left(2\pi \boldsymbol{\tau}^\top \boldsymbol{\mu_q}\right) \prod_{p=1}^{P} \exp\left(-2\pi^2 \tau_p^2 v_{q,p}^2\right), \end{aligned} \tag{6.3}$$

where $\boldsymbol{\tau} \odot \boldsymbol{v}_q$ denotes the Hadamard (also known as entrywise) product between the vectors.

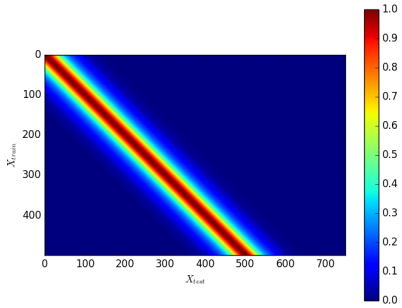
The SM kernel is easy to interpret, has a simple closed analytic form but is still very expressive. By setting one of the parameters to specific values it is possible to construct a range of other stationary kernels. For example, by choosing $\boldsymbol{\mu}_q = \boldsymbol{0},\ \forall\, q$ we form a combination of SE kernels.

Each parameter has its physical meaning which makes the kernel interpretable. The relative contribution of a mixture component is denoted by the weight w_q. The means $\boldsymbol{\mu}_q$ are the principal frequencies of the components. The third set of parameters, v_q, can be seen as the inverse of the lengthscales, encoding how fast a component varies with the input.

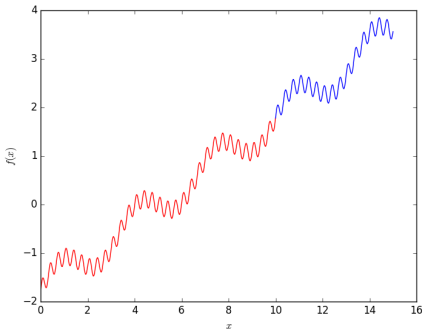
The large flexibility and expressiveness of the kernel comes at a cost: the kernel has many hyperparameters to optimize. Consider P to be the input dimension and Q to be the number of components of the SM kernel, then the kernel constitutes out of Q weight, $Q \times P$ frequency and $Q \times P$ lengthscale hyperparameters, a combined total of $2QP + Q$. Comparing this to, for example, the ARD SE kernel, which has P lengthscales and 1 variance hyperparameters, or the linear kernel which has only 2 hyperparameters, highlights the high number of hyperparameters. As we will use the SM kernel in a GP setting or in combination with Variational Inference (VI) we are not particularly concerned with overfitting. The consequences are more noticeable during optimization as the surface becomes highly multi-model and the optimization routine prone to local minima. For this reason, the SM kernel requires intelligent initialization techniques, such as Monte Carlo Markov Chain (MCMC) or *fingerspitzengefühl*, to work as desired. In the next paragraphs we give some best practices.



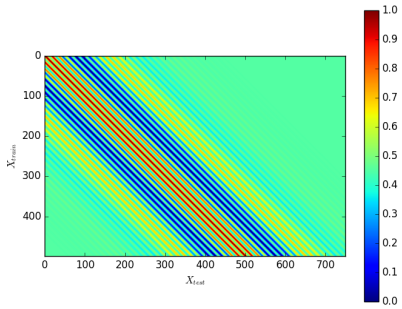
(a) Prediction - bad initialization



(b) Kernel matrix - bad initialization



(c) Prediction - exact initialization



(d) Kernel matrix - exact initialization

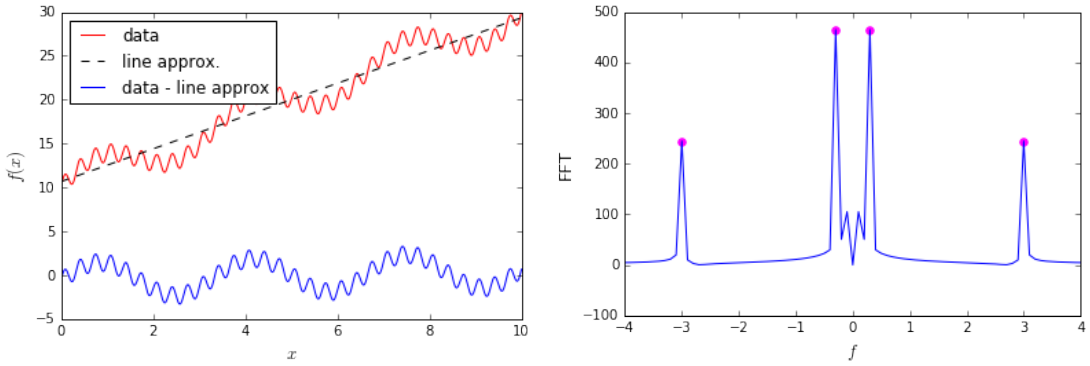
Figure 6.1: Figure highlighting the influence of initialization when predicting with the SM kernel. In the left plots are the red parts the training data and the blue parts the extrapolation. The right plots shows the covariance matrix $k(X, X_*)$ after optimization. The difference between the top and bottom plots is the SM kernel initialization, where good start values are chosen for the bottom plots and bad for the top plots.

We illustrate the difficulty of initialization by extrapolating a synthetic dataset using a standard GP and a SM kernel. The synthetic datasets follows the expression

$$f(x) = 10 + 2\,x + \sin(2\pi f_1 x) + 2\,\sin(2\pi f_2 x),$$

where $f_1 = 3.0$ and $f_2 = 0.3$. The training data $\{\boldsymbol{X}, f(\boldsymbol{X})\}$ consists out of 500 samples, where $\boldsymbol{X}$ is chosen uniformly between 0.0 and 10.0. The test data $\{\boldsymbol{X}_*, f(\boldsymbol{X}_*)\}$ are the next 250 samples between 10.0 and 15.0. Both dataset are shown in Figure 6.1c in red and blue, respectively. The objective is to extrapolate the function into the test region, given the data in the training region.

To fulfill the objective, initialization and optimization of the SM kernel hyperparameters are required. This dataset is constructed synthetically which means that the true frequencies $f_1 = 3.0$ and $f_2 = 0.3$ are known. However, in Figure 6.1a the result is shown when the SM kernel is initialized with *bad* frequencies: $f_1 = 1.0$ and $f_2 = 1.0$. It is clearly visible that the optimization routine is not able to retrieve the exact frequencies. The numerical values of the found frequency hyperparameters are 0.26 and 0.07, which are too low to extrapolate the data correctly. The plot on the right shows the covariance matrix $k(\boldsymbol{X}, \boldsymbol{X}_*)$, which also highlights that the high frequency component f_1 is not learned by the kernel. Contrarily, if we initialize the SM kernel



(a) Data and preprocessed data

(b) Fourier analysis

Figure 6.2: In the left plot is the original un-normalized dataset shown in red. The black dashed line shows the first degree polynomial approximation. The blue line is the residue between the data and the line approximation and will be used in the Fourier analysis. In the right plot the Fourier analysis is shown together with its local maxima. The local maxima correspond to the principal frequency components of the original function.

with almost the true, exact frequencies $f_1 = 3.0$ and $f_2 = 0.3$, we obtain Figure 6.1c. In this case, the correct hyperparameters are found. The corresponding covariance matrix, plotted in Figure 6.1d, is now showing the additional high frequency component. In conclusion, this toy example illustrates the paramount importance of intelligent initialization as choosing slightly different start values can easily result in bad models.

In real-world scenario's the dominant frequencies of a dataset are often unknown. In this case other techniques can be used to initialize the SM kernel as good as possible. When, for example, faced with a time-series dataset measuring temperature at a certain location, a plausible initialization would be to select $Q = 3$ with a daily, monthly and yearly frequency component. Unfortunately, it is not always possible to rely on common sense, then the Fast Fourier Transform (FFT) transform can be another useful technique. If we consider the dataset, shown in Figure 6.2a (this is the same dataset as in Figure 6.1c), we can extract the dominant frequencies through the Fourier analysis. However, in order to get an interpretable and clean Fourier transform, some preprocessing is required. We start by extracting the mean by fitting a first degree polynomial, shown in black in Figure 6.2a, using least square. The resulting function is shown in blue and its FFT is plotted in Figure 6.2b. The dominant frequencies at ± 0.3 and ± 3.0 are clearly visible but are in practice most easily found by using a peak detection routine. Initializing the kernel with these obtained dominant frequencies will greatly aid the SM kernel optimization.

6.2 Experiments

In this section we cover a couple of experiments, showing the performance of the SM kernel and comparing it with other covariance functions. In Section 6.2.1 the SM kernel in combination with a standard GP will be used to forecast the number of airline passengers in the years after 1964. The second application is the extrapolation of a highly structured tread plate image, based

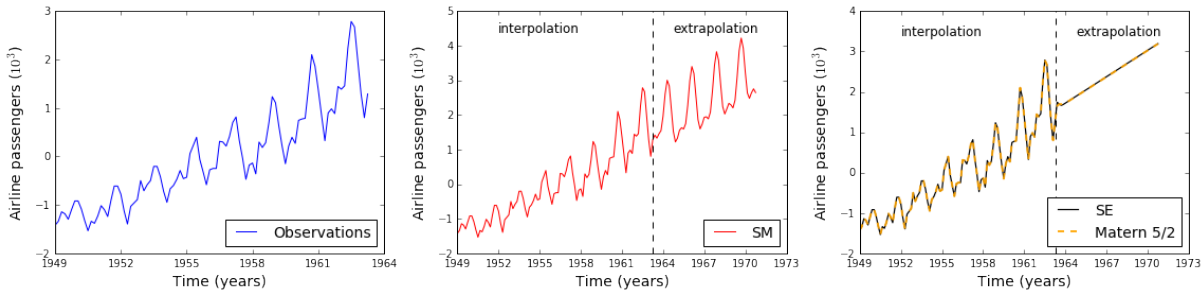


Figure 6.3: Forecasting airline passenger numbers based on observations between 1949 and 1964. The left plot (in blue) shows the training data (i.e. true observations). The red line in the middle plot shows the prediction of a GP using a SM kernel in the training (interpolation) and test region (extrapolation). The right plot shows the prediction of two GP models. One using a Matérn 5/2 kernel, plotted in dashed-yellow, and another one using a SE kernel, plotted in black.

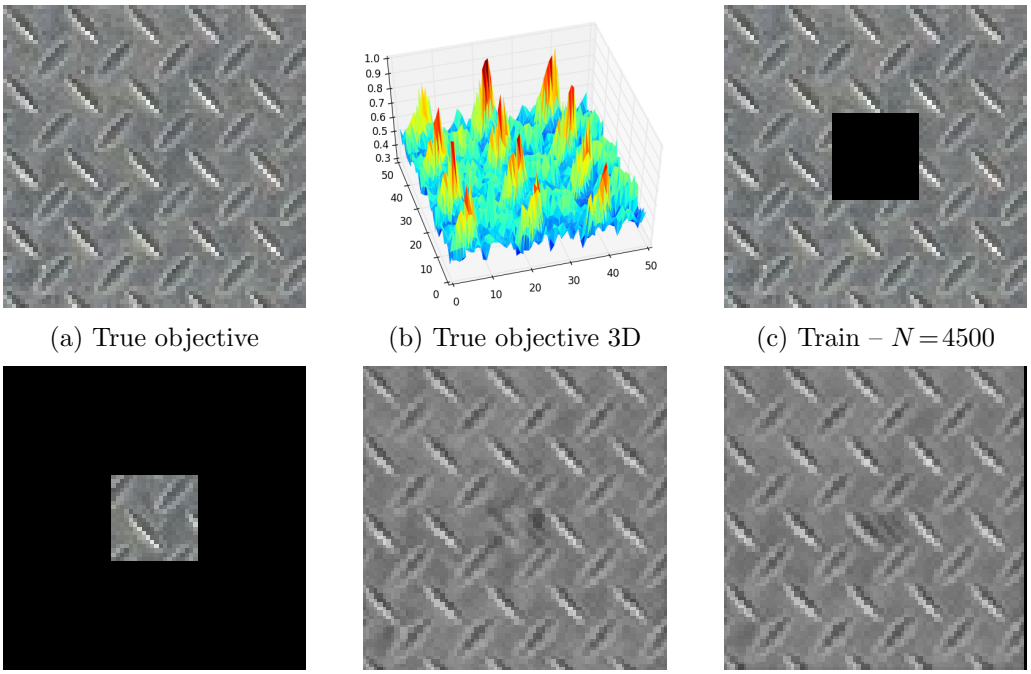
on the idea of Samo and Roberts (2015). For all the experiments conducted in this section, we make use of GPflow (Matthews et al. 2017). At the moment of writing GPflow did not featured the SM kernel and therefore an implementation was made, conform Eq. 6.3, and integrated into the framework.

6.2.1 Airline data

The left plot of Figure 6.3 shows the number of airline passengers from 1949 to 1964 (Hyndman 2005). The objective is to forecast the number of passengers for the next 10 years, based on the past observed measurements.

The dataset has several distinctive features: (1) there is a long-term rising trend, (2) there are short yearly fluctuations depending on the season and (3) there is the absence of white noise on the observations, as we assume that these measurements are precise. Most of the well-known kernels, including the SE and the Matérn 5/2, are forced in a trade-off during optimization. On the one hand they can choose to model the highly fluctuating seasonal trends, and ignore the long-term increase. On the other hand, they can choose to model the long-term trend, which will cause the seasonal variations to be ignored as they will be considered noise variations instead.

The right plot of Figure 6.3 shows the forecast made with the Matérn 5/2 and the SE kernel. Both stationary kernels choose to model the long-term rising trend, at the expense of the short term fluctuations. They perform satisfactory in the training region, but move very quickly to the mean beyond that, having learned no general structure in the data. In conclusion, these kernels are easily able to capture the trend in the training region (interpolation) but do not have the expressiveness to extrapolate into the future. This again emphasizes that they simply act as intelligent smoothing functions.



(a) True objective (b) True objective 3D (c) Train – $N = 4500$

(d) Test – $N_* = 400$ (e) Prediction @ 10 iter. (f) Prediction @ 160 iter.

Figure 6.4: Extrapolation of a metal tread plate pattern. In (a) and (b) the true metal tread plate is shown, (c) shows the training data with the test data left out (black), (d) shows the test data with the training data left out. (e) en (f) show the full prediction (i.e. the prediction of the training and the test region) at 10 and 160 optimization iterations, respectively. The predictions are done with a SM-GP model.

By contrast, the middle plot shows the prediction of the GP model using the SM kernel. The SM kernel interpolates satisfactory in the training region while extrapolating the complex seasonal trends and long-term rise beyond the training data. We used a SM kernel with initially 30 components (i.e. $Q = 30$), and after optimization 14 were used. The impact of unused components is artificially reduced during optimization by shrinking the weight w_q to zero. The optimized frequencies constitutes out of high values, required to model the short seasonal trends, and low values, necessary to model the long-term rising trend.

6.2.2 Tread plate

In this second experiment we try to extrapolate the missing part of a tread plate image. The experiment is presented in Figure 6.4. Figure 6.4a and 6.4b show the true objective: in 6.4a the objective is shown as an image in which the pixel intensities are gray-scale values and in 6.4b the pixel intensities are plotted as a 3D surface. Both images show the high peaks and the periodic structure of the pattern. However, there are no two identical diagonal markings in the objective as the texture contains shadows and subtle irregularities. In the experiment we use 2D locations $\boldsymbol{x} \in \mathbb{N}^2$ as input and pixel intensities as output $y \in \mathbb{R}$. The model is trained on the dataset shown in Figure 6.4c, where the center 20 $\times$ 20 pixels are missing. The objective is to extrapolate the pattern in the missing region.

To achieve this, we use a traditional GP configured with a SM kernel. Two reconstructions are shown in Figure 6.4e and 6.4f. The former shows the prediction at the start of the optimization and the latter at the end. Comparing both, we observe that the extrapolation in the missing area is quite blurred at the beginning but becomes significantly sharper at the end. This clearly illustrates that SM the kernel is *learning* the particular structure of the pattern under study.

Good initialization of the SM kernel hyperparameters $\{w_q, \boldsymbol{v}_q, \boldsymbol{\mu}_q\}_{q=1}^{Q}$ is the most important part for a satisfactory outcome of this experiment. We used a SM kernel with $Q = 30$, which are actually more components than needed for this problem. However, thanks to the complexity penalty in the marginal likelihood of GP models, extraneous components are pruned by shrinking the weights w_q. This effect is similar to Automatic Relevance Determination (ARD) and can be seen as a sort of automatic model selection. The SM kernel used was initialized using the following procedure (based on Wilson et al. (2014)): the frequencies $\boldsymbol{\mu}_q$ are drawn from a uniform distribution from 0 to the Nyquist frequency (i.e. half of the sampling rate), the lengthscales $\boldsymbol{v}_q$ are drawn from a truncated Gaussian distribution, with mean proportional to the range of the data, and weights w_q are initialized as the empirical standard deviation of the data divided by the number of components Q.

As mentioned in Chapter 2, the training of a standard GP model is expensive in terms of number of computations. This translates into unpractical training times. To give an indication, the training of the model, depicted in Figure 6.4f, takes approximately 2 hours on a high-end machine powered by 40 Intel Xeon (E5-2640) CPUs at 2.40GHz, each having a cache of 25.6 MB.

Overall, these two experiments highlight the expressiveness and flexibility of the SM kernel. It has automatically discovered the underlying structure of a metal pattern image and extrapolated that pattern across a large missing region. Furthermore, it was also the only kernel able to extrapolate the airline data properly.

6.3 Kernel Expectations of the SM kernel

In most variational models, such as Bayesian GP-LVMs and DGPs, it is required to calculate kernel expectations during inference and learning. In those models, the kernel expectations are used to propagate distributions through (non) linear kernel functions in order to traverse the variational layers. Also for dealing with input uncertainty and for using GPs in reinforcement learning applications, are kernel expectation necessary. As covered in Section 4.4, kernel expectations are the continuous convolution of the kernel function with a Gaussian density. In particular, the models require three statistics $\{\xi, \Omega, \Phi\}$ (Girard et al. 2003, Damianou 2015) defined as

$$\xi = \int k(\boldsymbol{x}, \boldsymbol{x})\, \mathcal{N}(\boldsymbol{x}|\boldsymbol{\gamma}, \boldsymbol{S})\, \mathrm{d}\boldsymbol{x}, \tag{6.4}$$

$$\Omega(\boldsymbol{y}) = \int k(\boldsymbol{x}, \boldsymbol{y})\, \mathcal{N}(\boldsymbol{x}|\boldsymbol{\gamma}, \boldsymbol{S})\, \mathrm{d}\boldsymbol{x}, \tag{6.5}$$

$$\Phi(\boldsymbol{y}, \boldsymbol{z}) = \int k(\boldsymbol{x}, \boldsymbol{y})\, k(\boldsymbol{x}, \boldsymbol{z})\, \mathcal{N}(\boldsymbol{x}|\boldsymbol{\gamma}, \boldsymbol{S})\, \mathrm{d}\boldsymbol{x}. \tag{6.6}$$

Analytic tractability of the $\{\xi, \Omega, \Phi\}$-statistics greatly depend on the type of kernel used. They can be computed analytically for the ARD SE and linear kernel. The kernel expectations of the SM kernel are, at the moment of writing, not available in the literature. However, if one wishes to use the SM kernel in a DGP, Bayesian GP-LVM, or another advanced GP model, they are necessary. For this reason, in the coming section the kernel expectations of the SM kernel are derived.

6.3.1 Derivation

ξ-statistic

We begin the derivation of the first kernel statistic ξ (Eq. 6.4), which equals

$$\xi = \int k(\boldsymbol{x}, \boldsymbol{x})\ \mathcal{N}(\boldsymbol{x}|\boldsymbol{\gamma}, \boldsymbol{S})\, \mathrm{d}\boldsymbol{x}$$

Substituting the definition of the SM kernel (Eq. 6.3) with $\boldsymbol{\tau} = \boldsymbol{x} - \boldsymbol{x} = \mathbf{0}$ gives

$$\begin{aligned}
\xi &= \int \sum_{q=1}^{Q} w_q \cos\left(2\pi \mathbf{0}^\top \boldsymbol{\mu_q}\right) \exp\left(-2\pi^2 \|\mathbf{0} \odot \boldsymbol{v}_q\|^2\right)\ \mathcal{N}(\boldsymbol{x}|\boldsymbol{\gamma}, \boldsymbol{S})\ d\boldsymbol{x} \\
&= \int \sum_{q=1}^{Q} w_q\ \mathcal{N}(\boldsymbol{x}|\boldsymbol{\gamma}, \boldsymbol{S})\ d\boldsymbol{x} \\
&= \sum_{q=1}^{Q} w_q \int \mathcal{N}(\boldsymbol{x}|\boldsymbol{\gamma}, \boldsymbol{S})\ d\boldsymbol{x}. \qquad (6.7)
\end{aligned}$$

Due to normalization, it follows that ξ is simply the sum of the weights, i.e. $\xi = \sum_{q=1}^{Q} w_q$, and is thus independent of the input and the probability distribution $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\gamma}, \boldsymbol{S})$.

Ω-statistic

The integrand in Ω is less trivial than in ξ, which makes it already more complicated to calculate. We again depart from the definition (Eq. 6.5), which equals

$$\Omega(\boldsymbol{y}) = \int k(\boldsymbol{x}, \boldsymbol{y})\ \mathcal{N}(\boldsymbol{x}|\boldsymbol{\gamma}, \boldsymbol{S}) d\boldsymbol{x}.$$

Filling in the definition of the SM kernel for k (Eq. 6.3) and substitution $\boldsymbol{x}$ by $\boldsymbol{\tau} + \boldsymbol{y}$, we obtain

$$\Omega(\boldsymbol{y}) = \int \sum_{q=1}^{Q} w_q \cos\left(2\pi \boldsymbol{\tau}^\top \boldsymbol{\mu_q}\right) \exp\left(-2\pi^2 \|\boldsymbol{\tau} \odot \boldsymbol{v}_q\|^2\right) \mathcal{N}(\boldsymbol{\tau} + \boldsymbol{y}|\boldsymbol{\gamma}, \boldsymbol{S})\ d\boldsymbol{\tau}.$$

The Hadamard product between the lengthscales and $\boldsymbol{\tau}$ can be written as a matrix product $\|\boldsymbol{\tau} \odot \boldsymbol{v}_q\|^2 = \boldsymbol{\tau}^\top \boldsymbol{\Lambda}_q \boldsymbol{\tau}$, where $\boldsymbol{\Lambda}_q = \text{diag}(v_q^{(1)}, v_q^{(2)}, \ldots, v_q^{(P)})^2$.

$$\begin{aligned}\Omega(\boldsymbol{y}) &= \sum_{q=1}^{Q} \int w_q \cos\left(2\pi \boldsymbol{\tau}^\top \boldsymbol{\mu_q}\right) \exp\left(-2\pi^2 \boldsymbol{\tau}^\top \boldsymbol{\Lambda}_q \boldsymbol{\tau}\right) \mathcal{N}(\boldsymbol{\tau} + \boldsymbol{y} | \boldsymbol{\gamma}, \boldsymbol{S})\ d\boldsymbol{\tau} \\ &= \sum_{q=1}^{Q} \int w_q \cos\left(2\pi \boldsymbol{\tau}^\top \boldsymbol{\mu_q}\right) \exp\left(-\frac{\boldsymbol{\tau}^\top (4\pi^2 \boldsymbol{\Lambda}_q) \boldsymbol{\tau}}{2}\right) \mathcal{N}(\boldsymbol{\tau} + \boldsymbol{y} | \boldsymbol{\gamma}, \boldsymbol{S})\ d\boldsymbol{\tau}.\end{aligned}$$

The input of the $\exp(\cdot)$ was rewritten to reassemble the input of the exponential term in a multivariate Gaussian distribution, with a covariance matrix $\boldsymbol{\Sigma}_q = (4\pi^2 \boldsymbol{\Lambda}_q)^{-1}$ and mean zero. The factors $(2\pi)^{P/2} |\Sigma_q|^{1/2}$ are added to complete the normal distribution $\mathcal{N}(\boldsymbol{\tau} | \boldsymbol{0}, \boldsymbol{\Sigma}_q)$. We find

$$\Omega(\boldsymbol{y}) = \sum_{q=1}^{Q} w_q (2\pi)^{P/2} |\Sigma_q|^{1/2} \int \cos\left(2\pi \boldsymbol{\tau}^\top \boldsymbol{\mu_q}\right) \mathcal{N}(\boldsymbol{\tau} | \boldsymbol{0}, \boldsymbol{\Sigma}_q)\ \mathcal{N}(\boldsymbol{\tau} + \boldsymbol{y} | \boldsymbol{\gamma}, \boldsymbol{S})\ d\boldsymbol{\tau}.$$

Both Gaussians can be combined into a single one using Eq. A.3: $\mathcal{N}(\boldsymbol{\tau} | \boldsymbol{0}, \boldsymbol{\Sigma}_q)\ \mathcal{N}(\boldsymbol{\tau} + \boldsymbol{y} | \boldsymbol{\gamma}, \boldsymbol{S}) = c_q\ \mathcal{N}(\boldsymbol{\tau} | \boldsymbol{m}, \boldsymbol{A})$, with $c_q = \mathcal{N}(\boldsymbol{0} | \boldsymbol{\gamma} - \boldsymbol{y}, \boldsymbol{S} + \boldsymbol{\Sigma}_q)$, $\boldsymbol{m} = \boldsymbol{A}\, \boldsymbol{S}^{-1}(\boldsymbol{\gamma} - \boldsymbol{y})$ and $\boldsymbol{A} = \left(\boldsymbol{\Sigma}_q^{-1} + \boldsymbol{S}^{-1}\right)^{-1}$:

$$\Omega(\boldsymbol{y}) = \sum_{q=1}^{Q} w_q (2\pi)^{P/2} |\Sigma_q|^{1/2} c_q \int \cos\left(2\pi \boldsymbol{\tau}^\top \boldsymbol{\mu_q}\right) \mathcal{N}(\boldsymbol{\tau} | \boldsymbol{m}, \boldsymbol{A})\ d\boldsymbol{\tau}.$$

The final step is to calculate the expectation of the angle $2\pi \boldsymbol{\tau}^\top \boldsymbol{\mu}_q$ with respect to the normal distribution $\mathcal{N}(\boldsymbol{\tau} | \boldsymbol{m}, \boldsymbol{A})$, which gives

$$\Omega(\boldsymbol{y}) = \sum_{q=1}^{Q} w_q (2\pi)^{P/2} |\Sigma_q|^{1/2}\, c_q\, \cos\left((2\pi \boldsymbol{\mu}_q^\top \boldsymbol{m})\right) \exp\left(-\frac{1}{2}(2\pi \boldsymbol{\mu}_q^\top)\, \boldsymbol{A}\, (2\pi \boldsymbol{\mu}_q)\right). \tag{6.8}$$

Φ-statistic

The third kernel expectation Φ consists of the convolution of a normal distribution and a product of two kernels (Eq. 6.6):

$$\Phi(\boldsymbol{y}, \boldsymbol{z}) = \int k(\boldsymbol{x}, \boldsymbol{y})\ k(\boldsymbol{z}, \boldsymbol{x})\ \mathcal{N}(\boldsymbol{x} | \boldsymbol{\gamma}, \boldsymbol{S})\ d\boldsymbol{x}.$$

Substituting both k's with the definition of the SM kernel (Eq. 6.3) and taking the independent factors out of the integral, we find

$$\begin{aligned}\Phi(\boldsymbol{y}, \boldsymbol{z}) = \sum_{k,q} w_k\, w_q \int &\exp\left(-2\pi^2 \|(\boldsymbol{x} - \boldsymbol{y}) \odot \boldsymbol{v}_k\|^2\right) \cos\left(2\pi (\boldsymbol{x} - \boldsymbol{y})^\top \boldsymbol{\mu_k}\right) \\ &\exp\left(-2\pi^2 \|(\boldsymbol{x} - \boldsymbol{z}) \odot \boldsymbol{v}_q\|^2\right) \cos\left(2\pi (\boldsymbol{x} - \boldsymbol{z})^\top \boldsymbol{\mu_q}\right) \mathcal{N}(\boldsymbol{x} | \boldsymbol{\gamma}, \boldsymbol{S})\ d\boldsymbol{x}.\end{aligned}$$

Note that we did not introduce $\boldsymbol{\tau}$ as was done in the derivation of Ω, but instead keep both vectors in the equations. Further, both Hadamard products can again be written as a product of matrices: $(\boldsymbol{x} - \boldsymbol{y})^\top \boldsymbol{\Lambda}_k (\boldsymbol{x} - \boldsymbol{y})$ and $(\boldsymbol{x} - \boldsymbol{z})^\top \boldsymbol{\Lambda}_q (\boldsymbol{x} - \boldsymbol{z})$, where $\boldsymbol{\Lambda}_i = \text{diag}(v_i^{(1)}, v_i^{(2)}, \ldots, v_i^{(P)})^2$,

which gives

$$\Phi(\boldsymbol{y},\boldsymbol{z}) = \sum_{k,q} w_k\, w_q \int \exp\left(-2\pi^2(\boldsymbol{x}-\boldsymbol{y})^\top \boldsymbol{\Lambda}_k(\boldsymbol{x}-\boldsymbol{y})\right)\cos\left(2\pi(\boldsymbol{x}-\boldsymbol{y})^\top \boldsymbol{\mu_k}\right) \\ \exp\left(-2\pi^2(\boldsymbol{x}-\boldsymbol{z})^\top \boldsymbol{\Lambda}_q(\boldsymbol{x}-\boldsymbol{z})\right)\cos\left(2\pi(\boldsymbol{x}-\boldsymbol{z})^\top \boldsymbol{\mu_q}\right)\mathcal{N}(\boldsymbol{x}|\boldsymbol{\gamma},\boldsymbol{S})\; d\boldsymbol{x}.$$

The next step is to slightly rearrange the arguments of the exponentials

$$\Phi(\boldsymbol{y},\boldsymbol{z}) = \sum_{k,q} w_k w_q \int \exp\left(-\tfrac{(\boldsymbol{x}-\boldsymbol{y})^\top(4\pi\boldsymbol{\Lambda}_k)(\boldsymbol{x}-\boldsymbol{y})}{2}\right)\cos\left(2\pi(\boldsymbol{x}-\boldsymbol{y})^\top \boldsymbol{\mu_k}\right) \\ \exp\left(-\tfrac{(\boldsymbol{x}-\boldsymbol{z})^\top(4\pi\boldsymbol{\Lambda}_q)(\boldsymbol{x}-\boldsymbol{z})}{2}\right)\cos\left(2\pi(\boldsymbol{x}-\boldsymbol{z})^\top \boldsymbol{\mu_q}\right)\mathcal{N}(\boldsymbol{x}|\boldsymbol{\gamma},\boldsymbol{S})\; d\boldsymbol{x}.$$

Written like this, the input of the exponentials now closely reassembles the input of a normal distribution's exponential factor. Defining $\boldsymbol{\Sigma}_i$ as $(4\pi\boldsymbol{\Lambda}_i)^{-1}$ and adding the correct constants, we can construct two normal distributions: $\mathcal{N}(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{\Sigma}_k)$ and $\mathcal{N}(\boldsymbol{x}|\boldsymbol{z},\boldsymbol{\Sigma}_q)$. Integrating both normals into the equation, yields

$$\Phi(\boldsymbol{y},\boldsymbol{z}) = \sum_{k,q} w_k\, w_q (2\pi)^{P/2}|\boldsymbol{\Sigma}_k|^{-1/2}(2\pi)^{P/2}|\boldsymbol{\Sigma}_q|^{-1/2} \\ \int \mathcal{N}(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{\Sigma}_k)\cos\left(2\pi(\boldsymbol{x}-\boldsymbol{y})^\top \boldsymbol{\mu_k}\right)\mathcal{N}(\boldsymbol{x}|\boldsymbol{z},\boldsymbol{\Sigma}_q)\cos\left(2\pi(\boldsymbol{x}-\boldsymbol{z})^\top \boldsymbol{\mu_q}\right)\mathcal{N}(\boldsymbol{x}|\boldsymbol{\gamma},\boldsymbol{S})\; d\boldsymbol{x}.$$

Next, we define $w_k w_q (2\pi)^{P/2}|\boldsymbol{\Sigma}_k|^{-1/2}(2\pi)^{P/2}|\boldsymbol{\Sigma}_q|^{-1/2}$ as $C_{k,q}$ and combine the normal distributions $\mathcal{N}(\boldsymbol{x}|\boldsymbol{y},\boldsymbol{\Sigma}_k)$ and $\mathcal{N}(\boldsymbol{x}|\boldsymbol{z},\boldsymbol{\Sigma}_q)$ into a single one $c_1\,\mathcal{N}(\boldsymbol{x}|\boldsymbol{m}_1,\boldsymbol{V}_1)$, with $c_1 = \mathcal{N}(\boldsymbol{y}|\boldsymbol{z},\boldsymbol{\Sigma}_k+\boldsymbol{\Sigma}_q)$, $\boldsymbol{m}_1 = \boldsymbol{V}_1\left(\boldsymbol{\Sigma}_k^{-1}\boldsymbol{y}+\boldsymbol{\Sigma}_q^{-1}\boldsymbol{z}\right)$ and $\boldsymbol{V}_1 = \left(\boldsymbol{\Sigma}_k^{-1}+\boldsymbol{\Sigma}_q^{-1}\right)^{-1}$. Substituting this into the equation gives

$$\Phi(\boldsymbol{y},\boldsymbol{z}) = \sum_{k,q} C_{k,q}\, c_1 \int \mathcal{N}(\boldsymbol{x}|\boldsymbol{m}_1,\boldsymbol{V}_1)\cos\left(2\pi(\boldsymbol{x}-\boldsymbol{y})^\top \boldsymbol{\mu_k}\right)\cos\left(2\pi(\boldsymbol{x}-\boldsymbol{z})^\top \boldsymbol{\mu_q}\right)\mathcal{N}(\boldsymbol{x}|\boldsymbol{\gamma},\boldsymbol{S})\; d\boldsymbol{x}.$$

For a second time we combine the two remaining normal distributions into a single one, using Eq. A.3: $\mathcal{N}(\boldsymbol{x}|\boldsymbol{m}_1,\boldsymbol{V}_1)\;\mathcal{N}(\boldsymbol{x}|\boldsymbol{\gamma},\boldsymbol{S}) = c_2\;\mathcal{N}(\boldsymbol{x}|\boldsymbol{m}_2,\boldsymbol{V}_2)$, with $c_2 = \mathcal{N}(\boldsymbol{m}_1|\boldsymbol{\gamma},\boldsymbol{V}_1+\boldsymbol{S})$, $\boldsymbol{m}_2 = \boldsymbol{V}_2\left(\boldsymbol{V}_1^{-1}\boldsymbol{m}_1+\boldsymbol{S}^{-1}\boldsymbol{\gamma}\right)$ and $\boldsymbol{V}_2 = \left(\boldsymbol{V}_1^{-1}+\boldsymbol{S}^{-1}\right)^{-1}$. We now obtain an expression with a single Gaussian, which is characterizing the previous three

$$\Phi(\boldsymbol{y},\boldsymbol{z}) = \sum_{k,q} C_{k,q}\, c_1\, c_2 \int \mathcal{N}(\boldsymbol{x}|\boldsymbol{m}_2,\boldsymbol{V}_2)\cos\left(2\pi(\boldsymbol{x}-\boldsymbol{y})^\top \boldsymbol{\mu_k}\right)\cos\left(2\pi(\boldsymbol{x}-\boldsymbol{z})^\top \boldsymbol{\mu_q}\right)\; d\boldsymbol{x}. \quad (6.9)$$

Using trigonometry rules, it is possible to transform the product of cosines into a sum of two: $\cos\left(2\pi(\boldsymbol{x}-\boldsymbol{y})^\top \boldsymbol{\mu_k}\right)\;\cos\left(2\pi(\boldsymbol{x}-\boldsymbol{z})^\top \boldsymbol{\mu_q}\right) = \frac{1}{2}\left(\cos(\alpha)+\cos(\beta)\right)$, where

$$\alpha(\boldsymbol{x}) = 2\pi\left((\boldsymbol{\mu}_q+\boldsymbol{\mu}_k)^\top\boldsymbol{x} - \boldsymbol{\mu}_k^\top\boldsymbol{y} - \boldsymbol{\mu}_q^\top\boldsymbol{z}\right), \quad (6.10)$$

$$\beta(\boldsymbol{x}) = 2\pi\left((\boldsymbol{\mu}_k-\boldsymbol{\mu}_q)^\top\boldsymbol{x} - \boldsymbol{\mu}_k^\top\boldsymbol{y} + \boldsymbol{\mu}_q^\top\boldsymbol{z}\right). \quad (6.11)$$

Substituting Eqs. 6.10 and 6.11 in Eq. 6.9 gives

$$\Phi(\boldsymbol{y}, \boldsymbol{z}) = \tfrac{1}{2} \sum_{k,q} C_{k,q}\, c_1\, c_2 \left[\int \mathcal{N}(\boldsymbol{x}|\boldsymbol{m}_2, \boldsymbol{V}_2) \cos(\alpha(\boldsymbol{x}))\ d\boldsymbol{x} + \int \mathcal{N}(\boldsymbol{x}|\boldsymbol{m}_2, \boldsymbol{V}_2) \cos(\beta(\boldsymbol{x}))\ d\boldsymbol{x} \right].$$

The final step is to calculate the expectation of $\alpha(\boldsymbol{x})$ and $\beta(\boldsymbol{x})$ with respect to the normal distribution $\mathcal{N}(\boldsymbol{x}|\boldsymbol{m}_2, \boldsymbol{V}_2)$, yielding in

$$\Phi(\boldsymbol{y}, \boldsymbol{z}) = \frac{1}{2} \sum_{k,q} C_{k,q}\, c_1\, c_2 \left(\exp\left(-\tfrac{1}{2}\sigma_\alpha\right) \cos(m_\alpha) + \exp\left(-\tfrac{1}{2}\sigma_\beta\right) \cos(m_\beta) \right), \tag{6.12}$$

with

$$\begin{aligned}
m_\alpha &= 2\pi \left((\boldsymbol{\mu}_k + \boldsymbol{\mu}_q)^\top \boldsymbol{m}_2 - \boldsymbol{\mu}_k^\top \boldsymbol{y} - \boldsymbol{\mu}_q^\top \boldsymbol{z} \right), \\
\sigma_\alpha &= 4\pi^2 (\boldsymbol{\mu}_k + \boldsymbol{\mu}_q)^\top \boldsymbol{V}_2 (\boldsymbol{\mu}_k + \boldsymbol{\mu}_q), \\
m_\beta &= 2\pi \left((\boldsymbol{\mu}_k - \boldsymbol{\mu}_q)^\top \boldsymbol{m}_2 - \boldsymbol{\mu}_k^\top \boldsymbol{y} + \boldsymbol{\mu}_q^\top \boldsymbol{z} \right), \\
\sigma_\beta &= 4\pi^2 (\boldsymbol{\mu}_k - \boldsymbol{\mu}_q)^\top \boldsymbol{V}_2 (\boldsymbol{\mu}_k - \boldsymbol{\mu}_q).
\end{aligned}$$

This concludes the derivations of the kernel expectations of the SM kernel, the analytical closed forms of ξ, Ω and Φ are given in Eqs. 6.7, 6.8 and 6.12. In the next section it is briefly explained how the correctness of these formula is verified. In Section 6.4 we integrate the kernel expectations into a Bayesian GP-LVM and an uncertain predict GP.

Empirical Proof Correctness

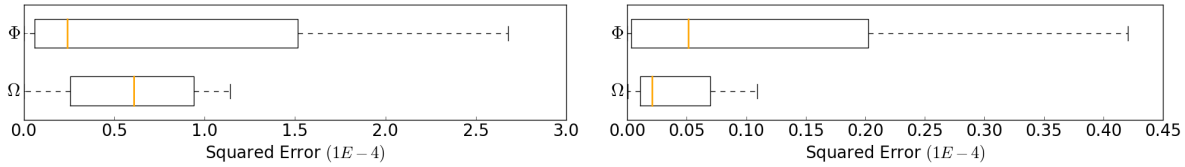
The correctness of the above derivations and their accompanying TensorFlow implementation[1] needs to be determined before integrating them into other models. Due to the nature of the implementations, we choose for an empirical proof using Monte Carlo integration. The Monte Carlo routine for Ω is given in Algorithm 1 (Φ it is very analogous).

Figure 6.5 is generated by executing Algorithm 1 20 times, for both Ω and Φ, and reporting the statistics of the measured error ε. The dimension of the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is chosen to be 2 in order to keep the MC routine efficient. The difference between the left and right plot is the number of Monte Carlo iterations (MC_{iters}) within each routine. We observe that if we multiply the MC_{iters} by a factor 10, the median squared error ε is also reduced by a factor 10 approximately. Consequently, for MC_{iters} tending towards infinity, $\tilde{\Omega}$ will hold the exact kernel expectation value Ω. To conclude, we consider our derivation and implementation correct as the corresponding median error ε is progressively moving towards 0.

[1] https://github.com/vdutor/GPflow/blob/sm-kernel/GPflow/ekernels.py

Algorithm 1 Monte Carlo verification Ω-statistic

1: Randomly initialize the hyperparameters of the SM kernel k_{SM}, the parameters of a Gaussian distribution $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\gamma}, \boldsymbol{S})$ and vector $\boldsymbol{y}$
2: **for** $i \leftarrow 1$ to MC_{iters} **do**
3: Draw $\boldsymbol{x}_i$ from $\mathcal{N}(\boldsymbol{\gamma}, \boldsymbol{S})$
4: $k_i \leftarrow k_{\text{SM}}(\boldsymbol{x}_i, \boldsymbol{y})$ ▷ using Eq. 6.3
5: $\tilde{\Omega} \leftarrow \tilde{\Omega} + k_i / \text{MC}_{\text{iters}}$ ▷ update approximation
6: **end for**
7: $\hat{\Omega} \leftarrow \Omega(\boldsymbol{y})$ ▷ using the closed-form equation, Eq. 6.8
8: $\varepsilon \leftarrow \left(\tilde{\Omega} - \hat{\Omega}\right)^2$ ▷ calculate squared error
9: **return** ε



(a) $\text{MC}_{\text{iters}} = 1000$ (b) $\text{MC}_{\text{iters}} = 10\,000$

Figure 6.5: Boxplots of the squared error, obtained by executing the Monte Carlo routine (Algorithm 1) 20 times for both Ω and Φ. The input dimension of the vectors $\boldsymbol{\gamma}, \boldsymbol{x}$ and $\boldsymbol{y}$ is only 2 and $\boldsymbol{S}$ is a 2×2 matrix, which keeps the MC routine efficient.

6.4 Experiments

In Chapter 4 we presented three different advanced GP-based models, which all require kernel expectations for their internal working: models dealing with input uncertainty (Section 4.1), Bayesian GP-LVMs (Section 4.2) and DGPs (Section 4.3). In this section we cover an experiment with the first two models and use the newly derived kernel expectation of the SM covariance function. The objective is to compare these advanced models with standard models and to asses the performance of the SM kernel expectations.

6.4.1 Visualization Task

When a high-dimensional dataset, with an unknown underlying structure, is given, it is often very difficult to extract useful properties from the data. Manifold learning algorithms have the goal of constructing a lower dimensional (latent) space, which can then be used for visualization, compression or for downstream supervised tasks. We illustrate this with the multi-phase oil flow data, where each of the 1000 observations belongs to one of the 3 known classes, corresponding to different phases of oil flow (Bishop and James 1993). Each sample in the dataset is 12 dimensional, which makes it extremely difficult to directly apply classifications methods on the dataset, as they typically suffer from the *curse of dimensionality*. Figure 6.6 shows 8 different projections from the 12D to a 2D subspace. If the goal is to separate the observations belonging to a certain class, we observe that some subspaces are clearly more relevant than others.

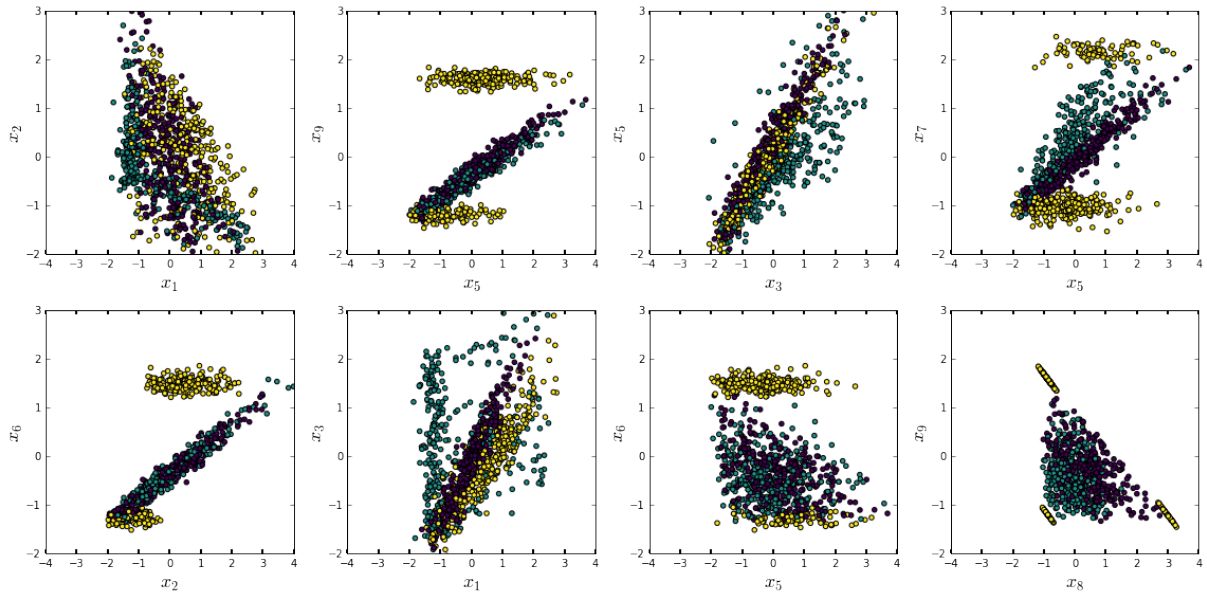


Figure 6.6: The three phase oil dataset constitutes out of 1000 samples. Each sample is 12 dimensional and belongs to one of the three phases (which are the classes). In these 8 plots, we randomly selected 2 out of the 12 dimensions to visualize the data.

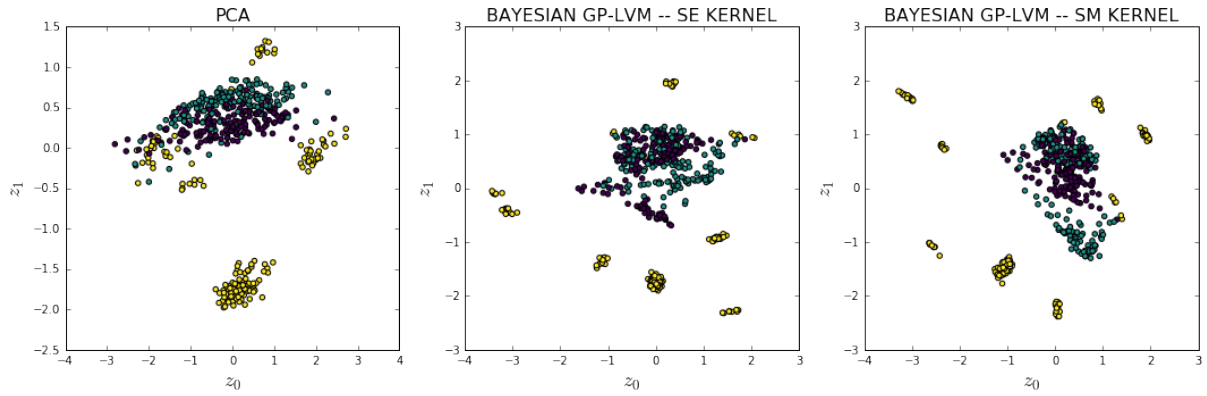


Figure 6.7: Results of dimensionality reduction on the three phase oil dataset. Nearest Neighbor Error (NNE): PCA: 0.19, BGP-LVM SE: 0.21, BGP-LVM SM: 0.18.

In order to prune the irrelevant features from this dataset we use the Bayesian GP-LVM. Additionally, we compare the GP-LVM's subspace-projection with the one from the non-probabilistic and linear Principal Component Analysis (PCA) algorithm. Both dimensionality reduction techniques are discussed in Section 4.2. Figure 6.7 shows the results obtained by applying the PCA and Bayesian GP-LVM with 2 latent dimensions using the ARD SE and the SM kernel. In the Bayesian models are the variances of the variational distribution initialized uniformly between 0 and 0.1, while the means of the distribution are initialized based on PCA. Furthermore, the Bayesian GP-LVM is configured with only 20 inducing points, which makes the training very efficient. As shown in the figures, both algorithms are able to obtain an acceptable quality 2D visualization, automatically revealing the dataset's lower dimensional nature.

The quality of the class separation in the latent space can also be assessed numerically in terms of the Nearest Neighbor Error (NNE): the relative percentage of points whose closest neighbor

in the latent space corresponds to a data point of a different class. For comparison, the left plot shows the manifold provided by the PCA method, and has a NNE of 0.19. The middle and right plot, show the latent space obtained by a Bayesian GP-LVM, configured with the popular SE kernel and the SM kernel, respectively. The middle visualization corresponds with a NNE of 0.21, while to right one has the lowest NNE of 0.18. In conclusion, the integration of the SM kernel, accompanied with its kernel expectations, in the Bayesian GP-LVM is advantageous as it is able to outperform the other methods.

6.4.2 State-Space Model Extrapolation

The Mackey-Glass chaotic time series is a well-known benchmark for the prediction of uncertain time series (Mackey et al. 1977). In this experiment we consider the continuous model

$$\frac{d\,z(t)}{d\,t} = \frac{a\,z(t-\tau)}{1+z(t-\tau)^{10}} - b\,z(t),$$

where $a = 0.2,\ b = 0.1$ and $\tau = 17$. This differential equation is numerically solved and then sampled H times with frequency 1. The obtained vector $\boldsymbol{z} \in \mathbb{R}^H$ is normalized and corrupted by white Gaussian noise with variance $1E-3$. We construct a state-space Gaussian Process Regression (GPR) model for the transition function $f(\cdot)$, which has as states (inputs) and targets (outputs)

$$\begin{cases} \boldsymbol{x}_t = [z_{t-1}, z_{t-2}, \ldots, z_{t-L}]^\top, \\ y_t = z_t \end{cases}$$

In other words, the 1D time series is transformed into a state-space model, where the state at a certain time t is a function of the previous L states. In the dataset $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{y}\}$ used for training, is the target vector $\boldsymbol{y} \in \mathbb{R}^N$ constructed by randomly picking N samples from the H initial ones. The input matrix $\boldsymbol{X} \in R^{N \times L}$, are the corresponding L previous states for each output in $\boldsymbol{y}$. In Figure 6.8 this approach is illustrated graphically: the black dots are outputs, picked at random timestamps between 0 and H. The blue dots, preceding a black dot, are the corresponding L inputs, which are stored as rows in $\boldsymbol{X}$. In this experiment $N = 100$ and $H = 8030$.

The objective is to extrapolate the time series 100-steps ahead, starting from H. We shall consider two cases: (1) a naive approach, named *certain predict* and (2) a more intelligent way, where output uncertainty is taken into account, dubbed as *uncertain predict*.

Certain predict

The naive approach will iteratively predict the next outcome and use this estimate, together with the previous outputs (up to the lag L), as the input for the next prediction. Particularly, we ignore the uncertainty over an output y_{H+i} and only use the predicted mean to perform the next prediction of y_{H+i+1}. The procedure works as follows, assuming outputs y_1 up to y_H are known and we have an optimized GP model:

1. $y_{H+1} \sim \mathcal{N}\left(\mu(\boldsymbol{x}_{H+1}), \sigma^2(\boldsymbol{x}_{H+1})\right)$ with $\boldsymbol{x}_{H+1} = [y_H, y_{H-1}, \ldots, y_{H-L+1}]^\top$ and where $\mu(\cdot)$

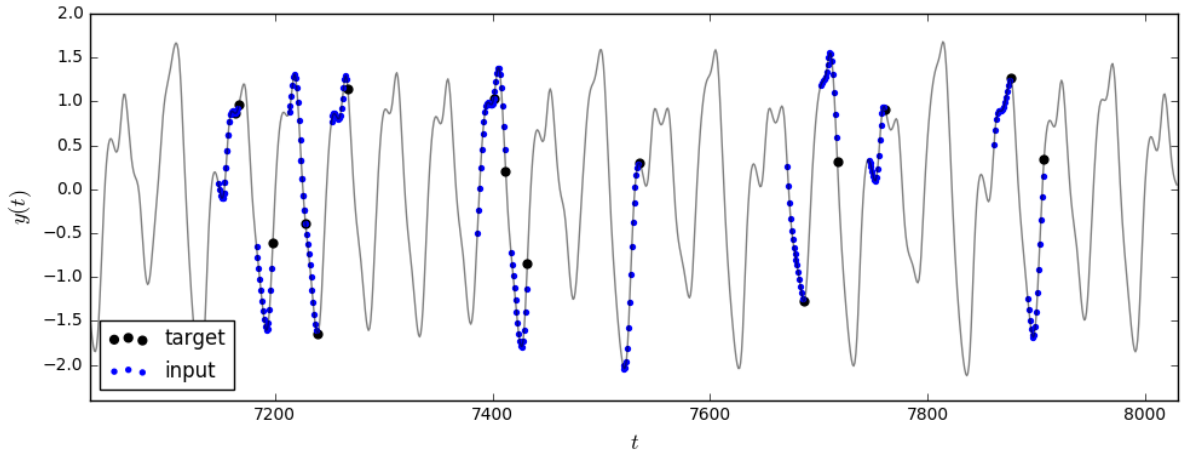
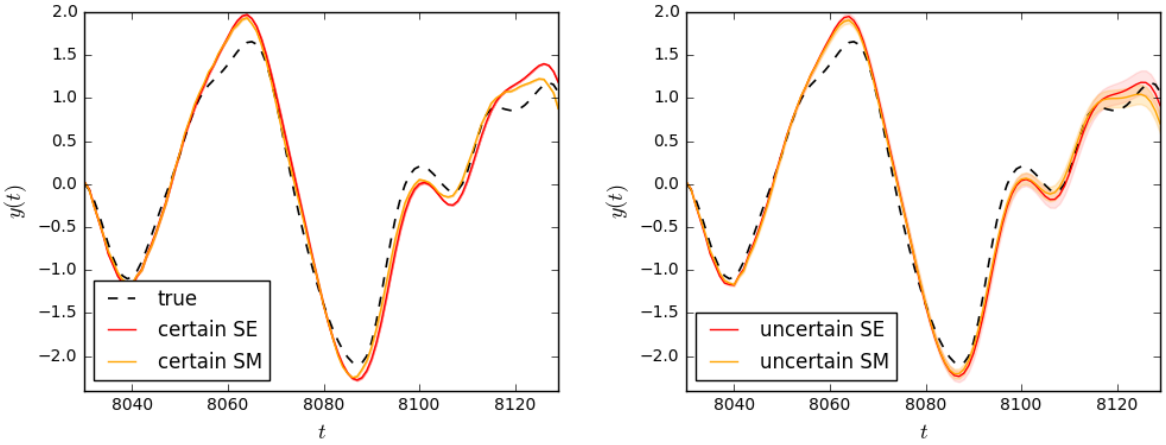


Figure 6.8: Extract from the MacKey-Glass time series.



(a) certain (b) uncertain

Figure 6.9: 100-steps ahead predictions of the MacKey-Glass time series.

and $\sigma^2(\cdot)$ are given by Eq. 2.6 and 2.7, respectively.

2. Use $\hat{y}_{H+1} \triangleq \mu(\boldsymbol{x}_{H+1})$ as estimate for y_{H+1}, then the next input $\boldsymbol{x}_{H+2} = [\hat{y}_{H+1}, y_H, y_{H-1}, \ldots, y_{H-L+2}]^\top$ and $y_{H+2} \sim \mathcal{N}\left(\mu(\boldsymbol{x}_{H+2}), \sigma^2(\boldsymbol{x}_{H+2})\right)$.

3. Use $\hat{y}_{H+2} \triangleq \mu(\boldsymbol{x}_{H+2})$ as estimate for y_{H+2}, then the next input $\boldsymbol{x}_{H+3} = [\hat{y}_{H+2}, \hat{y}_{H+1}, y_H, \ldots, y_{H-L+3}]^\top$ and $y_{H+3} \sim \mathcal{N}\left(\mu(\boldsymbol{x}_{H+3}), \sigma^2(\boldsymbol{x}_{H+3})\right)$.

4. Repeat this until the desired horizon is reached.

This approach is followed to obtain Figure 6.9a, once for a GP using a SM kernel (yellow) and once for a GP using a SE kernel (red). The uncertainty over the predictions, i.e. $\pm 2\sigma$, is also plotted. However, as σ is almost 0 these uncertainty regions are barely visible. This is also the most important problem with the naive approach: it consistently throws away the uncertainty over its predictions which leads to severely underestimating the variance over the next predictions.

Uncertain predict

Using the theory derived in Section 4.1 it is possible to incorporate the uncertainty information about a prediction when using it as an input. Particularly, as we predict ahead in time, we considered the inputs as a random variable, with a mean formed by the predicted means and a covariance matrix holding the predicted variances. The same notation as in Section 4.1 will be used: an uncertain input is Gaussian distributed with mean $\boldsymbol{w}$ and covariance $\boldsymbol{S}$. Like in the previous approach, we again assume that outputs $y_1, \ldots, y_H$ are known and that we have an optimized GP function. Then the procedure works as follows:

1. $y_{H+1} \sim \mathcal{N}\left(m(\boldsymbol{w}_{H+1}, \boldsymbol{S}_{H+1}), v(\boldsymbol{w}_{H+1}, \boldsymbol{S}_{H+1})\right)$, where $m(\cdot)$ and $v(\cdot)$ are respectively given by Eq. 4.4 and 4.5. Since this is the first prediction, all the inputs are observed outputs so we assume they are certain. This gives

$$\boldsymbol{w}_{H+1} = \begin{bmatrix} y_H \\ y_{H-1} \\ \vdots \\ y_{H-L+1} \end{bmatrix} \quad \text{and} \quad \boldsymbol{S}_{H+1} = \begin{bmatrix} 0 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & 0 \end{bmatrix}.$$

   As $\boldsymbol{S}_{H+1}$ is the zero matrix, the predictive distribution $\mathcal{N}(m(\boldsymbol{w}_{H+1}, \boldsymbol{S}_{H+1}), v(\boldsymbol{w}_{H+1}, \boldsymbol{S}_{H+1}))$ equals $\mathcal{N}(\mu(\boldsymbol{w}_{H+1}), \sigma^2(\boldsymbol{w}_{H+1}))$. Indeed, as the uncertain input in the first step has a zero covariance matrix, the deterministic mean and variance formula (Eq. 2.6 and 2.7) apply.

2. The input $\boldsymbol{x}_{H+2}$ for the next prediction y_{H+2} will be a random variable and incorporate the uncertainty over y_{H+1}, which we obtained in the previous step. This gives

$$\boldsymbol{w}_{H+2} = \begin{bmatrix} m(\boldsymbol{w}_{H+1}, \boldsymbol{S}_{H+1}) \\ y_H \\ \vdots \\ y_{H-L+2} \end{bmatrix} \quad \text{and} \quad \boldsymbol{S}_{H+2} = \begin{bmatrix} v(\boldsymbol{w}_{H+1}, \boldsymbol{S}_{H+1}) & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & 0 \end{bmatrix}.$$

   Thus, $\boldsymbol{x}_{H+2} \sim \mathcal{N}(\boldsymbol{w}_{H+2}, \boldsymbol{S}_{H+2})$ and the predictive distribution of y_{H+2} becomes

$$y_{H+2} \sim \mathcal{N}\left(m(\boldsymbol{w}_{H+2}, \boldsymbol{S}_{H+2}), v(\boldsymbol{w}_{H+2}, \boldsymbol{S}_{H+2})\right).$$

3. **Prediction of y_{H+k} with $2 < k < L$:**
   We now consider the general case for the k^{th}-step ahead prediction. However, as $k < L$ we still have some certain outputs with no variance. This gives

$$\boldsymbol{w}_{H+k} = \begin{bmatrix} m(\boldsymbol{w}_{H+k-1}, \boldsymbol{S}_{H+k-1}) \\ \vdots \\ m(\boldsymbol{w}_{H+1}, \boldsymbol{S}_{H+1}) \\ y_H \\ \vdots \\ y_{H-L+k} \end{bmatrix} \quad \text{and}$$

$$\boldsymbol{S}_{H+1} = \left[\begin{array}{ccc|c} v(\boldsymbol{w}_{H+k-1}, \boldsymbol{S}_{H+k-1}) & \dots & 0 & \mathbf{0} \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & v(\boldsymbol{w}_{H+1}, \boldsymbol{S}_{H+1}) & \mathbf{0} \\ \hline \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \end{array}\right],$$

where the lower-right zero matrix of $\boldsymbol{S}_{H+1}$ has dimensions $(L-k+1) \times (L-k+1)$. The predictive distribution of y_{H+k} is again given by $\mathcal{N}\left(m(\boldsymbol{w}_{H+k}, \boldsymbol{S}_{H+k}), v(\boldsymbol{w}_{H+k}, \boldsymbol{S}_{H+k})\right)$.

4. **Prediction of y_{H+k} with $k \geq L$:**
   Starting from timestamps that are larger or equal than L, there are no certain outputs in the history anymore. Consequently, the input covariance $\boldsymbol{S}$ becomes a diagonal matrix containing the variance of the L previous predictions and the input mean vector $\boldsymbol{w}$ now solely consists out of the previous L predicted means.

Figure 6.9b shows the result of using this uncertainty propagation approach to perform 100-steps ahead predictions. As in Figure 6.9a the experiment is conducted for a GPR model with a SE (red) and a SM (yellow) kernel. We observe that the predicted means, for both approaches and for both kernels, are relatively similar. However, in the case of uncertain predict (6.9b) we notice that the variance over the predictions increases with k and that the true curve is more often within the uncertainty interval $\pm 2\sigma$. This is definitely more realistic compared to first approach (6.9a) where the variance over the predictions is negligible and constant over time. Having a more accurate uncertainty over the predictions is one of the most important advantages of this technique.

When assessing the extrapolations numerically (see Table 6.1) the lowest error is found for the uncertainty propagation approach in combination with the SM kernel. Section 4.1 showed that the kernel expectations (i.e. ξ, Ω and Φ) are necessary to implement this approach, which highlights the usefulness of the SM kernel expectation derivations carried out above.

Table 6.1: Test metrices for 100-steps ahead predictions of the MacKey-Glass time series.

	certain		uncertain	
	SE	SM	SE	SM
MSE	0.036	0.021	0.023	**0.019**
RMSE	0.191	0.145	0.152	**0.138**
RRSE	0.183	0.139	0.146	**0.132**

Chapter 7

Conclusion

Summary

The Gaussian Process (GP) framework is non-parametric and fully probabilistic. It enables specifying a prior distribution over a latent function and provides a tractable posterior for hyperparameter optimization and inference. The ability to quantify the uncertainty about a prediction and the data-efficiency are two of its most important advantages. GP-based models compete with the state-of-the-art in many machine learning fields, such as regression, dimensionality reduction (e.g., Latent Variable Models), reinforcement learning (e.g., PILCO) and Bayesian optimization.

Chapter 2 provides an elaborate discussion of this non-parametric model and situates it within the bigger framework of probabilistic machine learning. The most important aspects, including GP regression and classification, hyperparameter learning and covariance functions are discussed in detail.

Even when taking recent advances into consideration, GPs are still not fully applicable for modeling big data. This is due to two important limitations. Firstly, GPs using standard covariance functions, such as the stationary Squared Exponential (SE), can typically only be used successfully for smoothing and interpolation. Where other techniques, such as Neural Networks (NNs), are intrinsically better to discover meaningful patterns, GPs have difficulties with this. Secondly, the required computations and storage during training and inference scales cubically in the number of data. Limiting the overall size of the datasets leveraged for GP modeling. This dissertation addresses both shortcomings and suggests multiple approaches to remedy them.

The first contribution of this thesis is the development and implementation of an algorithm that significantly speeds up the Cholesky computation in TensorFlow. Most GP software-frameworks require the Cholesky decomposition for the inversion of a positive-definite matrix. It is therefore important to have a fast implementation available. The developed implementation is GPU and CPU compatible and makes use of efficient matrix-matrix computations (BLAS Level 3) to optimally exploit the available hardware in modern workstations. When integrating it in a

standard GP it achieves a 3× speedup on large datasets, compared to the default TensorFlow implementation.

The second contribution addresses the extrapolation issues of GPs. The traditional stationary kernels, which are the de-facto standard, are only parametrized by lengthscales, controlling the smoothing capabilities of the model. While this suffice for intelligent interpolation, it is often not powerful enough to extrapolate the function into new regions. A possible solution for this problem is the use of the Spectral Mixture (SM) kernel. The usefulness of the SM kernel is highlighted through different experiments, but more importantly, the kernel expectations (i.e. $\{\xi, \Omega, \Phi\}$) of the SM kernel are derived in closed analytical form. This is novel research and not available in the literature. Having the kernel expectations allows one to use the SM kernel for tackling real-world problems with more advanced and efficient GP models, like input uncertainty propagation and Bayesian GP-LVMs (BGP-LVMs).

Thirdly, surrogate modeling is studied, where the goal is to approximate a simulation response surface with a cheap-to-evaluate mathematical approximation. In particular, we focused on surrogate modeling in combination with sequential design as this makes it much more data and time efficient. While GPs are often used for this, we showed that using the default stationary kernels can be inappropriate in certain scenarios. As a solution Deep Gaussian Processes (DGPs) are proposed and a series of experiments is conducted to validate this. The results of the empirical experiments proofed that DGPs can be a significant contribution to metamodeling in combination with sequential design as they are able to leverage non-uniform sampling schemes, while retaining the advantages of GPs.

Finally, throughout this dissertation we covered in-depth the mathematical derivation of many advanced GP models. For example, the bound of the Sparse Pseudo-input Gaussian Process (SPGP) in Chapter 3 and the variational lower bound of the Sparse Variational Gaussian Process (SVGP) in Chapter 4. To the best of our knowledge a more detailed derivation of these bounds is not found in the literature. With this work, we hope to help future students gain more insight into the models, familiarize them with some the mathematical tools and lower the steep learning curve of GPs in general.

Future Work

GP-based modeling is still an open field of study. More than ever, new advances are required to keep *competing* with state-of-the-art deep learning techniques, which are currently the dominating model in big data machine learning. In this section we point out some interesting research directions.

A lot of datasets, nowadays considered as big data, consists of images (e.g. ImageNet, MNIST, CIFAR, etc.). Therefore, we believe it is important to focus on GPs in combination with 2D image inputs. Currently, 2D images are transformed into 1D vectors before using them as inputs, which leads to a considerable loss of spatial information. Incorporating this spatial information in model could significantly increase the performance. This approach is comparable with convolutional layers in NNs, which are also tailored for image data.

A second interesting study is the use of Deep Bayesian networks for generative modeling. Traditional Generative Adversarial Networks (GANs) use (deep) neural networks as mapping function for both the generative and discriminative model (Goodfellow et al. 2014). Training GANs is due to the high number of parameters expensive and extremely prone to overfitting. A Bayesian treatment of GANs would involve using GPs as mapping functions, instead of the traditional neural networks, effectively reducing the number of parameters. This could lead to the development of Bayesian GANs.

A last valuable research direction we foresee is data-efficient Reinforcement learning (RL) using GPs. GPs have intrinsically difficulties with handling large datasets, but are, compared to other techniques, extremely powerful in learning from a limited amount of data. Throughout this dissertation this characteristic was most considered as a shortcoming as we wish to model big data. However, in the context of RL it is actually a convenient property as it can reduce the required exploration time of an agent. While traditional RL techniques (e.g., Deep Q-Learning models) need several hours to days to learn a complex task, GPs could be used to learn these tasks in a fraction of the time. A notable example of this is the PILCO framework, which is able to learn the cart-pole control problem in only 17.5 seconds (Deisenroth and Rasmussen 2011). Furthermore, GPs are able to give uncertainty over their predictions (or actions in the context of RL), which can come in extremely handy when making important decisions.

References

Bauer, M., M. van der Wilk, and C. E. Rasmussen. 2016. "Understanding probabilistic sparse gaussian process approximations". In *Advances in Neural Information Processing Systems*, 1525–1533.

Bishop, C. M. 2006. *Pattern Recognition and Machine Learning*. Springer.

Bishop, C. M., and G. D. James. 1993. "Analysis of multiphase flows using dual-energy gamma densitometry and neural networks". *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 327 (2-3): 580–593.

Bochner, S. 1959. *Lectures on Fourier Integrals.(AM-42)*, Volume 42. Princeton University Press.

Bui, T. D., J. M. Hernández-Lobato, D. Hernández-Lobato, and D. H. Es. 2016. "Deep Gaussian Processes for Regression using Approximate Expectation Propagation". In *Proceedings of The 33rd International Conference on Machine Learning (ICML)*.

Chatfield, C. 1989. *Time Series Analysis: An Introduction*. Chapman and Hall.

Choi, J., J. J. Dongarra, L. S. Ostrouchov, A. P. Petitet, D. W. Walker, and R. C. Whaley. 1996. "Design and implementation of the ScaLAPACK LU, QR, and Cholesky factorization routines". *Scientific Programming* 5 (3): 173–184.

Cisco 2016. "Visual Networking Index: Forecast and Methodology, 2015–2020".

Crombecq, K., L. De Tommasi, D. Gorissen, and T. Dhaene. 2009, December. "A novel sequential design strategy for global surrogate modeling". In *Proceedings of the 2009 Winter Simulation Conference (WSC)*, 731–742.

Csató, L., and M. Opper. 2002, March. "Sparse On-line Gaussian Processes". *Neural Comput.* 14 (3): 641–668.

Damianou, A. 2015. *Deep Gaussian Processes and Variational Propagation of Uncertainty*. Ph. D. thesis, University of Sheffield.

Damianou, A. C., and N. D. Lawrence. 2013. "Deep Gaussian Processes". In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2013, Scottsdale, AZ, USA, April 29 - May 1, 2013*, 207–215.

Deisenroth, M., and C. E. Rasmussen. 2011. "PILCO: A model-based and data-efficient approach to policy search". In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, 465–472.

Fonseca, D., D. Navaresse, and G. Moynihan. 2003. "Simulation metamodeling through artificial neural networks". *Engineering Applications of Artificial Intelligence* 16 (3): 177–183.

Forrester, A., A. Sobester, and A. Keane. 2008. *Engineering Design Via Surrogate Modelling: A Practical Guide*. Chichester: Wiley.

Ghahramani, Z. 2015. "Probabilistic machine learning and artificial intelligence". *Nature* 521 (7553): 452–459.

Gibbs, M. N., and D. J. MacKay. 2000. "Variational Gaussian process classifiers". *IEEE Transactions on Neural Networks* 11 (6): 1458–1464.

Girard, A., C. E. Rasmussen, J. Q. Candela, and R. Murray-Smith. 2003. "Gaussian process priors with uncertain inputs-application to multiple-step ahead time series forecasting". *Advances in neural information processing systems*:545–552.

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

Gorissen, D., I. Couckuyt, P. Demeester, T. Dhaene, and K. Crombecq. 2010, August. "A Surrogate Modeling and Adaptive Sampling Toolbox for Computer Based Design". *J. Mach. Learn. Res.* 11:2051–2055.

Gorissen, D., L. D. Tommasi, J. Croon, and T. Dhaene. 2008, June. "Automatic model type selection with heterogeneous evolution: An application to RF circuit block modeling". In *IEEE Congress on Evolutionary Computation*, 989–996.

GPy since 2014. "GPy: A Gaussian process framework in python". http://github.com/SheffieldML/GPy.

Gramacy, R. B., and H. K. H. Lee. 2008. "Bayesian Treed Gaussian Process Models With an Application to Computer Modeling". *Journal of the American Statistical Association* 103 (483): 1119–1130.

Hensman, J., N. Fusi, and N. D. Lawrence. 2013. "Gaussian processes for big data". In *Conference on Uncertainty in Artificial Intellegence*, 282–290. auai.org.

Hensman, J., and N. D. Lawrence. 2014. "Nested variational compression in deep Gaussian processes". visited on 12/03/2016, 2014.

Hensman, J., A. G. d. G. Matthews, and Z. Ghahramani. 2015. "Scalable Variational Gaussian Process Classification". In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*.

Higham, N. J. 2002. *Accuracy and stability of numerical algorithms*. SIAM.

Hyndman, R.J. 2005. "Time series data library". http://www-personal.buseco.monash.edu.au/~hyndman/TSDL/. visited on 07/04/2017.

Jolliffe, I. 2002. *Principal component analysis*. Wiley Online Library.

Kleijnen, J. P. 2015. *Design and Analysis of Simulation Experiments*. 2 ed. Springer International Publishing.

Kostantinos, N. 2000. "Gaussian mixtures and their applications to signal processing". *Advanced signal processing handbook: theory and implementation for radar, sonar, and medical imaging real time systems*:3–1.

Kuss, M., and C. E. Rasmussen. 2005. "Assessing approximate inference for binary Gaussian process classification". *Journal of machine learning research* 6 (Oct): 1679–1704.

Lawrence, N. 2003. "Gaussian Process Latent Variable Models for Visualisation of High Dimensional Data". In *In NIPS*, 2004.

Lawrence, N. 2005. "Probabilistic non-linear principal component analysis with Gaussian process latent variable models". *Journal of Machine Learning Research* 6 (Nov): 1783–1816.

Lawrence, N. D., and J. Quiñonero-Candela. 2006. "Local distance preservation in the GP-LVM through back constraints". In *Proceedings of the 23rd international conference on Machine learning*, 513–520. ACM.

Lehmensiek, R., P. Meyer, and M. Müller. 2002. "Adaptive sampling applied to multivariate, multiple output rational interpolation models with application to microwave circuits". *International Journal of RF and Microwave Computer-Aided Engineering* 12 (4): 332–340.

Li, X. R., and Z. Zhao. 2006, October. "Evaluation of estimation algorithms part I: incomprehensive measures of performance". *IEEE Transactions on Aerospace and Electronic Systems* 42 (4): 1340–1358.

MacKay, D. 1991. *Bayesian methods for adaptive modelling*. Ph. D. thesis, PhD thesis, California Institute of Technology.

MacKay, D. J. 1997. "Gaussian processes-a replacement for supervised neural networks?".

MacKay, D. J. C. 2002. *Information Theory, Inference & Learning Algorithms*. New York, NY, USA: Cambridge University Press.

Mackey, M. C., L. Glass et al. 1977. "Oscillation and chaos in physiological control systems". *Science* 197 (4300): 287–289.

Matthews, A. G. G. M., M. van der Wilk, T. Nickson, K. Fujii, A. Boukouvalas, P. León-Villagrá, Z. Ghahramani, and J. Hensman. 2017. "GPflow: A Gaussian Process Library using TensorFlow". *Journal of Machine Learning Research* 18 (40): 1–6.

Minka, T. P. 2001. "Expectation Propagation for Approximate Bayesian Inference". In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, UAI'01, 362–369. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Murray, I. 2016. "Differentiation of the Cholesky decomposition". *arXiv preprint arXiv:1602.07527*.

Neal, R. M. 1994. *Bayesian Learning for Neural Networks*. Ph.D. thesis, Dept. of Computer Science, University of Toronto.

Neal, R. M. 1997. “Monte Carlo implementation of Gaussian process models for Bayesian regression and classification”. *arXiv preprint physics/9701026*.

Nickisch, H., and C. E. Rasmussen. 2008. “Approximations for binary Gaussian process classification”. *Journal of Machine Learning Research* 9 (Oct): 2035–2078.

Park, J.-S. 1994. “Optimal Latin-hypercube designs for computer experiments”. *Journal of Statistical Planning and Inference* 39 (1): 95–111.

K. B. Petersen and M. S. Pedersen 2012, nov. “The Matrix Cookbook”. Version 2012/11/15.

Quiñonero-Candela, J., A. Girard, and C. E. Rasmussen. 2002. “Prediction at an uncertain input for Gaussian processes and relevance vector machines-application to multiple-step ahead time-series forecasting”. Technical report, Technical Report, IMM, Danish Technical University.

Quiñonero-Candela, J., and C. E. Rasmussen. 2005. “A unifying view of sparse approximate Gaussian process regression”. *Journal of Machine Learning Research* 6 (Dec): 1939–1959.

Rasmussen, and Williams. 2006. *Gaussian Processes for Machine Learning*. MIT Press.

Rasmussen, C. E. 1996. *Evaluation of Gaussian processes and other methods for non-linear regression*. Ph. D. thesis, Citeseer.

Salimbeni, H., and M. Deisenroth. 2017. “Doubly Stochastic Variational Inference for Deep Gaussian Processes”. visited on 28/05/2017.

Samo, Y.-L. K., and S. Roberts. 2015. “Generalized Spectral Kernels”. visited on 06/04/2017, 2015.

Santner, T., B. Williams, and W. Notz. 2013. *The Design and Analysis of Computer Experiments*. New York, NY, USA: Springer-Verlag.

Seeger, M., C. K. I. Williams, and N. D. Lawrence. 2003. “Fast Forward Selection to Speed Up Sparse Gaussian Process Regression”. In *workshop on AI and Statistics 09*.

Smola, A. J., and P. L. Bartlett. 2001. “Sparse Greedy Gaussian Process Regression”. In *Advances in Neural Information Processing Systems 13*, edited by T. K. Leen, T. G. Dietterich, and V. Tresp, 619–625. MIT Press.

Snelson, E., and Z. Ghahramani. 2006. “Sparse Gaussian Processes using Pseudo-inputs”. In *Advances in Neural Information Processing Systems 18*, edited by Y. Weiss, B. Schölkopf, and J. C. Platt, 1257–1264. MIT Press.

Stein, M. 1999. *Interpolation of Spatial Data: Some Theory for Kriging*. Springer Verlag.

Suykens, J. A. K., and J. Vandewalle. 1999, June. “Least Squares Support Vector Machine Classifiers”. *Neural Process. Lett.* 9 (3): 293–300.

Tipping, M. E., and C. M. Bishop. 1999. “Probabilistic principal component analysis”. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 61 (3): 611–622.

Titsias, M. K. 2009. “Variational learning of inducing variables in sparse Gaussian processes”. In *In Artificial Intelligence and Statistics 12*, 567–574.

Titsias, M. K., and N. D. Lawrence. 2010. “Bayesian Gaussian Process Latent Variable Model.”.

In *AISTATS*, Volume 9, 844–851.

Turner, R., M. Deisenroth, and C. Rasmussen. 2010. "State-space inference and learning with Gaussian processes". In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 868–875.

van der Herten, J., I. Couckuyt, D. Deschrijver, and T. Dhaene. 2015. "A fuzzy hybrid sequential design strategy for global surrogate modeling of high-dimensional computer experiments". *SIAM Journal on Scientific Computing* 37 (2): 1020–1039.

Vijayakumar, S., A. D'souza, T. Shibata, J. Conradt, and S. Schaal. 2002. "Statistical learning for humanoid robots". *Autonomous Robots* 12 (1): 55–69.

Williams, C. K., and D. Barber. 1998. "Bayesian classification with Gaussian processes". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20 (12): 1342–1351.

Williams, C. K. I., and M. Seeger. 2001. "Using the Nyström Method to Speed Up Kernel Machines". In *Advances in Neural Information Processing Systems 13*, edited by T. K. Leen, T. G. Dietterich, and V. Tresp, 682–688. MIT Press.

Wilson, A., E. Gilboa, J. P. Cunningham, and A. Nehorai. 2014. "Fast kernel learning for multi-dimensional pattern extrapolation". In *Advances in Neural Information Processing Systems*, 3626–3634.

Wilson, A. G., and R. P. Adams. 2013. "Gaussian Process Kernels for Pattern Discovery and Extrapolation.". In *ICML*, 1067–1075.

Wilson, A. G., Z. Hu, R. Salakhutdinov, and E. P. Xing. 2016. "Deep kernel learning". In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, 370–378.

Yang, Z., A. J. Smola, L. Song, and A. G. Wilson. 2014. "A la carte-learning fast kernels". visited on 07/04/2017.

Zhu, C., R. H. Byrd, P. Lu, and J. Nocedal. 1997, December. "Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-scale Bound-constrained Optimization". *ACM Trans. Math. Softw.* 23 (4): 550–560.

Appendix A

Mathematical Identities

In this appendix, we include some identities that are used throughout the thesis. It also gives a good overview of the mathematical *tricks* that are used in a lot of GP-based models. The main resources for this appendix are the corresponding Wikipedia pages and the matrix cookbook (Petersen and Pedersen 2012).

A.1 Matrix Identities

The matrix inversion lemma, also known as the Woodbury, Sherman and Morrison formula states:

$$(\boldsymbol{A} + \boldsymbol{U}\boldsymbol{C}\boldsymbol{U}^\top)^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{U}(\boldsymbol{C}^{-1} + \boldsymbol{U}^\top\boldsymbol{A}^{-1}\boldsymbol{U})^{-1}\boldsymbol{U}^\top\boldsymbol{A}^{-1}, \tag{A.1}$$

where $\boldsymbol{A}$ and $\boldsymbol{C}$ are invertible square matrices of possibly different sizes, and $\boldsymbol{U}$ is a rectangular matrix.

When $\boldsymbol{C}$ is an identity matrix and $\boldsymbol{u}$ and $\boldsymbol{v}$ are vectors, the Woodbury identity can be shortened and become the Sherman-Morrison formula:

$$(\boldsymbol{A} + \boldsymbol{u}\boldsymbol{v}^\top)^{-1} = \boldsymbol{A}^{-1} - \frac{\boldsymbol{A}^{-1}\boldsymbol{u}\boldsymbol{v}^\top\boldsymbol{A}^{-1}}{1 + \boldsymbol{v}^\top\boldsymbol{A}^{-1}\boldsymbol{u}}.$$

Another useful identity is the matrix determinant lemma:

$$|\boldsymbol{A} + \boldsymbol{U}\boldsymbol{C}\boldsymbol{U}^\top| = |\boldsymbol{C}^{-1} + \boldsymbol{U}^\top\boldsymbol{A}^{-1}\boldsymbol{U}|\,|\boldsymbol{C}|\,|\boldsymbol{A}|.$$

A.2 Gaussian Identities

When the marginal distribution $p(\boldsymbol{y})$ is the integral of a product between two Gaussian distributions with the following particular form:

$$p(\boldsymbol{f}) = \mathcal{N}(\boldsymbol{f}|\boldsymbol{m}, \boldsymbol{B})$$
$$p(\boldsymbol{y}|\boldsymbol{f}) = \mathcal{N}(\boldsymbol{y}|\boldsymbol{V}\boldsymbol{f}, \boldsymbol{A}),$$

then $p(\boldsymbol{y}) = \int p(\boldsymbol{y}|\boldsymbol{f})p(\boldsymbol{f})d\boldsymbol{f}$ equals

$$p(\boldsymbol{y}) = \mathcal{N}(\boldsymbol{y}|\boldsymbol{V}\boldsymbol{m}, \boldsymbol{A} + \boldsymbol{V}\boldsymbol{B}\boldsymbol{V}^\top). \tag{A.2}$$

And the conditional distribution $\boldsymbol{f}$ given $\boldsymbol{y}$, is given by

$$p(\boldsymbol{f}|\boldsymbol{y}) = \mathcal{N}(\boldsymbol{f}|\boldsymbol{C}\left(\boldsymbol{V}^\top\boldsymbol{A}^{-1}\boldsymbol{y} + \boldsymbol{B}^{-1}\boldsymbol{m}\right), \boldsymbol{C}),$$

where $\boldsymbol{C} = \left(\boldsymbol{B}^{-1} + \boldsymbol{V}^\top\boldsymbol{A}^{-1}\boldsymbol{V}\right)^{-1}$.

The product of a Gaussian in $\boldsymbol{x}$ with a Gaussian in a linear projection $\boldsymbol{P}\boldsymbol{x}$ is again a Gaussian, although unnormalized

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{m}_1, \boldsymbol{A}_1)\ \mathcal{N}(\boldsymbol{P}\boldsymbol{x}|\boldsymbol{m}_2, \boldsymbol{A}_2) = c\mathcal{N}(\boldsymbol{x}|\boldsymbol{m}, \boldsymbol{A}), \tag{A.3}$$

with

$$\boldsymbol{A} = \left(\boldsymbol{A}_1^{-1} + \boldsymbol{P}^\top\boldsymbol{A}_2^{-1}\boldsymbol{P}\right)^{-1},$$
$$\boldsymbol{m} = \boldsymbol{A}(\boldsymbol{A}_1^{-1}\boldsymbol{m}_1 + \boldsymbol{P}^\top\boldsymbol{A}_2^{-1}\boldsymbol{m}_2),$$
$$c = \mathcal{N}(\boldsymbol{m}_2|\boldsymbol{P}\boldsymbol{m}_1, \boldsymbol{A}_2 + \boldsymbol{P}\boldsymbol{A}_1\boldsymbol{P}^\top).$$

A.3 KL Divergence

The Kullback–Leibler (KL) divergence is a measure used to asses the similarity of two probability distributions. It measures how one distribution diverges from a second. It is defined as follows

$$\mathrm{KL}\left[p(x)\|q(x)\right] = \int_{-\infty}^{\infty} p(x)\ \log\frac{p(x)}{q(x)}\,\mathrm{d}x \geq 0.$$

An interesting property of the KL divergence is that it is always non-negative for all distributions p and q. Also note that $\mathrm{KL}\left[p\|q\right] \neq \mathrm{KL}\left[q\|p\right]$. For most distributions is the KL divergence not analytically tractable. Luckily, for D dimensional multivariate Gaussian distributions, with means $\boldsymbol{\mu}_0$, $\boldsymbol{\mu}_1$ and covariances $\boldsymbol{\Sigma}_0, \boldsymbol{\Sigma}_1$, the KL divergence has the following closed-form

$$\mathrm{KL}\left[\mathcal{N}_0\|\mathcal{N}_1\right] = \frac{1}{2}\left\{\mathrm{tr}\left(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\Sigma}_0\right) + \left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0\right)^\top\boldsymbol{\Sigma}_1^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) - D + \ln\left(\frac{|\boldsymbol{\Sigma}_1|}{|\boldsymbol{\Sigma}_0|}\right)\right\}. \tag{A.4}$$

Appendix B

Proof Unblocked Cholesky Algorithm

In Section 3.2.2 we gave an intuitive algorithm for constructing the Cholesky decomposition. As explained in the section, the Cholesky decomposition $\boldsymbol{L}$ of a symmetric positive definite matrix $\boldsymbol{\Sigma}$ is the unique lower-triangular matrix with positive diagonal elements satisfying $\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^\top$. In this appendix we proof the correctness of the proposed recursive algorithm.

The algorithm and the proof start by partitioning the matrices $\boldsymbol{\Sigma}$ and $\boldsymbol{L}$ into the following block matrices

$$\boldsymbol{\Sigma} = \begin{pmatrix} \sigma_{11} & \boldsymbol{\sigma}_{21}^\top \\ \boldsymbol{\sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix} \text{ and } \boldsymbol{L} = \begin{pmatrix} \ell_{11} & \mathbf{0}^\top \\ \boldsymbol{\ell}_{21} & \boldsymbol{L}_{22} \end{pmatrix}, \tag{B.1}$$

where σ_{11} and ℓ_{11} are scalars, $\boldsymbol{\sigma}_{21}$ and $\boldsymbol{\ell}_{21}$ are column vectors of length $N-1$ and $\boldsymbol{\Sigma}_{22}$ and $\boldsymbol{L}_{22}$ are the right-lower submatrices of $\boldsymbol{\Sigma}$ and $\boldsymbol{L}$, respectively, with dimension $N-1 \times N-1$. Substituting these block-partitioned matrices into the Cholesky factorization $\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^\top$, we find

$$\begin{pmatrix} \sigma_{11} & \boldsymbol{\sigma}_{21}^\top \\ \boldsymbol{\sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix} = \begin{pmatrix} \ell_{11} & \mathbf{0}^\top \\ \boldsymbol{\ell}_{21} & \boldsymbol{L}_{22} \end{pmatrix} \begin{pmatrix} \ell_{11} & \boldsymbol{\ell}_{21}^\top \\ \mathbf{0} & \boldsymbol{L}_{22}^\top \end{pmatrix} = \begin{pmatrix} \ell_{11}^2 & \ell_{11}\boldsymbol{\ell}_{21}^\top \\ \ell_{11}\boldsymbol{\ell}_{21} & \boldsymbol{\ell}_{21}\boldsymbol{\ell}_{21}^\top + \boldsymbol{L}_{22}\boldsymbol{L}_{22}^\top \end{pmatrix},$$

and therefore

$$\begin{aligned} \ell_{11} &= \sqrt{\sigma_{11}}, \\ \boldsymbol{\ell}_{21} &= \boldsymbol{\sigma}_{21}/\ell_{11}, \\ \boldsymbol{L}_{22}\boldsymbol{L}_{22}^\top = \boldsymbol{\Sigma}_{22} - \boldsymbol{\ell}_{21}\boldsymbol{\ell}_{21}^\top &\Rightarrow \boldsymbol{L}_{22} = \textsc{Cholesky}\left(\boldsymbol{\Sigma}_{22} - \boldsymbol{\ell}_{21}\boldsymbol{\ell}_{21}^\top\right). \end{aligned}$$

We will prove that recursively applying this approach will result in constructing the Cholesky decomposition. In order to proof this, the following lemma is needed.

Lemma 1. Let $\boldsymbol{\Sigma} \in \mathbb{R}^{N \times N}$ be a symmetric positive definite matrix, and let $\boldsymbol{\ell}_{21} = \boldsymbol{\sigma}_{21}/\sqrt{\sigma_{11}}$, then $\boldsymbol{\Sigma}_{22} - \boldsymbol{\ell}_{21}\boldsymbol{\ell}_{21}^\top$ is also a symmetric positive definite matrix. The variables σ_{11}, $\boldsymbol{\sigma}_{21}$ and $\boldsymbol{\Sigma}_{22}$ are defined as in Eq. B.1.

Proof. We have $\boldsymbol{\Sigma}$, which is a symmetric matrix, therefore the submatrix $\boldsymbol{\Sigma}_{22}$ is also a symmetric. Further, because of the commutativity of the multiplication and the fact that $\boldsymbol{\Sigma}_{22}$ is symmetric, $\boldsymbol{\Sigma}_{22} - \boldsymbol{\ell}_{21}\boldsymbol{\ell}_{21}^\top$ is also a symmetric.

Now, as we known that $\boldsymbol{\Sigma}_{22} - \boldsymbol{\ell}_{21}\boldsymbol{\ell}_{21}^\top$ is symmetric, the only thing that remains is proving that $\boldsymbol{\Sigma}_{22} - \boldsymbol{\ell}_{21}\boldsymbol{\ell}_{21}^\top$ is also positive definite. Therefore, let $\boldsymbol{x}_1$ be any vector of length $N-1$ different from $\mathbf{0}$. Construct the column vector $\boldsymbol{x} = (x_0, \boldsymbol{x}_1)^\top$, where $x_0 = -\boldsymbol{\sigma}_{21}^\top \boldsymbol{x}_1 / \sigma_{11}$. Then, since $\boldsymbol{\Sigma}$ is positive definite and $\boldsymbol{x}$ is non-zero, we have

$$\begin{aligned} 0 < \boldsymbol{x}^\top \boldsymbol{\Sigma}\, \boldsymbol{x} &= (x_0, \boldsymbol{x}_1) \begin{pmatrix} \sigma_{11} & \boldsymbol{\sigma}_{21}^\top \\ \boldsymbol{\sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix} (x_0, \boldsymbol{x}_1)^\top \\ &= \sigma_{11} x_0^2 + 2 x_0 \boldsymbol{\sigma}_{21}^\top \boldsymbol{x}_1 + \boldsymbol{x}_1^\top \boldsymbol{\Sigma}_{22} \boldsymbol{x}_1. \end{aligned}$$

As x_0 is a scalar, we can write the square x_0^2 as $x_0^\top x_0$. Substituting the definition of x_0 gives

$$\begin{aligned} 0 &< \sigma_{11} \frac{-\boldsymbol{x}_1^\top \boldsymbol{\sigma}_{21}}{\sigma_{11}} \frac{-\boldsymbol{\sigma}_{21}^\top \boldsymbol{x}_1}{\sigma_{11}} + 2 \frac{-\boldsymbol{x}_1^\top \boldsymbol{\sigma}_{21}}{\sigma_{11}} \boldsymbol{\sigma}_{21}^\top \boldsymbol{x}_1 + \boldsymbol{x}_1^\top \boldsymbol{\Sigma}_{22} \boldsymbol{x}_1 \\ &= \boldsymbol{x}_1^\top \left(\boldsymbol{\Sigma}_{22} - \frac{\boldsymbol{\sigma}_{21} \boldsymbol{\sigma}_{21}^\top}{\sigma_{11}} \right) \boldsymbol{x}_1 \\ &= \boldsymbol{x}_1^\top \left(\boldsymbol{\Sigma}_{22} - \boldsymbol{\ell}_{21} \boldsymbol{\ell}_{21}^\top \right) \boldsymbol{x}_1. \end{aligned}$$

As $\boldsymbol{x}_1$ is any non-zero vector, and $\boldsymbol{x}_1^\top \left(\boldsymbol{\Sigma}_{22} - \boldsymbol{\ell}_{21}\boldsymbol{\ell}_{21}^\top \right) \boldsymbol{x}_1 > 0$ we can conclude that $\left(\boldsymbol{\Sigma}_{22} - \boldsymbol{\ell}_{21}\boldsymbol{\ell}_{21}^\top \right)$ is positive definite as well. □

Proof Cholesky decomposition. (By induction)

Base case. For a matrix of size 1 (i.e. $N = 1$), $\ell_{11} = \sqrt{\sigma_{11}}$, which is the unique Cholesky decomposition of the symmetric positive definite matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{1\times 1}$.

Inductive Hypothesis. For every symmetric positive definite matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{N\times N}$ we assume that the matrix $\boldsymbol{L}$ exists and is the Cholesky decomposition, satisfying $\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^\top$

Inductive step. Let $\boldsymbol{\Sigma}$ and $\boldsymbol{L}$ be matrices in $\mathbb{R}^{(N+1)\times(N+1)}$ and partition both as in Eq. B.1. Let $\ell_{11} = \sqrt{\sigma_{11}} \in \mathbb{R}$, $\boldsymbol{\ell}_{21} = \boldsymbol{\sigma}_{21}/\ell_{11} \in \mathbb{R}^N$ and $\boldsymbol{L}_{22} = \textsc{Cholesky}\left(\boldsymbol{\Sigma}_{22} - \boldsymbol{\ell}_{21}\boldsymbol{\ell}_{21}^\top \right) \in \mathbb{R}^{N\times N}$. $\boldsymbol{L}_{22}$ exists as a result of the Inductive Hypothesis and Lemma 1. $\boldsymbol{L}$ is now the desired Cholesky decomposition of $\boldsymbol{\Sigma}$. □

This page is intentionally
left blank.

Non-Stationary Surrogate Modeling with Deep Gaussian Processes

Vincent Dutordoir

Supervisors: Prof. dr. ir. Tom Dhaene, Dr. ir. Ivo Couckuyt
Counsellors: Joachim van der Herten, Nicolas Knudde

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Department of Information Technology
Chair: Prof. dr. ir. Bart Dhoedt
Faculty of Engineering and Architecture
Academic year 2016-2017

GHENT
UNIVERSITY