

ADVANCES IN SCALABLE BAYESIAN INFERENCE:  
GAUSSIAN PROCESSES & DISCRETE VARIABLE MODELS

by

Trefor W. Evans

A thesis submitted in conformity with the requirements  
for the degree of Doctor of Philosophy  
Graduate Department of the Institute for Aerospace Studies  
University of Toronto

© Copyright 2022 by Trefor W. Evans

Advances in Scalable Bayesian Inference:  
Gaussian Processes & Discrete Variable Models

Trefor W. Evans  
Doctor of Philosophy

Graduate Department of the Institute for Aerospace Studies  
University of Toronto  
2022

## Abstract

Gaussian processes are exactly the models we would like to use for modern machine learning tasks; they are non-parametric models whose capacity naturally adapts to the quantity of training data, are highly interpretable, offer powerful opportunities to incorporate prior knowledge, and they deal with uncertainty due to lack of data in a rigorous manner through Bayesian inference. Unfortunately, the generic algorithm for Gaussian process training and inference scales with  $\mathcal{O}(n^3)$  time and  $\mathcal{O}(n^2)$  storage on a problem with  $n$  training observations. Given present-day computational resources, this scaling makes Gaussian processes struggle to scale beyond modestly sized datasets. This thesis explores approaches to scale Gaussian process training and inference to large datasets without sacrificing the benefits of these models. Specifically, we present theoretical analyses alongside algorithmic advances in Gaussian process modelling and inference for regression and classification problems.

First, we consider Gaussian process inference on a problem structure present in many spatiotemporal problems and develop several algorithms that dramatically reduce the complexity of exact Gaussian process inference using Kronecker matrix algebra.

We then discuss a novel Gaussian process approximation by showing how an accurate Nyström approximation of kernel eigenfunctions can use as many as  $10^{33}$  inducing points with little computational expense. We subsequently consider a highly general class of Gaussian process covariance kernels and show that the Gaussian process marginal likelihood can be computed with a complexity that is independent of the number of training observations and as low as linear in the number of kernel basis functions.

We then consider a variational inference approximation to the Gaussian process posterior that exploits a stochastic training strategy whose per iteration complexity is independent of

both the number of training examples and the number of kernel basis functions. We show that this unique approach enables the use of high-capacity Gaussian process models on large datasets for regression and classification.

Finally, we consider a discrete relaxation of continuous priors that enables fast inferencing on devices with limited computing resources. We develop a novel variational inference procedure that exploits Kronecker matrix algebra to compute the variational bound exactly and with a complexity that is independent of the dataset size.

## Acknowledgments

I would like to thank the members of my doctoral examination committee: Professors Tim Barfoot and Masayuki Yano for their support over the years, as well as Professors Karthik Duraisamy and Costas Sarris for their thoughts and feedback on my thesis. I am also grateful to Prof. Prasanth Nair for his supervision throughout my graduate studies.

I would like to thank all of my lab mates, past and present. Christophe Audouze, Justin Beland, Isaac Chung, Lin Gao, and Marc Palaci – it was a pleasure to work with you all. It was also great to work with Geoff Donoghue and Aaron Klein who were always happy to talk about research, no matter how technical, and to share their thoughts. It was also a great pleasure to work with Aravind Shaj who always brought a cheery demeanour and bright enthusiasm. I would also like to thank Behrad Vatankhahghadim, my close friend throughout all of our numerous university years.

I would like to sincerely thank Jialian Wu for her love and support. She would bear the load whenever I wanted to talk about my work and she would share in my excitement about any new idea. Late at night before a submission deadline, she would be proofreading my papers, and she was also the one who pushed me to finally finish my thesis!

I am eternally grateful to my parents Janice and Bryn for their constant love and support. I am also grateful to my siblings Rhiannon, Bryn and Davy for never failing to take me down a notch or two when I needed it.

Lastly, I gratefully acknowledge the financial support for this project from the Natural Sciences and Engineering Research Council of Canada (NSERC), the University of Toronto, and the Kenneth M. Molson foundation.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Background</b>	<b>4</b>
2.1	Bayesian Learning . . . . .	4
2.2	Gaussian Processes . . . . .	9
2.2.1	Basis Function Models . . . . .	9
2.2.2	Function-space View . . . . .	12
2.2.3	Covariance Kernels . . . . .	17
2.3	Model Selection & Model Evidence . . . . .	20
2.3.1	Model Evidence . . . . .	21
2.3.2	Type-I Inference . . . . .	22
2.3.3	Type-II Empirical Bayes . . . . .	24
2.4	Gaussian Process Approximations . . . . .	26
2.4.1	Sparse Gaussian Processes . . . . .	26
2.4.2	Gaussian Processes with Non-Gaussian Likelihoods . . . . .	29
2.5	Concluding Remarks . . . . .	30
<b>3</b>	<b>Scaling Exact Gaussian Processes on Multi-dimensional Grids</b>	<b>32</b>
3.1	Introduction . . . . .	32
3.2	Notation & Problem Structure . . . . .	34
3.3	Linear Algebra aspects of GP Training . . . . .	37
3.3.1	Gap Penalization Strategy . . . . .	37
3.3.2	Selection Matrix Strategy . . . . .	38
3.3.3	Fill Gaps Strategy . . . . .	38
3.3.4	Preconditioning strategies . . . . .	39
3.4	Model Selection & Fast Inferencing . . . . .	40
3.5	Experiments . . . . .	42
3.5.1	Stress Tests . . . . .	42
3.5.2	Preconditioner Studies . . . . .	43
3.5.3	Ontario Weather Stations . . . . .	44
3.5.4	Particle Image Velocimetry Flow Reconstruction . . . . .	47

3.6	Concluding Remarks . . . . .	50
<b>4</b>	<b>Scaling Gaussian Processes with the Nyström Approximation</b>	<b>52</b>
4.1	Introduction . . . . .	52
4.2	Eigenfunction Kernel Approximation . . . . .	54
4.3	Grid-Structured Eigenfunctions (GRIEF) . . . . .	55
4.3.1	Eigenvalue Search . . . . .	58
4.3.2	Stable Computation in High Dimensions . . . . .	58
4.3.3	Preconditioning Applications . . . . .	59
4.3.4	SKI Applications . . . . .	59
4.4	Experimental Studies . . . . .	60
4.4.1	Two-Dimensional Visualization . . . . .	60
4.4.2	Kernel Reconstruction Accuracy . . . . .	60
4.4.3	UCI Regression Studies . . . . .	62
4.5	Conclusions . . . . .	63
<b>5</b>	<b>Scaling Type-I Inference with Gaussian Processes</b>	<b>64</b>
5.1	Introduction . . . . .	64
5.2	Re-weighted Basis Kernel . . . . .	66
5.2.1	$\mathcal{O}(m^3)$ Evidence Computations . . . . .	67
5.2.2	$\mathcal{O}(m^2)$ Evidence Computations . . . . .	68
5.2.3	$\mathcal{O}(m)$ Evidence Computations . . . . .	69
5.2.4	Prior over Noise Variance . . . . .	69
5.2.5	Non-Zero Prior Mean . . . . .	70
5.3	Applications . . . . .	70
5.3.1	Importance-Weighted Random Fourier Features . . . . .	70
5.3.2	The Relevance Vector Machine . . . . .	75
5.3.3	Fixed Basis Kernel Interpolation . . . . .	75
5.3.4	Eigenfunction Bases . . . . .	76
5.3.5	Non-Gaussian Likelihoods . . . . .	77
5.4	Experimental Studies . . . . .	77
5.4.1	One-Dimensional Visualization . . . . .	78
5.4.2	UCI Regression Studies . . . . .	79
5.5	Conclusions . . . . .	80
<b>6</b>	<b>Scaling Gaussian Processes using Variational Inference</b>	<b>81</b>
6.1	Introduction . . . . .	81
6.2	Unbiased ELBO Estimator in $\mathcal{O}(1)$ for Regression . . . . .	83
6.2.1	Learning the Variational Mean . . . . .	85
6.2.2	Learning the Variational Covariance . . . . .	86

6.2.3	Empirical Bayes . . . . .	89
6.2.4	Control Variates . . . . .	90
6.3	ELBO Lower Bound Estimator in $\mathcal{O}(1)$ for Classification . . . . .	91
6.4	Numerical Studies . . . . .	94
6.4.1	Classification Stress Testing . . . . .	94
6.4.2	Control Variate Studies . . . . .	94
6.4.3	Regression Studies with Type-II Inference . . . . .	95
6.4.4	Optimization Trace Studies . . . . .	96
6.4.5	Regression Studies with Inducing Point Kernel Approximations . . . . .	97
6.4.6	Relevance Vector Machines . . . . .	99
6.5	Conclusion . . . . .	100
<b>7</b>	<b>Variational Inference with Discrete Variable Models</b>	<b>101</b>
7.1	Introduction . . . . .	101
7.2	Variational Inference Background . . . . .	103
7.3	DIRECT: Efficient ELBO computations . . . . .	104
7.3.1	Generalized Linear Regression . . . . .	106
7.3.2	Deep Neural Networks for Regression . . . . .	109
7.4	Limitations & Extensions . . . . .	110
7.4.1	Generalized Linear Logistic Regression . . . . .	110
7.4.2	Unfactorized Variational Distributions . . . . .	112
7.4.3	Unfactorized Prior Distributions . . . . .	113
7.4.4	Unbiased Entropy and Prior Expectation Gradients . . . . .	114
7.5	Numerical Studies . . . . .	114
7.5.1	Comparison with REINFORCE . . . . .	114
7.5.2	One-Dimensional Regression Visualization . . . . .	115
7.5.3	Relaxing Gaussian Priors on UCI Regression Datasets . . . . .	116
7.6	Conclusions . . . . .	119
<b>8</b>	<b>Concluding Remarks</b>	<b>121</b>
8.1	Overview of Main Results . . . . .	121
8.2	Opportunities for Future Research . . . . .	122
8.3	Conclusion . . . . .	123
<b>A</b>	<b>Kronecker Matrix Algebra</b>	<b>132</b>
<b>B</b>	<b>Regularization Kernel Method for Multi-dimensional Grids</b>	<b>134</b>
<b>C</b>	<b>Re-Weighted Basis Kernel Log-Marginal Likelihood Derivatives</b>	<b>137</b>

<b>D</b>	<b>Additional Content: Quadruply Stochastic Gaussian Processes</b>	<b>140</b>
D.1	Control Variates: Additional Details . . . . .	140
D.1.1	Sparse Unbiased Gradient Computation . . . . .	140
D.1.2	Additional Control Variates . . . . .	140
D.1.3	Control Variates with Empirical Bayes . . . . .	142
D.2	Predictive Posterior Augmentation . . . . .	142
D.3	Site Projection Notes . . . . .	144
D.3.1	Logistic Likelihood . . . . .	145
D.3.2	Laplace Likelihood . . . . .	145
D.3.3	Gaussian Likelihood . . . . .	145
<b>E</b>	<b>DIRECT Bayesian Neural Networks</b>	<b>147</b>



# Chapter 1

## Introduction

Machine learning aims to construct a predictive model whose performance on a particular task is improved by observing data. For example, we may want a model to use examples of handwritten digits to learn how to recognize them. In a *Bayesian* approach to machine learning, we work with probabilistic models and uncertainty. In Bayesian inference, we take a group of hypotheses, and weight those hypotheses based on how well predictions match the observed data, and how well they match prior knowledge about the problem. This approach is appealing for two reasons. First, Bayesian inference provides a natural mechanism to incorporate prior knowledge about a problem, which can enable powerful predictive capabilities even when little data has been observed. Second, keeping all hypotheses that match the observed data helps to guard against overfitting, and admits confidence bounds for data analysis and decision making, a crucially important element in many applications where it is necessary to be able to tell whether a model is certain about its prediction. For instance, if a safety critical decision is being made based upon a machine learning model (such as an autonomous car deciding whether to brake or not), it is important to be aware of how certain the model is about this decision.

Despite the attractive aspects of a Bayesian approach to machine learning, this principled statistical technique does not easily scale to complex models and to large quantities of observed data. This is problematic due to the tremendous growth of data generation in modern times, leading to a push in the engineering and machine learning community for the ability to process and extract actionable information from ever larger datasets in applications from business (Chen et al., 2012) and government policy (Kim et al., 2014) to data-intensive science (Bell et al., 2009). These problems need to be addressed with highly scalable algorithms that can enable fast, or even real-time, processing. Unfortunately, Bayesian approaches to machine learning struggle to scale while addressing these big data problems. Instead, simpler heuristic techniques are often adopted that cannot provide accurate prediction statistics. This can lead to poor generalization robustness, which is dangerous in a society that is placing a growing trust in artificially intelligent systems (Bradshaw et al., 2017; Ghahramani, 2015). This thesis will focus on the development of scalable Bayesian techniques to fill this gap.

Gaussian processes (GPs) are one such Bayesian approach to machine learning. Gaussian

processes, which define probability distributions over functions, can be used to learn more likely and less likely ways to generalize from observed data. Perhaps the most impressive feature of GPs are that computation of the conditional distribution of a Gaussian process on observed data can be done analytically, and also results in a Gaussian process. Gaussian processes also reason directly over functions. This is in contrast to many other machine learning models that reason over abstract objects (for instance in deep neural networks, reasoning and inference is performed over a high dimensional vector of the weights/parameters of the network). Reasoning over functions leads to interpretable models and priors that can easily encode high-level knowledge about the problem at hand (for instance function smoothness, stationarity, additivity, symmetry, or periodicity could all be specified *a priori*). Lastly, being non-parametric models with potentially infinite capacity, Gaussian processes are also exactly the type of models we desire for large quantities of observed data, and the modelling of complex functions. Therefore, Gaussian processes are precisely the model we would like to consider for processing and extracting actionable information from massive datasets of the modern era.

The focus of this thesis is on the development of *scalable* Gaussian process models. We consider two perspectives of scalability: i) the ability to perform inference on Gaussian process models using large quantities of observed data, and ii) the ability to perform inference with Gaussian processes that have the capacity to model complicated functions. Scalable solutions to Gaussian processes often consider one of these goals at the expense of the other. For instance, general (exact) Gaussian processes may have an infinite model capacity but cannot handle large datasets, whereas Gaussian process approximations limit model capacity, allowing the methods to scale to large datasets at the expense of the ability to model complex functions. Scaling Gaussian processes to both of these definitions of scalability is an open and active area of research.

The challenges of scalability are encountered at several stages of the Gaussian process value chain. For instance, during the so-called training stage, where model selection is often carried out and precomputations are performed, necessary operations for exact Gaussian processes generally scale with  $\mathcal{O}(n^2)$  memory and  $\mathcal{O}(n^3)$  time, given  $n$  data observations. During test time when a GP would be deployed, computation of an exact GP predictive posterior generally requires  $\mathcal{O}(n^2)$  memory and time (see section 2.2). These complexities can be prohibitive for both the training and deployment stages, particularly in applications where hardware limitations are restrictive and when predictions need to be performed in real-time. We will therefore discuss implications on GP scaling through both training and deployment stages.

We now provide a brief overview of the thesis structure. Note that specific contributions and associated publications are detailed at the beginning of each respective chapter. In chapter 2, we begin by providing a background on common techniques and tools that are required for further chapters. Specifically, we provide an approachable and comprehensive background on Bayesian inference and Gaussian processes.

In chapter 3, two methods are introduced for exact GP inference and learning on massive image, video, spatiotemporal, or multi-output datasets with missing values (or “gaps”) in the

observed responses. Both of these novel approaches make extensive use of Kronecker matrix algebra to design massively scalable algorithms that have low memory requirements. We demonstrate exact GP inference for a particle image velocimetry (PIV) problem with 20 million training observations, a spatiotemporal climate modelling problem with 3.7 million training observations, as well as a video reconstruction problem with one billion training observations.

In chapter 4, we introduce a kernel approximation strategy that enables Gaussian process training and inference in  $\mathcal{O}(np^2)$  time. The developed ‘‘GRIEF’’ kernel consists of  $p$  eigenfunctions approximated on a dense Cartesian tensor product grid of inducing points. We show that by exploiting algebraic properties of Kronecker and Khatri-Rao tensor products, computational complexity of the training procedure can be *independent* of the number of inducing points, allowing us to use arbitrarily many to achieve a globally accurate kernel approximation. We benchmark our algorithm on real-world problems with as many as two million training observations and up to  $10^{33}$  inducing points.

In chapter 5, we describe a powerful GP model that permits exact inference using a Markov chain Monte Carlo (MCMC) sampling scheme where the cost of each MCMC sample is independent of the size of the training dataset. We also provide an asymptotic result showing that any stationary kernel can be recovered when using the developed parameterization. We benchmark this algorithm on real-world problems with up to two million training observations.

In chapter 6, we introduce a new approach to GP scaling that we call ‘‘quadruply stochastic Gaussian processes’’ (QSGP). The central contribution of the QSGP is a stochastic training procedure that uses a Monte Carlo estimator to make four unbiased estimates of the training loss and is presented in theorems 6.1 and 6.2. These estimators allow stochastic gradient descent to be performed such that each optimization iteration is independent of both the quantity of training data and the complexity of the model. Ultimately, this allows the QSGP model to perform inference on huge datasets using large capacity GP models. We demonstrate accurate inference on large classification and regression datasets using GPs and relevance vector machines with up to  $m = 10^7$  basis functions.

In chapter 7, we introduce a variational inference technique for discrete latent variable models such that the posterior samples consist of sparse and low-precision quantized integers. This method (referred to as ‘‘DIRECT’’) admits deterministic gradients and the training complexity is, again, independent of the size of the training dataset. The DIRECT approach is not practical for all likelihoods; however, we identify a popular model structure that is practical, and demonstrate accurate inference using latent variables discretized as low-precision 4-bit quantized integers. While the ELBO computations considered in the numerical studies require over  $10^{2352}$  log-likelihood evaluations, we train on datasets with over two million points in just seconds.

The thesis concludes in chapter 8 with a summary of research contributions and a discussion of future work.

## Chapter 2

# Background

This chapter provides a background on common techniques and tools that are used in subsequent chapters of this thesis. Specifically, it provides an introduction to Bayesian inference for machine learning on the way to a comprehensive overview of Gaussian processes. There is a notable lack of clean introductory material for Gaussian processes in particular and this chapter aims to help fill that gap with a liberal use of visuals. As an overview, section 2.1 begins by introducing a Bayesian approach to machine learning followed by section 2.2, which then focuses on the particular modelling choice of Gaussian processes (GPs). Section 2.3 continues with a discussion of techniques to perform Bayesian model selection, and section 2.4 concludes with a brief overview of some approximate Gaussian process techniques in the literature that either allow GPs to scale, or to handle more general types of machine learning problems.

### 2.1 Bayesian Learning

Machine learning ultimately aims to create algorithms that improve their performance by leveraging observed data. For example, after observing the results of two experiments, we may want a computer to predict the result of a third experiment. Machine learning algorithms rely on statistical models that will hopefully reflect reality, and these models contain parameters whose value is unknown or uncertain *a priori*. In the process of machine learning, we would like to determine the parameter values that give predictions as close to reality as possible. Given any dataset of finite size, we cannot expect to get completely certain answers about the parameter values. For example, consider fig. 2.1a where two noisy observations have been collected for a one-dimensional regression problem that were generated from the dashed black line with *i.i.d.* Gaussian noise applied. The goal here is to predict a value of  $y^*$  at a given value of  $x^*$ . We have chosen a linear statistical model for this problem and it is visually evident that multiple different straight lines could fit the data, each of which would give different predictions beyond the dataset. Bayesian learning differs from other approaches to machine learning in how to infer the model's parameters given the dataset. For instance, one might suggest selecting the parameters that best fits the data; however, this approach may perform

very poorly as we move away from the training data, as seen in fig. 2.1b. This is known as *overfitting*, a degeneracy that must be accounted for in many approaches to machine learning. In contrast, a Bayesian approach to machine learning finds a distribution over parameter settings that agrees with both the observed data and with prior knowledge. The Bayesian predictions can be seen in fig. 2.1c, which evidently does not provide a point estimate for  $y^*$  given a value of  $x^*$  but rather a distribution over plausible values of  $y^*$ . In this example, these probabilistic predictions compare favourably to the point predictions in fig. 2.1b since it reflects uncertainty due to lack of data. We will return to this example frequently throughout this chapter.

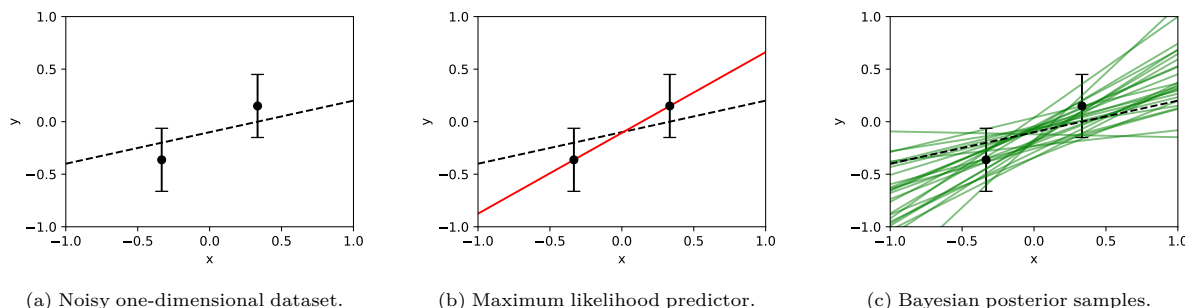


Figure 2.1: Comparison of maximum likelihood and Bayesian approaches to inference. The training data is shown in black which was generated by the dashed black line and corrupted by independent Gaussian noise. The maximum likelihood predictor is given in red whereas samples from the Bayesian posterior are given by the green lines.

In Bayesian inference, probability distributions are used to reflect uncertainty. From the example in fig. 2.1c, samples from the probability distribution over  $y^*$  were used to visualize the fact that  $y^*$  is a random variable at any  $x^*$ . Probability theory provides a rigorous foundation that allows us to reason under uncertainty (Jaynes, 2003). We will now proceed to describe how we can use this theory in machine learning.

## Parameter Inference

We begin Bayesian inference for machine learning by choosing a statistical model. Returning to the example in fig. 2.1, the chosen statistical model takes the form

$$\mathbf{y} = \begin{bmatrix} f(x_1; \mathbf{w}) + \epsilon_1 \\ f(x_2; \mathbf{w}) + \epsilon_2 \end{bmatrix}, \quad (2.1)$$

where

$$f(x; \mathbf{w}) = w_1 x + w_2 \quad (2.2)$$

is a linear model,  $\mathbf{w} = [w_1, w_2]^T$  are model parameters,  $(x_1, y_1)$  and  $(x_2, y_2)$  are the two data points in fig. 2.1a, and  $\epsilon_1, \epsilon_2$  are random variables describing additive noise. In a Bayesian treatment, we will consider both  $\mathbf{y}$  and  $\mathbf{w}$  as random variables, since before observing any data we are uncertain about these variables. We represent the uncertainty about these variables in the joint distribution under the model,  $\Pr(y, \mathbf{w})$ . Using the chain rule (also known as the

product rule) of probability theory, we can write the joint distribution of  $\mathbf{y}$  and  $\mathbf{w}$  as

$$\underbrace{\Pr(\mathbf{y}, \mathbf{w})}_{\text{joint under the model}} = \underbrace{\Pr(\mathbf{y}|\mathbf{w})}_{\text{likelihood}} \underbrace{\Pr(\mathbf{w})}_{\text{prior}}. \quad (2.3)$$

While it is not necessary to decompose the joint under the model in this manner to perform Bayesian inference, it is often convenient for the purposes of interpretability. In our example, eq. (2.1) contains all the information needed to define the *likelihood* (provided we know the statistical properties of  $\epsilon_1$  and  $\epsilon_2$ ). Additionally, the *prior*  $\Pr(\mathbf{w})$  can be selected based upon our belief about the value of the model parameters *a priori* (before observing any data). Specifying a good prior is important for Bayesian inference to be effective. It requires a practitioner to express their belief explicitly as a probability distribution, which can take practice.

Equation (2.3) contains all the information that is required to start a Bayesian modelling procedure. While decomposing the joint under the model into a prior and likelihood is attractive for the purposes of interpretability, it is often useful to consider the joint as its own entity that summarizes all information about the statistical model and the practitioner's prior beliefs.

After specifying the joint under the model, we are now ready for data. The term *inference* (or more specifically statistical inference) refers to making conclusions about uncertain variables given the observational data. This is precisely what we would like to do. To begin, consider rewriting the joint from eq. (2.3) using the chain rule of probability theory in a symmetric manner

$$\underbrace{\Pr(\mathbf{y}, \mathbf{w})}_{\text{joint under the model}} = \underbrace{\Pr(\mathbf{w}|\mathbf{y})}_{\text{posterior}} \underbrace{\Pr(\mathbf{y})}_{\text{model evidence}}. \quad (2.4)$$

Our goal is to compute the *posterior* which provides an update to our belief about  $\mathbf{w}$  *after* observing the dataset. We can compute the posterior by rearranging eq. (2.4) to give

$$\underbrace{\Pr(\mathbf{w}|\mathbf{y})}_{\text{posterior}} = \frac{\underbrace{\Pr(\mathbf{y}, \mathbf{w})}_{\text{joint under the model}}}{\underbrace{\Pr(\mathbf{y})}_{\text{model evidence}}} = \frac{\underbrace{\Pr(\mathbf{y}|\mathbf{w})}_{\text{likelihood}} \underbrace{\Pr(\mathbf{w})}_{\text{prior}}}{\underbrace{\Pr(\mathbf{y})}_{\text{model evidence}}}. \quad (2.5)$$

This simple relation is referred to as *Bayes' rule* and describes how we can update our beliefs after observing data. The only element in the preceding equation that has not yet been discussed is the *model evidence* (also known as the *marginal likelihood*) which is the joint under the model with  $\mathbf{w}$  marginalized. The model evidence can be defined as follows:

$$\underbrace{\Pr(\mathbf{y})}_{\text{model evidence}} = \int \underbrace{\Pr(\mathbf{y}, \mathbf{w})}_{\text{joint under the model}} d\mathbf{w}. \quad (2.6)$$

Unfortunately, this expression is usually challenging to compute since it is an integral which is often high-dimensional and cannot be computed in closed form in many instances. Evaluating

the model evidence is typically the most computationally challenging aspect of Bayesian inference, and this integral alone typically makes Bayesian inference more computationally taxing than a point-estimate approach such as the maximum likelihood procedure shown in fig. 2.1b for our simple example. This is perhaps not too surprising considering that Bayesian inference requires inferring distributions rather than point estimates. A wealth of approaches have been developed to ease computational burden through *approximate* Bayesian inference techniques. Several approximate techniques will be introduced throughout this thesis where required; however, we shall see in section 2.3.1 that the model evidence can be computed in closed form for the majority of the models we will consider.

## Predictive Inference

We previously showed how to update beliefs about the model parameters given data observations through the posterior,  $\Pr(\mathbf{w}|\mathbf{y})$ . While in some scenarios, a practitioner might care about the model parameters directly, in most machine learning scenarios, we only care about the ability to predict  $f(\mathbf{x}^*)$  at a value of  $\mathbf{x}^*$  that was not in the training dataset. In other words, we would like to perform *predictive inference*. This can be performed as follows:

$$\underbrace{\Pr(f(\mathbf{x}^*)|\mathbf{y})}_{\text{predictive posterior}} = \int \underbrace{\Pr(f(\mathbf{x}^*)|\mathbf{w})}_{\text{predictive likelihood}} \underbrace{\Pr(\mathbf{w}|\mathbf{y})}_{\text{posterior}} d\mathbf{w}, \quad (2.7)$$

which is evidently a weighted average of the predictions at all values of parameters  $\mathbf{w}$ , weighted by the posterior. In this way, uncertainty of the parameters are taken into account to express uncertainty over predictions. Quantifying predictive uncertainty is crucial for safe or optimal decision making. In these cases, the predictive posterior would be used for downstream decision making procedures to directly assess risk and potential reward.

In fig. 2.1c, the predictive posterior was sampled 25 times for visualization. In that example, the *predictive likelihood* was the degenerate distribution  $\Pr(f(x^*)|\mathbf{w}) = \delta(f(x^*; \mathbf{w}) - y^*)$  since there is a deterministic relationship between  $f$  and  $\mathbf{w}$  given by eq. (2.2), where  $\delta(\cdot)$  denotes the Dirac delta.

## Example

Returning to our original example in fig. 2.1, we now analyze the modelling choices made, including the likelihood and prior, to better understand the inference procedures conducted.

Beginning with the likelihood, the two observations in the dataset were corrupted with *i.i.d.* Gaussian noise, and therefore the random variables  $\epsilon_1$  and  $\epsilon_2$  from eq. (2.1) are *i.i.d.* Gaussian with variance  $\sigma^2$ . The assumption of *i.i.d.* Gaussian noise is a common assumption and one that is frequently employed throughout this thesis, and this assumption alone allows

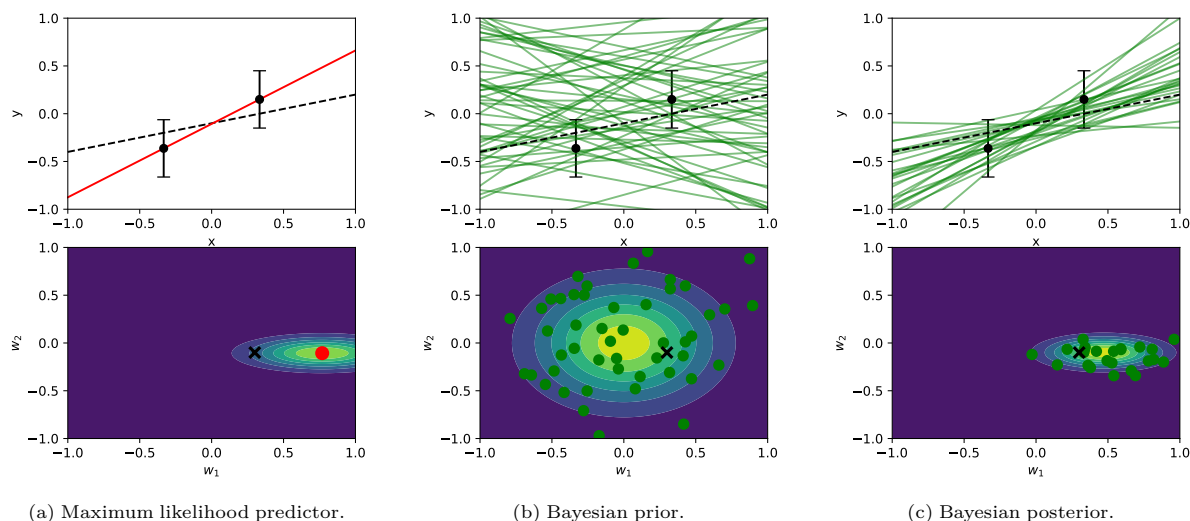


Figure 2.2: Comparison of maximum likelihood and Bayesian approaches to inference. A linear model  $f(x) = w_1x + w_2$  is employed and the top plots demonstrate inference in function space  $(x, y)$  whereas the bottom plots show inference in weight space  $(w_1, w_2)$ . The green lines and dots denote samples drawn from the function and weight space, respectively. The contour plots in the bottom row from left-to-right contain the likelihood, prior and posterior. Additionally, the black  $\times$  in the bottom row indicates the exact parameters that we would like to recover, i.e., the parameters used to draw the black dashed line the top row.

us to define our likelihood as follows:

$$\underbrace{\Pr(\mathbf{y}|\mathbf{w})}_{\text{likelihood}} = \mathcal{N}\left(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \middle| \begin{bmatrix} f(x_1; \mathbf{w}) \\ f(x_2; \mathbf{w}) \end{bmatrix}, \sigma^2 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right). \quad (2.8)$$

The likelihood of the dataset is plotted on the bottom of fig. 2.2a as a contour plot for various values of parameters  $(w_1, w_2)$ . It is evident that the red *maximum likelihood* line in the top of fig. 2.2a corresponds to the red dot in the bottom plot that maximizes the likelihood. If a point estimate is to be made, this choice does seem sensible; however, this simple example alone showcases the danger of making point estimates and its susceptibility to *overfitting*.

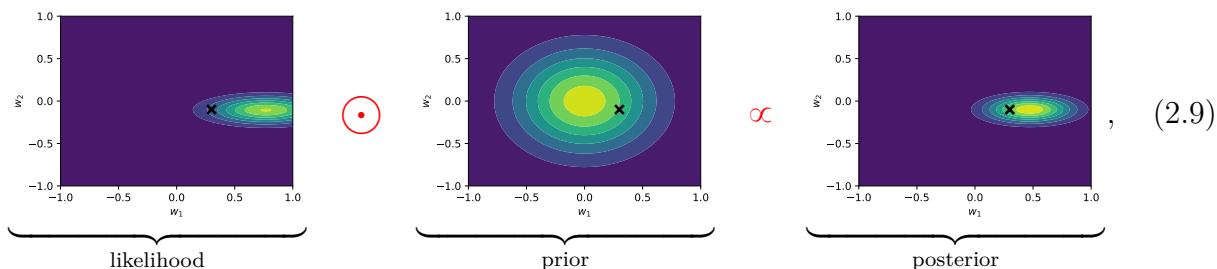
Next, we can define a prior,  $\Pr(\mathbf{w})$ . In this case we choose an *i.i.d.* Gaussian prior which is plotted on the bottom of fig. 2.2b as a contour plot. We can visualize the effect of this prior by drawing samples from it (green dots) and plotting each sample in the top plot (green lines). Evidently, this is not a particularly informative prior and indicates that we are not very aware of what the values of  $(w_1, w_2)$  should be.

Now that we have specified the likelihood and prior, the posterior can be computed. There are many advanced techniques that can be applied to compute or approximate a posterior that will be detailed later in the thesis<sup>1</sup>. For this simple example, we can consider a naive approach that can help to understand the posterior more clearly. From Bayes' rule in eq. (2.5), the posterior is evidently proportional to the product of the likelihood and prior. Visually, we

<sup>1</sup>In fact, this posterior can be computed in closed form as we will show in section 2.2.1.



can write this as



where  $\odot$  denotes an “elementwise” product that multiplies the likelihood and prior at each value of  $(w_1, w_2)$ . The relation on the left-hand side of eq. (2.9) is proportional to the posterior up to a multiplicative factor. The multiplicative factor is the inverse model evidence as given by Bayes’ rule in eq. (2.5), and this factor ensures that the posterior integrates to unity; a requirement for a valid probability distribution. Once again, we can visualize the effect of the posterior in the bottom plot of fig. 2.2c by drawing samples from it (green dots) and plotting each sample in the top plot (green lines). It can be noted that the posterior distribution is a much tighter distribution than the prior, and it is consistent with the data.

## 2.2 Gaussian Processes

The previous section discussed Bayesian learning in a general setting. We now proceed towards a particular (albeit powerful) modelling choice, Gaussian processes (GPs). Beginning with a general class of basis function models, we demonstrate how a prior in weight space ( $\mathbf{w}$ ) implies a prior in function space ( $f$ ). From there, we demonstrate an equivalent view of basis function models in terms of *kernels* and show how this perspective enables i) a powerful specification of priors directly in the function space, and ii) the use of infinitely many basis functions.

### 2.2.1 Basis Function Models

Basis function models (also known as generalized linear models) are those that can be written in the form

$$f(\mathbf{x}) = \sum_{i=1}^m w_i \phi_i(\mathbf{x}), \quad (2.10)$$

where  $\mathbf{x} \in \mathbb{R}^d$  is a  $d$ -dimensional input,  $\phi_i : \mathbb{R}^d \rightarrow \mathbb{R}$  for  $i = 1, \dots, m$  are  $d$ -dimensional basis functions, and  $\mathbf{w} \in \mathbb{R}^m$  are parameters (or weights). This form is extremely general. In fact, almost all machine learning models can be interpreted in this way from linear models to deep learning models and Gaussian processes. The one-dimensional linear example from the previous section in eq. (2.2) can be written in this form where

$$\phi_1(x) = x, \quad \text{and} \quad \phi_2 = 1. \quad (2.11)$$

In this example, the basis functions are linear; however, they can be non linear functions in general. We will proceed with the same likelihood as given in eq. (2.8) that assumes the training observations are corrupted by *i.i.d.* Gaussian noise with variance  $\sigma^2$ . This gives

$$\underbrace{\Pr(\mathbf{y}|\mathbf{w})}_{\text{likelihood}} = \mathcal{N}(\mathbf{y}|\Phi\mathbf{w}, \sigma^2\mathbf{I}_n), \quad (2.12)$$

where we have generalized our notation such that  $\mathbf{y} \in \mathbb{R}^n$  contains the observations from the dataset  $\{\mathbf{x}_i, y_i\}_{i=1}^n$  of size  $n$ ,  $\Phi \in \mathbb{R}^{n \times m}$  is a matrix whose  $i$ th column contains the evaluation of  $\phi_i$  on all  $n$  training points, and  $\mathbf{I}_n \in \mathbb{R}^{n \times n}$  is the identity matrix. As in the previous example, we will also proceed assuming a Gaussian prior on the weights  $\Pr(\mathbf{w})$  to give

$$\underbrace{\Pr(\mathbf{w})}_{\text{prior}} = \mathcal{N}(\mathbf{w}|\mathbf{0}, \mathbf{S}^{-1}), \quad (2.13)$$

where  $\mathbf{S} \in \mathbb{R}^{m \times m}$  is a symmetric positive definite precision matrix, and we have assumed a zero-prior mean for ease of exposition. Conveniently, because of the choice of the Gaussian prior, the posterior of the discussed model is also Gaussian and can be directly computed in closed form. A simple derivation of this result can be seen by writing the natural logarithm of Bayes' rule eq. (2.5) to give

$$\begin{aligned} \log\Pr(\mathbf{w}|\mathbf{y}) &= \log\Pr(\mathbf{w}) + \log\Pr(\mathbf{y}|\mathbf{w}) - \log\Pr(\mathbf{y}) \\ &= -\frac{1}{2}\mathbf{w}^T\mathbf{S}\mathbf{w} - \frac{1}{2\sigma^2}(\Phi\mathbf{w} - \mathbf{y})^T\mathbf{I}_n(\Phi\mathbf{w} - \mathbf{y}) + \text{const.}, \end{aligned}$$

where “const.” contains all the terms that do not depend on  $\mathbf{w}$  which includes normalizing terms from the likelihood and prior, as well as the entirety of the model evidence. Expanding, and completing the square gives

$$\log\Pr(\mathbf{w}|\mathbf{y}) = -\frac{1}{2}(\mathbf{w} - \boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{w} - \boldsymbol{\mu}) + \text{const.},$$

where

$$\boldsymbol{\mu} = \sigma^{-2}\boldsymbol{\Sigma}\Phi^T\mathbf{y}, \quad \text{and} \quad \boldsymbol{\Sigma}^{-1} = \sigma^{-2}\Phi^T\Phi + \mathbf{S}. \quad (2.14)$$

We can recognize this quadratic form as the log probability density of the multivariate Gaussian

$$\Pr(\mathbf{w}|\mathbf{y}) = \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma}). \quad (2.15)$$

The proceeding equation demonstrates how inference can be analytically performed on a basis function model, requiring  $\mathcal{O}(m^2n + m^3)$  time for the matrix operations involved, in general. Returning to our example, the posterior in the bottom plot of fig. 2.2c is evidently a Gaussian distribution. In this way, sampling from the posterior (the green dots) was performed using

multivariate Gaussian sampling techniques.

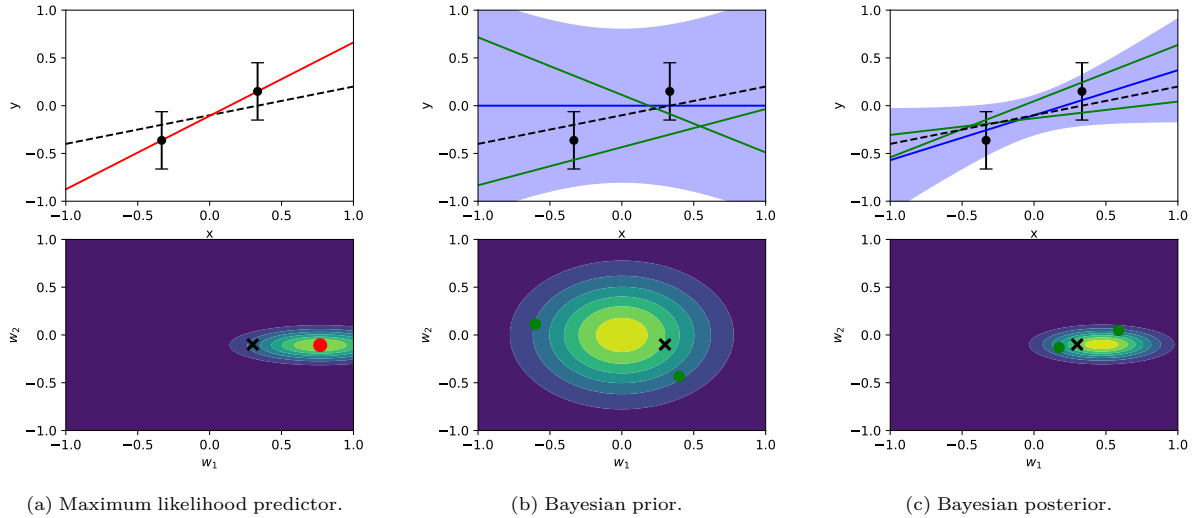


Figure 2.3: Comparison of maximum likelihood and Bayesian approaches to inference. The statistical model employed is the linear model  $y = w_1x + w_2$  and the top plots demonstrate inference in function space  $(x, y)$  whereas the bottom plots show inference in weight space  $(w_1, w_2)$ . In the top plots, the training data is shown in black which was generated by the dashed black line and corrupted by independent Gaussian noise. The parameter values of the dashed black line is shown by a black  $\times$  in the bottom plots. The maximum likelihood predictor is given in red. For the Bayesian models, the blue line denotes the posterior mean, the shaded region denotes the 95% confidence interval, and the green lines and dots denote samples from function and weight space, respectively. The contour plots in the bottom row from left-to-right contain the likelihood, prior and posterior.

### Predictive Posterior

To derive the predictive posterior, observe that the basis function model in eq. (2.10) is linear in  $\mathbf{w}$ . Observing that a linear function of Gaussian random variables is also Gaussian, we can conclude that the predictive posterior is Gaussian since the posterior  $\Pr(\mathbf{w}|\mathbf{y})$  is Gaussian. Its form is given by

$$\Pr(f(\mathbf{x}^*)|\mathbf{y}, \mathbf{x}^*) = \mathcal{N}(f(\mathbf{x}^*)|\phi(\mathbf{x}^*)^T \boldsymbol{\mu}, \phi(\mathbf{x}^*)^T \boldsymbol{\Sigma} \phi(\mathbf{x}^*)), \quad (2.16)$$

where  $\phi(\mathbf{x}^*) \in \mathbb{R}^m$  contains the evaluations of all  $m$  basis functions at  $\mathbf{x}^*$ . As a result of symmetry of the Gaussian distribution, it is not surprising that the predictive posterior mean is simply the evaluation of the basis function model (eq. (2.10)) using the posterior mean, i.e., using  $\mathbf{w} = \boldsymbol{\mu}$ . Additionally, note the quadratic form of the predictive posterior variance which shows that the predictive uncertainty grows with the magnitude of the basis functions. Using our example of linear basis functions in eq. (2.11), the uncertainty grows with the magnitude of  $x$ , as we would expect for a linear model. This can be seen in the top plot of fig. 2.3c where the shaded region shows two standard deviations from the mean. The top plot of fig. 2.3b also shows the “predictive” prior mean (blue line) and two standard deviations (shaded). This is also Gaussian (since the prior is Gaussian) and is given by

$$\Pr(f(\mathbf{x}^*)) = \mathcal{N}(f(\mathbf{x}^*)|0, \phi(\mathbf{x}^*)^T \mathbf{S}^{-1} \phi(\mathbf{x}^*)). \quad (2.17)$$

### 2.2.2 Function-space View

Equivalent relations to those derived in the previous section can be found by taking an alternative view. We call this different perspective a *function-space* view since inference is performed directly in function space without ever explicitly discussing basis functions or parameters/weights. We will see that in some scenarios this perspective will be computationally preferable to the previous approach (that we call a *weight-space* view), and it is certainly more interpretable.

To begin, consider the  $n$  noise-free function values  $\mathbf{f} = \{f(\mathbf{x}_1), \dots, f(\mathbf{x}_n)\}$  at the inputs  $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ . Extending the prior from eq. (2.17) to multiple function values, it is easy to see that the prior over  $\mathbf{f}$  is jointly Gaussian and is given by

$$\Pr(\mathbf{f}) = \mathcal{N}(\mathbf{f} | \mathbf{0}, \Phi \mathbf{S}^{-1} \Phi^T) = \mathcal{N}(\mathbf{f} | \mathbf{0}, \mathbf{K}(\mathbf{X}, \mathbf{X})), \quad (2.18)$$

where we have introduced  $\mathbf{K}(\mathbf{X}, \mathbf{X}) = \Phi \mathbf{S}^{-1} \Phi^T \in \mathbb{R}^{n \times n}$ . The matrix  $\mathbf{K}(\mathbf{X}, \mathbf{X})$  describes the prior covariance between the random variables  $\mathbf{f}$  such that

$$[\mathbf{K}(\mathbf{X}, \mathbf{X})]_{i,j} = \phi(\mathbf{x}_i)^T \mathbf{S}^{-1} \phi(\mathbf{x}_j) = \mathbb{E}[f(\mathbf{x}_i) f(\mathbf{x}_j)] = k(\mathbf{x}_i, \mathbf{x}_j), \quad (2.19)$$

where we have introduced  $k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$  which we shall refer to the prior covariance *kernel* (also called the *covariance function*). The kernel describes the prior covariance between the function values at two arbitrary points in  $d$ -dimensional input space. It is all that is required to define a zero mean Gaussian process:

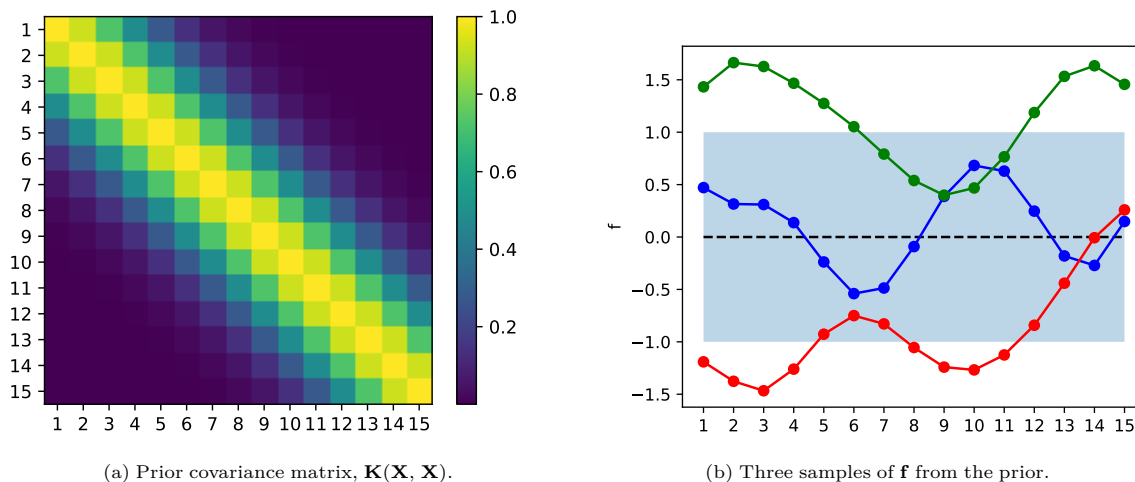
A *Gaussian process* (GP) is a collection of random variables, any finite number of which have a joint Gaussian distribution.

A Gaussian process is defined entirely by a covariance function  $k$ , and a mean function which we have assumed to be zero for the purpose of clarity<sup>2</sup>. Equation (2.18) demonstrates precisely that when using the covariance kernel  $k$ , a finite collection of ( $n$ ) observations of the target are jointly Gaussian.

#### Predictions with Noise-Free Observations

Writing the prior in eq. (2.18) using a covariance matrix constructed by the kernel  $k$  immediately suggests a simple alternative inference procedure when the observations are noise-free. Consider  $\mathbf{f} \in \mathbb{R}^n$  to be the  $n$  noise-free observations of the target, and take  $f^*$  to be the response at a test input  $\mathbf{x}^*$ . Since a finite collection of targets is assumed to have a joint Gaussian distribution in Gaussian process modelling, we can extend the collection  $\mathbf{f}$  of  $n$  random variables in eq. (2.18) to write the joint prior over training observations  $\mathbf{f}$  and the test observation  $f^*$  as the following joint Gaussian distribution whose  $(n+1) \times (n+1)$  covariance

<sup>2</sup>A mean function can be easily incorporated as well. One way to account for a non-zero prior mean is to simply consider the random variables  $\mathbf{f}$  to be realizations of a function with the mean function subtracted. Therefore, any realizations of  $\mathbf{f}$  need to have the mean function added to it before interpretation.

(a) Prior covariance matrix,  $\mathbf{K}(\mathbf{X}, \mathbf{X})$ .(b) Three samples of  $\mathbf{f}$  from the prior.Figure 2.4: Visualization of the Gaussian process covariance, and samples on a finite set of points  $\mathbf{X}$ .

matrix is formed by the kernel  $k$

$$\Pr(\mathbf{f}, f^*) = \mathcal{N} \left( \begin{bmatrix} \mathbf{f} \\ f^* \end{bmatrix} \middle| \begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{K}(\mathbf{X}, \mathbf{X}) & \mathbf{k}(\mathbf{X}, \mathbf{x}^*) \\ \mathbf{k}(\mathbf{x}^*, \mathbf{X}) & k(\mathbf{x}^*, \mathbf{x}^*) \end{bmatrix} \right). \quad (2.20)$$

To derive the posterior distribution, we need to restrict this joint prior distribution to contain only realizations that are consistent with the training observations  $\mathbf{f}$ . In probabilistic terms, this simply describes *conditioning* the joint distribution on the training observations  $\mathbf{f}$ . This can be performed using standard Gaussian identities as follows (e.g., (Rasmussen and Williams, 2006, appendix A.2)):

$$\Pr(f^* | \mathbf{f}) = \mathcal{N}(f^* | \mathbf{k}(\mathbf{x}^*, \mathbf{X}) \mathbf{K}(\mathbf{X}, \mathbf{X})^{-1} \mathbf{f}, k(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}(\mathbf{x}^*, \mathbf{X}) \mathbf{K}(\mathbf{X}, \mathbf{X})^{-1} \mathbf{k}(\mathbf{X}, \mathbf{x}^*)). \quad (2.21)$$

This is an elegant result since we were able to go from the prior directly to the posterior without explicitly considering the weight space at all. This posterior is also easy to compute, requiring only linear algebra operations, and the computations can be trivially extended to evaluate the predictive posterior on a set of test points  $\mathbf{X}^*$  by simply replacing  $\mathbf{x}^*$  with  $\mathbf{X}^*$ .

### Infinite Basis Functions

Unfortunately, the posterior in eq. (2.21) is not defined for all basis function modelling choices. This is because the prior covariance matrix  $\mathbf{K}(\mathbf{X}, \mathbf{X})$  in eq. (2.18) is a semi-positive definite matrix and may be singular such that  $\mathbf{K}(\mathbf{X}, \mathbf{X})^{-1}$  is not defined. For instance, it will be singular if  $m < n$  since the covariance is of rank at most  $m$ . One approach to help deal with this singularity is to expand the number of features  $m$ . At first glance this would appear to be an expensive solution. After all, we are considering increasing the complexity of our basis function model so it is reasonable to expect this would come with an increase in cost. When we were considering inference from a weight-space perspective, computation of the posterior in

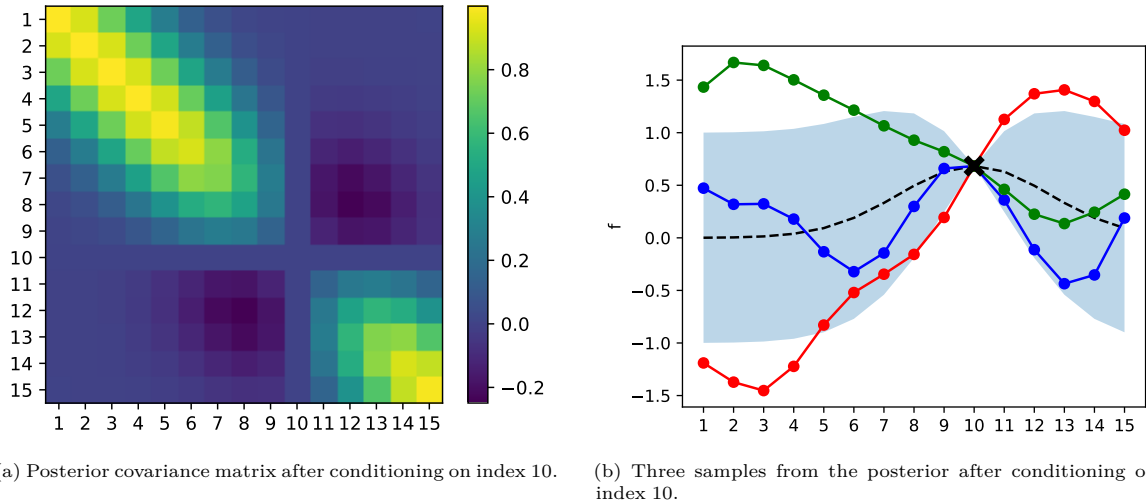


Figure 2.5: Visualization of the Gaussian process posterior, and samples on a finite set of points. A single noise-free observation is assumed.

eq. (2.15) cost  $\mathcal{O}(m^2n + m^3)$  time, giving an expensive cubic scaling in the number of features. However, we will see that by taking a function-space perspective, the cost of inference can be *independent* of the number of features if we use the kernel in a clever way.

To begin, consider that eq. (2.21) only requires evaluations of the kernel  $k$ , not the basis functions themselves. Therefore, if we can evaluate a kernel without explicitly computing the inner product between basis functions then the cost of kernel evaluations will be independent of the number of features, and so will the cost of evaluating the posterior in eq. (2.21). This is possible but we require some properties for this kernel, namely that it must admit a symmetric and positive semi-definite covariance matrix  $\mathbf{K}(\mathbf{X}, \mathbf{X})$  for any collection of points in  $\mathbb{R}^d$ . These come directly from the requirements of the covariance of a Gaussian distribution. Fortunately, such kernels exist, for example, consider the exponentiated quadratic kernel (also known as the squared exponential kernel):

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{1}{2}\|\mathbf{x}_i - \mathbf{x}_j\|_2^2\right). \quad (2.22)$$

It can be shown that the exponentiated quadratic kernel corresponds to the inner product of an infinite number of basis functions, i.e.,  $m = \infty$  when using this kernel. For example, we can obtain the exponentiated quadratic kernel as the inner product of an infinite number of Gaussian-shaped basis functions (see (Rasmussen and Williams, 2006, sec. 4.2.1)). This is a remarkable property: by using such a kernel, we can expand the capacity of our model from one using a finite number of basis functions to one using an infinite number of basis functions without incurring any additional cost.

Let us now try to get a better grasp of the Gaussian process prior defined by the covariance kernel  $k$ . Figure 2.4a shows the prior covariance matrix using the exponentiated quadratic kernel in eq. (2.22). The matrix shows the covariance between 15 points  $\mathbf{X}$  in  $d = 1$  dimension such that  $\mathbf{x}_i = i$ , i.e., the value of  $\mathbf{x}$  is the same as its index. By observing fig. 2.4a it is immediately

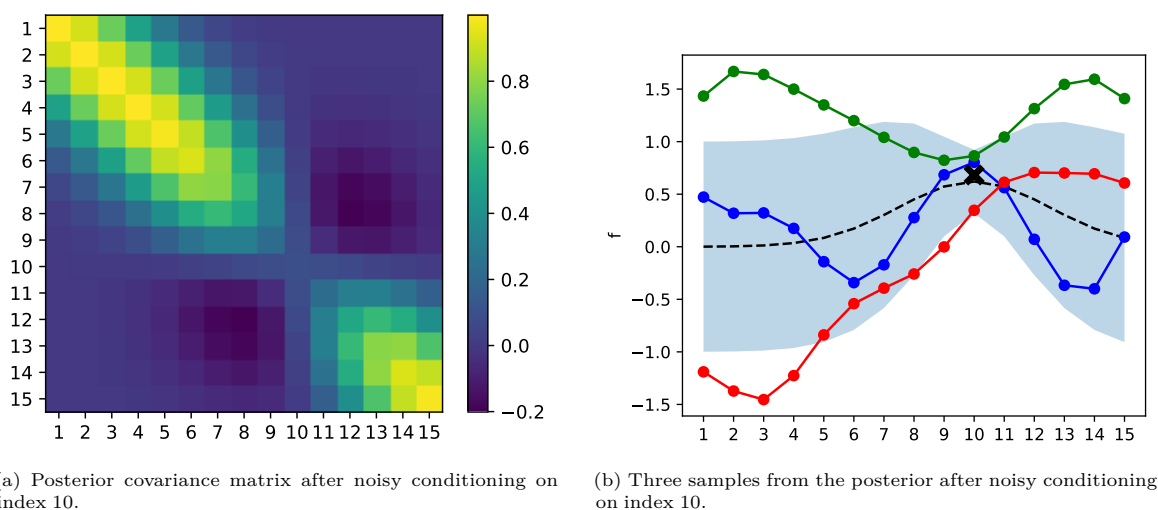


Figure 2.6: Visualization of the Gaussian process posterior, and samples on a finite set of points. An observation that is corrupted with additive Gaussian noise is assumed.

evident that points closer together have higher covariance. For example,  $k(\mathbf{x}_1=1, \mathbf{x}_2=2)$  at index (1, 2) has greater covariance than  $k(\mathbf{x}_1=1, \mathbf{x}_3=3)$  at index (1, 3). It seems sensible that locations far from one another should have lesser covariance, this is one of the properties of the exponentiated quadratic kernel. The three coloured curves in fig. 2.4b show three samples drawn from the zero-mean Gaussian distribution given in eq. (2.18) with the covariance matrix given in fig. 2.4a, and lines were drawn between the 15 points in  $\mathbf{X}$  to aid visualization. Since the set  $\mathbf{X}$  is arbitrary, the procedure used to create the samples in fig. 2.4b could be extended by expanding the set  $\mathbf{X}$  to contain infinitely many points in  $\mathbb{R}$ , giving a distribution over functions.

While fig. 2.4 visualized the GP prior, fig. 2.5 visualizes the GP posterior after the prior has been conditioned on a single observation at  $\mathbf{x}_{10} = 10$  given by the black  $\times$  in fig. 2.5b. Once again, the three coloured curves in fig. 2.5b show three samples, but this time they are drawn from the posterior Gaussian distribution given in eq. (2.21) with the covariance matrix given in fig. 2.5a and the mean given by the dashed black line in fig. 2.5b. Also shown in fig. 2.5b, is the shaded region that shows one standard deviation (square-root of the diagonal of fig. 2.5a) from posterior mean. Evidently the posterior variance vanishes at  $\mathbf{x}_{10}$  as would be expected since we are absolutely certain about the value of the function at this point. Additionally, it can be seen that the posterior returns to the prior as we move away from  $\mathbf{x}_{10}$ , indicating that we know little about the function values far from this point. One of the beautiful features of Gaussian processes is that the predictive posterior statistical moments are given in closed form. This allows easy interpretation without needing to sample the posterior since the posterior mean (dashed black line) gives the expectation of the prediction as a point estimate, while the posterior variance (visualized by the shaded region) gives the posterior uncertainty which gives some measure of confidence in a prediction. Lastly, note that it is trivial to extend these computations to multidimensional inputs by simply changing the evaluation of the covariance kernel in accordance with eq. (2.22).

### Predictions with Noisy Observations

Extending the number of features  $m$  to infinity helped improve the rank of  $\mathbf{K}(\mathbf{X}, \mathbf{X})$ ; however, it still does not guarantee that the matrix will be full rank. As an example, consider the scenario where two input points with indices  $i$  and  $j$  are identical such that  $\mathbf{x}_i = \mathbf{x}_j$ . In this case the  $i$ th row (or column) of  $\mathbf{K}(\mathbf{X}, \mathbf{X})$  will be identical to the  $j$ th row (or column) and the covariance will be rank deficient no matter what positive semi-definite kernel is used. A definitive way to deal with such singularities is to assume all observations are corrupted by additive independent Gaussian noise. Although other forms of noise are certainly possible, additive *i.i.d.* Gaussian noise is a common assumption and one that is made throughout this thesis. In fact, it is typical that we do not have access to the function values themselves, but noisy versions thereof (as we had assumed in eq. (2.1) in the original example of this chapter).

Additive *i.i.d.* Gaussian noise with variance  $\sigma^2$  gives the following prior over the training observations

$$\Pr(\mathbf{y}) = \mathcal{N}(\mathbf{y} | \mathbf{0}, \mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I}_n), \quad (2.23)$$

whose covariance differs from the prior over  $\mathbf{f}$  (in eq. (2.18)) by the addition of a diagonal matrix. We can then modify eq. (2.20) to give the joint prior over noisy training observations and a test point as follows:

$$\Pr(\mathbf{y}, f^*) = \mathcal{N} \left( \begin{bmatrix} \mathbf{y} \\ f^* \end{bmatrix} \middle| \begin{bmatrix} \mathbf{0} \\ 0 \end{bmatrix}, \begin{bmatrix} \mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I}_n & \mathbf{k}(\mathbf{X}, \mathbf{x}^*) \\ \mathbf{k}(\mathbf{x}^*, \mathbf{X}) & k(\mathbf{x}^*, \mathbf{x}^*) \end{bmatrix} \right). \quad (2.24)$$

We can now follow the Gaussian conditioning expression in eq. (2.21) to condition the joint in eq. (2.24) on the noisy observations  $\mathbf{y}$  as follows:

$$\begin{aligned} \Pr(f^* | \mathbf{y}) &= \mathcal{N}(f^* | \mathbb{E}[f^*], \text{cov}[f^*]), \quad \text{where} \\ \mathbb{E}[f^*] &= \mathbf{k}(\mathbf{x}^*, \mathbf{X}) \left( \mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I}_n \right)^{-1} \mathbf{f}, \quad \text{and} \\ \text{cov}[f^*] &= k(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}(\mathbf{x}^*, \mathbf{X}) \left( \mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma^2 \mathbf{I}_n \right)^{-1} \mathbf{k}(\mathbf{X}, \mathbf{x}^*). \end{aligned} \quad (2.25)$$

The preceding equation describes the key predictive equations for Gaussian process regression<sup>3</sup>. Also, it can be shown that the predictive posterior in the preceding equation is identical to the predictive posterior we had derived from a weight-space approach in eq. (2.17) provided we use the kernel  $k(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\phi}(\mathbf{x}_i)^T \mathbf{S}^{-1} \boldsymbol{\phi}(\mathbf{x}_j)$  from eq. (2.19). This observation means that we arrive at the same predictive posterior if we take a weight-space or a function-space perspective, although the two approaches have differing computational complexities. Based upon the algebraic operations in their respective relations, computation of the predictive

<sup>3</sup>For the outline of a stable algorithmic implementation of Gaussian process predictions, please see (Rasmussen and Williams, 2006, alg. 2.1).



posterior generally scales as follows for the two approaches.

Perspective	Time	Storage
Weight-Space (eq. (2.16))	$\mathcal{O}(nm^2 + m^3)$	$\mathcal{O}(nm + m^2)$
Function-Space (eq. (2.25))	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$

(2.26)

It is therefore evident that the weight-space view is generally preferred when the number of basis functions ( $m$ ) is less than the number of observations ( $n$ ). When the training dataset size  $n$  is not prohibitively large, the function-space perspective is generally preferred since it allows a potentially infinite number of features ( $m \rightarrow \infty$ ) with no additional cost, and it allows a wealth of interpretable kernels to be used for specification of the prior (as we will discuss in the following section).

In fig. 2.6, we extend the visualization of fig. 2.5 by again conditioning the prior Gaussian distribution on index  $\mathbf{x}_{10}$  but this time we assume the observation is corrupted by additive Gaussian noise with variance  $\sigma^2 = 0.1$ . In contrast to the example of fig. 2.5, we see that the variance does not vanish at  $\mathbf{x}_{10}$  since we are not completely certain about the value of the function at this point because of the noise in the observation.

### 2.2.3 Covariance Kernels

Here we discuss how the choice of covariance kernel  $k$  affects Gaussian process inference. Specifically (and quite simply), choosing a kernel is synonymous with selecting a Gaussian process prior. This connection is clear when the zero-mean GP prior eq. (2.23) is inspected, since the kernel  $k$  is the only element we have control over in this equation. Kernels offer an elegant and easily interpretable way to specify priors, allowing practitioners to incorporate high-level domain knowledge about a learning problem such as stationarity, differentiability, periodicity, scale, expected change over a distance, and can even enforce complicated linear operator constraints. Kernel selection is an important topic for effective inference with Gaussian processes and this section is intended as an introduction to this field. The interested reader is referred to (Rasmussen and Williams, 2006, chapter 4) for a thorough overview.

**Flexibility** To begin, it is important to consider the dimension  $m$  of the feature expansion of a covariance kernel since this does affect the flexibility of a Gaussian process model. These features  $\phi: \mathbb{R}^d \rightarrow \mathbb{R}^m$  can be seen in eq. (2.19) and we have considered cases where  $m$  is finite, as well as infinite. In the case where the kernel can be represented exactly by a finite basis function expansion (i.e., a finite  $m$ ), the resulting Gaussian process will have a finite capacity to model observations. We call a kernel with a finite basis function expansion *degenerate*. For example, consider the example in fig. 2.3 where linear basis functions were employed. Clearly the resultant Gaussian process does not have the flexibility to model an arbitrary non-linear function no matter how many observations are provided. Choosing a kernel with these basis functions are a

Matérn-5/2	$k(r) = (1 + \sqrt{5}r + \frac{5}{3}r^2)\exp(-\sqrt{5}r)$	Twice differentiable
Matérn-3/2	$k(r) = (1 + \sqrt{3}r)\exp(-\sqrt{3}r)$	Once differentiable
Matérn-1/2	$k(r) = \exp(-r)$	Non-differentiable

Table 2.1: Popular Matérn kernels. The shorthand  $r = \|\mathbf{x}_i - \mathbf{x}_j\|_2$  was used where  $\mathbf{x}_i, \mathbf{x}_j$  are the kernel inputs. The differentiability statements refer to mean-square differentiability of the Gaussian process that uses the respective covariance kernel. All kernels listed admit a mean-square continuous Gaussian process.

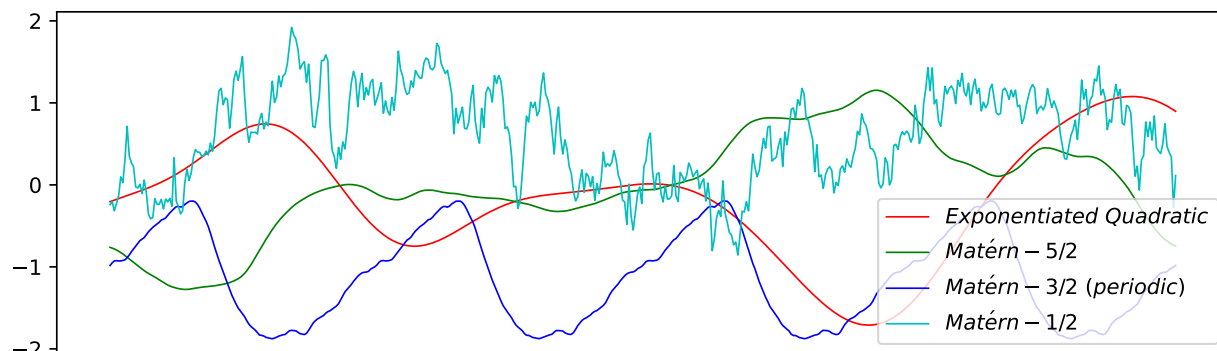


Figure 2.7: Realizations of zero-mean Gaussian process priors using various Matérn kernels.

good choice if we know the resultant function is linear *a priori*, but it will clearly be a poor choice if there is a possibility the function is non-linear. Conversely, many kernels used in practice have  $m = \infty$  such that the Gaussian process has the capacity to model increasingly complicated functions as more data arrives. Such a model is considered *non-parametric*. In addition, many kernels with an infinite basis function expansion admit a GP that will be *universally consistent*, meaning that any function can be approximated to arbitrary precision. The exponentiated quadratic kernel (eq. (2.28)) is one such example. Universal consistency is a remarkable property which implies that the Gaussian process prior has support over the space of all functions and therefore will be able to recover the true underlying function in the limit of infinite data, even if the initial prior is poorly specified. We will see throughout this thesis that in practice a trade-off occasionally needs to be made between flexibility and computational complexity.

**Differentiability** The kernel can be chosen to admit a Gaussian process that has a specified level of smoothness. For example, the Matérn class of kernels can be used to specify Gaussian processes with varying levels of differentiability. Table 2.1 specifies several popular Matérn kernels that are zero-, one- and two-times mean-squared differentiable. The exponentiated quadratic kernel (eq. (2.22)) is also in the Matérn family, being infinitely differentiable. Realizations of Gaussian processes using these kernels are shown in fig. 2.7. It is evident that the function behaviour varies dramatically based on the level of smoothness and therefore if a given level of differentiability is known *a priori*, this is powerful information that can be used to select an appropriate kernel to restrict the class of functions under the prior appropriately.

**Stationarity** A covariance kernel  $k(\mathbf{x}_i, \mathbf{x}_j)$  is stationary if it can be written as a function of  $\mathbf{x}_i - \mathbf{x}_j$ . Stationarity means that the covariance between two points in the input space depends only on the difference between the two points in input space and does not depend on the absolute location of the points. This has the effect of assuming the Gaussian process prior behaves similarly throughout the input space, which is often a logical presumption. For example, the exponentiated quadratic kernel in eq. (2.22) and the Matérn kernels in table 2.1 are all stationary. If desired, a stationary kernel can be made non-stationary using a non-linear input warping to give the following modified kernel

$$k(\mathbf{g}(\mathbf{x}_i), \mathbf{g}(\mathbf{x}_j)), \quad (2.27)$$

where  $\mathbf{g} : \mathbb{R}^d \rightarrow \mathbb{R}^p$  is an arbitrary non-linear function, and the number of outputs  $p \geq 1$  can also be arbitrary. For example, Snoek et al. (2014) introduce a simple non-linear warping that can account for non-stationarity.

**Periodicity** Function periodicity can also be modelled through an appropriately chosen covariance kernel. A periodic prior can be employed using the warping function  $\mathbf{g}(x) = [\cos(x), \sin(x)]^T$  for  $d = 1$  dimensional inputs and applied as described in eq. (2.27). This periodic warping is applied to the Matérn-3/2 kernel in fig. 2.7 where the realization exhibits periodic behaviour, as expected.

**Variance & Lengthscale** To account for functions that have differing observation magnitudes and input scales, modifications can be made to all discussed kernels. As an example, we will re-write the exponentiated quadratic kernel originally given in eq. (2.22) to introduce additional hyperparameters as follows:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_0^2 \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{\Lambda}^{-1}(\mathbf{x}_i - \mathbf{x}_j)\right), \quad (2.28)$$

where  $\sigma_0^2 > 0$  is the kernel variance, and  $\mathbf{\Lambda} \in \mathbb{R}^{d \times d}$  is a symmetric positive definite matrix describing the kernel lengthscale. The kernel variance describes the magnitude of the function values of the Gaussian process prior. Quite simply, if  $\sigma_0$  is doubled, the vertical magnitude of the realizations in fig. 2.7 would double.

The kernel lengthscale  $\mathbf{\Lambda}$  describes the rate at which the function is expected to change with respect to a change in input space. In the simplest case, if  $\mathbf{\Lambda} = \ell^2 \mathbf{I}_d$  then the kernel is commonly called an isotropic kernel since it is simply a function of the radius  $\|\mathbf{x}_i - \mathbf{x}_j\|_2$  from either of the input points. In this case, the parameter  $\ell > 0$  describes how “sharp” the realizations will be (a smaller  $\ell$  value gives sharper functions). As another interpretation, this prior states that you would not be able to extrapolate more than  $\mathcal{O}(\ell)$  units from your data. Figures 2.8a and 2.8b plot a sample from a two-dimensional Gaussian process prior using the isotropic exponentiated quadratic kernel with two different values of  $\ell$ . The lengthscale can

also be visualized by the black curve in each plot where the radius of the black curve from the black dot indicates the lengthscale in that respective direction. Specifically, the black curves show a contour of equal prior covariance with the function value at the black dot. It can be seen that the function realization with the smaller  $\ell$  in fig. 2.8b is more “wiggly” and changes value more rapidly with respect to the input coordinates.

If  $\mathbf{\Lambda} = \text{diag}[\ell_1^2, \dots, \ell_d^2]$  then the kernel has axis-aligned lengthscales and is commonly referred to as the ARD (automatic relevance determination) exponentiated quadratic kernel because it can prune irrelevant input dimensions by growing the corresponding lengthscales. Plotted in fig. 2.8c is a realization of a two-dimensional Gaussian process prior using an ARD exponentiated quadratic kernel where it can be seen that there are different lengthscales along the coordinate axes. Along the first input dimension  $x_1$ , the lengthscale  $\ell_1$  is large and the function values vary slowly. If we take  $\ell_1 \rightarrow \infty$  the realization in fig. 2.8c would not vary at all along  $x_1$  and the effect of this input dimension would be entirely eliminated.

A non-diagonal  $\mathbf{\Lambda}$  matrix can be seen as an application of the ARD exponentiated quadratic kernel applied after a rotation of the input space coordinate axes. Figure 2.8d plots a two-dimensional sample drawn from a GP prior with a dense  $\mathbf{\Lambda}$  matrix. It can be seen that the black ellipse indicating the lengthscale in fig. 2.8d is rotated with respect to the ellipse in fig. 2.8c such that its principal axes are no longer aligned with the input coordinate axes. To apply lengthscales to the Matérn kernels in table 2.1, substitute  $r = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{\Lambda}^{-1} (\mathbf{x}_i - \mathbf{x}_j)}$ . This is simply a replacement of Euclidean distance between inputs with the Mahalanobis distance.

**Neural Tangent Kernel** While not necessarily a popular kernel, the neural tangent kernel is an interesting covariance function that demonstrates the power and generality of Gaussian processes. It has been shown that deep, infinitely wide Bayesian neural networks are Gaussian processes under certain assumptions (G. Matthews et al., 2018; Lee et al., 2018; Neal, 1995). Further, Jacot et al. (2018) showed that during training by gradient descent, the same network converges to the so-called neural tangent kernel. This is a fascinating result which has led to many interesting and practical developments. For instance, it has allowed exact Bayesian inference to be performed on deep and infinitely wide neural networks, even when complex architectures are considered such as convolutional neural networks with global average pooling (Arora et al., 2019).

## 2.3 Model Selection & Model Evidence

Since the beginning of this chapter, we have assumed that a single model is selected *a priori* and then inference is performed. Unfortunately, in many scenarios, a practitioner may not have enough insight into a problem to specify a single good model for a learning problem. As an example, a practitioner may not know *a priori* which of two Gaussian process priors will perform better on a given learning problem. This section will discuss how to deal with

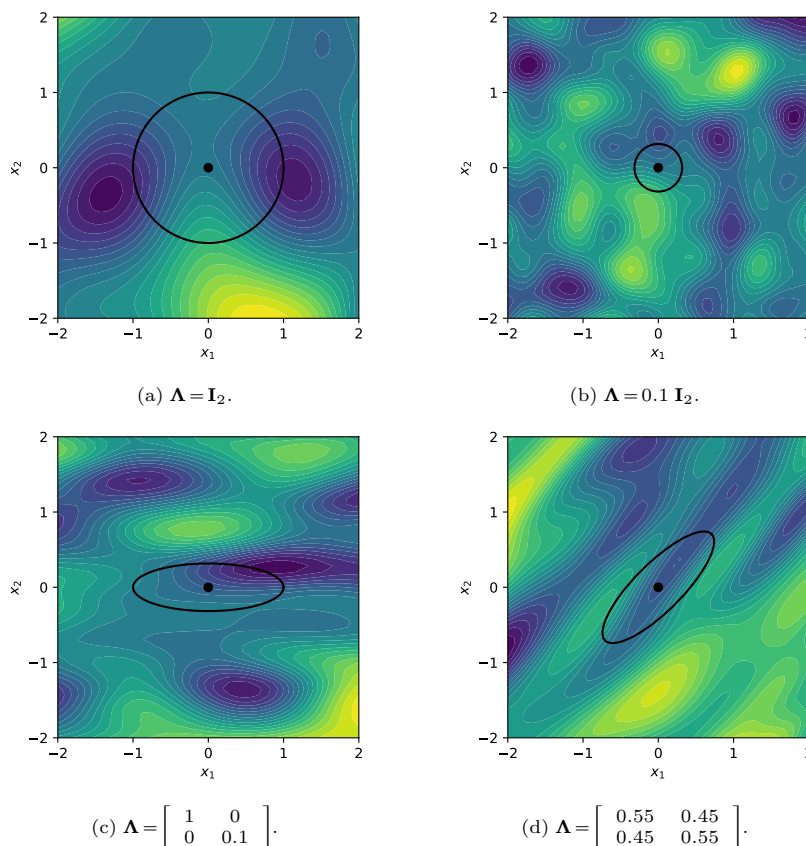


Figure 2.8: Samples from a two-dimensional Gaussian process prior using the exponentiated quadratic kernel in eq. (2.28) with various values of  $\Lambda$ . The black curves show a contour of equal prior covariance with the function value at the black dot. The radius of the black curve from the black dot indicates the lengthscale in that respective direction.

uncertainty over candidate models.

We begin by posing the model selection problem more concretely. We assume that a model is defined by the vector  $\theta$  such that all candidate models can be determined by a specific value of  $\theta$ . We refer to the vector  $\theta$  as a set of *hyperparameters*. As an example, the set of hyperparameters of a Gaussian process prior might include the kernel variance  $\sigma_0^2$  and lengthscale  $\Lambda$  of the exponentiated quadratic kernel in eq. (2.28), as well as the training observation noise variance  $\sigma^2$ . The model selection problem simply involves selection of the models that perform best out of the candidates within the space of  $\theta$ .

### 2.3.1 Model Evidence

The marginal likelihood or model evidence presented in eq. (2.6) is instrumental in Bayesian model selection. We will therefore begin by describing how the model evidence is computed for Gaussian processes, as discussed in section 2.2. To begin, we will update our notation such that  $\Pr(\mathbf{y}|\theta)$  denotes the model evidence for the model with hyperparameters  $\theta$ .

While evaluation of the model evidence is generally intractable for an arbitrary Bayesian model, in the case of a Gaussian process it can be performed analytically which is a tremendous advantage for the purposes of inference and model selection. Specifically, evaluating the

marginal likelihood, of a Gaussian process simply involves evaluating the GP prior on the training dataset. By the definition of a Gaussian process, the GP prior evaluated on the training dataset is a Gaussian distribution (given in eq. (2.23)) and therefore evaluating the model evidence simply involves evaluating a multivariate Gaussian distribution with dimension  $n$ . To evaluate this multivariate Gaussian, there are two computational approaches that one may wish to take, a weight-space approach, and a function-space approach. While the two approaches are mathematically equivalent, the computational complexities differ in the same manner as the predictive posterior computations, whose computational complexities are summarized in eq. (2.26).

**Weight-Space Approach** From the weight-space approach of section 2.2.1, the log of the model evidence can be computed as

$$\log\Pr(\mathbf{y}|\boldsymbol{\theta}) = -\frac{n}{2}\log(2\pi) - \frac{n}{2}\log(\sigma^2) + \frac{1}{2}\log(|\mathbf{S}|) + \frac{1}{2}\log(|\boldsymbol{\Sigma}|) - \frac{1}{2\sigma^2}\mathbf{y}^T\mathbf{y} + \frac{1}{2}\boldsymbol{\mu}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}, \quad (2.29)$$

where  $\boldsymbol{\mu}$  and  $\boldsymbol{\Sigma}$  are given in eq. (2.14), and  $|\mathbf{A}|$  denotes the determinant of a square matrix  $\mathbf{A}$ . For computational reasons, the weight-space approach is generally preferred when  $n > m$ .

**Function-Space Approach** From the function-space approach of section 2.2.2, the log of the model evidence can be computed as

$$\log\Pr(\mathbf{y}|\boldsymbol{\theta}) = \underbrace{-\frac{1}{2}\log|\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma^2\mathbf{I}_n|}_{\text{Complexity}} - \underbrace{\frac{1}{2}\mathbf{y}^T(\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma^2\mathbf{I}_n)^{-1}\mathbf{y}}_{\text{Data Fit}} - \underbrace{\frac{n}{2}\log(2\pi)}_{\text{Normalization}}. \quad (2.30)$$

For computational reasons, the function-space approach is generally preferred when  $n < m$ . The terms in the preceding equations have been labelled for reference in later discussions.

### 2.3.2 Type-I Inference

We can now begin discussing how the model selection problem can be addressed by introducing a Bayesian approach that allows us to move from a prior over models to a posterior over models. This is effectively an inference procedure over hyperparameters,  $\boldsymbol{\theta}$ , rather than an inference procedure over parameters,  $\mathbf{w}$ , described in the earlier sections of this chapter. In this way, it is effectively a meta-inference procedure. Applying Bayes' rule (eq. (2.5)) at the level of hyperparameters gives the posterior over  $\boldsymbol{\theta}$  as follows:

$$\Pr(\boldsymbol{\theta}|\mathbf{y}) = \frac{\Pr(\mathbf{y}|\boldsymbol{\theta}) \Pr(\boldsymbol{\theta})}{\Pr(\mathbf{y})}. \quad (2.31)$$

The distribution  $\Pr(\boldsymbol{\theta})$  is referred to as the *hyper-prior* and is a prior over models that reflects a practitioner's prior belief about which model is best.  $\Pr(\mathbf{y}|\boldsymbol{\theta})$  may be recognized as the marginal likelihood (eq. (2.6)) of a model with hyperparameters  $\boldsymbol{\theta}$ . Lastly, the marginal in

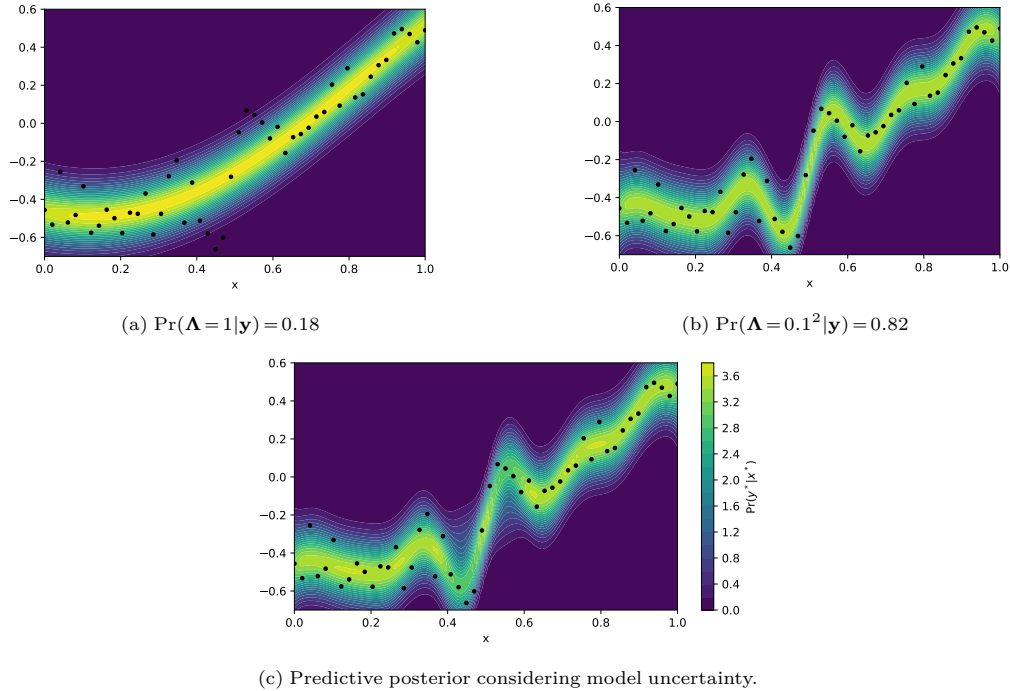


Figure 2.9: Demonstration of posterior computation over models (type-I inference). Both models use a Gaussian process prior given by an exponentiated quadratic covariance kernel in eq. (2.28) with  $\sigma_0 = 1$  and with varying values of lengthscale  $\Lambda$ . Also, training observations (black) are corrupted with *i.i.d.* Gaussian noise with variance  $\sigma^2 = 0.1^2$  for both models. The background contours show the predictive posterior probability (the colours use the same scale for all plots). Note that the predictive posterior is plotting  $\Pr(y^*|\mathbf{x}^*)$  rather than  $\Pr(f^*|\mathbf{x}^*)$  which takes into account the independent Gaussian noise corrupting the input data. The hyper-prior over both models are equivalent such that  $\Pr(\Lambda=1) = \Pr(\Lambda=0.1^2) = 0.5$ .

the denominator can be computed as

$$\Pr(\mathbf{y}) = \int \Pr(\mathbf{y}|\boldsymbol{\theta})\Pr(\boldsymbol{\theta})d\boldsymbol{\theta}. \quad (2.32)$$

Figure 2.9 shows an example computation of the model selection posterior for two Gaussian process models where  $\boldsymbol{\theta} = \Lambda$ , the lengthscale of the exponentiated quadratic kernel in eq. (2.28) for a  $d=1$  dimensional learning problem. For both models considered, the hyper-prior  $\Pr(\Lambda)$  is equivalent (both are 0.5), and the posterior probability mass is given under each plot. Each model has a different interpretation of the data with the long lengthscale model in fig. 2.9a seeing a smooth curve with several outliers in the middle of the plot whereas the shorter lengthscale model in fig. 2.9b sees a large vertical wave in the middle of the plot. The shorter lengthscale model has greater posterior probability; however, both models have reasonable mass under the posterior indicating that there is still uncertainty about which model is preferred.

When making predictions, we would like to take into account our uncertainty over models. Applying the same rules of probability, we can write the predictive distribution over the function  $f^*$  at test input  $\mathbf{x}^*$  as follows:

$$\Pr(f^*|\mathbf{y}) = \int \Pr(f^*|\mathbf{y}, \boldsymbol{\theta})\Pr(\boldsymbol{\theta}|\mathbf{y})d\boldsymbol{\theta}, \quad (2.33)$$

where  $\Pr(f^*|\mathbf{y}, \boldsymbol{\theta})$  is the predictive posterior of a model with hyperparameters  $\boldsymbol{\theta}$  that is given

by eq. (2.7) in general (in the case of Gaussian processes, it is given by eq. (2.16) or eq. (2.25)). Ultimately, Bayesian inference is conducted at two levels simultaneously: inference over parameters and inference over models (hyperparameters). Returning to the example in fig. 2.9, the predictive posterior that accounts for model uncertainty is a sum of the two models weighted by the posterior over models, as follows:

$$\Pr(f^*|\mathbf{y}) = 0.18 \Pr(f^*|\mathbf{y}, \Lambda = 1) + 0.82 \Pr(f^*|\mathbf{y}, \Lambda = 0.1^2),$$

where the predictive posteriors of each model are shown in figs. 2.9a and 2.9b, respectively, and the predictive posterior that accounts for model uncertainty is shown in fig. 2.9c. Note that this posterior is no longer Gaussian but is instead a mixture of Gaussian distributions.

In general, the posterior in eq. (2.31) cannot be tractably computed in closed form and must be estimated numerically or approximated. A common approach for numerical estimation of the posterior can be achieved through the use of Markov chain Monte Carlo (MCMC) techniques which are applied in chapter 5. The following section discusses a particular simplification of the Bayesian approach discussed here. To distinguish the two strategies, the Bayesian approach outlined here is commonly referred to as *type-I* inference. For interested readers, further details about the type-I inference procedure can be found in (Neal, 1995).

### 2.3.3 Type-II Empirical Bayes

This section discusses a simplification of the type-I Bayesian model selection problem outlined in the previous section. The simplified approach is commonly referred to as *type-II* inference, or *empirical Bayes*. Quite simply, in a type-II inference approach a single model is selected that maximizes the model evidence. This can be written as follows:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \Pr(\mathbf{y}|\boldsymbol{\theta}), \quad (2.34)$$

where  $\boldsymbol{\theta}^*$  describes the selected model. Under certain conditions, the posterior eq. (2.31) will be tightly peaked around  $\boldsymbol{\theta}^*$  and type-II inference can be seen as an approximation of type-I inference. In the presence of abundant data and relatively few hyperparameters, this approximation can be quite good<sup>4</sup>.

The model evidence is an attractive objective for hyperparameter estimation since it naturally balances model flexibility with the model’s ability to fit the dataset, admitting a Bayesian interpretation of Occam’s razor. This trade-off can be seen by observing the terms in eq. (2.30), each of which have a clear interpretation. The term labelled “complexity” depends only on the data inputs  $\mathbf{X}$  and penalizes high model flexibility<sup>5</sup>. The term labelled “data fit” is the only

<sup>4</sup>Note that in contrast, making this same maximum likelihood approximation to the posterior  $\Pr(\mathbf{w}|\mathbf{y})$  over the parameters  $\mathbf{w}$  will often give very poor results since the number of parameters  $m$  is typically very large and possibly infinite.

<sup>5</sup>Model flexibility may be difficult to envision for a non-parametric model. Specifically, the complexity term penalizes a slowly decaying eigenspectrum of the covariance matrix which generally occurs with smaller kernel lengthscales. Therefore, a kernel with a large lengthscales (admitting smoother functions) is favoured over a kernel with a small lengthscales (i.e., admitting sharper functions) under this penalty. Also influencing the complexity is the scale of the eigenvalues which relates the kernel variance



term containing the training responses  $\mathbf{y}$  and reflects how well the responses are modelled by the marginal Gaussian distribution. The final term is simply a normalization constant and depends on neither the training set or the hyperparameters. Gradient-based optimization is commonly used to maximize the model evidence when the hyperparameters  $\boldsymbol{\theta}$  are continuous but it should be considered that the model evidence may have multiple maxima in  $\boldsymbol{\theta}$ .

We can illustrate the Bayesian interpretation of Occam’s razor through a simple example of selecting a Gaussian process prior on a dataset with a single training instance. Specifically, we will consider the exponentiated quadratic kernel in eq. (2.28) with various values of kernel variance  $\sigma_0$ , and we will assume that the single training observation  $y = 1$  is noise-free (such that  $f = y$ ). Figure 2.10a plots the model evidence  $\Pr(y'|\sigma_0)$  for three different values of  $\sigma_0$  across a range of possible observation values  $y'$ . It is easily seen that of the three curves, the GP prior given by  $\sigma_0 = 1$  provides the maximal model evidence at the true observation value of  $y' = y = 1$  indicated by the grey vertical line. Figure 2.10b shows the breakdown of the log-evidence at the true training observation,  $\log\Pr(y|\sigma_0)$ , based upon the decomposition given in eq. (2.30). It can be seen that the choice of  $\sigma_0 = 1$  is a trade-off between data fit and complexity since of the three  $\sigma_0$  choices,  $\sigma_0 = 1.5$  provides the best data fit,  $\sigma_0 = 0.5$  is the least complex, and  $\sigma_0 = 1$  is in between on both data-fit and complexity. To understand what complexity means, observe in fig. 2.10a that the most complicated prior with  $\sigma_0 = 1.5$  has the ability to represent a far greater range of possible observation values  $y'$  than the simplest model  $\sigma_0 = 0.5$ .

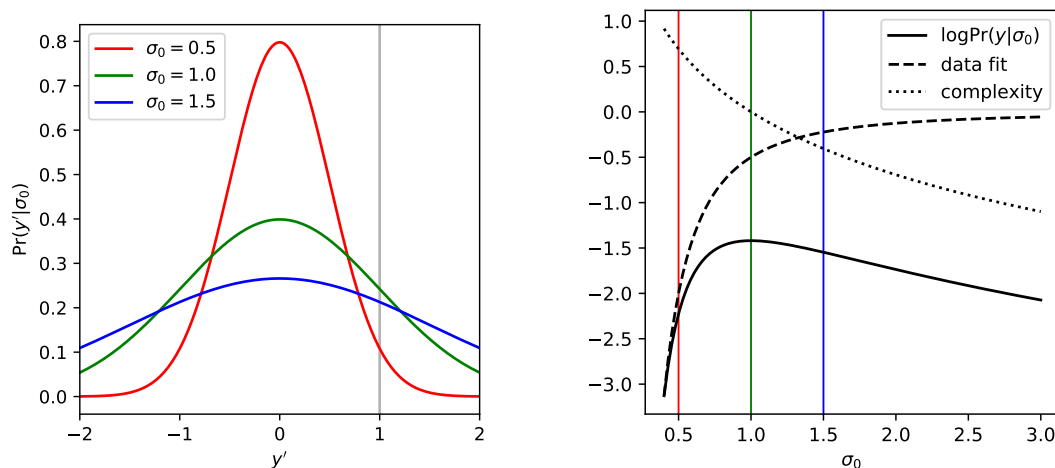
While empirical Bayes (evidence maximization) can be an effective means of model selection in many instances, it should be used cautiously since it can suffer from several of potential issues:

- **Type-II inference underestimates uncertainty.** This is not surprising since type-II inference is effectively ignoring uncertainty over models whereas a type-I approach takes this uncertainty into account. This effect can be seen in the example of fig. 2.9 where a shorter lengthscale of  $\boldsymbol{\Lambda} = 0.1^2$  would have been selected out of the two options considered from type-II approach. The type-II predictive posterior is therefore shown in fig. 2.9b where it is clear that the predictive posterior underestimates uncertainty around  $x = 0.5$  relative to the type-I predictive posterior shown in fig. 2.9c.
- **Type-II inference is not immune from overfitting.** Cases where many hyperparameters are being estimated by evidence maximization are liable to overfit, for instance. As an example, in fig. 2.10a if we were estimating the GP prior mean in addition to the variance  $\sigma_0$ , the maximum evidence would occur at a delta spike about  $y = 1$ , i.e.,  $\Pr(y'|\boldsymbol{\theta}) = \delta(y' - 1)$ . This pathological GP prior which would give infinite evidence but would clearly be a silly model to use given a single training observation.

In general, empirical Bayes is safe from these concerns when the number of hyperparameters is far less than then number of training observations (i.e., when  $|\boldsymbol{\theta}| \ll n$ ).

---

and noise variance (smaller variances are favoured).



(a) Evidence plotted over a range of possible observation values  $y'$  for three values of hyperparameter  $\sigma_0$ .

(b) Model evidence versus hyperparameter  $\sigma_0$ . The breakdown refers to the terms in eq. (2.30).

Figure 2.10: Comparison of three Gaussian process priors on a dataset with the single observation  $y = 1$ . The plots illustrate a Bayesian interpretation of Occam’s razor for model selection by maximization of model evidence.

## 2.4 Gaussian Process Approximations

This section considers approximate Bayesian inference on Gaussian processes. This may seem bizarre to some, since we have shown that Gaussian processes are a rare instance of Bayesian inference where computations can be performed analytically, so why would we want to approximate? One reason is that performing exact Gaussian process inference on massive datasets can be prohibitively expensive. Another reason is that for some applications (e.g., classification learning problems that require a likelihood function for which a GP prior is not conjugate), Gaussian processes are not analytically tractable. This section will briefly introduce some approximate Gaussian process methods and set the stage for the later chapters of this thesis.

### 2.4.1 Sparse Gaussian Processes

To begin, we will consider the problem of scaling Gaussian processes to large quantities of training observations. The concern is that the Gaussian process inference procedure of a non-degenerate kernel natively scales unfavourably with  $\mathcal{O}(n^2)$  storage and  $\mathcal{O}(n^3)$  time, as was discussed in section 2.2. These complexities generally make Gaussian process modelling challenging beyond modestly sized datasets. Towards this end, much effort has been placed in the literature to scale Gaussian processes more favourably, and indeed this is one of the goals of the present thesis. Most of these techniques are considered to be “sparse” in some sense and are therefore frequently called sparse Gaussian processes. Sparse approximations are made for tractable inference because large scale Gaussian process inference natively results in the manipulation of extremely large or even infinite-dimensional objects<sup>6</sup>. There are a variety of different types of sparse Gaussian process approximations which can be broadly categorized

<sup>6</sup>As an example, consider the weight-space approach of section 2.2.1 which would result in algebraic operations with infinite-dimensional matrices when a non-degenerate kernel is employed (such that  $m = \infty$ ).

into two groups. These groups are discussed in the following sections.

In all cases, sparse Gaussian processes result in a finite dimensional approximation to a non-parametric Gaussian process<sup>7</sup>. One may reasonably ask why it makes sense to go through the effort of setting up non-parametric Gaussian processes with an infinite number of basis functions only to approximate it by a finite dimensional model. Why not simply begin with a finite dimensional parametric model? The reason is because sparse Gaussian processes aim to preserve those properties of non-parametric Gaussian processes that are challenging to possess when beginning with a finite-dimensional parametric model. These desirable properties include:

- Quantifying uncertainty caused by lack of data by using Bayesian inference and specifically ensuring correctly large uncertainty estimates in regions with little data. The limited capacity of parametric models means that even when Bayesian inference is performed, uncertainty can be severely under-represented far from training data since there is no incentive for the model to contain basis functions that are unconstrained by the data in those regions.
- Imposing high-level functional priors on the learning problem. Gaussian process covariance kernels can encode complex and high-level constraints on the form of the functions. Additionally, kernels can be selected that will guarantee convergence to the correct function even if the prior is poorly specified (see section 2.2.3). Such prior information can be challenging to encode in parametric models since priors are typically specified in an abstract parameter space rather than in function space.
- A capacity to model phenomena that scales with the size of the dataset. A non-parametric Gaussian process can represent arbitrarily complex functions. Conversely, parametric models have a finite capacity to represent functions.

The degree to which prior work has been successful in preserving these properties varies. For instance, the third point about model capacity is often the first to be compromised in the presence of large quantities of data. This is somewhat illogical since as the size of a dataset increases so does its ability to explain complex phenomena. Therefore, it would make more sense to increase model capacity for a large dataset, rather than decrease it. Addressing this concern is one of the primary goals of this thesis and therefore many of the contributions within it are developments in various aspects of sparse Gaussian processes. The following two sections briefly outline several types of sparse Gaussian processes from prior works; however, specific techniques will be outlined in greater detail in later chapters, where appropriate.

---

<sup>7</sup>To avoid potential confusion, we note that the discussed sparse Gaussian process approximations will generally result in matrices and prior (cross-)covariance matrices that are still dense. This is not to be confused with approaches to scaling Gaussian processes that choose covariance kernels which admit sparse covariance matrices (for example compactly supported kernels). This compactly-supported approach will not be discussed.

### Approximating the Gaussian Process Posterior

One approach to sparse Gaussian process approximations is to approximate the Gaussian process posterior using variational inference (Titsias, 2009). This approach performs approximate inference in the original GP model, as opposed to the methods of the following section which approximate the model and then perform exact inference. These techniques present computational and optimization challenges when scaling to high-capacity models, however, the approach does very well for functions that can be represented compactly and comes with the advantage that the exact GP prior is maintained.

This thesis does not directly make contributions to Gaussian process approximations of this type; however, it is a competitive technique against which results are compared throughout the thesis. We will refer to this approximation as the variational free energy (VFE) approximation, or stochastic variational Gaussian processes (SVGP) if a stochastic mini-batch approach is taken as in (Hensman et al., 2013). For a detailed introduction to these techniques see (van der Wilk, 2019), for example.

### Approximating the Gaussian Process Prior

The second group of sparse Gaussian processes approximate the prior (eq. (2.23)) such that inference is easier. Typically, after approximation of the prior, inference is performed exactly; however, in some instances additional approximations are employed.

A common approach for sparse Gaussian processes of this type is to approximate a kernel with a finite basis approximation. Specifically, one may replace an exact kernel  $k$  with the approximate kernel  $\tilde{k}$  that is represented as follows:

$$k(\mathbf{x}_i, \mathbf{x}_j) \approx \tilde{k}(\mathbf{x}_i, \mathbf{x}_j) = \sum_{p=1}^{\tilde{m}} \tilde{\phi}_p(\mathbf{x}_i) \tilde{\phi}_p(\mathbf{x}_j), \quad (2.35)$$

where  $\tilde{m} < m$ , i.e., the number of basis functions in the approximate kernel are less than the exact kernel, and  $\tilde{\phi}_i$  for  $i = 1, \dots, \tilde{m}$  are the basis functions of the approximate kernel. These basis functions can take several different forms with perhaps the most common being approximate eigenfunctions of the exact kernel (discussed further in section 4.2) or random features from the exact kernel (discussed further in section 5.3.1). Commonly, exact Gaussian process inference is performed with the approximate kernel  $\tilde{k}$  by taking a weight-space approach along with its associated computational properties (see eq. (2.26)). When exact inference is performed in this way, computational savings are generally only realized when  $\tilde{m} < n$ , thus limiting the number of basis functions that can be retained and subsequently limiting the model capacity. Some approaches also introduce further approximations at inference time to bring further computational advantages. For example, chapter 6 introduces an approximate inference procedure that allows  $\tilde{m} \gg n$ , while chapter 7 provides an additional discrete relaxation of the prior which allows for fast evaluations.

The kernel approximation  $\tilde{k}$  results in a degenerate Gaussian process that can potentially underestimate uncertainty far from the training data. A popular solution to this is the following correction to the kernel approximation in eq. (2.35) that in some scenarios results in a non-degenerate kernel (Snelson and Ghahramani, 2006)

$$k(\mathbf{x}_i, \mathbf{x}_j) \approx \tilde{k}'(\mathbf{x}_i, \mathbf{x}_j) = \tilde{k}(\mathbf{x}_i, \mathbf{x}_j) + \delta(\mathbf{x}_i, \mathbf{x}_j)(k(\mathbf{x}_i, \mathbf{x}_j) - \tilde{k}(\mathbf{x}_i, \mathbf{x}_j)), \quad (2.36)$$

where  $\delta(\mathbf{x}_i, \mathbf{x}_j) = 1$  if  $\mathbf{x}_i = \mathbf{x}_j$ , else 0. A Gaussian process using the kernel  $\tilde{k}'$  has no additional capacity compared to the use of the kernel  $\tilde{k}$  and the additional correction term is only non-zero for some basis function approximations. However, this correction often provides a better approximation to the exact kernel, and can improve the posterior variance far from training observations. Complexity of exact inference using kernel  $\tilde{k}'$  is also the same as the complexity for exact inference with kernel  $\tilde{k}$  since the correction in eq. (2.36) only results in an additive diagonal correction to the prior covariance matrix between training observations.

Another approach that is commonly employed to improve the predictive posterior variance is to augment the set of basis functions at test time. Once again, this correction is only valid for some basis function approximations but can improve the posterior variance far from the training data. This technique is discussed further in appendix D.2.

For the interested reader, Quiñero-Candela and Rasmussen (2005) provide a thorough overview of various sparse Gaussian process techniques that approximate the Gaussian process prior.

## 2.4.2 Gaussian Processes with Non-Gaussian Likelihoods

So far, all discussion of Gaussian processes has assumed that a regression problem is being addressed where the training observations are all continuous valued. Another common type of learning problem is classification (for example) wherein the training responses belong to some discrete categorical set. Gaussian process inference can be performed analytically with a Gaussian likelihood (which we have implicitly assumed throughout section 2.2); however, a Gaussian likelihood is not appropriate for classification. A solution to this is to use an alternative, non-Gaussian likelihood, but this generally results in analytically intractable computations. To account for this, iterative sampling techniques such as MCMC can be used for inference (Neal, 1997, 1998). Alternatively, various approximations can be employed to perform approximate inference including a Laplace approximation (Williams and Barber, 1998), expectation propagation (Minka, 2001), and variational inference (Gibbs and MacKay, 2000), with the latter two being the preferred methods over the Laplace approximation in terms of accuracy. This thesis mainly considers Gaussian likelihoods since our contributions focus on fundamental developments for Gaussian processes; however, the abundance of classification problems in practice makes non-Gaussian likelihoods an important topic that will also be discussed. Specifically, a variational inference approximation is considered for classification problems and other non-

Gaussian likelihoods in chapters 6 and 7 where the method will be introduced in greater detail.

## 2.5 Concluding Remarks

In this chapter we presented the background of the methods that will be built upon in this thesis. Namely, a Bayesian approach to machine learning was introduced, followed by a detailed introduction to Gaussian processes, a particular modelling choice that is a focus of the present thesis. In the chapters that follow, we will also cover the technical background of other areas that are relevant to the particular developments of that respective chapter.

The following chapters present the contributions of the thesis whose objective is to develop scalable Gaussian processes from two perspectives: i) the ability to perform inference on Gaussian process models using large quantities of observed data, and ii) the ability to perform inference with Gaussian processes that have the capacity to model complicated functions. These objectives are typically competing since scalable solutions to Gaussian processes often consider one of these goals at the expense of the other. For instance, general (exact) Gaussian processes may have an infinite model capacity but cannot handle large datasets, whereas Gaussian process approximations limit model capacity at the expense of the ability to model complex functions (see section 2.4.1, for example).

With respect to the scope of these objectives, we will consider developing Gaussian processes that are scalable during both the so-called training stage where model selection is carried out, and during the so-called deployment (or testing) stage where computation of the GP predictive posterior is performed upon model deployment. We have seen in section 2.2 that at the training stage, the necessary operations for exact GPs scale with  $\mathcal{O}(n^2)$  memory and  $\mathcal{O}(n^3)$  time. Additionally, we had seen that the necessary computations at the deployment stage generally requires  $\mathcal{O}(n^2)$  memory and time. These complexities can be prohibitive for both the training and deployment stages, particularly in applications where hardware limitations are restrictive and when predictions need to be performed in real-time.

To briefly summarize the contributions of the following chapters, in chapter 3 we consider Gaussian process inference on a problem structure present in many spatiotemporal problems and develop several algorithms that dramatically reduce the complexity of exact Gaussian process inference using Kronecker matrix algebra. In chapter 4, we then discuss a novel Gaussian process approximation by showing how a highly accurate Nyström approximation of kernel eigenfunction can use as many as  $10^{33}$  inducing points with little computational expense, and also show that the eigenfunction approximation is asymptotically consistent in the limit of infinite inducing points. Chapter 5 subsequently considers a highly general class of Gaussian process covariance kernels where it is shown that the Gaussian process marginal likelihood can be computed with a complexity that is independent of the number of training instances and as low as linear in the number of kernel basis functions. We also show that this class of kernels is highly general, being able to asymptotically recover any stationary covariance

function. In chapter 6, a variational inference approximation to the Gaussian process posterior is considered that allows us to exploit a novel stochastic training strategy whose per iteration complexity is independent of both the number of training examples and the number of kernel basis functions. We show that this unique approach enables the use of high-capacity Gaussian process models on large datasets for regression and classification. Finally, in chapter 7 a discrete relaxation of a Gaussian process prior is considered that enables fast inferencing on devices with limited computing resources. We develop a novel variational inference procedure that exploits Kronecker matrix algebra to compute the variational bound exactly and with a complexity that is independent of  $n$ , allowing rapid inference on large problems.

## Chapter 3

# Scaling Exact Gaussian Processes on Multi-dimensional Grids

In this chapter, we propose two methods for exact Gaussian process (GP) inference and learning on massive image, video, spatiotemporal, or multi-output datasets with missing values (or “gaps”) in the observed responses. The first method ignores the gaps using sparse selection matrices and a highly effective low-rank preconditioner is introduced to accelerate computations. The second method introduces a novel approach to GP training whereby response values are inferred on the gaps *before* explicitly training the model. We find the introduced approaches greatly improve upon state-of-the-art methods in terms of speed, accuracy, and stability. Both of these novel approaches make extensive use of Kronecker matrix algebra to design massively scalable algorithms which have low memory requirements. We demonstrate exact GP inference for a spatiotemporal climate modelling problem with 3.7 million training points as well as a video reconstruction problem with 1 billion points. The following paper was published from some of the contents of this chapter:

T. W. Evans and P. B. Nair (2018b). “Exploiting Structure for Fast Kernel Learning”.  
In: *SIAM International Conference on Data Mining*. SIAM, pp. 414–422

While the chapter (and indeed this entire thesis) focuses on a Bayesian perspective, we note interesting connections and extensions of this work to a regularization (frequentist) perspective in appendix [B](#).

### 3.1 Introduction

We introduce techniques to perform *exact* Gaussian process (GP) training and inference on massive datasets by exploiting a structure that is present in many problems. We consider the problem of reconstructing spatiotemporal datasets structured on a Cartesian tensor product grid where there are missing values or “gaps” in the data. Many spatiotemporal measurements are sampled in this manner, including images and videos, fluid flow data, weather, geology,



medical imaging, etc. and it is important to reconstruct missing or corrupted measurements before analysis can be conducted. These gaps may be present because of obstructed views, the presence of water or government boundaries, missing pixels, image artifacts, or otherwise data corruption, perhaps due to proximity to walls, insufficient sampling frequency or low signal-to-noise ratios (Gunes et al., 2006; Saini et al., 2016; Wilson et al., 2014).

As mentioned in chapter 2, GP modelling is a powerful non-parametric framework for performing classification and regression; however, exact GPs are typically restricted to small datasets since they require  $\mathcal{O}(N^3)$  time and  $\mathcal{O}(N^2)$  storage where  $N$  is the number of training points. This has motivated a considerable amount of work on scalable *approximate* GP methods (discussed in section 2.4.1) that can be applied to large-scale datasets (Lázaro-Gredilla et al., 2010; Smola and Bartlett, 2001; Snelson and Ghahramani, 2006; Williams and Seeger, 2001; Wilson and Nickisch, 2015) and even though significant progress has been made on this topic, current methods cannot generally achieve significant gains in scalability without a noticeable deterioration in accuracy.

Saatçi (2011) introduced scalable GP modelling techniques for datasets whose inputs are distributed on a full Cartesian tensor product grid to leverage efficient Kronecker matrix algebra (Van Loan, 2000). Training datasets structured in this form arise in many important applications including the analysis of images, videos, spatiotemporal fields, sensor networks, or multi-output processes (Álvarez et al., 2012; Osborne et al., 2008). In most of these applications, there will be gaps in the training dataset which may be caused by missing observations, presence of obstructions or irregular domain boundaries, or data corruption (Gunes et al., 2006; Wilson et al., 2014). Unfortunately, the efficient Kronecker matrix algebra used by Saatçi (2011) can no longer be used in the presence of these gaps.

Wilson et al. (2014) introduce an extension to deal with gaps in structured data using a penalty method which works well provided a suitable choice is made for a free penalty parameter. In the present work, we provide alternative formulations that eliminate the need for a free parameter and we demonstrate significant empirical speed improvements, particularly on massive datasets.

We will consider datasets with  $N$  training points which form a subset of  $M$  points on a full Cartesian product grid in  $d$  dimensional input space. Also, we denote the number of missing points (or gaps) from the full grid to be  $L = M - N$ . Two GP formulations are developed which enable fast training and inference on a dataset with this partial grid structure. While we restrict our discussion to GP models, the methods proposed in this chapter can be readily applied to other kernel methods such as regularization networks (see appendix B). Here we summarize the main contributions of the chapter:

- The proposed algorithms perform exact GP training and inference in  $\mathcal{O}(dM^{\frac{d+1}{d}} + N)$  or  $\mathcal{O}(dM^{\frac{d+1}{d}} + L)$  time, and  $\mathcal{O}(dM^{\frac{2}{d}})$  storage; a significant improvement over standard GP models. We also demonstrate marked improvements in both speed and robustness of the proposed algorithms in comparison to the “penalty” method introduced by Wilson et al.

(2014).

- A novel covariance matrix preconditioner is derived which we show significantly accelerates convergence. Direct application of this preconditioner to structured kernel interpolation (SKI) (Wilson and Nickisch, 2015) is also discussed.
- We take a new approach to GP training where we infer the posterior mean on the gaps before training is complete. We show that this technique is greatly advantageous for the considered class of problems.
- For fast predictions, we demonstrate how the posterior mean can be efficiently computed on a grid and we discuss how to quickly compute the posterior covariance exactly, or approximately.
- Finally, we show that our methods are highly scalable and accurate; we train exact GPs on a massive video reconstruction problem and we introduce a powerful approach to climate analysis in the reconstruction of daily temperatures at 291 Ontario weather stations over 56 years. We also analyze the proposed methods on a massive particle image velocimetry (PIV) flow-field reconstruction problem with 19.4 million data points and demonstrate significant improvements in both speed *and* accuracy compared to state-of-the-art techniques on this problem such as gappy proper orthogonal decomposition (GPOD). The largest problem contains over 1 billion points; to the best of our knowledge exact GP inference has not been attempted before on this scale.

## 3.2 Notation & Problem Structure

We consider the set of inputs on a full Cartesian product grid  $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^M$  where  $\mathbf{x}_i \in \mathbb{R}^d$  is the  $d$ -dimensional input vector of the  $i^{\text{th}}$  of  $M$  points on the full grid. The  $M$  corresponding responses are denoted  $\mathbf{y} = \{y_i\}_{i=1}^M$  and we assume that some of these responses are missing from our dataset. Points in  $\mathcal{X}$  with known responses will be differentiated from points with missing responses through two index sets: the set  $X$  contains the indices of the  $N$  training points with *known* responses on the grid, and the set  $Z$  contains the indices of the *missing* training points on the grid such that  $X \cup Z = \{i\}_{i=1}^M$  is the indices of *all*  $M$  points on the full grid. Figure 3.1 provides an illustration of the types of acceptable training point distributions that allow the techniques of this chapter to be utilized. Figure 3.1a shows training input points distributed on a two-dimensional Cartesian product grid where no gaps are present. In fig. 3.1b gaps in the two-dimensional grid (points in the input space where the response is unknown) are shown as empty circles such that the set  $Z$  contains the indices of the points with empty circles and  $X$  contains the indices of all the other points shown. The grid need not be uniformly spaced along each input dimension and a Cartesian product grid in all dimensions can of course be extended to cases of three or more dimensions. Another structure that is amenable to the techniques of this chapter is shown in fig. 3.1c wherein a Cartesian product grid is only

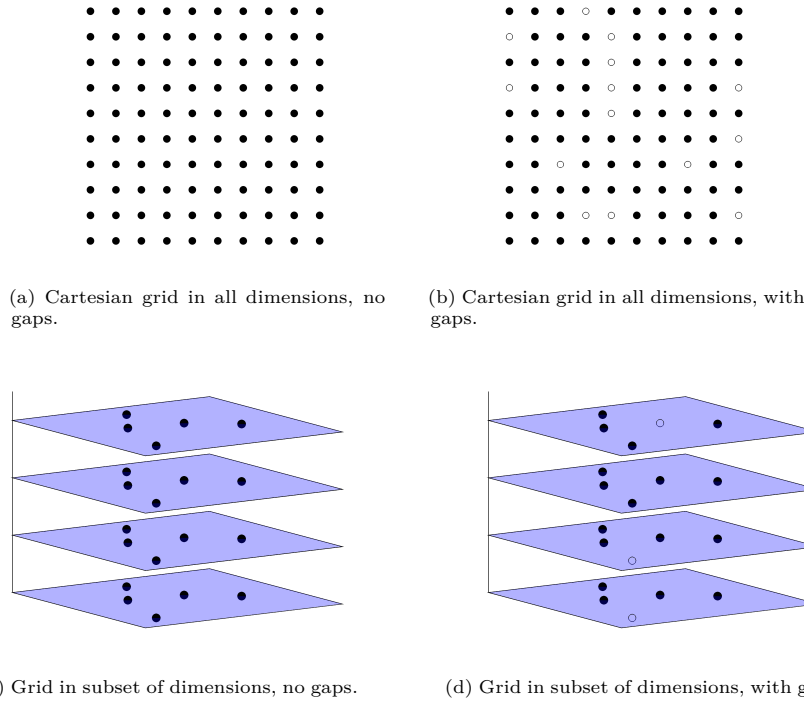


Figure 3.1: Illustration of acceptable training point distributions for the techniques of this chapter.

present in a subset of the input dimensions. In this particular visualization, the inputs are not structured on a grid in the two dimensions denoted by the drawn planes; however, these points are observed at the same locations for each value of the third (vertical) dimension. Therefore a Cartesian product grid is formed between *groups* of dimensions; in this case between the third (vertical) dimension and dimensions one and two (on the drawn planes). Once again, this structure can also contain gaps in the observed responses as visualized in fig. 3.1d. This structure is commonly seen in spatiotemporal problems and sensor networks. For the purpose of clarity, we will proceed assuming the training points are distributed on a Cartesian grid in all dimensions as seen in figs. 3.1a and 3.1b; however, an example problem of the generalized grid structure can be seen in the climate modelling studies of section 3.5.3.

When we write index sets in the subscript, we refer to a partition, i.e., we can write  $\alpha \in \mathbb{R}^M$ , and  $\mathbf{K} \in \mathbb{R}^{M \times M}$  in partitioned form as

$$\alpha = \begin{bmatrix} \alpha_X \\ \alpha_Z \end{bmatrix}, \quad \text{and} \quad \mathbf{K} = \begin{bmatrix} \mathbf{K}_{X,X} & \mathbf{K}_{X,Z} \\ \mathbf{K}_{Z,X} & \mathbf{K}_{Z,Z} \end{bmatrix}.$$

We would like to emphasize that the notation used in this chapter is not entirely consistent from the notation used in the GP overview of section 2.2 as well as the rest of the thesis. The reason for this inconsistency is that the data considered in this chapter is highly structured and requires specialized notation for exposition.

We will employ Gaussian processes (GPs) as non-parametric prior distributions over the latent function that generated the training dataset. It is assumed that the dataset is corrupted

by independent Gaussian noise with variance  $\sigma^2$  and that the latent function is drawn from a Gaussian process with zero mean and covariance determined by the kernel  $k$ . Using the notation defined above, the log marginal likelihood of the targets with known response,  $\mathbf{y}_X$ , can be written as follows (for a detailed introduction to Gaussian processes see section 2.2):

$$\log \Pr(\mathbf{y}_X | \boldsymbol{\theta}, \mathcal{X}_X) = -\frac{1}{2} \log |\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_N| - \frac{1}{2} \mathbf{y}_X^T (\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_N)^{-1} \mathbf{y}_X - \frac{N}{2} \log(2\pi), \quad (3.1)$$

where  $\mathcal{X}_X$  is the set of  $N$  training point input positions, and  $\mathbf{K}_{X,X} \in \mathbb{R}^{N \times N}$  is the kernel covariance matrix evaluated on the training dataset which is a partition of  $\mathbf{K} \in \mathbb{R}^{M \times M}$ , the covariance matrix evaluated on the full tensor product grid;  $[\mathbf{K}]_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$ . The kernel is parameterized by the hyperparameters,  $\boldsymbol{\theta}$ , which we estimate by maximizing the marginal likelihood.

After training, the response  $y_*$  at an arbitrary point  $\mathbf{x}_* \in \mathbb{R}^d$  can be inferred by evaluating the posterior distribution

$$\Pr(y_* | \boldsymbol{\theta}, \mathcal{X}_X, \mathbf{x}_*) \sim \mathcal{N}\left(y_* \mid \mathbf{g}_X^T (\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_N)^{-1} \mathbf{y}_X, k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{g}_X^T (\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_N)^{-1} \mathbf{g}_X\right), \quad (3.2)$$

where  $y_* \in \mathbb{R}$  is the test prediction and  $\mathbf{g} \in \mathbb{R}^M$  is the cross-covariance vector between all points on the full training grid and the test point,  $[\mathbf{g}]_i = k(\mathbf{x}_i, \mathbf{x}_*)$ .

We focus on the three computationally demanding calculations required for GP training and inference; *i*) solving the linear system  $\boldsymbol{\alpha}_X = (\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_N)^{-1} \mathbf{y}_X$  to compute the log marginal likelihood and posterior mean; *ii*) solving the linear system  $(\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_N)^{-1} \mathbf{g}_X$  for each test point to compute the posterior covariance; and *iii*) computing  $\log |\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_N|$  for the log likelihood.

We will consider a covariance kernel that obeys the product correlation rule (as many popular multidimensional kernels do), i.e.,  $k(\mathbf{x}_i, \mathbf{x}_j) = \prod_{l=1}^d k_l(x_{il}, x_{jl})$ , in which case the covariance between points on a full tensor product grid inherits a Kronecker product form;  $\mathbf{K} = \bigotimes_{l=1}^d \mathbf{K}_l$ , where  $\mathbf{K} \in \mathbb{R}^{M \times M}$  is the kernel covariance matrix between all points on the Cartesian product grid containing  $M = \prod_{l=1}^d m_l$  points,  $\mathbf{K}_l \in \mathbb{R}^{m_l \times m_l}$  is a one-dimensional kernel covariance matrix along a slice of the  $l^{\text{th}}$  input dimension, and  $m_l$  is the number of points along the  $l^{\text{th}}$  input dimension on the Cartesian product grid (Saatçi, 2011). Additionally,  $\otimes$  denotes the Kronecker product whose algebraic properties are summarized in appendix A. Exploiting this Kronecker product structure, we find that only  $\mathcal{O}(dN^{\frac{2}{d}})$  storage is required, and a matrix-vector product with  $\mathbf{K}$  requires only  $\mathcal{O}(dN^{\frac{d+1}{d}})$  time<sup>1</sup>. However, when there are missing responses from the full grid, the Kronecker product structure is broken and  $\mathbf{K}_{X,X} \in \mathbb{R}^{N \times N}$  becomes a large dense matrix with no structure. As a result, GP modelling storage and time increase to the nominal complexities of  $\mathcal{O}(N^2)$  and  $\mathcal{O}(N^3)$ , respectively. We next discuss some novel approaches for addressing this computational challenge.

<sup>1</sup>For ease of exposition, we assume  $m_1 = m_2 = \dots = m_d = \sqrt[d]{M}$  points along each dimension of the Cartesian product grid when denoting asymptotic complexities. Additionally, we refer to (Saatçi, 2011, algorithm 15) for an efficient matrix-vector product algorithm.

### 3.3 Linear Algebra aspects of GP Training

Here we propose new algorithms to compute a matrix-vector product with the covariance matrix inverse,  $(\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_N)^{-1}$ , which is the most computationally demanding component of performing exact GP regression. We first review the existing “penalize-gaps” (PG) formulation of Wilson et al. (2014) which requires solving an unstructured linear system of equations of size  $M \times M$  (number of points on full grid). We then proceed to develop two alternative formulations; the “ignore-gaps” (IG) method which reduces the size to  $N \times N$  (number of training points); and the “fill-gaps” (FG) method which only requires solving an unstructured linear system of size  $L \times L$  (number of gaps). We complete this section by developing a preconditioner for the two new formulations presented.

#### 3.3.1 Gap Penalization Strategy

Wilson et al. (2014) first approached this problem when using GPs to model observations on a spatiotemporal Cartesian product grid. The central idea of their solution was to fill in the gaps in the observations with arbitrary values and subsequently use a penalty approach to ensure that these pseudo-observations do not influence the final model. This may appear to be somewhat counterintuitive since they are essentially increasing the number of observations; however, this step allows fast matrix-vector products to be made with the  $M \times M$  covariance matrix  $\mathbf{K}$  which has a Kronecker product structure.

**Proposition 3.1** (Penalize-Gaps, PG). *The vector  $\boldsymbol{\alpha} \in \mathbb{R}^M$  obtained from the numerical solution of the penalized  $M \times M$  system of equations*

$$(\mathbf{K} + \gamma \mathbf{R} + \sigma^2 \mathbf{I}_M) \boldsymbol{\alpha} = \mathbf{y} \quad (3.3)$$

*satisfies  $(\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_N) \boldsymbol{\alpha}_X = \mathbf{y}_X$  in the limit of the penalty parameter  $\gamma \rightarrow \infty$ , where  $\mathbf{K} \in \mathbb{R}^{M \times M}$  is the kernel covariance matrix on the full product grid,  $\mathbf{R} \in \mathbb{R}^{M \times M}$  is an all zero matrix except  $\mathbf{R}_{X,X} = \mathbf{I}_N$ , and arbitrary numerical values are inserted in the missing entries of  $\mathbf{y} \in \mathbb{R}^M$ .*

A proof is provided in the supplementary material of (Wilson et al., 2014). A linear conjugate gradient (CG) solver (Atkinson, 2008) can be used to solve eq. (3.3) to take advantage of fast matrix-vector products in  $\mathcal{O}(dM^{\frac{d+1}{d}} + N)$  since  $\mathbf{K} = \bigotimes_{i=1}^d \mathbf{K}_i$  has a Kronecker product form. The preconditioner  $(\gamma \mathbf{R} + \sigma^2 \mathbf{I}_M)^{-\frac{1}{2}}$  was suggested for this formulation (Wilson et al., 2014).

Unfortunately, this method can suffer from numerical inaccuracies with a poor choice of  $\gamma$ . For instance, if  $\gamma$  is too small, the desired system will be inaccurately approximated. Conversely, a  $\gamma$  chosen to be too large will result in an ill-conditioned system of equations. It is therefore not clear how large the penalty parameter  $\gamma$  should be *a priori*. Two novel approaches that do not suffer from this limitation are presented next.

### 3.3.2 Selection Matrix Strategy

Using sparse selection matrices, we develop a technique to exploit Kronecker matrix algebra without the use of a penalty.

**Proposition 3.2** (Ignore-Gaps, IG). *The solution of the linear algebraic system of equations  $(\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_N) \boldsymbol{\alpha}_X = \mathbf{y}_X$  also satisfies the system of equations*

$$\mathbf{W}(\mathbf{K} + \sigma^2 \mathbf{I}_M) \mathbf{W}^T \boldsymbol{\alpha}_X = \mathbf{y}_X, \quad (3.4)$$

where  $\mathbf{W} \in \mathbb{R}^{N \times M}$  is a sparse selection matrix with one value per row set to unity in the column corresponding to each index in  $X$  such that  $\mathbf{W}\mathbf{K}\mathbf{W}^T = \mathbf{K}_{X,X}$ .

It is straightforward to see that this result holds since application of the selection matrices immediately recovers the target system of equations. This formulation also lends itself well to a conjugate gradient solver since it admits fast matrix-vector products in  $\mathcal{O}(dM^{\frac{d+1}{d}} + N)$  due to the form of  $\mathbf{K} = \bigotimes_{i=1}^d \mathbf{K}_i$ . Most importantly, the proposed method does not require any user-defined parameters unlike the previous penalize-gaps method.

We observe that this covariance matrix structure coincides with that of “structured kernel interpolation” (SKI) (Wilson and Nickisch, 2015) in the special case where training data coincides with the inducing point grid and so the SKI interpolation matrix becomes the sparse selection matrix  $\mathbf{W}$ . We emphasize that while SKI typically uses an approximate kernel, if the inducing point grid and training data coincide, then the exact kernel is recovered during training. At prediction time, the methods differ since the SKI kernel evaluated between train and test points becomes approximate whereas we continue to use the exact kernel.

### 3.3.3 Fill Gaps Strategy

While the previous two approaches determine  $\boldsymbol{\alpha}_X$  directly, this approach first infers the posterior mean on the gaps,  $\mathbf{y}_Z$ , thus recovering a non-gappy problem upon which we can fully exploit Kronecker matrix algebra.

**Proposition 3.3** (Fill-Gaps, FG). *The solution of  $(\mathbf{K} + \sigma^2 \mathbf{I}_M) \boldsymbol{\alpha} = \mathbf{y}$  satisfies  $(\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_N) \boldsymbol{\alpha}_X = \mathbf{y}_X$  where the missing values,  $\mathbf{y}_Z$ , are determined by*

$$\mathbf{V}\mathbf{Q}(\mathbf{T} + \sigma^2 \mathbf{I}_M)^{-1} \mathbf{Q}^T \mathbf{V}^T \mathbf{y}_Z = -\mathbf{V}\mathbf{Q}(\mathbf{T} + \sigma^2 \mathbf{I}_M)^{-1} \mathbf{Q}^T \mathbf{W}^T \mathbf{y}_X, \quad (3.5)$$

where  $\mathbf{V} \in \mathbb{R}^{L \times M}$  is a sparse selection matrix that has one value per row set to unity in the column corresponding to each index in  $Z$ , and  $\mathbf{Q}, \mathbf{T} \in \mathbb{R}^{M \times M}$  are unitary and diagonal matrices, respectively, formed from the eigendecomposition of  $\mathbf{K} = \mathbf{Q}\mathbf{T}\mathbf{Q}^T$ .

*Proof.* First, considering the special case  $\sigma^2 = 0$ , observe that we can use  $\boldsymbol{\alpha}_X = \mathbf{K}_{X,X}^{-1} \mathbf{y}_X$  and  $\boldsymbol{\alpha}_Z = 0$  to infer the posterior mean of the missing values on the gaps,  $\mathbf{y}_Z$ , by  $\mathbf{y} = \mathbf{K}\boldsymbol{\alpha}$ . Here we attempt to solve for  $\mathbf{y}_Z$  directly without first computing  $\boldsymbol{\alpha}_X$ . We start by writing  $\boldsymbol{\alpha} = \mathbf{K}^{-1} \mathbf{y}$

in partitioned form, and impose  $\boldsymbol{\alpha}_Z = \mathbf{0}$

$$\begin{bmatrix} [\mathbf{K}^{-1}]_{X,X} & [\mathbf{K}^{-1}]_{X,Z} \\ [\mathbf{K}^{-1}]_{Z,X} & [\mathbf{K}^{-1}]_{Z,Z} \end{bmatrix} \begin{bmatrix} \mathbf{y}_X \\ \mathbf{y}_Z \end{bmatrix} = \begin{bmatrix} \boldsymbol{\alpha}_X \\ \mathbf{0} \end{bmatrix},$$

where  $[\mathbf{K}^{-1}]_{(\cdot, \cdot)}$  is a partition of  $\mathbf{K}^{-1}$ . Rearranging the last row gives  $[\mathbf{K}^{-1}]_{Z,Z}\mathbf{y}_Z = -[\mathbf{K}^{-1}]_{Z,X}\mathbf{y}_X$ , and observing that  $\mathbf{V}\mathbf{K}^{-1}\mathbf{V}^T = [\mathbf{K}^{-1}]_{Z,Z}$  and  $\mathbf{V}\mathbf{K}^{-1}\mathbf{W}^T = [\mathbf{K}^{-1}]_{Z,X}$  gives the system of equations

$$\mathbf{V}\mathbf{K}^{-1}\mathbf{V}^T\mathbf{y}_Z = -\mathbf{V}\mathbf{K}^{-1}\mathbf{W}^T\mathbf{y}_X. \quad (3.6)$$

Next, if we consider  $\sigma^2 > 0$ , then we require a partition of  $(\mathbf{K} + \sigma^2\mathbf{I}_M)^{-1}$  instead of  $\mathbf{K}^{-1}$ . We can write this as  $(\mathbf{K} + \sigma^2\mathbf{I}_M)^{-1} = \mathbf{Q}(\mathbf{T} + \sigma^2\mathbf{I}_M)^{-1}\mathbf{Q}^T$  using the eigendecomposition  $\mathbf{K} = \mathbf{Q}\mathbf{T}\mathbf{Q}^T$  defined in the proposition statement which only requires  $\mathcal{O}(dM^{\frac{3}{d}})$  time to compute using Kronecker matrix algebra. Substituting this for  $\mathbf{K}^{-1}$  in eq. (3.6) completes the proof.  $\square$

Like the other methods (IG and PG), this formulation lends itself well to a conjugate gradient solver since it admits fast matrix-vector products in  $\mathcal{O}(dM^{\frac{d+1}{d}} + L)$  due to the structure of  $\mathbf{K} = \bigotimes_{i=1}^d \mathbf{K}_i$ . The eigendecomposition of  $\mathbf{K} = \mathbf{Q}\mathbf{T}\mathbf{Q}^T$  can be rapidly computed in  $\mathcal{O}(dM^{\frac{3}{d}})$  time using Kronecker matrix algebra, and as a result,  $\mathbf{Q} = \bigotimes_{i=1}^d \mathbf{Q}_i$  is also a Kronecker product matrix (Van Loan, 2000). Lastly, since  $\mathbf{T} + \sigma^2\mathbf{I}_M$  is diagonal, matrix-vector products with its inverse cost only  $\mathcal{O}(M)$  time. We would like to emphasize that the time complexity of matrix-vector products are *independent* of the number of training points,  $N$ . Additionally, we observe that since the system being solved by a CG method is  $L \times L$ , the number of iterations required is expected to be much less than  $L$ , the number of gaps. We therefore expect that this method would be extremely fast for massive datasets where there are few gaps but in section 3.5 we find that it also outperforms other methods well outside of this regime. Once the missing values  $\mathbf{y}_Z$  are inferred,  $\boldsymbol{\alpha}_X$  can be found by evaluating  $\boldsymbol{\alpha} = \mathbf{Q}(\mathbf{T} + \sigma^2\mathbf{I}_M)^{-1}\mathbf{Q}^T\mathbf{y}$ , which requires only  $\mathcal{O}(dM^{\frac{d+1}{d}})$  time.

### 3.3.4 Preconditioning strategies

Using the ignore-gaps technique outlined in section 3.3.2, and the eigendecomposition of  $\mathbf{K} = \mathbf{Q}\mathbf{T}\mathbf{Q}^T$ , observe that we can write the kernel covariance matrix as  $\mathbf{K}_{X,X} = \mathbf{W}\mathbf{Q}\mathbf{T}\mathbf{Q}^T\mathbf{W}^T$ . We can also approximate  $\mathbf{K}_{X,X}$  using the  $p$  largest eigenvalues and corresponding eigenvectors of  $\mathbf{K}$  as

$$\mathbf{K}_{X,X} \approx \tilde{\mathbf{K}}_{X,X} = \mathbf{W}\mathbf{Q}\mathbf{S}_p^T (\mathbf{S}_p^T \mathbf{T} \mathbf{S}_p^T) \mathbf{S}_p \mathbf{Q}^T \mathbf{W}^T = \mathbf{W}\mathbf{Q}\mathbf{S}_p^T \mathbf{T}_p \mathbf{S}_p \mathbf{Q}^T \mathbf{W}^T,$$

where  $\mathbf{S}_p \in \mathbb{R}^{p \times M}$  is a sparse selection matrix whose  $i^{\text{th}}$  row has one value set to unity in the column corresponding to the index of the  $i^{\text{th}}$  largest eigenvalue of  $\mathbf{K}$  on the diagonal of  $\mathbf{T}$ ; and

$\mathbf{T}_p \in \mathbb{R}^{p \times p}$  is a subset of  $\mathbf{T}$ . We can then use the matrix inversion lemma to invert  $\tilde{\mathbf{K}}_{X,X} + \sigma^2 \mathbf{I}_N$

$$(\tilde{\mathbf{K}}_{X,X} + \sigma^2 \mathbf{I}_N)^{-1} = \frac{1}{\sigma^2} \left[ \mathbf{I}_N - \mathbf{W} \mathbf{Q} \mathbf{S}_p^T (\sigma^2 \mathbf{I}_p + \mathbf{T}_p \mathbf{S}_p \mathbf{Q}^T \mathbf{W}^T \mathbf{W} \mathbf{Q} \mathbf{S}_p^T)^{-1} \mathbf{T}_p \mathbf{S}_p \mathbf{Q}^T \mathbf{W}^T \right], \quad (3.7)$$

which only requires the additional storage and inversion of a matrix of size  $p \times p$ . After the  $p \times p$  matrix has been factorized, multiplications with this preconditioner cost only  $\mathcal{O}(dM^{\frac{d+1}{d}} + p^2)$  time.

The use of  $\tilde{\mathbf{K}}_{X,X}$  for matrix preconditioning was explored with notable empirical success by Cutajar et al. (2016) where a sub-set of training data was used as ‘‘inducing points’’ giving  $M < N$ , as opposed to a super-set as it is here ( $M > N$ ) which we expect improves performance.

In section 3.3.2, we observed the duality between the ignore-gaps technique and SKI (Wilson and Nickisch, 2015). We would further like to point out that the preconditioner eq. (3.7) can similarly be used in the SKI framework, but we will not study its effectiveness in a general SKI setting.

Considering now a preconditioner for the fill-gaps method of section 3.3.3, it can be shown that the matrix on the left-hand side of eq. (3.5) can be rewritten as

$$\mathbf{V} \mathbf{Q} (\mathbf{T} + \sigma^2 \mathbf{I}_M)^{-1} \mathbf{Q}^T \mathbf{V}^T = \mathbf{J} + \zeta \mathbf{I}_L = \mathbf{V} \mathbf{Q} [(\mathbf{T} + \sigma^2 \mathbf{I}_M)^{-1} - \zeta \mathbf{I}_M] \mathbf{Q}^T \mathbf{V}^T + \zeta \mathbf{I}_L$$

where we have applied a spectral shift of  $\zeta \in \mathbb{R}$  to the first term, which we call  $\mathbf{J} \in \mathbb{R}^{L \times L}$ . We can now write a rank- $p$  approximation of  $\mathbf{J}$  as

$$\mathbf{J} \approx \tilde{\mathbf{J}} = \mathbf{V} \mathbf{Q} \bar{\mathbf{S}}_p^T \left( \bar{\mathbf{S}}_p [(\mathbf{T} + \sigma^2 \mathbf{I}_M)^{-1} - \zeta \mathbf{I}_M] \bar{\mathbf{S}}_p^T \right) \bar{\mathbf{S}}_p \mathbf{Q}^T \mathbf{V}^T = \mathbf{V} \mathbf{Q} \bar{\mathbf{S}}_p^T \bar{\mathbf{T}}_p \bar{\mathbf{S}}_p \mathbf{Q}^T \mathbf{V}^T,$$

where  $\bar{\mathbf{T}}_p \in \mathbb{R}^{p \times p}$  is a subset of  $[(\mathbf{T} + \sigma^2 \mathbf{I}_M)^{-1} - \zeta \mathbf{I}_M]$  containing the largest values on its diagonal, and  $\bar{\mathbf{S}}_p \in \mathbb{R}^{p \times M}$  is a sparse selection matrix whose  $i^{\text{th}}$  row has one non-zero value set to unity in the column corresponding to the index of the  $i^{\text{th}}$  smallest eigenvalue of  $\mathbf{K}$  on the diagonal of  $\mathbf{T}$ . To ensure no singularities, we choose  $0 < \zeta < (\lambda_p + \sigma^2)^{-1}$  where  $\lambda_p \in \mathbb{R}$  is the  $p^{\text{th}}$  smallest eigenvalue of  $\mathbf{K}$  (in practice we choose a value mid-range). The term  $\tilde{\mathbf{J}} + \zeta \mathbf{I}_L$  is then an approximation of the fill-gaps left-hand side matrix which we can cheaply invert to use as a preconditioner with the matrix inversion lemma

$$(\tilde{\mathbf{J}} + \zeta \mathbf{I}_L)^{-1} = \zeta^{-1} \left[ \mathbf{I}_L - \mathbf{V} \mathbf{Q} \bar{\mathbf{S}}_p^T (\zeta \mathbf{I}_p + \bar{\mathbf{T}}_p \bar{\mathbf{S}}_p \mathbf{Q}^T \mathbf{V}^T \mathbf{V} \mathbf{Q} \bar{\mathbf{S}}_p^T)^{-1} \bar{\mathbf{T}}_p \bar{\mathbf{S}}_p \mathbf{Q}^T \mathbf{V}^T \right]. \quad (3.8)$$

The preceding matrix similarly admits fast matrix vector products in  $\mathcal{O}(dM^{\frac{d+1}{d}} + p^2)$  time. We will analyze the efficacy of both of these preconditioners in section 3.5.2.

### 3.4 Model Selection & Fast Inferencing

This section aims to cohesively organize the developments outlined previously into a practical algorithm that can be used for training and inferencing. To compute the log marginal



likelihood in eq. (3.1), we need  $(\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_N)^{-1} \mathbf{y}_X$ , and  $\log |\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_N|$ . We already discussed efficient ways to compute the former and we will use a Nyström approximation for the latter, which is accurate for large  $N$  (Wilson et al., 2014);

$$\log |\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_N| \approx \sum_{i=1}^N \log \left( \frac{N}{M} \lambda_i + \sigma^2 \right),$$

where  $\lambda_i$  is the  $i^{\text{th}}$  largest eigenvalue of  $\mathbf{K} = \bigotimes_{j=1}^d \mathbf{K}_j$  which can be rapidly computed in  $\mathcal{O}(dM^{\frac{3}{2}})$  time due to its Kronecker product form (Van Loan, 2000). With some modification, other log determinant estimators that require only matrix vector multiplications could also be considered (see Dong et al. (2017) for a discussion of such techniques). We now have all the ingredients required to efficiently train our model; we can now compute and maximize the marginal likelihood with respect to the hyperparameters and compute  $\boldsymbol{\alpha}_X$ . We shall next outline how to rapidly compute the posterior mean and covariance during inference.

We frequently need to infer test points distributed on a tensor product grid (e.g., for search or visualization). Here we demonstrate how we can exploit Kronecker matrix algebra to evaluate the posterior mean of points on a grid extremely quickly. From eq. (3.2), the posterior mean for a single point is given by  $\mathbf{g}_X^T \boldsymbol{\alpha}_X$  which is equivalent to  $\mathbf{g}^T \boldsymbol{\alpha}$ , where  $\boldsymbol{\alpha}_X = (\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_N)^{-1} \mathbf{y}_X$  and  $\boldsymbol{\alpha}_Z = \mathbf{0}$ . If it is sought to perform inference at many points then we can replace  $\mathbf{g}$  with  $\mathbf{G} \in \mathbb{R}^{M \times Q}$  which is formed by horizontally stacking the column vectors  $\mathbf{g}$  for each of  $Q$  distinct test points. Further, if we take these test points to be on a Cartesian product grid then a Kronecker product structure is inherited,  $\mathbf{G} = \bigotimes_{i=1}^d \mathbf{G}_i$ , where  $\mathbf{G}_i \in \mathbb{R}^{m \times q}$ , and we take  $Q = q^d$ ,  $M = m^d$ . By recognizing and exploiting this structure, the time required to compute the posterior mean at the  $Q$  test points decreases from  $\mathcal{O}(NQ)$  to  $\mathcal{O}(\sqrt[d]{Q}M + \sqrt[d]{M}Q)$ . This can give significant computational advantages for large  $Q$ .

The posterior covariance computation poses a different problem, since, from eq. (3.2) it is evident that it requires the solution of an  $N \times N$  system of equations for *each* test point,  $(\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_N)^{-1} \mathbf{g}_X$ . We could compute this using any technique developed in section 3.3 since extension of the penalize-gaps and ignore-gaps techniques for this problem is trivial. However, applying the fill-gaps formulation gives an interesting interpretation: what we are “filling in” here is  $\mathbf{g}_Z$  which is the cross-covariance between gaps on the training grid and the test point such that the final solution is not influenced by the gaps. Once  $\mathbf{g}_Z$  is filled in, the fully structured problem can be solved rapidly to complete the posterior covariance computation. Since this problem is effectively identical to the training problem, we refer to section 3.5 for a comparison of the different computation methods; however, it must be considered that any time invested in formulating a preconditioner once can be used to perform inference at many test points.

We may alternatively consider an *approximation* of the posterior covariance where we make use of the matrix preconditioner developed in section 3.3.4 by replacing  $\mathbf{K}_{X,X}$  with  $\tilde{\mathbf{K}}_{X,X}$  in eq. (3.2). This gives us a similar posterior distribution as the Nyström method for GP modelling studied by Williams and Seeger (2001) and would enable the posterior covariance to

be computed in  $\mathcal{O}(dM^{\frac{d+1}{d}} + p^2)$  time per test point. We will not consider this approximation in the experiments of section 3.5 but will instead restrict our attention to exact GP inference.

## 3.5 Experiments

A python implementation of the methods discussed in this chapter along with several tutorials can be found at [https://github.com/treforevans/gp\\_grid](https://github.com/treforevans/gp_grid).

### 3.5.1 Stress Tests

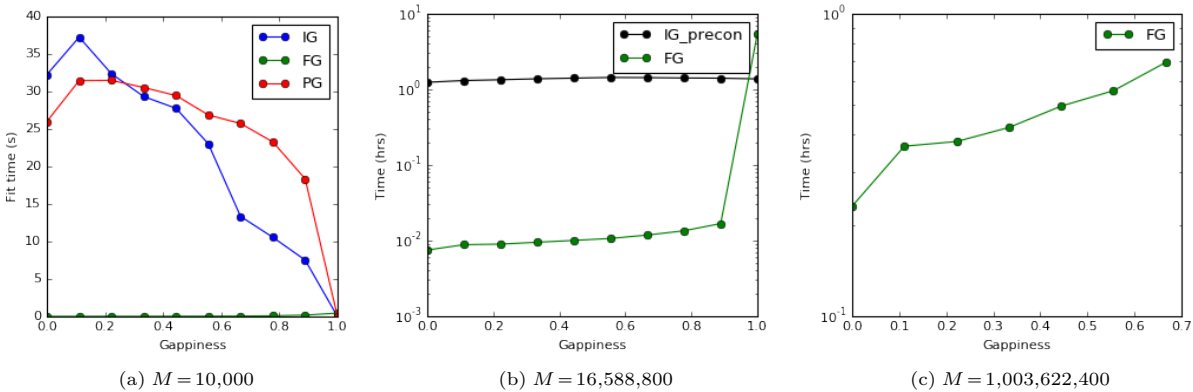


Figure 3.2: Reconstruction timings comparing the techniques outlined in section 3.3 across a range of gappiness on various problem sizes,  $M$ . The three techniques are FG - fill-gaps with no preconditioner (section 3.3.3), IG - ignore-gaps with no preconditioner (section 3.3.2), IG\_precon - ignore-gaps with a rank-5000 preconditioner (section 3.3.4) and PG - penalize-gaps with the preconditioner discussed in section 3.3.1. The timings include both the time to fill in the missing values, as well as compute  $\mathbf{K}^{-1}\mathbf{y} = \boldsymbol{\alpha}$ .

We test the robustness of techniques outlined in section 3.3 for training a GP regression model on massive datasets of varying “gappiness” =  $(M - N)/M = L/M$ . Results are shown in fig. 3.2 for gappiness sweeps across various grid sizes,  $M$ . For the  $M = 10,000$  case, data was generated from the two-dimensional Rastrigin function (Mühlenbein et al., 1991) and for the larger studies, we reconstruct a gappy 4K video of a resonating elastic membrane defined by the two-dimensional wave equation,  $\frac{\partial^2 y}{\partial x_1^2} + \frac{\partial^2 y}{\partial x_2^2} = \frac{\partial^2 y}{\partial x_3^2}$ , where  $x_1, x_2 \in \mathbb{R}$  are spatial coordinates, and  $x_3 \in \mathbb{R}$  is time. The membrane is constrained along the edges of the video frame and begins with random initial conditions. Our GP models use the squared-exponential kernel,  $k_l(\mathbf{x}, \mathbf{z}) = \exp(-\|\mathbf{x} - \mathbf{z}\|_2^2 / \theta_l^2)$ , and in this study hyperparameters are estimated beforehand on the fully structured dataset. We do not consider hyperparameter estimation using the gappy data in this experiment since the computation of  $\boldsymbol{\alpha}$  alone allows us to contrast differences between the considered techniques. Gaps are randomly applied to mask the training data and  $\boldsymbol{\alpha}$  is computed using a CG solver to a tolerance of  $10^{-6}$ . All experiments are run on a machine with two E5-2680 v3 processors and 128Gb RAM and we only report timings since the formulations are all mathematically identical so differences between the solutions found by each approach are negligible.

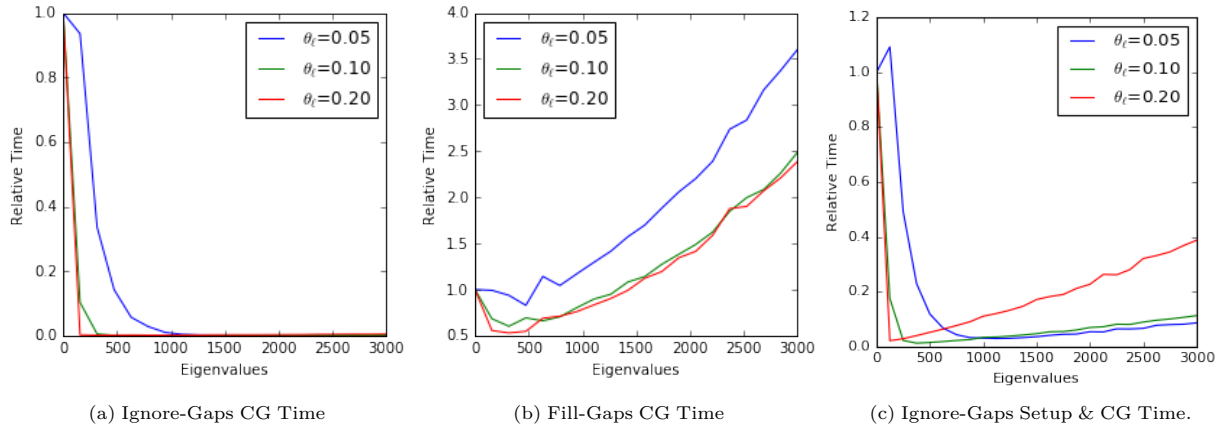


Figure 3.3: Results of a preconditioner efficacy study. We use a linear conjugate gradient solver to find  $\alpha_X$  using the preconditioner eq. (3.7) with varying values of the number of eigenvalues included,  $p$ , and kernel lengthscales,  $\theta_\ell$ . The timings in fig. 3.3a and fig. 3.3b only include the time required to solve the linear system using a CG solver while in fig. 3.3c the time required to construct the preconditioner is also included. All timings are presented relative to the time to perform the reconstruction with no preconditioner.

From fig. 3.2a we can see that the non-preconditioned ignore-gaps (IG) and the preconditioned penalize-gaps (PG) algorithms performed comparably, but the fill-gaps (FG) method significantly outperformed the others across much of the gappiness spectrum.

In the larger video reconstruction problem in fig. 3.2b, IG and PG are not shown, since after a wall-clock time of 5 days, only the  $N = 1$  case had converged. However, the FG method and the ignore-gaps algorithm with a rank-5000 preconditioner (IG\_precon) completed reconstructions across the full gappiness spectrum with each run taking  $\lesssim 1$  hour. We emphasize that although  $p = 5000$  is four orders of magnitude smaller than  $M$ , the preconditioner dramatically accelerated convergence.

Moving to the 1-billion-point video reconstruction problem in fig. 3.2c, the IG\_precon method was omitted since the time required to form the preconditioner became prohibitive; however, the FG routine completed all reconstructions up to 70% gappiness in under 1 hour.

In figures 3.2a & 3.2b, we observe that the fill-gaps technique is fastest where the gappiness is low, but the ignore-gaps technique is only faster when the gappiness is surprisingly close to one; in section 3.5.2 we provide insight into why the FG method converges so quickly.

### 3.5.2 Preconditioner Studies

Here we assess the efficacy of the preconditioners outlined in section 3.3.4 for both the ignore-gaps and fill-gaps methods. For these tests, we take the training data to be structured on a two-dimensional grid with 100 points evenly spaced in the range  $[0, 1]$  along each dimension, giving  $M = 10,000$ . We then randomly remove 50% of the training points and sample responses as  $\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_M)$ . For our GP model, we set  $\sigma^2 = 10^{-6}$ , and use a squared-exponential kernel, considering a range of kernel lengthscales,  $\theta_\ell$ . We then compute  $\alpha_X = (\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_N)^{-1} \mathbf{y}_X$  using various values of preconditioner rank,  $p$ , and average over several samples of  $\mathbf{y}$  and randomly applied gaps. Results are shown in fig. 3.3.

Firstly, observe that the ignore-gaps scheme benefits dramatically through the use of a preconditioner (eq. (3.7)), which is consistent with the results of the stress tests. Figure 3.3a shows the  $\alpha_X$  computation time *excluding* the preconditioner setup time where it is evident that massive computational savings can be realized. This is representative of posterior covariance computations since we can reuse the preconditioner for many test points after an initial setup (discussed in section 3.4). The timings in fig. 3.3c *include* setup time which is representative of using the preconditioner for training. Here a broad minimum is evident, indicating that an acceptable value of  $p$  can be easily chosen. We also see that as  $\theta_\ell$  decreases, a greater number of eigenvalues,  $p$ , are required to achieve equivalent savings, as expected.

In fig. 3.3b we analyze the fill-gaps preconditioner (eq. (3.8)) where it appears to only be effective for a very particular value of  $p$ , even when we exclude the preconditioner setup time. If the setup time is included, we find that the method converges fastest with no preconditioner. We expect this is because the eigenvalues of  $(\mathbf{K} + \sigma^2 \mathbf{I}_M)^{-1}$  tend to decay slowly from the largest possible value of  $\sigma^{-2}$  and so the reduced-rank approximation used in the preconditioner is inaccurate. We would also expect this eigenspectrum to make the non-preconditioned fill-gaps method particularly well suited to a CG solver, which is consistent with the fast convergence observed in the stress tests of section 3.5.1.

### 3.5.3 Ontario Weather Stations

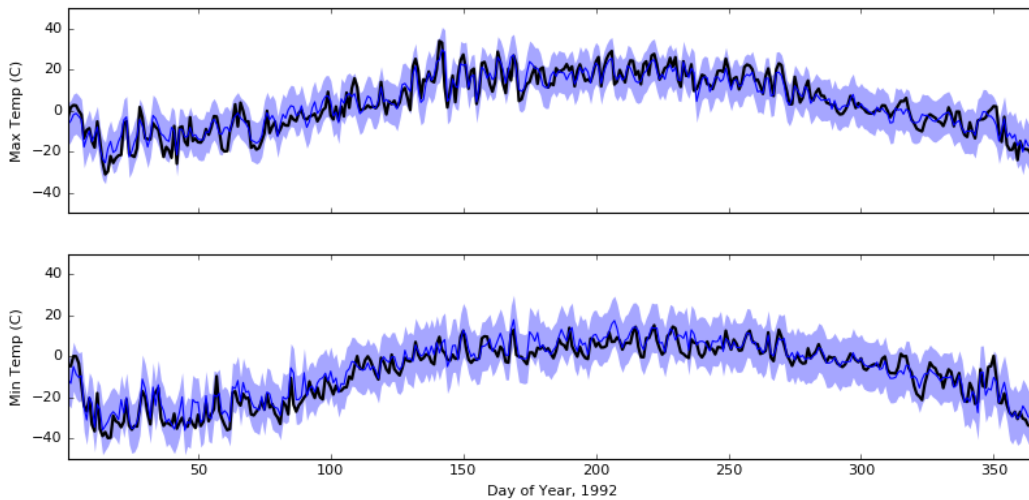


Figure 3.4: Reconstructed daily temperature observations for Moosonee in 1992. The black curves show the actual daily maximum (top) and minimum temperatures (bottom) which were both withheld from the model to compute the blue posterior distribution where the mean and three standard deviations (99.7% confidence) are illustrated.

Here we demonstrate an efficient and powerful analysis of weather patterns at 291 weather stations across Ontario every day from 1950 to 2005. The dataset (Natural Resources and Forestry, 2008) contains many missing observations, mostly due to corrupted recordings or because some weather stations were not in operation for the entire period. Reconstructing

these observations are important for environmental studies and typically this is done by interpolating responses spatially for each day, independent of all other days, and all other responses (Hutchinson et al., 2009; Natural Resources and Forestry, 2008). These dependencies are ignored because the problem size becomes prohibitive when temporal or multi-output correlations are considered. We identify structure in this problem, allowing us to consider correlations within space, time, and between responses to allow forecasting into the future with an accurate posterior distribution. This ability to forecast is not possible with existing techniques.

We use a kernel for our multi-output GP which admits the following non-gappy covariance matrix;

$$\mathbf{K} = \mathbf{K}_{\text{year}} \otimes \mathbf{K}_{\text{day}} \otimes \mathbf{K}_{\text{space}} \otimes \mathbf{B},$$

where  $\mathbf{K}_{\text{year}} \in \mathbb{R}^{56 \times 56}$  is the covariance between the integer year of observations ( $\in \{1950, \dots, 2005\}$ ),  $\mathbf{K}_{\text{day}} \in \mathbb{R}^{366 \times 366}$  is the covariance between the day of year of observations ( $\in \{1, \dots, 366\}$ ),  $\mathbf{K}_{\text{space}} \in \mathbb{R}^{291 \times 291}$  is the covariance between the location of the weather stations ( $\in \mathbb{R}^3, \{\text{latitude, longitude, elevation}\}$ ), and  $\mathbf{B} \in \mathbb{R}^{2 \times 2}$  is the covariance between the daily maximum and minimum temperature, which are the two observations we are modelling each day (see (Álvarez et al., 2012) for multi-output kernel details). We also apply a periodic transformation to the day-of-year kernel with a period of 365.25 days (Roberts et al., 2013). Although the weather stations themselves are not distributed on a grid, we have evidently identified significant structure to exploit.

The full grid size is  $M = 11,928,672$ , however, over 6.5 million points are missing and 30% of the remaining points are randomly withheld for testing, giving  $N = 3,742,547$  training points; an enormous problem for exact GP modelling.

Table 3.1 illustrates the results of the exact GP model constructed on the climate dataset using the different techniques outlined. Model training includes hyperparameter estimation through marginal likelihood maximization and the root mean squared error (RMSE) evaluated on the randomly withheld test set is reported for daily minimum and maximum temperature predictions from the multi-output GP. The test error is quite low and is almost constant down the columns as we would expect since the different training methods should ideally result in identical models. It is firstly evident that the penalize-gaps technique (PG) is the slowest method considered, even for a small penalty parameter ( $\gamma = 100$ ). The non-preconditioned ignore-gaps (IG) technique trained about 22% faster; however, we see that the training time can be further reduced by nearly an order of magnitude using a preconditioner with only  $p = 3000$ . The fill-gaps (FG) technique continues to be our most robust method, nearly halving the training time of any other.

Figure 3.4 compares the posterior distribution to the observed data at the Moosonee weather station in 1992. Moosonee was chosen since it is the farthest from its nearest neighbour, the Smoky Falls weather station which is 171km inland (see fig. 3.6 for a map). The GP does a good reconstruction of the withheld observations, especially considering the evident

	Run Time (hrs)	RMSE ( $^{\circ}C$ )	
		Minimum	Maximum
FG	<b>11.5</b>	2.018	1.453
IG	173.1	2.018	1.454
IG <sub>1000</sub>	37.3	2.018	1.453
IG <sub>3000</sub>	19.7	2.018	1.453
PG <sub>100</sub>	221.5	2.017	1.452

Table 3.1: Reconstruction time and accuracy of daily maximum and minimum temperatures on the withheld test set using different training techniques for the multi-output GP. PG<sub>#</sub> means the penalty  $\gamma = \#$  was used and IG<sub>#</sub> means a rank  $p = \#$  preconditioner was used.

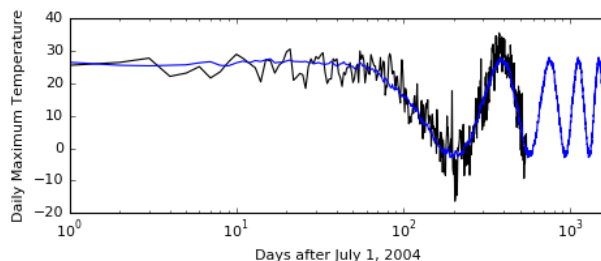


Figure 3.5: Forecast of Toronto maximum daily temperature. Actual observations are in black and the posterior mean is in blue. Note that the x-axis is on a log scale.

non-stationary behaviour. However, the model appears to struggle to predict the minimum temperature accurately during the summer months. This is perhaps the result of a localized climate effect caused by Moosonee’s close proximity to a large body of water (James bay). In the shaded region, the posterior distribution is also shown with respect to each output.

Figure 3.5 shows a multiple year forecast at the Toronto weather station where it is evident that the posterior mean is quite reasonable. For training, all data after July 1, 2004 was withheld along with all data at the Toronto weather station back to 1950. In this way the forecast makes full use of the spatiotemporal correlations, demonstrating the ability to forecast at a location where there is no historical data.

Figure 3.6 shows the log posterior variance of daily temperatures across Ontario. Also shown is the scattered distribution of weather stations across the province. As expected, the posterior variance rises away from the weather stations where the lack of data is reflected in our uncertainty.

Figure 3.7 shows the learned temporal kernel for the Ontario climate studies, found by maximization of the marginal likelihood (i.e., type-II inference). This stationary and decaying periodic kernel has a spike in correlation once every year indicating that the temperature on a given day is correlated to the same day of year the following years. On a daily scale, it is evident that the kernel amplitude decays very rapidly suggesting that daily temperature is only correlated a couple days into the future and past. On a yearly scale, it is evident that the kernel amplitude decays noticeably the first year and then essentially remains constant. This means that the temperature on a given day in 1950 is correlated to the same day of year 55 years later in 2005. This is consistent with our understanding of an annual climate.

Note that the kernel in fig. 3.7 is very different from the learned temporal kernel presented

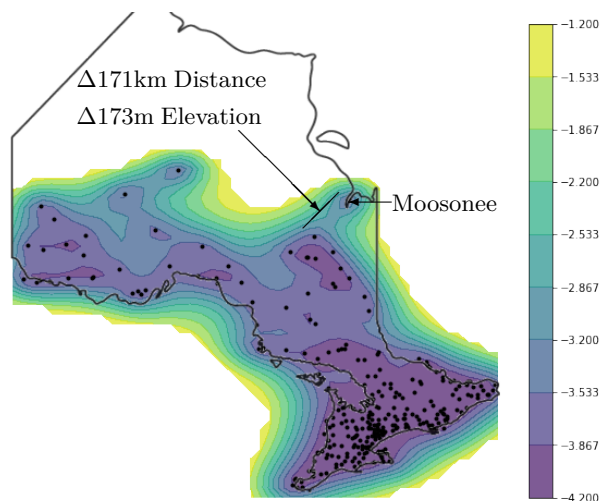


Figure 3.6: Log posterior variance of daily temperatures across Ontario. Black dots indicate weather station locations. Moosonee is indicated along with the distance and elevation change between its nearest neighbour, the Smoky Falls weather station. Temperature units are in degrees Celsius ( $^{\circ}\text{C}$ ).

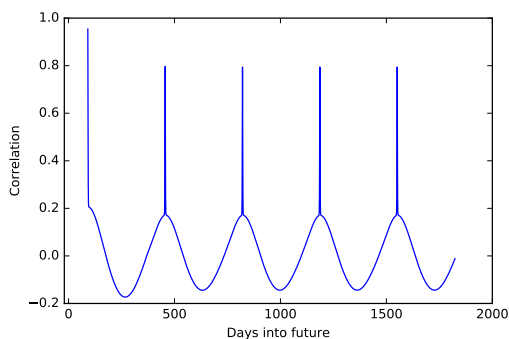


Figure 3.7: Learned temporal kernel for the Ontario climate study. This stationary and decaying periodic kernel has a spike in correlation once every year.

by Wilson et al. (2014) for a climate model constructed using pre-interpolated weather data on a Cartesian grid. While we initialized our temporal kernel to be similar to the kernel found by Wilson et al. (2014), it fit the data poorly and the marginal likelihood maximization procedure rapidly moved towards a different structure. The kernel in fig. 3.7 is also more consistent with our prior understanding of the climate. Perhaps the kernel presented by Wilson et al. (2014) was stuck in a local optimum of their marginal likelihood maximization procedure.

### 3.5.4 Particle Image Velocimetry Flow Reconstruction

In the analysis of fluid flows, particle image velocimetry (PIV) is a popular means of observing spatiotemporal fluid velocities where measurements can be resolved on a high-resolution spatial grid with a multi-kHz temporal repetition rate, giving datasets that can exceed hundreds of millions of observations from a single experiment. Datasets of this size require massively scalable techniques to enable accurate reconstructions. Further, PIV is the *de facto* standard two-dimensional flow-field measurement technique used for gas turbine combustors (GTCs) where it has become an important tool in design and analysis (e.g., (Arndt et al., 2015; Boxx

Method	Run Time	RMSE		
		$u$	$v$	$w$
GP-FG	<b>7.04 hrs</b>	<b>1.8264</b>	<b>2.2015</b>	<b>3.4697</b>
GP-IG	7.94 hrs	<b>1.8264</b>	<b>2.2015</b>	<b>3.4697</b>
GP-PG-10 <sup>2</sup>	8.00 hrs	1.8371	2.2119	3.4778
GP-PG-10 <sup>3</sup>	8.17 hrs	1.8274	2.2025	3.4704
GP-PG-10 <sup>4</sup>	8.38 hrs	1.8265	2.2016	3.4698
GP-PG-10 <sup>5</sup>	8.63 hrs	<b>1.8264</b>	<b>2.2015</b>	<b>3.4697</b>
GP-Local	47.17 hrs	1.9704	2.4879	3.7777
GPOD (Saini et al., 2016)	~40 hrs	2.1306	2.4071	3.8189

Table 3.2: Comparison of reconstruction time and accuracy on the PIV dataset with missing observations for different model types. The time listed is the total time to sequentially reconstruct all three velocity components ( $u$ ,  $v$ ,  $w$ ). The number listed after the penalize-gaps methods (i.e., GP-PG-#) indicates the value of the penalty parameter used. Top performance is identified in boldface. Note that the GPOD studies were run on different hardware than the other methods and so the timing results should be interpreted carefully.

et al., 2015; Caux-Brisebois et al., 2014; Ćosić et al., 2015; Fureby et al., 2007; Stopper et al., 2013; Temme et al., 2014)). However, gaps are abundant in PIV flow observations for GTC applications due to complex experimental setups, high-pressure flows, and a wide variety of gas densities which makes uniform particle seeding difficult while high-pressure liquid-fuelled combustion generates high background luminosity from flame chemiluminescence and soot (Saini et al., 2016). In this area, reconstruction of missing data is essential for the measurements to be utilized in data assimilation strategies, modal decompositions or other data-mining techniques that require continuous data in space or time (Gunes et al., 2006; Saini et al., 2016).

Gappy proper orthogonal decomposition (GPOD) is a popular technique used to approach PIV reconstruction problems (Everson and Sirovich, 1995; Raben et al., 2012). It is an efficient extension of proper orthogonal decomposition (POD) that provides high accuracy approximations for missing or erroneous data in measurements. POD is a data analysis method that provides a low dimensional approximation to high-dimensional processes in the form of a linear combination of basis functions. The POD basis functions, also called eigenmodes, are calculated from the data directly. The method extracts the most explanatory modes from the dataset and uses these for reconstruction under the assumption that the more energetic modes contain the majority of noise and are therefore excluded (Liang et al., 2002). The core of all GPOD techniques consists of an iterative implementation of POD where instantaneous gaps in the data set are filled-in. This prevents the POD from attempting to produce zeros (gaps) at the locations and times of missing data. After each POD calculation, the guess for the data in the gaps is updated based on a POD approximation using a particular number of modes. Interested readers should refer to (Saini et al., 2016) for a detailed discussion of GPOD.

We will compare the performance of GPOD to the developed Gaussian process algorithms on a large-scale PIV reconstruction problem. We consider a high vector yield particle image velocimetry (PIV) dataset taken from turbulent swirl flames in an atmospheric pressure gas turbine model combustor. This dataset was previously described by Caux-Brisebois et al. (2014), Saini et al. (2016), and Steinberg et al. (2013). To summarize, 10 kHz stereoscopic PIV vector fields were computed with a spatial vector spacing of approximately 0.5mm to give a



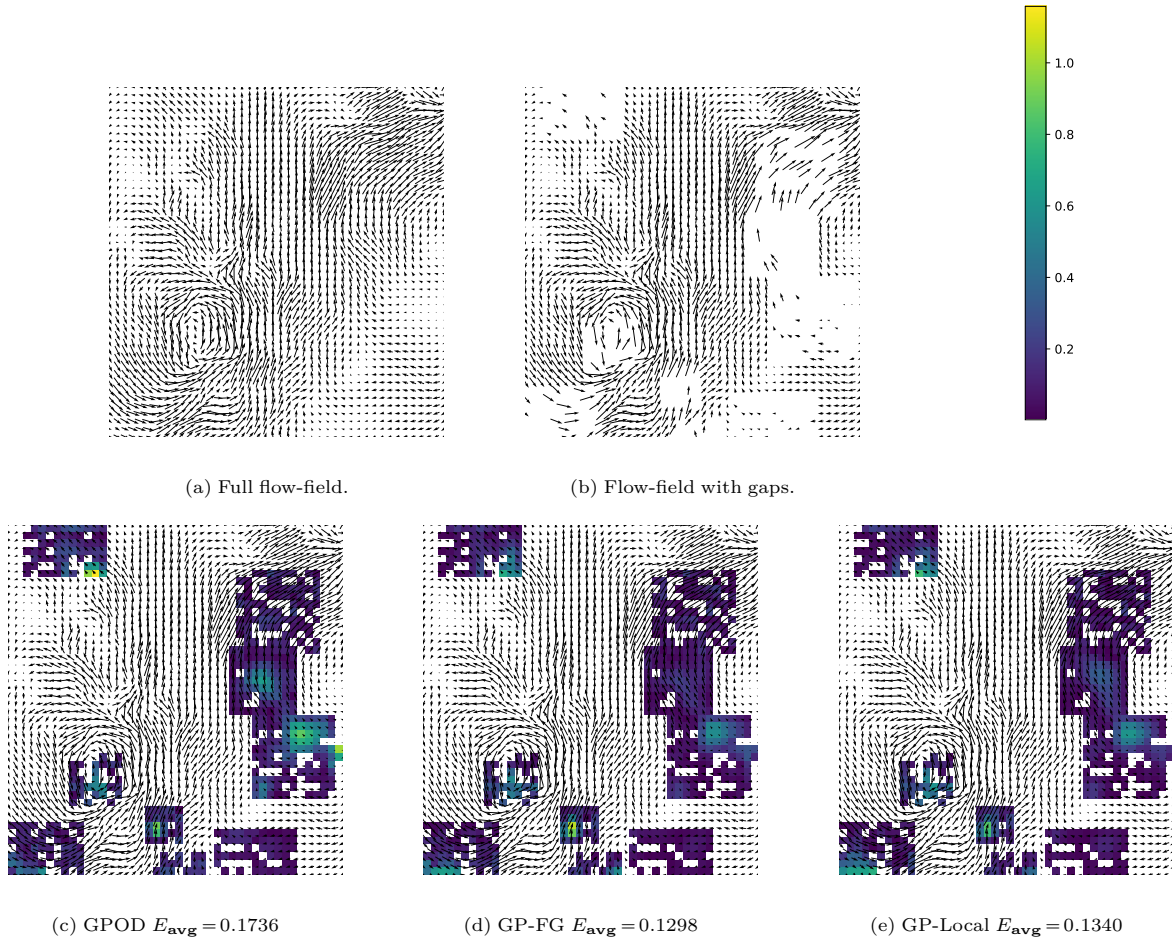


Figure 3.8: Instantaneous  $u, v$  flow velocities from a PIV temporal snapshot. Shown is the original flow field along with the artificially applied gaps and reconstructions of the gaps for various methods. Also shown in colour is the relative velocity vector error magnitude  $E(\tilde{u}, \tilde{v}) = \sqrt{(\tilde{u} - u)^2 + (\tilde{v} - v)^2} / \sqrt{u^2 + v^2}$ , where values with and without a tilde,  $\tilde{\cdot}$ , indicate the reconstructed and target velocity vectors, respectively, and  $E_{\text{avg}}$  gives the average value of  $E$  across all gaps in the figure.

dataset of flow velocity vectors (with velocity components  $u, v, w$  along three orthogonal spatial coordinates) on a Cartesian product grid of size  $m_x = 44$ ,  $m_y = 56$ , and  $m_t = 7870$ . The number of observations on the full grid is thus  $M = 44 \times 56 \times 7870 = 19,391,680$ . An instantaneous snapshot of the  $u, v$  flow vector field is shown in fig. 3.8a. The velocity component  $w$  is the velocity out-of-plane and is not shown in the image.

To compare various reconstruction techniques, we create artificial gaps in the dataset as shown in fig. 3.8b, train on the remaining observations and test the quality of the reconstruction on the held-out observations. We use the same gaps as Saini et al. (2016). Table 3.2 demonstrates the performance of the various model types discussed in the reconstruction of the PIV dataset for the three velocity components ( $u, v, w$ ). We compare various strategies to the gappy proper orthogonal decomposition (GPOD) reconstructions obtained by Saini et al. (2016) where computations were performed on a machine with an Intel(R) Core(TM) i5-2400 CPU and 16 GB of RAM.

All Gaussian process (GP) methods use the ARD exponentiated quadratic kernel shown in eq. (2.28). In addition to the GPOD method and the Gaussian process methods outlined in this chapter, we also compare to a method we call GP-Local which treats each temporal snapshot independently. This modelling choice is equivalent to choosing a prior covariance kernel  $k$  that gives zero correlation between two independent points in time and is commonly referred to as “local Kriging interpolation” in flow-reconstruction literature (Gunes et al., 2006). GP-local is implemented using the open-source GPy library (GPy, since 2012).

The penalize-gaps (GP-PG) method of section 3.3.1, and the developed ignore-gaps (GP-IG) method of section 3.3.2 and fill-gaps (GP-FG) method of section 3.3.3 were all implemented in python using the authors’ code. All GP computations were performed using sequential code (linked to a multi-threaded BLAS library) on a machine with two E5-2680 v3 processors and 128GB RAM. All GP training times include the time for hyperparameter estimation (with hyperparameters  $\theta = \{\ell_x, \ell_y, \ell_t, \sigma^2\}$ ) by maximization of the log-marginal likelihood as discussed in section 3.4.

It can be seen from the results obtained that the fill-gaps method (FG) was both the fastest and most accurate method considered. Not surprisingly, the ignore-gaps (IG) method achieves the same accuracy since it is constructing an identical model; however, the penalize-gaps method (PG) requires a high penalty parameter value to achieve this which affects conditioning and increases its reconstruction time. The timing of GPOD, while listed as significantly slower than the other methods, should be carefully interpreted since the experiments were performed on different hardware.

We can see that the reconstruction accuracy for  $u$  and  $v$  are similar, but the error in  $w$  is higher. This was similarly observed by Saini et al. (2016); they attributed this as a result of higher noise in the out-of-plane  $w$  measurements that is inherent to stereoscopic PIV techniques.

Figure 3.8 plots the reconstructed flow-field at an instantaneous time snapshot, comparing the reconstructions of GPOD, GP-Local, and GP-FG. This allows visualization of the error distributions between the methods. Both GP methods evidently out-perform the GPOD reconstruction, with the GP-FG construction improving slightly upon GP-local. Note that GP-IG and GP-PG with an infinite penalty parameter would all give identical reconstructions to GP-FG.

### 3.6 Concluding Remarks

In this chapter, we propose two novel and scalable exact GP regression algorithms for massive datasets on a partial grid. Both our algorithms make extensive use of Kronecker matrix algebra and we present novel preconditioners to accelerate computations. The proposed algorithms have modest memory requirements which enable us to showcase performance on a real-world climate modelling problem with over 3.7 million training points and on a synthetic

video dataset problem with over 1 billion training points. To the best of our knowledge, exact GP inference has not been attempted before on this scale.

Compared to the penalize-gaps strategy of Wilson et al. (2014), both developed formulations eliminate the need for any user defined parameters and both of them ultimately decrease the size of the problem to be solved through the use of sparse selection matrices, rather than a penalty approach. The fill-gaps technique from proposition 3.3 demonstrates a particularly interesting formulation wherein the gaps are “filled-in” *before* explicitly training the model. Additionally, both of the two developed techniques, fill-gaps and ignore-gaps, complement each other where the first is solving a problem whose size is equal to the number of gaps while the latter is solving a problem size equal to the number of training points. Thus, an appropriate method can always be selected based upon where the dataset lies in the “gappiness” spectrum. Further, we developed a Nyström-like preconditioner which is effective at accelerating computations.

Lastly, while the methods discussed assume some structure in the training data, we demonstrate through the climate modelling problem that with some additional insight, structure can be found in many applications to enable exact GP modelling on massive datasets.

## Chapter 4

# Scaling Gaussian Processes with the Nyström Approximation

In this chapter, we consider a novel Gaussian process approximation strategy and unlike the methods of chapter 3, this approach does not require any particular problem structure. Specifically, we introduce a novel kernel approximation strategy that consists of  $p$  eigenfunctions found using a Nyström approximation from a dense Cartesian product grid of inducing points. By exploiting algebraic properties of Kronecker and Khatri-Rao tensor products, computational complexity of the training procedure can be practically *independent* of the number of inducing points. This allows us to use arbitrarily many inducing points to achieve a globally accurate kernel approximation, even in high-dimensional problems. We benchmark our algorithms on real-world problems with up to two-million training points and  $10^{33}$  inducing points. The following paper was published from the contents of this chapter:

T. W. Evans and P. B. Nair (2018c). “Scalable Gaussian Processes with Grid-Structured Eigenfunctions (GP-GRIEF)”. in: *International Conference on Machine Learning*, pp. 1416–1425, long talk.

### 4.1 Introduction

As discussed in chapter 2, GP modelling is restricted to modestly sized datasets since training and inference require  $\mathcal{O}(n^3)$  time and  $\mathcal{O}(n^2)$  storage, where  $n$  is the number of training points (Rasmussen and Williams, 2006). This has motivated the development of approximate GP methods that use a set of  $m$  ( $\ll n$ ) inducing points to reduce time and memory requirements to  $\mathcal{O}(m^2n + m^3)$  and  $\mathcal{O}(mn)$ , respectively (Peng and Qi, 2015; Smola and Bartlett, 2001; Snelson and Ghahramani, 2006; Titsias, 2009). However, such techniques perform poorly if too few inducing points are used, and computational savings are lost on complex datasets that require  $m$  to be large.

Wilson and Nickisch (2015) exploited the structure of inducing points placed on a Cartesian product grid, allowing for  $m > n$  while dramatically reducing computational demands over

an exact GP. This inducing point structure enables significant performance gains in low dimensions, however, time and storage complexities scale *exponentially* with the dataset dimensionality, rendering the technique intractable for general learning problems unless a dimensionality reduction procedure is applied. In the present work, a Cartesian product grid of inducing points is also considered; however, we show that these computational bottlenecks can be eliminated by identifying and exploiting further structure of the resulting matrices. The proposed approach leads to a highly scalable algorithm which we call GP-GRIEF (Gaussian Processes with Grid-Structured Eigenfunctions). In general, the cost of log-marginal likelihood computations scale as  $\mathcal{O}(np^2 + dnp + dm^{3/d})$ , where  $d$  denotes the dataset dimensionality, and  $p$  is the number of eigenfunctions that we will describe next. We emphasize that our complexity is practically *independent* of  $m$ , which can generally be set arbitrarily high.

GP-GRIEF approximates an exact kernel as a finite sum of eigenfunctions which we accurately compute using the Nyström approximation conditioned on a huge number of inducing points. In other words, our model is sparse in the kernel eigenfunctions rather than the number of inducing points, which can greatly exceed the size of the training set due to the structure we introduce. This is attractive since it is well-known that eigenfunctions produce the most compact representation among orthogonal basis functions. Although the eigenfunctions used are approximate, we demonstrate convergence in the limit of large  $m$ . Additionally, our ability to fill out the input space with inducing points enables accurate global approximations of the eigenfunctions, even at test locations far from the training data. These basis functions also live in a reproducing kernel Hilbert space, unlike some other sparse GPs whose bases have a pre-specified form (e.g., Lázaro-Gredilla et al. (2010)). We summarize our main contributions below

- We break the *curse of dimensionality* incurred by placing inducing points on a full Cartesian product grid. Typically, a grid of inducing points results in a computational complexity that scales exponentially in  $d$ ; however, we reduce this complexity to *linear* in  $d$  by exploiting algebraic properties of Kronecker and Khatri-Rao products.
- We practically eliminate dependence of the inducing point quantity,  $m$ , on computational complexity. This allows us to choose  $m \gg n$  to provide a highly accurate kernel approximation, even at locations far from the training data.
- We show that the Nyström eigenfunction approximation becomes exact for large  $m$ , which is achievable thanks to the structure and algebra we introduce.
- Applications of the developed algebra are discussed to enable the extension of structured kernel interpolation methods for high-dimensional problems. We also develop an efficient preconditioner for general kernel matrices.
- Finally, we demonstrate accurate inference on real-world datasets with up to 2 million training points and  $m = 10^{33}$  inducing points.

We begin by introducing an eigenfunction kernel approximation in section 4.2. Section 4.3 demonstrates why we should use many inducing points and subsequently develops the algebra necessary to make  $m \gg n$  efficient and stable. We finish with numerical studies in section 4.4, demonstrating the performance of GP-GRIEF on real-world datasets.

## 4.2 Eigenfunction Kernel Approximation

We consider a compact representation of the GP prior using a truncated Mercer expansion of the kernel  $k$ . We use the first  $p$  eigenfunctions which we approximate numerically using a Nyström approximation (Peng and Qi, 2015)

$$\tilde{k}(\mathbf{x}, \mathbf{z}) = \sum_{i=1}^p \underbrace{(\lambda_i^{-\frac{1}{2}} \mathbf{K}_{\mathbf{x}, \mathbf{U}} \mathbf{q}_i)}_{\phi_i(\mathbf{x})} \underbrace{(\lambda_i^{-\frac{1}{2}} \mathbf{K}_{\mathbf{z}, \mathbf{U}} \mathbf{q}_i)}_{\phi_i(\mathbf{z})} = \mathbf{K}_{\mathbf{x}, \mathbf{U}} \mathbf{Q} \mathbf{S}_p^T \mathbf{\Lambda}_p^{-1} \mathbf{S}_p \mathbf{Q}^T \mathbf{K}_{\mathbf{U}, \mathbf{z}} \approx k(\mathbf{x}, \mathbf{z}), \quad (4.1)$$

where  $\mathbf{U} = \{\mathbf{u}_i \in \mathbb{R}^d\}_{i=1}^m$  refers to the set of  $m$  inducing point locations;  $\mathbf{\Lambda}, \mathbf{Q} \in \mathbb{R}^{m \times m}$  are diagonal and unitary matrices containing the eigenvalues and eigenvectors of  $\mathbf{K}_{\mathbf{U}, \mathbf{U}}$ , respectively;  $\lambda_i$  and  $\mathbf{q}_i$  denote the  $i$ th largest eigenvalue and corresponding eigenvector of  $\mathbf{K}_{\mathbf{U}, \mathbf{U}}$ , respectively;  $\mathbf{S}_p \in \mathbb{R}^{p \times m}$  is a sparse selection matrix where the  $i$ th row of  $\mathbf{S}_p$ , denoted  $\mathbf{S}_p(i, :)$ , contains one value set to unity in the column corresponding to the index of the  $i$ th largest value on the diagonal of  $\mathbf{\Lambda}$ ; and we use the shorthand notation  $\mathbf{\Lambda}_p = \mathbf{S}_p \mathbf{\Lambda} \mathbf{S}_p^T = \text{diag}(\boldsymbol{\lambda}_p) \in \mathbb{R}^{p \times p}$  to denote a diagonal matrix containing the  $p$  largest eigenvalues of  $\mathbf{K}_{\mathbf{U}, \mathbf{U}}$ , sorted in descending order. In this chapter, we use the notation  $[\mathbf{K}_{\mathbf{A}, \mathbf{B}}]_{i,j} = k(\mathbf{a}_i, \mathbf{b}_j)$  such that  $\mathbf{K}_{\mathbf{X}, \mathbf{X}} \in \mathbb{R}^{n \times n}$  is the kernel covariance matrix evaluated on the training dataset and  $\mathbf{K}_{\mathbf{x}, \mathbf{U}} \in \mathbb{R}^{1 \times m}$  is the cross-covariance between  $\mathbf{x}$  and the set of inducing points  $\mathbf{U}$ , for instance. Also,  $\phi_i(\mathbf{x})$  is the numerical approximation of the  $i$ th eigenfunction evaluated at the input  $\mathbf{x}$ , scaled by the square root of the  $i$ th eigenvalue. We only explicitly compute this scaled eigenfunction for numerical stability, as we will discuss later. Using the kernel  $\tilde{k}$ , the prior covariance matrix on the training set becomes

$$\tilde{\mathbf{K}}_{\mathbf{X}, \mathbf{X}} = \underbrace{\mathbf{K}_{\mathbf{X}, \mathbf{U}} \mathbf{Q} \mathbf{S}_p^T \mathbf{\Lambda}_p^{-\frac{1}{2}}}_{\boldsymbol{\Phi}} \underbrace{\mathbf{\Lambda}_p^{-\frac{1}{2}} \mathbf{S}_p \mathbf{Q}^T \mathbf{K}_{\mathbf{U}, \mathbf{X}}}_{\boldsymbol{\Phi}^T}, \quad (4.2)$$

where the columns of  $\boldsymbol{\Phi} \in \mathbb{R}^{n \times p}$  contain the  $p$  scaled eigenfunctions of our kernel evaluated on the training set. Observe that if  $\mathbf{U}$  is randomly sampled from the set of training inputs  $\mathbf{X}$ , then  $\tilde{\mathbf{K}}_{\mathbf{X}, \mathbf{X}}$  is the same covariance matrix from the ‘‘Nyström method’’ of Williams and Seeger (2001), however, since we have replaced the kernel and not just the covariance matrix, we recover a valid probabilistic model (Peng and Qi, 2015).

While  $\tilde{\mathbf{K}}_{\mathbf{X}, \mathbf{X}}$  has a rank of at most  $p$ , Peng and Qi (2015) show how a correction can be added to  $\tilde{k}$  (eq. (4.1)) to give a full rank covariance matrix (provided  $k$  does also). The resulting GP will be non-degenerate. We can write this correction as  $\delta(\mathbf{x} - \mathbf{z})(k(\mathbf{x}, \mathbf{z}) - \tilde{k}(\mathbf{x}, \mathbf{z}))$ , where  $\delta(a) = 1$  if  $a = 0$ , else 0. This correction term was also discussed in eq. (2.36) and does not affect

the computational complexity of GP training; however, we find it does not generally improve performance over the unmodified  $\tilde{k}$ . We do not consider this correction in further discussion.

### 4.3 Grid-Structured Eigenfunctions (GRIEF)

Previous work employing Nyström approximations in kernel methods require  $m$  to be small (often  $\ll n$ ) to yield computational benefits. As a result, the choice of inducing point locations,  $\mathbf{U}$ , has a great influence on the approximation accuracy, and many techniques have been proposed to choose  $\mathbf{U}$  effectively (Belabbas and Wolfe, 2009; Drineas and Mahoney, 2005; Gittens and Mahoney, 2013; Kumar et al., 2012; Li et al., 2016a; Musco and Musco, 2017; Smola and Schölkopf, 2000; Wang and Zhang, 2013; Zhang et al., 2008). In this work, we would instead like to use *so many* inducing points that carefully optimizing the distribution of  $\mathbf{U}$  is unnecessary. We will even consider  $m \gg n$ . The following result shows how an eigenfunction approximation can be improved by using many inducing points.

**Theorem 4.1.** *If the  $i$ th eigenvalue of  $k$  is simple and non-zero and  $\mathbf{U} \supset \mathbf{X}$ , a Nyström approximation of the  $i$ th kernel eigenfunction converges in the limit of large  $m$ ,*

$$\mathbf{q}_i^{(n)} = \lim_{m \rightarrow \infty} \sqrt{\frac{m}{n}} \frac{1}{\lambda_i^{(m)}} \mathbf{K}_{\mathbf{X}, \mathbf{U}} \mathbf{q}_i^{(m)}, \quad (4.3)$$

where  $\lambda_i^{(m)} \in \mathbb{R}$  and  $\mathbf{q}_i^{(m)} \in \mathbb{R}^m$  are the  $i$ th largest eigenvalue and corresponding eigenvector of  $\mathbf{K}_{\mathbf{U}, \mathbf{U}}$ , respectively.  $\mathbf{q}_i^{(n)}$  is the kernel eigenfunction corresponding to the  $i$ th largest eigenvalue, evaluated on the set  $\mathbf{X}$ .

*Proof.* We begin by constructing a Nyström approximation of the eigenfunction evaluated on  $\mathbf{U}$ , using  $\mathbf{X}$  as inducing points. From theorem 3.5 of Baker (1977), as  $m \rightarrow \infty$ ,

$$\mathbf{q}_i^{(m)} = \sqrt{\frac{n}{m}} \frac{1}{\lambda_i^{(n)}} \mathbf{K}_{\mathbf{U}, \mathbf{X}} \mathbf{q}_i^{(n)}, \quad (4.4)$$

where we assume that the  $i$ th eigenvalue of  $k$  is simple and non-zero. Multiplying both sides by  $\mathbf{K}_{\mathbf{X}, \mathbf{U}} \mathbf{K}_{\mathbf{U}, \mathbf{U}}^{-1}$ ,

$$\mathbf{K}_{\mathbf{X}, \mathbf{U}} \mathbf{K}_{\mathbf{U}, \mathbf{U}}^{-1} \mathbf{q}_i^{(m)} = \sqrt{\frac{n}{m}} \frac{1}{\lambda_i^{(n)}} \mathbf{K}_{\mathbf{X}, \mathbf{U}} \mathbf{K}_{\mathbf{U}, \mathbf{U}}^{-1} \mathbf{K}_{\mathbf{U}, \mathbf{X}} \mathbf{q}_i^{(n)}. \quad (4.5)$$

Since  $\mathbf{U} \supset \mathbf{X}$ ,  $\mathbf{K}_{\mathbf{X}, \mathbf{U}} = \mathbf{S}_n \mathbf{K}_{\mathbf{U}, \mathbf{U}}$  is a subset of the rows of  $\mathbf{K}_{\mathbf{U}, \mathbf{U}}$ , where  $\mathbf{S}_n \in \mathbb{R}^{n \times m}$  is a selection matrix. We can write  $\mathbf{K}_{\mathbf{X}, \mathbf{U}} \mathbf{K}_{\mathbf{U}, \mathbf{U}}^{-1} \mathbf{K}_{\mathbf{U}, \mathbf{X}} = \mathbf{S}_n \mathbf{K}_{\mathbf{U}, \mathbf{U}} \mathbf{K}_{\mathbf{U}, \mathbf{U}}^{-1} \mathbf{K}_{\mathbf{U}, \mathbf{X}} \mathbf{S}_n^T = \mathbf{S}_n \mathbf{K}_{\mathbf{U}, \mathbf{U}} \mathbf{S}_n^T = \mathbf{K}_{\mathbf{X}, \mathbf{X}}$ . Additionally, since the eigenvector  $\mathbf{q}_i^{(m)}$  satisfies,  $\mathbf{K}_{\mathbf{U}, \mathbf{U}}^{-1} \mathbf{q}_i^{(m)} = \frac{1}{\lambda_i^{(m)}} \mathbf{q}_i^{(m)}$ , we get

$$\frac{1}{\lambda_i^{(m)}} \mathbf{K}_{\mathbf{X}, \mathbf{U}} \mathbf{q}_i^{(m)} = \sqrt{\frac{n}{m}} \frac{1}{\lambda_i^{(n)}} \mathbf{K}_{\mathbf{X}, \mathbf{X}} \mathbf{q}_i^{(n)}. \quad (4.6)$$

Noting that  $\mathbf{K}_{X,X}\mathbf{q}_i^{(n)} = \lambda_i^{(n)}\mathbf{q}_i^{(n)}$  completes the proof.  $\square$

For multiple eigenvalues, it can similarly be shown that the  $i$ th approximated eigenfunction converges to lie within the linear space of eigenfunctions corresponding to the  $i$ th eigenvalue of  $k$  as  $m \rightarrow \infty$ .

We can use a large  $m$  by distributing inducing points on a Cartesian tensor product grid<sup>1</sup>. Saatçi (2011) demonstrated efficient GP inference when training points are distributed in this way by exploiting Kronecker matrix algebra. We will assume this grid structure for our inducing points, i.e.,  $\mathbf{U}$  will form a grid. If the covariance kernel satisfies the product correlation rule (as many popular multidimensional kernels do), i.e.,  $k(\mathbf{x}, \mathbf{z}) = \prod_{i=1}^d k_i(x_i, z_i)$ , then  $\mathbf{K}_{U,U} \in \mathbb{R}^{m \times m}$  inherits the Kronecker product form  $\mathbf{K}_{U,U} = \bigotimes_{i=1}^d \mathbf{K}_{U,U}^{(i)}$ , where  $\bigotimes$  is the Kronecker product (Van Loan, 2000).  $\mathbf{K}_{U,U}^{(i)} \in \mathbb{R}^{\bar{m} \times \bar{m}}$  are one-dimensional kernel covariance matrices for a slice of the input space grid along the  $i$ th dimension, and  $\bar{m} = \sqrt[d]{m} \approx \mathcal{O}(10)$  is the number of inducing points we choose along each dimension of the full grid. It is evident that the Kronecker product leads to a large, expansive matrix from smaller ones, therefore, it is very advantageous to manipulate and store these small matrices without “unpacking” them, or explicitly computing the Kronecker product. Exploiting this structure decreases the storage of  $\mathbf{K}_{U,U}$  from  $\mathcal{O}(m^2) \rightarrow \mathcal{O}(dm^{2/d}) = \mathcal{O}(d\bar{m}^2)$ , and the cost of matrix-vector products with  $\mathbf{K}_{U,U}$  from  $\mathcal{O}(m^2) \rightarrow \mathcal{O}(dm^{(d+1)/d}) = \mathcal{O}(d\bar{m}^{d+1})$ . Additionally, the cost of the eigendecomposition of  $\mathbf{K}_{U,U}$  decreases from  $\mathcal{O}(m^3) \rightarrow \mathcal{O}(dm^{3/d}) = \mathcal{O}(d\bar{m}^3)$ , and the eigenvector matrix  $\mathbf{Q} = \bigotimes_{i=1}^d \mathbf{Q}^{(i)}$  and eigenvalue matrix  $\mathbf{\Lambda} = \text{diag}(\bigotimes_{i=1}^d \boldsymbol{\lambda}^{(i)})$  both inherit a Kronecker product structure, enabling matrix-vector products with  $\tilde{\mathbf{K}}_{X,X}$  in  $\mathcal{O}(d\bar{m}^{d+1})$  operations. See appendix A for an overview of Kronecker matrix algebra.

In low-dimensions, exploiting the Kronecker product structure of  $\mathbf{K}_{U,U} = \bigotimes_{i=1}^d \mathbf{K}_{U,U}^{(i)}$  can be greatly advantageous; however, we can immediately see from the above complexities that the cost of matrix-vector products<sup>2</sup> with  $\tilde{\mathbf{K}}_{X,X}$  increases *exponentially* in  $d$ . The storage requirements will similarly increase exponentially since a vector of length  $m = \bar{m}^d$  needs to be stored when a matrix-vector product is made with  $\mathbf{Q} = \bigotimes_{i=1}^d \mathbf{Q}^{(i)}$ , and  $\mathbf{K}_{X,U}$  requires  $\mathcal{O}(\bar{m}^d n)$  storage. This poor scaling poses a serious impediment to the application of this approach to high-dimensional datasets.

We now show how to dramatically decrease time and storage requirements from exponential to *linear* in  $d$  by identifying further matrix structure in our problem.

We begin by identifying structure in the exact cross-covariance between train (or test) points and inducing points. These matrices, e.g.,  $\mathbf{K}_{X,U}$ , admit a row-partitioned Khatri-Rao

<sup>1</sup>We want  $\mathbf{U}$  to be sampled from the same distribution as the training data. Approximating the data distribution by placing  $\mathbf{U}$  on a grid is easy to do by various means as a quick preprocessing step since it implicitly assumes that the data distribution factorizes between inputs.

<sup>2</sup>We assume that a conjugate gradient method would be employed for GP training requiring matrix-vector products. Alternative formulations would require columns of  $\mathbf{Q} = \bigotimes_{i=1}^d \mathbf{Q}^{(i)}$  to be expanded which similarly scales exponentially ( $\mathcal{O}(\bar{m}^d)$ ).



product structure as follows (Nickson et al., 2015):

$$\mathbf{K}_{X,U} = \underset{i=1}{*}^d \mathbf{K}_{X,U}^{(i)} = \begin{pmatrix} \mathbf{K}_{X,U}^{(1)}(1,:) \otimes \mathbf{K}_{X,U}^{(2)}(1,:) \otimes \cdots \otimes \mathbf{K}_{X,U}^{(d)}(1,:) \\ \mathbf{K}_{X,U}^{(1)}(2,:) \otimes \mathbf{K}_{X,U}^{(2)}(2,:) \otimes \cdots \otimes \mathbf{K}_{X,U}^{(d)}(2,:) \\ \vdots \\ \mathbf{K}_{X,U}^{(1)}(n,:) \otimes \mathbf{K}_{X,U}^{(2)}(n,:) \otimes \cdots \otimes \mathbf{K}_{X,U}^{(d)}(n,:) \end{pmatrix}, \quad (4.7)$$

where  $*$  is the Khatri-Rao product whose computation gives a block Kronecker product matrix (Liu and Trenkler, 2008). We will always mention how Khatri-Rao product blocks are partitioned. Since  $\mathbf{K}_{X,U}^{(i)}$  are only of size  $n \times \bar{m}$ , the storage of  $\mathbf{K}_{X,U}$  has decreased from exponential to linear in  $d$ :  $\mathcal{O}(\bar{m}^d n) \rightarrow \mathcal{O}(dn\bar{m})$ . We also observe that the selection matrix  $\mathbf{S}_p = \underset{i=1}{*}^d \mathbf{S}_p^{(i)}$  can be written as a row-partitioned Khatri-Rao product matrix where each sub-matrix contains one non-zero per row. Further, by exploiting both Kronecker and Khatri-Rao matrix algebra, our main result below shows that  $\mathbf{K}_{X,U} \mathbf{Q} \mathbf{S}_p^T$  can be computed in  $\mathcal{O}(dnp)$  time. This is a dramatic reduction over the cost of  $\mathcal{O}(\bar{m}^d np)$  time incurred by the naive approach.

**Theorem 4.2.** *The product of a row-partitioned Khatri-Rao matrix  $\mathbf{K}_{X,U} = \underset{i=1}{*}^d \mathbf{K}_{X,U}^{(i)} \in \mathbb{R}^{n \times \bar{m}^d}$ , a Kronecker product matrix  $\mathbf{Q} = \bigotimes_{i=1}^d \mathbf{Q}^{(i)} \in \mathbb{R}^{\bar{m}^d \times \bar{m}^d}$ , and a column-partitioned Khatri-Rao matrix  $\mathbf{S}_p^T = \underset{i=1}{*}^d (\mathbf{S}_p^{(i)})^T \in \mathbb{R}^{\bar{m}^d \times p}$  can be computed as follows:*

$$\mathbf{K}_{X,U} \mathbf{Q} \mathbf{S}_p^T = \bigodot_{i=1}^d \mathbf{K}_{X,U}^{(i)} \mathbf{Q}^{(i)} (\mathbf{S}_p^{(i)})^T, \quad (4.8)$$

where  $\odot$  is the (elementwise) Hadamard product. This computation only requires products of the smaller matrices  $\mathbf{K}_{X,U}^{(i)} \in \mathbb{R}^{n \times \bar{m}}$ ,  $\mathbf{Q}^{(i)} \in \mathbb{R}^{\bar{m} \times \bar{m}}$  and  $\mathbf{S}_p^{(i)} \in \mathbb{R}^{p \times \bar{m}}$ .

*Proof.* First, observe that  $\mathbf{K}_{X,U} \mathbf{Q} = \underset{i=1}{*}^d \mathbf{K}_{X,U}^{(i)} \mathbf{Q}^{(i)} = \underset{i=1}{*}^d \mathbf{R}^{(i)}$  is a row-partitioned Khatri-Rao product matrix using theorem 2 of (Liu and Trenkler, 2008). Now we must compute a matrix product of row- and column-partitioned Khatri-Rao matrices. We observe that each element of this matrix product is an inner product between two Kronecker product vectors, i.e.,  $[\mathbf{K}_{X,U} \mathbf{Q} \mathbf{S}_p^T]_{ij} = (\bigotimes_{l=1}^d \mathbf{R}^{(l)}(i,:)) (\bigotimes_{l=1}^d \mathbf{S}_p^{T(l)}(:,j)) = \prod_{l=1}^d \mathbf{R}^{(l)}(i,:) \mathbf{S}_p^{T(l)}(:,j)$ . Writing this in matrix form completes the proof.  $\square$

If all the sub-matrices were dense,  $\mathcal{O}(dn\bar{m}p)$  time would be required to compute  $\mathbf{K}_{X,U} \mathbf{Q} \mathbf{S}_p^T$ ; however, since  $\mathbf{S}_p$  is a sparse selection matrix with one non-zero per row, computation requires just  $\mathcal{O}(dn \max(p, \bar{m}c)) \approx \mathcal{O}(dnp)$  time. We achieve this time by computing only the necessary columns of  $\mathbf{K}_{X,U}^{(i)} \mathbf{Q}^{(i)}$  and avoiding redundant computations. The constant  $c$  is the average number of non-zeros columns in each of  $\{\mathbf{S}_p^{(i)}\}_{i=1}^d$  which is typically  $\mathcal{O}(1)$ , but may be  $\bar{m}$  in the worst case. In other words, the time complexity of our approach is effectively *independent* of the number of inducing points. Even in the rare worst case where  $c = \bar{m}$  and  $\bar{m}^2 > p$ , the scaling is weak;  $\mathcal{O}(dnm^{2/d})$ .

What results is a kernel composed of basis eigenfunctions that are accurately approximated on a grid of inducing points using a Nyström approximation. Although  $m$  increases

exponentially in  $d$  given this inducing point structure, the cost of GP training and inference is not affected. We refer to the resulting model GP-GRIEF (GP with GRId-structured EigenFunctions). Completing the computations required for GP training and inference require straightforward application of the matrix-determinant and matrix-inversion lemmas which are presented in eq. (5.3).

### 4.3.1 Eigenvalue Search

What has not been addressed is how to form  $\mathbf{S}_p = \ast_{i=1}^d \mathbf{S}_p^{(i)}$  and compute  $\boldsymbol{\lambda}_p$  efficiently. This requires finding the index locations and values of the largest  $p$  eigenvalues in a vector of length  $m = \bar{m}^d$ . In high dimensions, this task is daunting considering  $m$  can easily exceed the number of atoms in the observable universe in practice. Fortunately, the resulting vector of eigenvalues,  $\text{diag}(\boldsymbol{\Lambda}) = \otimes_{i=1}^d \boldsymbol{\lambda}^{(i)}$ , has a Kronecker product structure which we can exploit to develop a fast search algorithm that requires only  $\mathcal{O}(d\bar{m}p)$  time. To do this, we compute a truncated Kronecker product expansion by keeping only the  $p$  largest values after each sequential Kronecker product such that only Kronecker products between length  $p$  and length  $\bar{m}$  vectors are computed. Algorithm 4.1 outlines a more numerically stable version of this search strategy that computes the log of the eigenvalues and also demonstrates how  $\mathbf{S}_p$  is computed.

---

**Algorithm 4.1** Computes  $\mathbf{S}_p$ , and  $\log(\boldsymbol{\lambda}_p)$  (the log of the  $p$  largest eigenvalues of  $\mathbf{K}_{\mathbf{U},\mathbf{U}}$ ). We use zero-based array indexing,  $\text{mod}(a,b)$  computes  $a \bmod b$ ,  $|\mathbf{a}|$  computes the length of  $\mathbf{a}$ ,  $\text{sort}_b(\mathbf{a})$  returns the  $\min(|\mathbf{a}|, b)$  largest elements of  $\mathbf{a}$  in descending order, as well as the indices of these elements in  $\mathbf{a}$ , and  $\lfloor \mathbf{a} \rfloor$  computes the floor of the elements in  $\mathbf{a}$ .

---

**Input:**  $\{\boldsymbol{\lambda}^{(i)} \in \mathbb{R}^{\bar{m}}\}_{i=1}^d$   
**Output:**  $\{\mathbf{S}_p^{(i)} \in \mathbb{R}^{p \times \bar{m}}\}_{i=1}^d$  &  $\log(\boldsymbol{\lambda}_p) \in \mathbb{R}^p$   
 $\log(\boldsymbol{\lambda}_p)$ ,  $\text{idxs} = \text{sort}_p(\log(\boldsymbol{\lambda}^{(1)}))$   
**for**  $i = 2$  **to**  $d$  **do**  
     $\log(\boldsymbol{\lambda}_p)$ ,  $\text{ord} = \text{sort}_p(\log(\boldsymbol{\lambda}_p) \otimes \mathbf{1}_{\bar{m}} + \mathbf{1}_{|\log(\boldsymbol{\lambda}_p)|} \otimes \log(\boldsymbol{\lambda}^{(i)}))$   
     $\text{idxs} = \left[ \text{idxs}(\lfloor \text{ord}/\bar{m} \rfloor, :), \text{mod}(\text{ord}, \bar{m}) \right]$   
**end for**  
 $\{\mathbf{S}_p^{(i)} = \mathbf{I}_{\bar{m}}(\text{idxs}(:, i-1), :)\}_{i=1}^d$

---

### 4.3.2 Stable Computation in High Dimensions

Direct use of theorem 4.2 may lead to finite-precision rounding inaccuracies and overflow errors in high dimensions because of the Hadamard product over  $d$  matrices. We can write a more numerically stable version of this algorithm by taking the log of eq. (4.8), allowing us to write the computation as a sum of  $d$  matrices, rather than a product

$$\mathbf{K}_{\mathbf{X},\mathbf{U}} \mathbf{Q} \mathbf{S}_p^T = \odot_{i=1}^d \text{sign}(\mathbf{B}^{(i)}) \odot \exp\left(\sum_{i=1}^d \log(\text{abs } \mathbf{B}^{(i)})\right),$$

where  $\mathbf{B}^{(i)} = \mathbf{K}_{X,U}^{(i)} \mathbf{Q}^{(i)} (\mathbf{S}_p^{(i)})^T$ , and  $\exp$ ,  $\log$  are computed element-wise. While the sign matrix is the Hadamard product of  $d$  matrices, it contains only  $\{-1, 0, 1\}$  so it is not susceptible to numerical issues. Also, when the sign of an element is zero, we do not compute the log. Unfortunately, the  $\exp$  computation can still lead to numerical issues; however,  $\Phi$  suffers less because of the rescaling provided by the eigenvalues (i.e., elements of  $\Phi$  are the quotient of possibly very large or small values). Since all eigenvalues are positive, we can stably compute  $\Phi$  as follows:

$$\Phi = (\mathbf{K}_{X,U} \mathbf{Q} \mathbf{S}_p^T) \Lambda_p^{-\frac{1}{2}} = \bigodot_{i=1}^d \text{sign}(\mathbf{B}^{(i)}) \odot \exp\left(\sum_{i=1}^d \log(\text{abs } \mathbf{B}^{(i)}) - \frac{1}{2} \mathbf{1}_n \log(\boldsymbol{\lambda}_p)^T\right), \quad (4.9)$$

where  $\log(\boldsymbol{\lambda}_p)$  is computed by algorithm 4.1.

### 4.3.3 Preconditioning Applications

As a side remark, we discuss the application of  $(\tilde{\mathbf{K}}_{X,X} + \sigma^2 \mathbf{I}_n)^{-1}$  as a preconditioner for the exact kernel matrix  $\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_n$  in moderately sized problems where  $\mathcal{O}(n^2)$  storage is not prohibitive. The use of  $\tilde{\mathbf{K}}_{X,X}$  for matrix preconditioning was explored with notable empirical success by Cutajar et al. (2016) where a sub-set of training data was used as inducing points giving  $U \subset X$  and  $m < n$ . By theorem 4.1, we know that the Nyström approximation converges for large  $m$ , and we have shown that we can accommodate  $m \gg n$  to provide an accurate low-rank kernel matrix approximation.

### 4.3.4 SKI Applications

As a further aside, we discuss how the developed algebra can be applied in a general kernel interpolation setting. Wilson and Nickisch (2015) introduced a kernel interpolation perspective to unify inducing point methods wherein the kernel is interpolated from a covariance matrix on the inducing points. For instance, the subset of regressors (SoR) method (Quiñonero-Candela and Rasmussen, 2005; Silverman, 1985) can be viewed as a zero-mean GP interpolant of the true kernel while Wilson and Nickisch (2015) proposed a sparse approximate interpolant. We can denote the interpolated covariance matrix as  $\mathbf{E} \mathbf{K}_{U,U} \mathbf{E}^T$ , where  $\mathbf{E} \in \mathbb{R}^{n \times m}$  is the interpolation matrix.

In structured kernel interpolation (SKI) the inducing points form a grid such that  $\mathbf{K}_{U,U}$  inherits a Kronecker product form. This can provide dramatic computational advantages; however, SKI suffers from the exponential scaling discussed earlier and so is recommended only for very low-dimensional problems,  $d \leq 5$  (Wilson et al., 2016). We observe that in the case of the GP interpolant (e.g., SoR), as well as the sparse interpolant suggested by Wilson and Nickisch (2015), the interpolation matrix  $\mathbf{E}$  inherits a row-partitioned Khatri-Rao structure. This enables direct use of theorem 4.2 to reduce the exponential scaling in  $d$  to a linear scaling, and allows SKI to scale to high-dimensional problems. However, time complexity would scale quadratically in  $n$ , unlike the proposed GRIEF method.

## 4.4 Experimental Studies

---

**Algorithm 4.2** Computation of the log marginal likelihood for GP-GRIEF.

---

**Input:** Grid of inducing points  $\mathbf{U} = \{\mathbf{u}_i \in \mathbb{R}^d\}_{i=1}^m$ , base kernel  $k_i$  for  $i = 1, \dots, d$  with hyperparameters  $\boldsymbol{\theta}$ .

**Output:** Log marginal likelihood  $\log\Pr(\mathbf{y}|\boldsymbol{\theta})$ .

Compute  $\mathbf{K}_{\mathbf{U},\mathbf{U}}^{(i)} \in \mathbb{R}^{\bar{m} \times \bar{m}}$ , using  $k_i$  for  $i = 1, \dots, d$ .

Compute the eigenvalues of  $\{\mathbf{K}_{\mathbf{U},\mathbf{U}}^{(i)}\}_{i=1}^d$  as  $\{\boldsymbol{\lambda}^{(i)} \in \mathbb{R}^{\bar{m}}\}_{i=1}^d$  and the eigenvectors in the columns of  $\{\mathbf{Q}^{(i)}\}_{i=1}^d$ .

Compute  $\{\mathbf{S}_p^{(i)} \in \mathbb{R}^{p \times \bar{m}}\}_{i=1}^d$  and  $\log(\boldsymbol{\lambda}_p) \in \mathbb{R}^p$  using algorithm 4.1.

Compute  $\boldsymbol{\Phi} = (\mathbf{K}_{\mathbf{X},\mathbf{U}} \mathbf{Q} \mathbf{S}_p^T) \boldsymbol{\Lambda}_p^{-\frac{1}{2}}$  using eq. (4.9).

Compute  $\log\Pr(\mathbf{y}|\boldsymbol{\theta})$  using eq. (2.29).

---

This section empirically assesses the proposed GP-GRIEF method. Algorithm 4.2 summarizes the computation of the Gaussian process log-marginal likelihood using the GP-GRIEF kernel. This demonstrates the operations necessary to estimate or marginalize the base kernel hyperparameters which is typically the most expensive component of GP modelling. The predictive posterior can also be computed using similar operations by replacing the computation of  $\boldsymbol{\Phi}$  with the basis functions evaluated at a test point and with the use of eq. (2.16). A python implementation of the methods discussed in this chapter along with several tutorials can be found at [https://github.com/treforevans/gp\\_grief](https://github.com/treforevans/gp_grief).

### 4.4.1 Two-Dimensional Visualization

Figure 4.1 shows a comparison between GP-GRIEF and the Variational Free Energy (VFE) inducing point approximation (Titsias, 2009) on a two-dimensional test problem with  $n = 10$  training points generated by the function  $f(\mathbf{x}) = \sin(x_1)\sin(x_2)$  and corrupted with  $\mathcal{N}(0, 0.1)$  noise. For both models, we use a squared-exponential base kernel, and we estimate the kernel lengthscale and noise variance,  $\sigma^2$ , by maximizing the log marginal likelihood. VFE can also select its inducing point locations. In this study, we do not consider optimizing the GRIEF weights,  $\mathbf{w}$ . VFE with  $m = 4$  inducing points achieves a root-mean squared error (RMSE) of 0.47 on the test set, whereas GP-GRIEF with the same number of basis functions<sup>3</sup>,  $p = 4$ , achieves an RMSE of 0.34, the same reconstruction error obtained by a full GP using the exact kernel. While GP-GRIEF uses a dense grid of  $m = 25$  inducing points, it has a computational complexity equivalent to VFE. This demonstrates the reconstruction power of GP-GRIEF, even when very few eigenfunctions are considered.

### 4.4.2 Kernel Reconstruction Accuracy

We compare the kernel covariance reconstruction error of the GRIEF Nyström method to competing techniques in fig. 4.2. We sample 5000 points from  $\mathcal{U}(-\sqrt{3}, \sqrt{3})$  in  $d = 100$  dimensions, randomly taking half for training and half for testing, and we consider a squared-exponential kernel. Given only the training set, we attempt to reconstruct the exact prior covariance

---

<sup>3</sup>Technically, VFE gives an infinite basis function expansion through a correction term; however, we will assume  $p = m = 4$ .

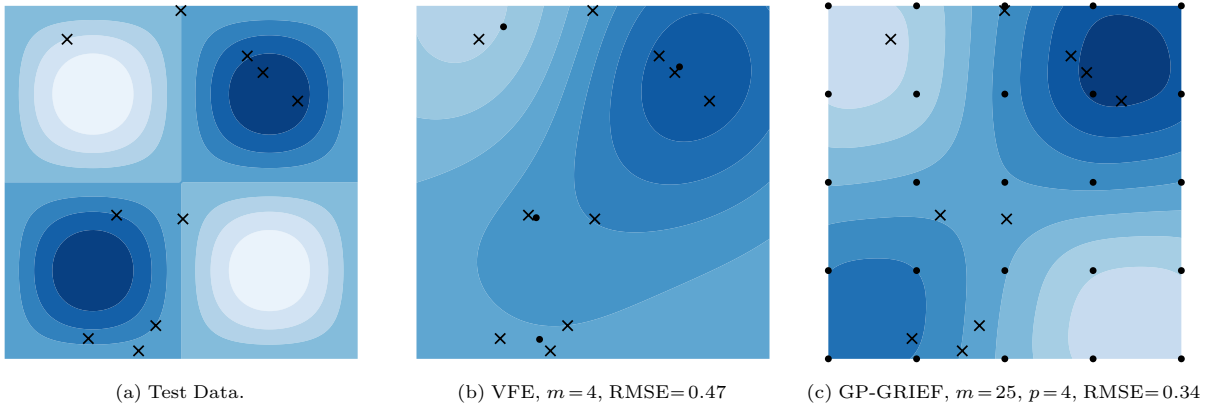
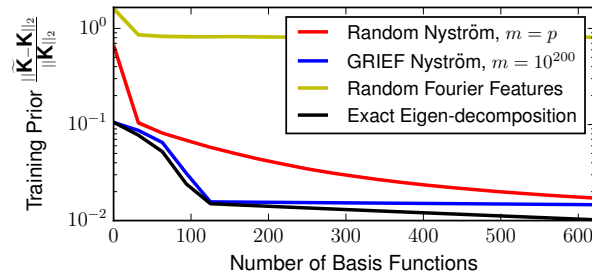


Figure 4.1: Reconstruction using GP-GRIEF outperforms VFE. Both techniques use an equal number of basis functions and have the same computational complexity. Crosses denote training point positions and dots denote inducing point locations.

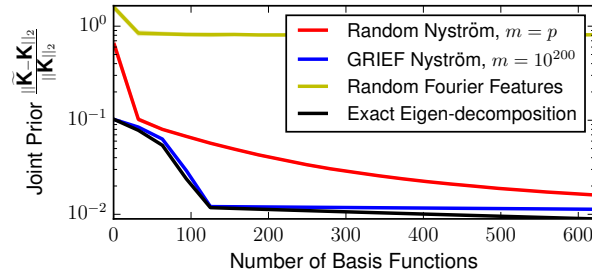
matrices between the training set (fig. 4.2a), and the joint train/test set (4.2b). This allows us to study the kernel reconstruction accuracy between points both within, and beyond the training set. In both studies, the proposed GRIEF Nyström method greatly outperforms a random Fourier features reconstruction (Rahimi and Recht, 2007), and a randomized Nyström reconstruction where  $m = p$  inducing points are uniformly subsampled from the training set. We emphasize that both randomized Nyström and GRIEF Nyström have the same computational complexity for a given number of basis functions,  $p$ , even though the GRIEF Nyström method uses  $m = 10^{200}$  inducing points ( $\bar{m} = 100$ ).

In the joint train/test study of fig. 4.2b, we observe a larger gap between GRIEF Nyström and randomized Nyström than in fig. 4.2a. This is not surprising since the goal of the randomized Nyström method (and indeed all existing extensions to this technique) is to improve the accuracy of eigenfunctions evaluated on the training set which does not guarantee performance of the eigenfunctions evaluated on points outside this set. For example, if a test point is placed far from the training set then we expect a poor approximation from existing Nyström methods. However, our GRIEF Nyström approach attempts to fill out the input space with inducing points everywhere, not just near training points. This guarantees an accurate approximation at test locations even if they are far from training data. Comparatively, the random Fourier features technique samples from a distribution that is independent of the training data, so it is also expected to perform no worse on the joint set than the training set. However, we observe that it provides an equally poor reconstruction on both sets due to the slow convergence of the stochastic Monte Carlo approximation employed therein.

The black curves in fig. 4.2 show the exact eigen-decomposition of the covariance matrices which demonstrates the optimal reconstruction accuracy for a kernel approximation with a given number of basis functions. We observe that GP-GRIEF approaches this optimal accuracy in both studies, even though the test point distribution is not known at training time.



(a) Training prior covariance error.



(b) Train/test joint prior covariance error.

Figure 4.2: Covariance matrix reconstruction error of GP-GRIEF beats randomized Nyström with uniform sampling averaged over 10 samples. GP-GRIEF approaches the optimal reconstruction accuracy of the black curve.

### 4.4.3 UCI Regression Studies

We next assess performance of GP-GRIEF on real-world regression datasets from the UCI repository. Using the authors’ code, we report the mean and standard deviation of the RMSE from 10-fold cross validation<sup>4</sup>. Also presented is the mean training time per fold on a machine with two E5-2680 v3 processors. We use a squared-exponential kernel with automatic relevance determination (SE-ARD), and we compare our test errors to those reported by Yang et al. (2015) using type-II inference on the same train-test splits. Yang et al. (2015) used an exact GP with an SE-ARD kernel for datasets with  $n < 2000$ , and Fastfood expansions were used to approximate the SE-ARD kernel for the larger datasets ( $n > 2000$ ).

Before training the GP-GRIEF model, we initialize the base kernel hyperparameters,  $\theta$ , by maximizing the marginal likelihood of an exact GP constructed on  $\min(n, 1000)$  points randomly selected from the dataset. We then train the model which we refer to as GP-GRIEF-II since a type-II inference approach is taken to estimate kernel hyperparameters (see section 2.3). GP-GRIEF-II uses the kernel from eq. (4.1) which is parametrized by the base kernel hyperparameters:  $\{\theta, \sigma^2\}$ . The presented training time includes log marginal likelihood maximization to estimate the hyperparameters, beginning with the initialized values. For all studies, we fix  $\bar{m} = 10$  and we make  $p$  proportional to  $n$  by rounding  $n$  down to the nearest power of ten, or take 1000 if it is lesser, i.e.,  $p = \min(1000, 10^{\lfloor \log_{10} n \rfloor})$ .

It can be observed from table 4.1 that GP-GRIEF-II outperforms the exact GP presented

<sup>4</sup>90% train, 10% test per fold. We use the same splits as Yang et al. (2015)

Dataset	$n$	$d$	$m=\bar{m}^d$	$p$	GP-GRIEF-II		Yang et al. (2015)
					Time	RMSE	RMSE
challenger	23	4	$10^4$	10	0	<b>0.554±0.277</b>	0.63±0.26
fertility	100	9	$10^9$	100	0.001	<b>0.172±0.055</b>	0.21±0.05
slump	103	7	$10^7$	100	0	<b>3.972±1.891</b>	4.72±2.42
automobile	159	25	$10^{25}$	100	0.007	<b>0.145±0.057</b>	0.18±0.07
servo	167	4	$10^4$	100	0	0.280±0.085	0.28±0.09
cancer	194	33	$10^{33}$	100	0.007	<b>27.843±3.910</b>	35±4
hardware	209	7	$10^7$	100	0	<b>0.408±0.046</b>	0.43±0.04
yacht	308	6	$10^6$	100	0.001	0.170±0.083	<b>0.16±0.11</b>
autopg	392	7	$10^7$	100	0.001	<b>2.607±0.356</b>	2.63±0.38
housing	506	13	$10^{13}$	100	0.004	3.212±0.864	<b>2.91±0.54</b>
forest	517	12	$10^{12}$	100	0.001	1.386±0.14	1.39±0.16
stock	536	11	$10^{11}$	100	0.001	0.005±0.000	0.005±0.001
energy	768	8	$10^8$	100	0.002	0.49±0.057	<b>0.46±0.07</b>
concrete	1030	8	$10^8$	1000	0.008	5.232±0.723	<b>4.95±0.77</b>
solar	1066	10	$10^{10}$	1000	0.003	<b>0.786±0.198</b>	0.83±0.20
wine	1599	11	$10^{11}$	1000	0.012	0.483±0.052	<b>0.47±0.08</b>
skillcraft	3338	19	$10^{19}$	1000	0.011	<b>0.248±0.016</b>	0.25±0.02
pumadyn	8192	32	$10^{32}$	1000	0.156	0.20±0.00	0.20±0.00
elevators	16599	18	$10^{18}$	1000	0.283	0.091±0.002	<b>0.090±0.001</b>
kin40k	40000	8	$10^8$	1000	0.38	<b>0.206±0.004</b>	0.28±0.01
kegg	63608	27	$10^{27}$	1000	3.642	<b>0.118±0.003</b>	<b>0.12±0.00</b>
3droad	434874	3	$10^3$	1000	0.869	11.648±0.281	<b>10.91±0.05</b>
electric	2049280	11	$10^{11}$	1000	8.019	<b>0.064±0.002</b>	0.12±0.12

Table 4.1: Mean and standard deviation of test error and average training time (including hyperparameter estimation, in hours) from 10-fold cross validation (90% train, 10% test per fold) on UCI regression datasets.

by Yang et al. (2015) on nearly every small dataset ( $n < 2000$ ). On the larger datasets, the GP-GRIEF technique shows comparable test error to Yang et al. (2015) but performs considerably better on kin40k and the electric dataset with two-million training points.

The independence of computational complexity on  $m$  allows enormous numbers of inducing points to be used. We use  $m = 10^{33}$  inducing points for the cancer dataset which demonstrates the efficiency of the matrix algebra employed since storing a double-precision vector of this length requires 8 quadrillion exabytes; far more than all combined hard-disk space on earth.

## 4.5 Conclusions

The new technique, GP-GRIEF, has demonstrated a kernel approximation strategy whose complexity is nearly independent of  $m$ , allowing us to break the *curse of dimensionality* inherent to methods that manipulate distributions of points on a full Cartesian product grid. Asymptotic results were also presented to show why a choice of large  $m$  is important to provide an accurate global kernel approximation. Lastly, we considered the use of up to  $10^{33}$  inducing points in our regression studies, demonstrating the efficiency of the matrix algebra employed. We discussed how the developed algebra can be used in areas beyond the focus of the numerical studies, such as in a general kernel interpolation framework, or in general kernel matrix preconditioning applications. However, it will be interesting to explore what other applications could exploit the developed matrix algebra techniques.

## Chapter 5

# Scaling Type-I Inference with Gaussian Processes

In this chapter, we introduce a Gaussian process covariance function that enables computation of the log marginal likelihood and all hyperparameter derivatives with a complexity that is independent of the quantity of training data. After an initial setup cost of  $\mathcal{O}(nm^2)$  time, the approach requires only  $\mathcal{O}(m)$  time and  $\mathcal{O}(m)$  memory per log marginal likelihood evaluation where  $n$  is the number of training examples, and  $m$  is the number of basis functions used in the kernel parameterization. The unique kernel parameterization approach is flexible, and the basis function can be of any form, making the approach highly general. The fast likelihood evaluation enables type-I or II Bayesian inference on large-scale datasets. We also provide an asymptotic result showing that any stationary kernel can be recovered when using the developed kernel. We benchmark our algorithms on real-world problems with up to two-million training points. The following paper was published from some of the content in this chapter:

T. W. Evans and P. B. Nair (2018c). “Scalable Gaussian Processes with Grid-Structured Eigenfunctions (GP-GRIEF)”. in: *International Conference on Machine Learning*, pp. 1416–1425, section 5 and appendices A & B.

### 5.1 Introduction

As discussed in section 2.3, there are a couple of common ways to deal with free covariance function hyperparameters: through estimation of the hyperparameters by maximization of the marginal likelihood (referred to as type-II inference, or empirical Bayes), or through marginalization of the hyperparameters using Bayes’ rule (which we refer to as type-I inference). In both cases, an iterative approach is generally required: typically an iterative numerical optimization procedure for type-II inference or a Markov chain Monte Carlo (MCMC) method for type-I inference.

All else being equal, a type-I inference approach is generally preferred since a type-II approach does not take into account the uncertainty over hyperparameters and tends to provide



over confident predictions, as was discussed in section 2.3. Unfortunately, a type-I approach is far more computationally expensive in practice. To understand why this is the case, first note that the per-iteration complexity of type-II inference (to perform numerical optimization) or type-I inference (to perform MCMC) are nearly identical with the most expensive computation required at each iteration being an evaluation of the marginal likelihood. This evaluation generally costs  $\mathcal{O}(n^3)$  time for a Gaussian process (eq. (2.30)) which is a significant expense for large datasets. Type-I inference also requires far more iterations (evaluations of the marginal likelihood) than type-II inference and therefore has to perform many more  $\mathcal{O}(n^3)$  computations. In practice, type-I marginalization of hyperparameters by MCMC often use several orders of magnitude more iterations than type-II optimization of hyperparameters. The reason for this largely discrepancy in the iteration count is partially due to the differences in convergence rates for the (stochastic) MCMC method verses the (deterministic) optimization approach. In addition, while optimality conditions are easily assessed, convergence of MCMC is much more difficult to quantify which makes termination conditions difficult to design.

Towards this end there has been significant work to design sparse Gaussian processes which reduce the cost of GP marginal likelihood computations as discussed in section 2.4.1. For instance, a degenerate Gaussian process using  $m$  basis functions to approximate a desired GP prior admits a marginal likelihood evaluation in  $\mathcal{O}(m^2n + m^3)$  time using eq. (2.29). While this is an improvement, this model still contains a linear dependence on the number of training observations  $n$  as well as a cubic complexity on the number of kernel basis functions which is generally required to be large for complicated modelling problems. Such models therefore still struggle to perform type-I inference by MCMC on large and complicated problems. In this chapter we significantly improve upon this asymptotic scaling. Here we summarize the main contributions of the chapter:

- A new covariance function parameterization is developed that enables computation of the GP marginal likelihood in as little as  $\mathcal{O}(m)$  time. Note that this is independent of the quantity of training data.
- It is shown that this parameterization recovers several popular kernels where fast evidence computations have not yet been exploited.
- The flexibility of the parameterization is demonstrated through a theoretical result that shows how any stationary kernel can be asymptotically recovered.
- Finally, we demonstrate accurate and rapid type-I GP inference on real-world datasets with up to 2 million training points.

Section 5.2 begins by introducing the employed kernel parameterization and discusses how computation of the Gaussian log marginal likelihood can be performed with a complexity that is independent of the quantity of training data. Section 5.3 discusses various applications of the developed kernels for specific choices of basis functions and basis function priors. Finally, experimental studies are performed in section 5.4 and conclusions are presented in section 5.5.

## 5.2 Re-weighted Basis Kernel

In this chapter, we consider a sparse Gaussian process model using the following finite basis kernel,

$$k(\mathbf{x}, \mathbf{z}) = \boldsymbol{\phi}(\mathbf{x})^T \mathbf{S}^{-1} \boldsymbol{\phi}(\mathbf{z}), \quad (5.1)$$

where  $\mathbf{S} \in \mathbb{R}^{m \times m}$ , and  $\boldsymbol{\phi}: \mathbb{R}^d \rightarrow \mathbb{R}$ . As was discussed in detail in section 2.2, using this kernel for a Gaussian process covariance function directly specifies a function space prior; however, we can also consider a weight space perspective to describe the following equivalent model: consider a generalized linear model of the form  $f(\mathbf{x}) = \sum_{i=1}^m w_i \phi_i(\mathbf{x})$ , where the weight space prior is  $\Pr(\mathbf{w}) = \mathcal{N}(\mathbf{w} | \mathbf{0}, \mathbf{S}^{-1})$ , and the likelihood is  $\Pr(\mathbf{y} | \mathbf{X}, \mathbf{w}) = \mathcal{N}(\mathbf{y} | \boldsymbol{\Phi} \mathbf{w}, \sigma^2 \mathbf{I}_n)$ . In our notation,  $\mathbf{S} \in \mathbb{R}^{m \times m}$  is a symmetric and positive definite weight prior precision matrix,  $\boldsymbol{\Phi} \in \mathbb{R}^{n \times m}$  where  $\phi_{ij} = \phi_j(\mathbf{x}_i)$ , contains the evaluations of all  $m$  basis functions at all  $n$  training points, and  $\mathbf{w} \in \mathbb{R}^m$  are the basis function weights. Using the kernel in eq. (5.1), the covariance matrix evaluated on the training set inputs becomes

$$\mathbf{K}_{\mathbf{X}, \mathbf{X}} = \boldsymbol{\Phi} \mathbf{S}^{-1} \boldsymbol{\Phi}^T. \quad (5.2)$$

In this chapter, we use the notation  $[\mathbf{K}_{\mathbf{A}, \mathbf{B}}]_{i,j} = k(\mathbf{a}_i, \mathbf{b}_j)$  such that  $\mathbf{K}_{\mathbf{X}, \mathbf{X}} \in \mathbb{R}^{n \times n}$  is the kernel covariance matrix evaluated on the training dataset and  $\mathbf{K}_{\mathbf{z}, \mathbf{X}} \in \mathbb{R}^{1 \times n}$  is the cross-covariance between  $\mathbf{z} \in \mathbb{R}^d$  and the training dataset  $\mathbf{X}$ , for instance. For the kernel in eq. (5.1), we consider the set of kernel hyperparameters to define the prior precision matrix  $\mathbf{S}$ . We call this the “re-weighted basis” kernel since we will infer the prior over the basis function weights. If the matrix  $\mathbf{S}$  is diagonal (which assumes no prior correlation between the basis functions), a parameterization could simply be the positive diagonal values themselves. If  $\mathbf{S}$  is to be dense, see (Pineiro and Bates, 1996) for a discussion on parameterization options. We will introduce specific parameterizations later in this chapter where needed.

Since the kernel is now heavily parametrized (e.g., the diagonal parameterization of  $\mathbf{S}$  contains  $\mathcal{O}(m)$  hyperparameters), maximizing the Gaussian process log marginal likelihood (or log evidence) for type-II Bayesian inference could be susceptible to overfitting (see discussion in section 2.3). Instead, we may choose to take a fully Bayesian type-I approach and integrate out the hyperparameters using hybrid Markov chain Monte Carlo (MCMC) sampling. This type-I approach requires orders of magnitude more log-marginal likelihood (LML) evaluations than a type-II approach; however, we show that fast evaluations make this tractable even for very large problems. Namely, we show that the cost per iteration can be *independent* of the number of training points. What follows is an in-depth discussion about computations of the LML. We show that various algebraic manipulations allow significantly accelerated computations.

Computation of the Gaussian process log marginal likelihood in eq. (2.30) simply requires the following operations which can be computed efficiently using the matrix inversion lemma

and the matrix determinant lemma, respectively (Harville, 2006),

$$\begin{aligned} \mathbf{y}^T (\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_n)^{-1} \mathbf{y} &= \sigma^{-2} \left( \mathbf{y}^T \mathbf{y} - (\Phi^T \mathbf{y})^T (\sigma^2 \mathbf{S} + \Phi^T \Phi)^{-1} (\Phi^T \mathbf{y}) \right), \quad \text{and} \\ \log |\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_n| &= \log |\sigma^2 \mathbf{S} + \Phi^T \Phi| - \log |\mathbf{S}| + (n-m) \log \sigma^2, \end{aligned} \quad (5.3)$$

The above computations enable evaluation of the LML with a finite basis kernel in  $\mathcal{O}(nm^2 + m^3)$  time and with  $\mathcal{O}(nm + m^2)$  storage. In fact, we had already used these algebraic operations to demonstrate efficient computation of the LML in eq. (2.29). Here we revisit the form of this computation to identify and exploit further algebraic properties. Specifically, we demonstrate that after an initial setup step, the log marginal likelihood (LML) and all hyperparameter derivatives can be computed in as little as  $\mathcal{O}(m)$  time. To do this, we first assume that  $\mathbf{y}^T \mathbf{y} \in \mathbb{R}$ ,  $\Phi^T \mathbf{y} = \mathbf{r} \in \mathbb{R}^m$ , and  $\Phi^T \Phi = \mathbf{A} \in \mathbb{R}^{m \times m}$  are precomputed, which requires  $\mathcal{O}(nm^2)$  time; however, this step only needs to be done once before LML iterations begin. We will next proceed to demonstrate how all computations can be performed in  $\mathcal{O}(m^3)$  for a general prior structure and then discuss how this can be further accelerated to  $\mathcal{O}(m^2)$  time. Finally, we show how computations can be reduced again to  $\mathcal{O}(m)$  when additional assumptions are made.

### 5.2.1 $\mathcal{O}(m^3)$ Evidence Computations

To begin, observe that we can re-write the computations required for LML evaluation in eq. (5.3) to give

$$\begin{aligned} \mathbf{y}^T (\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_n)^{-1} \mathbf{y} &= \sigma^{-2} (\mathbf{y}^T \mathbf{y} - \mathbf{r}^T \mathbf{P}^{-1} \mathbf{r}), \quad \text{and} \\ \log |\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_n| &= \log |\mathbf{P}| - \log |\mathbf{S}| + (n-m) \log \sigma^2, \end{aligned} \quad (5.4)$$

where we used the shorthand  $\mathbf{P} = \sigma^2 \mathbf{S} + \mathbf{A} \in \mathbb{R}^{m \times m}$ . Using these relations, LML iterations can be computed in  $\mathcal{O}(m^3)$  time and  $\mathcal{O}(m^2)$  storage. Observe that the complexity of these computations are now independent of the quantity of training data,  $n$ . This means that performing inference on larger datasets does not require additional computational resources, an attractive property.

The LML derivatives with respect to all hyperparameters can also be computed in  $\mathcal{O}(m^3)$  as shown in the following expression which is derived in appendix C. Note that here we assume the parameterization  $\mathbf{S} = \text{diag}(\boldsymbol{\gamma}^{-1})$ , where  $\boldsymbol{\gamma} \in (0, \infty)^m$ , and the precision is evidently a diagonal matrix. The derivatives can be written as follows:

$$\begin{aligned} \frac{\partial \text{LML}}{\partial \boldsymbol{\gamma}} &= \frac{(\mathbf{r} - \mathbf{A} \mathbf{P}^{-1} \mathbf{r})^2}{2\sigma^4} - \frac{\text{diag}(\mathbf{A}) - (\mathbf{A} \odot \mathbf{P}^{-1} \mathbf{A})^T \mathbf{1}_p}{2\sigma^2}, \quad \text{and} \\ \frac{\partial \text{LML}}{\partial \sigma^2} &= \frac{\mathbf{y}^T \mathbf{y} - 2\mathbf{r}^T \mathbf{P}^{-1} \mathbf{r} + \mathbf{r}^T \mathbf{P}^{-1} \mathbf{A} \mathbf{P}^{-1} \mathbf{r}}{2\sigma^4} - \frac{n - \text{Tr}(\mathbf{P}^{-1} \mathbf{A})}{2\sigma^2}, \end{aligned} \quad (5.5)$$

where  $\odot$  denotes the elementwise product. Note that if a dense prior precision is chosen, automatic differentiation can alternatively be used to compute derivatives rapidly. These

computations are summarized in algorithm 5.1.

### 5.2.2 $\mathcal{O}(m^2)$ Evidence Computations

To further reduce the per-iteration computational complexity from  $\mathcal{O}(m^3) \rightarrow \mathcal{O}(m^2)$  we apply a linear transformation to the basis functions to make them mutually orthogonal when evaluated on the training data. We can write the  $i$ th transformed basis function as  $\tilde{\phi}_i(\mathbf{x}) = \sum_{j=1}^m \tilde{v}_{ji} \tilde{\Sigma}_{ii}^{-1} \phi_i(\mathbf{x})$ , where  $\tilde{\Sigma} \in \mathbb{R}^{\tilde{m} \times \tilde{m}}$  is a diagonal matrix containing the non-zero singular values of  $\Phi$ ,  $\tilde{\mathbf{V}} \in \mathbb{R}^{m \times \tilde{m}}$  contains the corresponding right-singular vectors of  $\Phi$ , and  $\tilde{m} \leq \min(m, n)$ . With this transformation, we can re-write eq. (5.2) as

$$\mathbf{K}_{\mathbf{X}, \mathbf{X}} = \tilde{\Phi} \tilde{\mathbf{S}}^{-1} \tilde{\Phi}^T, \quad (5.6)$$

where  $\tilde{\Phi} = \Phi \tilde{\mathbf{V}} \tilde{\Sigma}^{-1} \in \mathbb{R}^{n \times \tilde{m}}$  is an orthogonal matrix composed of the transformed basis functions evaluated on the training dataset (its columns contain the left-singular vectors of  $\Phi$ ), and  $\tilde{\mathbf{S}}^{-1} = \tilde{\Sigma} \tilde{\mathbf{V}}^T \mathbf{S}^{-1} \tilde{\mathbf{V}} \tilde{\Sigma} \in \mathbb{R}^{\tilde{m} \times \tilde{m}}$  is the prior covariance matrix over the transformed basis functions. Using this parameterization,  $\mathbf{A} = \tilde{\Phi}^T \tilde{\Phi} = \mathbf{I}_{\tilde{m}}$  becomes a diagonal matrix, however,  $\tilde{\mathbf{S}}^{-1}$  will generally be dense even if  $\mathbf{S}^{-1}$  is diagonal. We therefore must consider a dense parameterization of  $\tilde{\mathbf{S}}^{-1}$ . Namely we make the choice of the following spectral (or rotational) parameterization which can represent any precision matrix (Pinheiro and Bates, 1996)

$$\tilde{\mathbf{S}} = \mathbf{U} \mathbf{D} \mathbf{U}^T, \quad (5.7)$$

where  $\mathbf{U} \in \mathbb{R}^{\tilde{m} \times \tilde{m}}$  is a unitary matrix with eigenvectors in its columns, and  $\mathbf{D} \in \mathbb{R}^{\tilde{m} \times \tilde{m}}$  is a diagonal matrix containing the eigenvalues. With this parameterization, we can write

$$\mathbf{P} = \sigma^2 \tilde{\mathbf{S}} + \tilde{\Phi}^T \tilde{\Phi} = \mathbf{U} (\sigma^2 \mathbf{D} + \mathbf{I}_{\tilde{m}}) \mathbf{U}^T. \quad (5.8)$$

Given these spectral forms, the computations in eq. (5.4) can be performed rapidly. Namely,  $\log |\mathbf{P}|$  and  $\log |\tilde{\mathbf{S}}|$  can evidently be performed in  $\mathcal{O}(\tilde{m})$ , and the computation of  $\mathbf{r}^T \mathbf{P}^{-1} \mathbf{r} = \mathbf{r}^T \mathbf{U} (\sigma^2 \mathbf{D} + \mathbf{I}_{\tilde{m}})^{-1} \mathbf{U}^T \mathbf{r}$  is dominated by the cost of the matrix-vector product  $\mathbf{U}^T \mathbf{r}$ . The cost of this matrix-vector product may depend on the parameterization of  $\mathbf{U}$ ; however, we choose a particular parameterization composed of a matrix product of  $\tilde{m}$  Householder matrices. This parameterization is general; it can be shown that any unitary matrix  $\mathbf{U}$  can be represented as a product of  $\tilde{m}$  Householder matrices (e.g., (Tomczak and Welling, 2016)). Since a matrix-vector product with each Householder matrix can be computed in  $\mathcal{O}(\tilde{m})$ ,<sup>1</sup> the matrix-vector product  $\mathbf{U}^T \mathbf{r}$  can be computed within  $\mathcal{O}(\tilde{m}^2) \leq \mathcal{O}(m^2)$ .

We have therefore demonstrated that all LML operations can be performed within  $\mathcal{O}(m^2)$ . We also note that while the linear transformation of the basis functions requires the singular-value decomposition of  $\Phi$  before LML iterations, this precomputation is no more

<sup>1</sup>This is because a Householder matrix is simply the addition of the identity matrix and a rank-one matrix. See (Tomczak and Welling, 2016, eqn. 7), for example.

expensive than those discussed previously at  $\mathcal{O}(n\tilde{m}^2) \leq \mathcal{O}(nm^2)$ . The approach is also just as general as the  $\mathcal{O}(m^3)$  approach described in section 5.2.1 and is therefore a clear algorithmic improvement; however, there are reasons that the  $\mathcal{O}(m^3)$  approach may be preferred in practice. For instance, specifying a hyper-prior over the parameters of  $\tilde{\mathbf{S}}$  may be less-intuitive than specifying a prior over the parameters of  $\mathbf{S}$  as a result of the linear transformation applied to the basis functions (especially since this transformation is data-dependent). Also, while the suggested Householder parameterization enables reduction of computational complexity to  $\mathcal{O}(m^2)$ , it is challenging to parallelize the resulting matrix operations and this may result in slower runtimes than the  $\mathcal{O}(m^3)$  approach in practice.

As a final remark, the full  $m$  basis functions should ideally be used for predictive posterior covariance computations (see eq. (2.16)), not just the  $\tilde{m} \leq \min(m, n)$  terms used for LML computations. The  $m - \tilde{m}$  basis functions were eliminated from computations in this section since they are orthogonal to the training data and therefore do not effect the LML computations; however, they may effect the predictive posterior variance (uncertainty) far from the training data. Ignoring these basis functions will result in uncertainty being underestimated at test time. Note that including these  $m - \tilde{m}$  basis functions will not effect the predictive posterior mean when the GP prior has zero mean.

### 5.2.3 $\mathcal{O}(m)$ Evidence Computations

As mentioned in the preceding section,  $\tilde{\mathbf{S}}^{-1}$  will generally be dense even if  $\mathbf{S}^{-1}$  is diagonal; however, if we directly parameterize  $\tilde{\mathbf{S}}$  as a diagonal matrix then  $\mathbf{P} = \sigma^2\tilde{\mathbf{S}} + \mathbf{A}$  will become a diagonal matrix and evaluation of the LML and all derivatives can be performed in  $\mathcal{O}(m)$  using eq. (5.4) and the derivative expressions in eq. (5.5) (recall that  $\mathbf{A} = \mathbf{I}_{\tilde{m}}$  becomes a diagonal matrix when we use the transformed basis functions  $\tilde{\Phi}$ ).

An alternative way of viewing the choice of a diagonal  $\tilde{\mathbf{S}}$  is to impose  $\mathbf{U} = \mathbf{I}_{\tilde{m}}$  in eq. (5.7).<sup>2</sup> Clearly this choice would ensure that  $\mathbf{U}^T \mathbf{r}$  could be computed within  $\mathcal{O}(m)$  and therefore would ensure an overall complexity of  $\mathcal{O}(m)$ . In appendix C, the LML derivatives are also presented which can be computed in  $\mathcal{O}(m)$  as well.

Imposing a diagonal  $\tilde{\mathbf{S}}$  is somewhat strange in the sense that the structure of the basis function prior precision is being selected based upon the data. We therefore consider that this  $\mathcal{O}(m)$  approach involves a further approximation and is not as general as the approaches in sections 5.2.1 and 5.2.2; however, we have found that it performs well in practice.

### 5.2.4 Prior over Noise Variance

Inference may be performed over the observation noise variance  $\sigma^2$  if there is uncertainty over this value *a priori*. Observing the computations required for the Gaussian process LML

<sup>2</sup>More generally, we can achieve  $\mathcal{O}(m)$  LML computations if  $\mathbf{U}$  is parameterized by a product of  $\mathcal{O}(1)$  Householder matrices. This would give improved parameterization flexibility over the choice of  $\mathbf{U} = \mathbf{I}_{\tilde{m}}$  while still ensuring an  $\mathcal{O}(m)$  runtime (albeit with a slightly greater computational expense).

evaluation in eq. (5.4), it is evident that recomputing the LML at a different value of  $\sigma^2$  has an incremental cost of only  $\mathcal{O}(1)$ . Therefore specifying a prior over the noise variance in addition to the basis function prior precision does not effect the computational complexities reported in sections 5.2.1 to 5.2.3.

### 5.2.5 Non-Zero Prior Mean

We may also consider specifying a non-zero prior mean for the basis function weights;  $\mathbf{w} \sim \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}, \mathbf{S}^{-1})$ , where  $\boldsymbol{\mu} \in \mathbb{R}^m$ . This is equivalent to specifying a GP with the kernel in eq. (5.1) and the prior mean function  $\sum_{i=1}^m \mu_i \phi_i(\mathbf{x})$ . In this case, it can be observed that computation of the log-marginal likelihood can still be performed with a complexity independent of  $n$  through the relations discussed in sections 5.2.1 to 5.2.3 where  $\mathbf{r}$  is replaced by  $\mathbf{r} - \mathbf{A}\boldsymbol{\mu}$ , and  $\mathbf{y}^T \mathbf{y}$  is replaced by  $\mathbf{y}^T \mathbf{y} - 2\mathbf{r}^T \boldsymbol{\mu} + \boldsymbol{\mu}^T \mathbf{A}\boldsymbol{\mu}$ . Similar to the parameterization of  $\mathbf{S}$ , we may also specify a hyper-prior on the elements of  $\boldsymbol{\mu}$  and either estimate or marginalize these variables as well. Both of these extensions increase the flexibility of the Bayesian model while ensuring that the computational complexity remains independent of the training dataset size, a necessity for performing type-I inference on massive datasets.

## 5.3 Applications

In this section, we discuss various applications of re-weighted basis kernels for specific choices of basis functions and basis function priors.

### 5.3.1 Importance-Weighted Random Fourier Features

We outline a novel approach to GP inference with random Fourier features which is amenable to the fast LML computation methods outlined in section 5.2. Before outlining the new approach, we provide background on the random Fourier features kernel approximation.

#### Background on Random Fourier Features

Bochner’s theorem (Stein, 1999, p. 24) states that any stationary covariance function,  $k(\mathbf{x}_i, \mathbf{x}_j) = k(\mathbf{x}_i - \mathbf{x}_j) = k(\boldsymbol{\tau})$ , can be represented as the Fourier transform of a positive finite measure. We refer to this finite measure as the *power spectrum*  $\mathcal{S}(\boldsymbol{\xi})$  and since it is positive, we can define a probability measure that is proportional to  $\mathcal{S}$ , i.e.,  $\mathcal{S}(\boldsymbol{\xi}) \propto \Pr(\boldsymbol{\xi})$ . The proportionality constant can be found by evaluating the kernel at  $\boldsymbol{\tau} = \mathbf{0}$  as follows:

$$\mathcal{S}(\boldsymbol{\xi}) = k(\mathbf{0}) \Pr(\boldsymbol{\xi}) = \sigma_0^2 \Pr(\boldsymbol{\xi}),$$

where  $\sigma_0^2 > 0$  is simply the variance (or amplitude) of the kernel  $k$ . We can now define the kernel  $k$  as the Fourier transform of the power spectrum  $\mathcal{S}$  as follows (Lázaro-Gredilla et al.,

2010, eq. 11):

$$\begin{aligned} k(\mathbf{x}_i, \mathbf{x}_j) &= k(\boldsymbol{\tau}) = \int_{\mathbb{R}^d} \exp(2\pi i \boldsymbol{\xi}^T (\mathbf{x}_i - \mathbf{x}_j)) \mathcal{S}(\boldsymbol{\xi}) d\boldsymbol{\xi} = \sigma_0^2 \int_{\mathbb{R}^d} \exp(2\pi i \boldsymbol{\xi}^T \mathbf{x}_i) \left( \exp(2\pi i \boldsymbol{\xi}^T \mathbf{x}_j) \right)^* \Pr(\boldsymbol{\xi}) d\boldsymbol{\xi}, \\ &= \sigma_0^2 \mathbb{E}_{\Pr(\boldsymbol{\xi})} \left[ \exp(2\pi i \boldsymbol{\xi}^T \mathbf{x}_i) \left( \exp(2\pi i \boldsymbol{\xi}^T \mathbf{x}_j) \right)^* \right], \end{aligned}$$

where we have replaced the power spectrum with the probability measure and identified the resulting expression as an expectation. Note that a superscript asterisk (\*) denotes complex conjugation. To recover a real-valued kernel, the power spectrum must be symmetric around zero (i.e.,  $\Pr(\boldsymbol{\xi}) = \Pr(-\boldsymbol{\xi})$ ). Rewriting the preceding equation we see that this causes the imaginary terms to cancel, as follows:

$$\begin{aligned} k(\mathbf{x}_i, \mathbf{x}_j) &= \frac{\sigma_0^2}{2} \mathbb{E}_{\Pr(\boldsymbol{\xi})} \left[ \exp(2\pi i \boldsymbol{\xi}^T \mathbf{x}_i) \left( \exp(2\pi i \boldsymbol{\xi}^T \mathbf{x}_j) \right)^* + \left( \exp(2\pi i \boldsymbol{\xi}^T \mathbf{x}_i) \right)^* \exp(2\pi i \boldsymbol{\xi}^T \mathbf{x}_j) \right], \\ &= \sigma_0^2 \mathbb{E}_{\Pr(\boldsymbol{\xi})} \left[ \operatorname{Re} \left\{ \exp(2\pi i \boldsymbol{\xi}^T \mathbf{x}_i) \left( \exp(2\pi i \boldsymbol{\xi}^T \mathbf{x}_j) \right)^* \right\} \right], \\ &= \sigma_0^2 \mathbb{E}_{\Pr(\boldsymbol{\xi})} \left[ \cos(2\pi \boldsymbol{\xi}^T (\mathbf{x}_i - \mathbf{x}_j)) \right], \end{aligned} \quad (5.9)$$

$$= \sigma_0^2 \mathbb{E}_{\Pr(\boldsymbol{\xi})} \left[ \cos(2\pi \boldsymbol{\xi}^T \mathbf{x}_i) \cos(2\pi \boldsymbol{\xi}^T \mathbf{x}_j) + \sin(2\pi \boldsymbol{\xi}^T \mathbf{x}_i) \sin(2\pi \boldsymbol{\xi}^T \mathbf{x}_j) \right]. \quad (5.10)$$

The integral in eq. (5.10) can be approximated by a Monte Carlo estimator using  $m/2$  samples as follows:

$$\begin{aligned} k(\mathbf{x}_i, \mathbf{x}_j) &\simeq \frac{2\sigma_0^2}{m} \sum_{r=1}^{m/2} \left[ \cos(2\pi \boldsymbol{\xi}_r^T \mathbf{x}_i) \cos(2\pi \boldsymbol{\xi}_r^T \mathbf{x}_j) + \sin(2\pi \boldsymbol{\xi}_r^T \mathbf{x}_i) \sin(2\pi \boldsymbol{\xi}_r^T \mathbf{x}_j) \right], \\ &= \frac{2\sigma_0^2}{m} \boldsymbol{\phi}(\mathbf{x}_i)^T \boldsymbol{\phi}(\mathbf{x}_j), \end{aligned} \quad (5.11)$$

where  $\boldsymbol{\xi}_r \sim \Pr(\boldsymbol{\xi})$ ,  $m$  is assumed to be divisible by two, and we denote the  $m$  basis functions of the kernel approximation as

$$\boldsymbol{\phi}(\mathbf{x}) = \left[ \cos(2\pi \boldsymbol{\xi}_1^T \mathbf{x}), \sin(2\pi \boldsymbol{\xi}_1^T \mathbf{x}), \dots, \cos(2\pi \boldsymbol{\xi}_{m/2}^T \mathbf{x}), \sin(2\pi \boldsymbol{\xi}_{m/2}^T \mathbf{x}) \right]^T. \quad (5.12)$$

The basis functions of the kernel approximation are simply trigonometric functions, and through comparison with eq. (5.1), the basis function prior precision is given by  $\mathbf{S} = \frac{m}{2\sigma_0} \mathbf{I}_m$ . This kernel approximation can be used for any real-valued stationary kernel and has the attractive property that it will remain stationary for any  $m$  (this can be seen by inspection of eq. (5.9), and also note that we still require  $m$  to be divisible by two). Additionally, by the strong law of large numbers, this Monte Carlo approximation procedure will converge to the true kernel as  $m \rightarrow \infty$ .

### Example: Exponentiated Quadratic Kernel Power Spectrum

The stationary kernel being approximated in eq. (5.11) depends only on the power spectrum  $\mathcal{S}$ . As an example, we now derive the power spectrum of the exponentiated quadratic

kernel (eq. (2.28)). This can be achieved by the following Fourier transform (note we divide by the kernel variance to give a probability measure) (Lázaro-Gredilla et al., 2010, eq. 12)

$$\Pr(\boldsymbol{\xi}) = \frac{1}{k(\mathbf{0})} \int_{\mathbb{R}^d} \exp(-2\pi i \boldsymbol{\xi}^T \boldsymbol{\tau}) k(\boldsymbol{\tau}) d\boldsymbol{\tau} = \sqrt{|2\pi\boldsymbol{\Lambda}|} \exp(-2\pi^2 \boldsymbol{\xi}^T \boldsymbol{\Lambda} \boldsymbol{\xi}) = \mathcal{N}(\boldsymbol{\xi} | \mathbf{0}, \frac{1}{(2\pi)^2} \boldsymbol{\Lambda}^{-1}), \quad (5.13)$$

which is evidently a multivariate Gaussian distribution. Note that since the basis functions in eq. (5.12) are all functions of  $\boldsymbol{\xi}' = 2\pi\boldsymbol{\xi}$ , it is more convenient to simply sample from the distribution  $\mathcal{N}(\boldsymbol{\xi}' | \mathbf{0}, \boldsymbol{\Lambda}^{-1})$  in a practical implementation.

### Importance Weighting Spectral Points

We have previously demonstrated how the finite basis kernel approximation in eq. (5.11) is derived from a Monte Carlo approximation using the finite set of spectral points,  $\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \dots, \boldsymbol{\xi}_{m/2}$ . Typically, during hyperparameter estimation or marginalization when the kernel being approximated is changed, the spectral points are modified to deliver a consistent Monte Carlo approximation (Lázaro-Gredilla et al., 2010). This unfortunately means that when kernel hyperparameters are changed, the basis functions  $\boldsymbol{\phi}(\mathbf{x})$  also change and so we cannot use the techniques introduced in section 5.2 for fast type-I or type-II inference. The following result introduces a novel way to adjust the random Fourier features kernel approximation in a way that is amenable to the fast inference techniques outlined in section 5.2.

**Theorem 5.1.** *Let  $k_1 : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$  and  $k_2 : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$  denote two real stationary kernels whose power spectra share the same support and let  $Pr_1$  and  $Pr_2$  denote  $d$ -dimensional probability distributions proportional to the power spectra of  $k_1$  and  $k_2$ , respectively. Then the following equality holds*

$$k_2(\mathbf{x}, \mathbf{z}) = \lim_{m \rightarrow \infty} \boldsymbol{\phi}(\mathbf{x})^T \mathbf{S}^{-1} \boldsymbol{\phi}(\mathbf{z}),$$

where  $\boldsymbol{\phi}(\mathbf{x})$  is given by eq. (5.12) with  $\boldsymbol{\xi}_i \sim Pr_1(\boldsymbol{\xi})$  for  $i = 1, 2, \dots, m/2$ , and the basis function prior precision matrix is given by

$$\mathbf{S} = \text{diag} \left( \left\{ \frac{m}{2k_2(\mathbf{0})} \frac{Pr_1(\boldsymbol{\xi}_i)}{Pr_2(\boldsymbol{\xi}_i)} \right\}_{i=1}^{m/2} \right) \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

*Proof.* We begin by rewriting eq. (5.10) to approximate  $k_2$  as follows:

$$\begin{aligned} k_2(\mathbf{x}, \mathbf{z}) &= k_2(\mathbf{0}) \mathbb{E}_{Pr_2(\boldsymbol{\xi})} \left[ \cos(2\pi \boldsymbol{\xi}^T \mathbf{x}) \cos(2\pi \boldsymbol{\xi}^T \mathbf{z}) + \sin(2\pi \boldsymbol{\xi}^T \mathbf{x}) \sin(2\pi \boldsymbol{\xi}^T \mathbf{z}) \right], \\ &= k_2(\mathbf{0}) \int_{\mathbb{R}^d} \left( \cos(2\pi \boldsymbol{\xi}^T \mathbf{x}) \cos(2\pi \boldsymbol{\xi}^T \mathbf{z}) + \sin(2\pi \boldsymbol{\xi}^T \mathbf{x}) \sin(2\pi \boldsymbol{\xi}^T \mathbf{z}) \right) Pr_2(\boldsymbol{\xi}) d\boldsymbol{\xi}. \end{aligned}$$

We now multiply and divide the integrand in the preceding equation by  $Pr_1$  to give an expect-



tation over  $\text{Pr}_1$ . This operation is equivalent to the introduction of a *proposal distribution*  $\text{Pr}_1$  as used in importance sampling (Owen, 2013) and gives

$$\begin{aligned} k_2(\mathbf{x}, \mathbf{z}) &= k_2(\mathbf{0}) \int_{\mathbb{R}^d} \left( \cos(2\pi \boldsymbol{\xi}^T \mathbf{x}) \cos(2\pi \boldsymbol{\xi}^T \mathbf{z}) + \sin(2\pi \boldsymbol{\xi}^T \mathbf{x}) \sin(2\pi \boldsymbol{\xi}^T \mathbf{z}) \right) \frac{\text{Pr}_2(\boldsymbol{\xi})}{\text{Pr}_1(\boldsymbol{\xi})} \text{Pr}_1(\boldsymbol{\xi}) d\boldsymbol{\xi}, \\ &= k_2(\mathbf{0}) \mathbb{E}_{\text{Pr}_1(\boldsymbol{\xi})} \left[ \frac{\text{Pr}_2(\boldsymbol{\xi})}{\text{Pr}_1(\boldsymbol{\xi})} \cos(2\pi \boldsymbol{\xi}^T \mathbf{x}) \cos(2\pi \boldsymbol{\xi}^T \mathbf{z}) + \frac{\text{Pr}_2(\boldsymbol{\xi})}{\text{Pr}_1(\boldsymbol{\xi})} \sin(2\pi \boldsymbol{\xi}^T \mathbf{x}) \sin(2\pi \boldsymbol{\xi}^T \mathbf{z}) \right]. \end{aligned}$$

We can now take a Monte Carlo estimator of this expectation with  $m/2$  samples

$$\begin{aligned} k_2(\mathbf{x}, \mathbf{z}) &\simeq \frac{2k_2(\mathbf{0})}{m} \sum_{r=1}^{m/2} \left[ \frac{\text{Pr}_2(\boldsymbol{\xi}_r)}{\text{Pr}_1(\boldsymbol{\xi}_r)} \cos(2\pi \boldsymbol{\xi}_r^T \mathbf{x}) \cos(2\pi \boldsymbol{\xi}_r^T \mathbf{z}) + \frac{\text{Pr}_2(\boldsymbol{\xi}_r)}{\text{Pr}_1(\boldsymbol{\xi}_r)} \sin(2\pi \boldsymbol{\xi}_r^T \mathbf{x}) \sin(2\pi \boldsymbol{\xi}_r^T \mathbf{z}) \right], \\ &= \sum_{i=1}^m \frac{2k_2(\mathbf{0})}{m} \frac{\text{Pr}_2(\boldsymbol{\xi}_{[i/2]})}{\text{Pr}_1(\boldsymbol{\xi}_{[i/2]})} \phi_i(\mathbf{x}) \phi_i(\mathbf{z}), \end{aligned}$$

where  $\boldsymbol{\xi}_r \sim \text{Pr}_1(\boldsymbol{\xi})$  for  $r = 1, 2, \dots, m/2$ , the operator  $[\cdot]$  rounds its argument up to the nearest integer, the basis functions  $\phi(\mathbf{x})$  are given by eq. (5.12), and the equality holds in the limit of  $m \rightarrow \infty$  by the strong law of large numbers provided  $\text{Pr}_1$  and  $\text{Pr}_2$  share the same support (Owen, 2013). Writing the preceding equation in matrix form completes the proof.  $\square$

The proof involves an application of importance sampling, a technique commonly used to reduce the variance of Monte Carlo estimators (Owen, 2013). In the situation outlined here, the proposal distribution is not selected to reduce the variance, but rather to eliminate bias such that the estimator converges in the limit of large  $m$ . This result allows a practitioner to begin with spectral samples from kernel  $k_1$  and re-use those same samples (and thus basis functions) to approximate a different kernel  $k_2$  by simply importance weighting the basis functions. This importance weighting simply requires changing the basis function prior precision matrix but does not require any changes to the basis functions. Ultimately, the result enables a practitioner to change hyperparameters of an interpretable and non-degenerate kernel, re-approximate the kernel by a finite-basis function expansion, and re-evaluate the log-marginal likelihood with a complexity that is independent of the quantity of training data through the use of the techniques discussed in section 5.2.

Theorem 5.1 does have a condition for convergence; the power spectra of the original kernel ( $k_1$ ) and the modified kernel ( $k_2$ ) must share the same support. For many commonly used kernels, the power spectrum support spans all real numbers irrespective of kernel hyperparameters, so this condition is not particularly restrictive. For instance, this condition is met for an exponentiated quadratic kernel since  $\text{Pr}(\boldsymbol{\xi})$  is Gaussian as shown in eq. (5.13).

We visualize the importance weighted random Fourier features kernel in fig. 5.1. The approximations used  $m/2 = 250$  spectral points which were sampled according to the power spectrum of the exponentiated quadratic kernel with lengthscale hyperparameter  $\ell^2 = 0.1$ . The same spectral points (and thus the same basis functions) were used for all three kernels plotted but the fea-

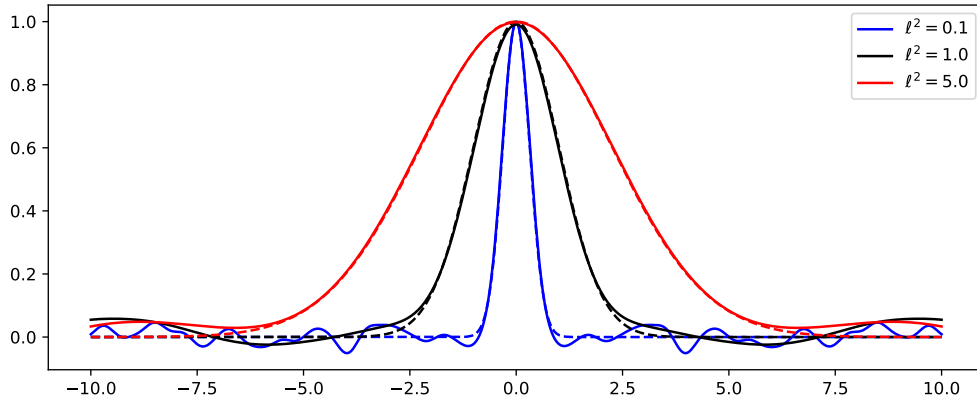


Figure 5.1: Importance weighted random Fourier features kernel with varying lengthscales. The exact exponentiated quadratic kernel being approximated is shown by the broken lines. The approximations used  $m/2=250$  spectral points which were sampled according to the power spectrum of the  $\ell^2=0.1$  kernel.

tures were importance weighted according to theorem 5.1. Evidently the kernel approximation remains accurate despite the significant change in kernel lengthscale from the original ( $\ell^2=0.1$ ) kernel. Note that the random Fourier features kernel approximation remains stationary.

We can also extend the result of theorem 5.1 to asymptotically recover any stationary kernel as  $m \rightarrow \infty$  so long as the power spectra of  $k_1$  has support over all real numbers.

**Theorem 5.2.** *Any real stationary kernel  $k_2$  can be represented as*

$$k_2(\mathbf{x}, \mathbf{z}) = \lim_{m \rightarrow \infty} \phi(\mathbf{x})^T \mathbf{S}^{-1} \phi(\mathbf{z}),$$

where  $\phi: \mathbb{R}^d \rightarrow \mathbb{R}$  is given by eq. (5.12) with  $\xi_i \sim Pr_1(\xi)$  for  $i = 1, 2, \dots, m/2$ ,  $Pr_1(\xi) > 0$  and  $Pr_1(\xi) = Pr_1(-\xi)$  for all  $\xi \in \mathbb{R}^d$ , and  $\mathbf{S}$  is given in theorem 5.1.

This result is a logical extension of theorem 5.1; however, suggests an interesting parameterization. Namely, if the conditions in theorem 5.2 are met and we choose to parameterize  $\mathbf{S}$  by

$$\mathbf{S} = \text{diag}(\mathbf{v}) \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

where  $\mathbf{v} \in \mathbb{R}_+^{m/2}$  is a vector of kernel parameters, modifying the values of  $\mathbf{v}$  allows this parameterization to recover any stationary kernel as  $m \rightarrow \infty$ . This is a powerful parameterization that enables LML evaluations with a complexity that is independent of the quantity of training data while enabling broad class of kernels to be recovered.

As a final remark, theorem 5.2 can be extended to recover any non-stationary covariance function by considering non-stationary random Fourier features as introduced by Ton et al. (2018).

### 5.3.2 The Relevance Vector Machine

The re-weighted basis kernel (eq. (5.1)) has the same structure as the relevance vector machine (Tipping, 2000) and thus the techniques developed in this chapter can be directly applied to this technique. In the simplest form of a relevance vector machine, the basis function weight prior precision is parameterized as  $\mathbf{S} = \text{diag}(\mathbf{s})$ , where  $\mathbf{s} \in (0, \infty)^m$  are separate prior precision hyperparameters for each basis function. When we maximize the evidence with respect to  $\mathbf{s}$  by empirical Bayes, a significant fraction of them will tend to infinity and the posterior distribution over the corresponding weight parameters will be concentrated at zero, thus achieving model sparsity. Training of the relevance vector machine can evidently take advantage of the fast  $\mathcal{O}(m^3)$  LML computations introduced in section 5.2.1 to optimize the precision parameters. The  $\mathcal{O}(m)$  LML computation approach outlined in section 5.2.3 could also be applied to relevance vector machines; however, the resulting model would then be sparse in the transformed parameters which may not be ideal for some applications.

Another approach to relevance vector machines is to place an independent Gamma hyperprior over the precision parameters  $\mathbf{s}$  (such that the effective prior over the basis function weights is an independent Student- $t$  distribution) and then compute the maximum posterior estimate of  $\mathbf{s}$  which will be sparse (Tipping, 2001). While the techniques of this chapter can again be used here to rapidly find the maximum posterior estimate of  $\mathbf{s}$ , the fast LML computations of section 5.2 could alternatively be used to perform type-I inference to marginalize the precision parameters  $\mathbf{s}$ . Such an approach is not typically considered for relevance vector machines since the cost to do so is typically prohibitive; however, this concern is alleviated based upon the techniques outlined in this chapter. Relevance vector machines are considered again in section 6.4.6.

### 5.3.3 Fixed Basis Kernel Interpolation

The re-weighted basis kernel (eq. (5.1)) could be considered as a finite basis function approximation of a non-degenerate kernel  $\bar{k}$  with an infinite basis function expansion. Such a kernel approximation is often used to reduce computational requirements over exact GPs (see section 2.4.1). One approach to taking a finite basis expansion to an exact kernel is using kernel interpolation (Wilson and Nickisch, 2015). In this case,  $k(\mathbf{x}, \mathbf{z}) = \mathbf{w}(\mathbf{x})^T \bar{\mathbf{K}}_{\mathbf{U}, \mathbf{U}} \mathbf{w}(\mathbf{z})$ , where  $\mathbf{w}(\mathbf{x}) \in \mathbb{R}^m$  is the interpolation function between the input and the  $m$  inducing points at locations  $\mathbf{U}$  in input space. This effectively approximates the kernel by interpolating the matrix  $\bar{\mathbf{K}}_{\mathbf{U}, \mathbf{U}} \in \mathbb{R}^{m \times m}$  created by exact evaluations of  $\bar{k}$  on the inducing points  $\mathbf{U}$ .

Natural choices for the interpolation functions are a global GP interpolant  $\mathbf{w}(\mathbf{x}) = \bar{\mathbf{K}}_{\mathbf{U}, \mathbf{U}}^{-1} \mathbf{K}_{\mathbf{U}, \mathbf{x}}$  which gives the Nyström approximation (Smola and Bartlett, 2001; Williams and Seeger, 2001), a Sheppard interpolant or one of its variants such as a nearest neighbours interpolant (Shepard, 1968), or if the set  $\mathbf{U}$  lies on a grid then a sparse Lagrange interpolant (Wilson and Nickisch, 2015).

If the interpolation function is fixed<sup>3</sup>, then we can change the exact kernel being approximated entirely and compute the LML in a time independent of  $n$  using the techniques from section 5.2. To see this, consider the case where  $\Phi = \mathbf{W} \in \mathbb{R}^{n \times m}$ , and  $\mathbf{S}^{-1} = \bar{\mathbf{K}}_{\mathbf{U}, \mathbf{U}}$ .

For this kernel interpolation strategy, we are not restricted to any particular form of kernel that can be used to represent  $\bar{\mathbf{K}}_{\mathbf{U}, \mathbf{U}}$ . For example, if  $\bar{k}$  were an exponentiated quadratic kernel (eq. (2.28)), the kernel lengthscale could be adjusted and the LML could be recomputed in a time that is independent of the number of training observations. We also have the ability to use flexible families of kernels such as spectral mixture kernels (Remes et al., 2017), or deep kernels (Wilson et al., 2016). Alternatively, we do not need to specify a kernel for  $\bar{\mathbf{K}}_{\mathbf{U}, \mathbf{U}}$  at all; we can simply directly parameterize an arbitrary symmetric positive definite matrix allowing any kernel to be approximated. It is important to note that since we can perform LML iterations so rapidly, it is possible to take a fully Bayesian type-I approach to marginalize the hyperparameters of the exact kernel and so are not at risk of overfitting since the uncertainty of these parameters are accounted for.

**Remark:** This approach can be easily extended for additive kernels. One particular case of interest is where the interpolation function for each of  $\ell$  additive kernel components are identical, in which case we would have  $\Phi = \mathbf{1}_\ell^T \otimes \mathbf{W}$ , where  $\mathbf{1}_\ell \in \mathbb{R}^\ell$  is a vector of ones and  $\mathbf{W}$  is the interpolation matrix used for all additive kernel components. This matrix structure, along with the fact that the prior covariance  $\mathbf{S}^{-1}$  is block-diagonal, can give substantial computational savings.

### 5.3.4 Eigenfunction Bases

Considering kernel eigenfunctions for basis functions are attractive since we can approximately recover a wide class of kernels by modifying the weights associated with the kernel eigenfunctions (Buhmann, 2003). For example, as basis functions we could consider grid-structured eigenfunctions (GRIEF) outlined in chapter 4. To do this we can apply the re-weighted basis kernel in eq. (5.1) using the basis functions  $\Phi$  defined in eq. (4.2), and parameterize  $\mathbf{S}$  as a diagonal matrix. We can then fix hyperparameters  $\theta$  such that  $\Phi$  remains constant.

Also, Solin and Särkkä (2020) introduce a finite basis kernel approximation strategy wherein the basis functions used are eigenfunctions of the Laplace operator in a compact subset of  $\mathbb{R}^d$ . Since the eigenfunctions do not change when kernel hyperparameters are changed, the basis function prior is all that needs to be modified to update the hyperparameters. Thus, the use of these kernel approximation strategies together with the techniques presented in section 5.2 could naturally enable LML computations with a complexity that is independent of  $n$  to facilitate fast type-I inference.

---

<sup>3</sup>The interpolation function is naturally fixed with a Sheppard or sparse Lagrange interpolant. A global GP interpolant can be used by fixing the interpolation kernel. Note that by fixing the interpolation kernel, the GP interpolant no longer gives a valid Nyström approximation; however, this fixed interpolation scheme is useful for embedding prior information.

### 5.3.5 Non-Gaussian Likelihoods

So far, only Gaussian likelihoods have been considered; however, this can be extended using a latent Gaussian process, e.g., (Neal, 1997). We can maintain updates to be independent of  $n$  using sparse latent variable updates that only update some of the latent variables between each hyperparameter MCMC iteration. Such a scheme would proceed as follows:

1. Update kernel hyperparameters  $\boldsymbol{\theta}$  given the current latent variable values.
2. Update a subset of latent variables of size  $\tilde{n}$  using the current hyperparameters  $\boldsymbol{\theta}$  and the remaining  $n - \tilde{n}$  latent variables.
3. Return to step 1.

This approach allows the user to select any value of  $1 \leq \tilde{n} \leq n$  to be updated at each iteration, although selecting a value that is too small could lead to slow mixing. Neal (1997) suggested using  $\tilde{n}$  iterations of Gibbs sampling in step two of the preceding algorithm and so could be used for sparse latent variable updates<sup>4</sup>. Titsias et al. (2008) generalized this approach for other sampling schemes that are effective for more general likelihood structures, and discuss other latent variable update schemes that can be used for sparse latent variable updates in step two above (e.g., block-based Metropolis-Hastings). The complexity of step two in the preceding loop depends on the sampling scheme employed; however, a scheme can easily be derived whose complexity is independent of  $n$  and scales in  $m$  with the same complexity as the update in step one.

## 5.4 Experimental Studies

---

**Algorithm 5.1** Summary of the approach outlined in section 5.2.1 to compute the Gaussian process log marginal likelihood (or log evidence) and its derivatives with respect to all kernel hyperparameters with a per-iteration complexity of  $\mathcal{O}(m^3)$ . It is assumed that the prior precision has the diagonal structure  $\mathbf{S} = \text{diag}(\boldsymbol{\gamma}^{-1})$ , where  $\boldsymbol{\gamma} \in \mathbb{R}_+^m$ , and that the algorithm has access to the following quantities that are precomputed before LML iterations begin (see page 67):  $\mathbf{y}^T \mathbf{y} \in \mathbb{R}$ ,  $\mathbf{r} = \boldsymbol{\Phi}^T \mathbf{y} \in \mathbb{R}^m$ , and  $\mathbf{A} = \boldsymbol{\Phi}^T \boldsymbol{\Phi} \in \mathbb{R}^{m \times m}$ .

---

**Input:** Hyperparameters  $\boldsymbol{\gamma} \in \mathbb{R}_+^m$  and  $\sigma^2 \in \mathbb{R}_+$ .

**Output:** Log marginal likelihood  $\text{LML}(\boldsymbol{\gamma}, \sigma^2) \in \mathbb{R}$  and its derivatives  $\frac{\partial \text{LML}(\boldsymbol{\gamma}, \sigma^2)}{\partial \boldsymbol{\gamma}} \in \mathbb{R}^m$ ,  $\frac{\partial \text{LML}(\boldsymbol{\gamma}, \sigma^2)}{\partial \sigma^2} \in \mathbb{R}$ .

$\mathbf{P} = \sigma^2 \text{diag}(\boldsymbol{\gamma}^{-1}) + \mathbf{A} \in \mathbb{R}^{m \times m}$

$\text{LML}(\boldsymbol{\gamma}, \sigma^2) = -\frac{1}{2} \sigma^{-2} (\mathbf{y}^T \mathbf{y} - \mathbf{r}^T \mathbf{P}^{-1} \mathbf{r}), -\frac{1}{2} \log |\mathbf{P}| - \log |\mathbf{S}| + (n - m) \log \sigma^2 - \frac{n}{2} \log(2\pi) \in \mathbb{R}$  (eq. (5.4))

Compute  $\frac{\partial \text{LML}(\boldsymbol{\gamma}, \sigma^2)}{\partial \boldsymbol{\gamma}} \in \mathbb{R}^m$  and  $\frac{\partial \text{LML}(\boldsymbol{\gamma}, \sigma^2)}{\partial \sigma^2} \in \mathbb{R}$  using eq. (5.5).

---

This section empirically assesses the Gaussian process methods introduced in this chapter to accelerate type-I inference. Algorithm 5.1 summarizes the approach outlined in section 5.2.1 to compute the Gaussian process log marginal likelihood (or log evidence) and its derivatives with respect to all kernel hyperparameters with a per-iteration complexity of  $\mathcal{O}(m^3)$ . It

<sup>4</sup>Neal (1997) suggested that the Gibbs sampler loop through the entire training set once or even more between hyperparameter updates since, in the situation considered, the latent variable updates were far cheaper than the hyperparameter updates. In our case, hyperparameter updates are of comparable expense.

is also assumed that the prior precision has the diagonal structure  $\mathbf{S} = \text{diag}(\boldsymbol{\gamma}^{-1})$ , where  $\boldsymbol{\gamma} \in \mathbb{R}_+^m$ . These are the most expensive computations involved in performing type-I inference with hybrid MCMC. Approaches to perform LML computations in  $\mathcal{O}(m^2)$  or  $\mathcal{O}(m)$  use a modification of algorithm 5.1 whose differences are discussed in section 5.2.2 and section 5.2.3, respectively. A python implementation of these methods, along with several tutorials can be found at [https://github.com/treforevans/gp\\_grief](https://github.com/treforevans/gp_grief).

### 5.4.1 One-Dimensional Visualization

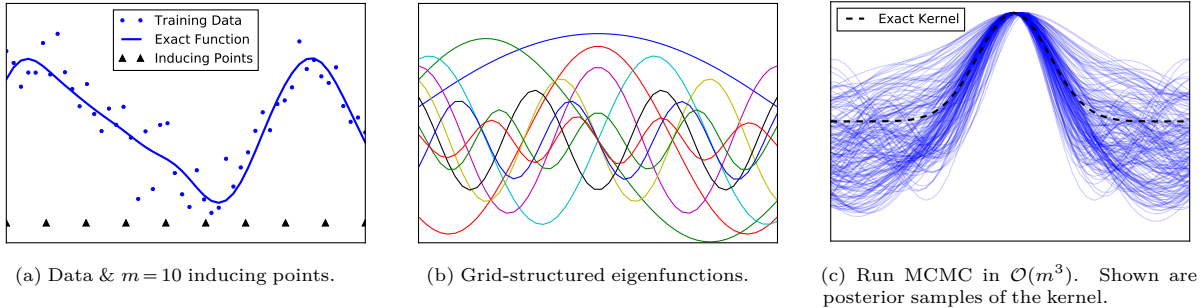


Figure 5.2: One-dimensional regression example demonstrating the GP-GRIEF type-I inference procedure in  $\mathcal{O}(m^3)$ . The posterior samples of the kernel demonstrate the flexibility of this parameterization.

We begin with a one-dimensional visualization showing the flexibility of a type-I inference procedure using the re-weighted basis kernel and the  $\mathcal{O}(m^3)$  LML computations outlined in section 5.2.1. We begin by generating a random dataset with  $n = 50$  points (shown in fig. 5.2a) that is a sample from Gaussian process using the kernel identified by the black dashed line in fig. 5.2c, and corrupted by *i.i.d.* Gaussian noise. As basis functions for the re-weighted basis kernel (eq. (5.1)) we use  $\Phi$  defined in eq. (4.2), and parameterize  $\mathbf{S}$  as a diagonal matrix. The basis functions are therefore approximate kernel eigenfunctions found using the GRIEF approach of chapter 4, as was discussed in section 5.3.4. The GRIEF basis functions are plotted in fig. 5.2b where  $m = 10$ . These basis functions were computed using the inducing points whose locations are shown as black triangles in fig. 5.2a. We place an independent inverse log-normal hyper-prior over the diagonal elements of  $\mathbf{s}$  and a log-normal prior over the Gaussian process noise variance  $\sigma^2$ . We marginalize  $\{\text{diag}(\mathbf{S}), \sigma^2\}$  using Metropolis adjusted Langevin dynamics Markov chain Monte Carlo (MCMC) which uses gradient information (Girolami and Calderhead, 2011). Beginning sampling at the prior mode, we burn the first 1000 samples before running the Markov chain for 10000 iterations and thinning every 50 iterations. Figure 5.2c shows posterior samples of the re-weighted basis function kernel where it is evident that the parameterization has a flexibility to explore kernels with a wide range of effective lengthscales even though only  $m = 10$  basis functions are employed and the MCMC sampling approach is independent of the number of training observations.

Dataset	$n$	$d$	GP-GRIEF-II		GP-GRIEF-I		Yang et al. (2015)
			Time	RMSE	Time	RMSE	RMSE
challenger	23	4	0	0.554±0.277	0.178	<b>0.519±0.261</b>	0.63±0.26
fertility	100	9	0.001	0.172±0.055	0.66	<b>0.166±0.051</b>	0.21±0.05
slump	103	7	0	3.972±1.891	0.566	<b>3.470±1.712</b>	4.72±2.42
automobile	159	25	0.007	0.145±0.057	0.604	<b>0.111±0.036</b>	0.18±0.07
servo	167	4	0	0.280±0.085	0.265	<b>0.268±0.075</b>	0.28±0.09
cancer	194	33	0.007	<b>27.843±3.910</b>	0.667	30.568±3.340	35±4
hardware	209	7	0	0.408±0.046	0.637	<b>0.402±0.045</b>	0.43±0.04
yacht	308	6	0.001	0.170±0.083	0.595	<b>0.120±0.070</b>	0.16±0.11
automp	392	7	0.001	2.607±0.356	0.594	<b>2.563±0.369</b>	2.63±0.38
housing	506	13	0.004	3.212±0.864	0.62	<b>2.887±0.489</b>	2.91±0.54
forest	517	12	0.001	1.386±0.14	0.621	<b>1.384±0.139</b>	1.39±0.16
stock	536	11	0.001	<b>0.005±0.000</b>	0.567	<b>0.005±0.000</b>	0.005±0.001
energy	768	8	0.002	0.49±0.057	0.622	<b>0.461±0.064</b>	0.46±0.07
concrete	1030	8	0.008	5.232±0.723	0.57	5.156±0.766	<b>4.95±0.77</b>
solar	1066	10	0.003	<b>0.786±0.198</b>	0.628	0.809±0.193	0.83±0.20
wine	1599	11	0.012	0.483±0.052	0.583	0.477±0.047	<b>0.47±0.08</b>
skillcraft	3338	19	0.011	<b>0.248±0.016</b>	0.573	<b>0.248±0.016</b>	0.25±0.02
pumadyn	8192	32	0.156	<b>0.20±0.00</b>	0.645	0.212±0.004	<b>0.20±0.00</b>
elevators	16599	18	0.283	0.091±0.002	0.664	0.097±0.001	<b>0.090±0.001</b>
kin40k	40000	8	0.38	<b>0.206±0.004</b>	0.649	<b>0.206±0.004</b>	0.28±0.01
kegg	63608	27	3.642	<b>0.118±0.003</b>	0.704	0.134±0.005	<b>0.12±0.00</b>
3droad	434874	3	0.869	11.648±0.281	0.221	12.966±0.077	<b>10.91±0.05</b>
electric	2049280	11	8.019	0.064±0.002	0.418	<b>0.058±0.006</b>	0.12±0.12

Table 5.1: Mean and standard deviation of test error and average training time (including hyperparameter estimation or MCMC sampling, in hours) from 10-fold cross validation (90% train, 10% test per fold) on UCI regression datasets.

## 5.4.2 UCI Regression Studies

We next assess performance on real-world regression datasets from the UCI repository. Note that the setup of this study follows closely the setup of the studies in section 4.4.3. We report the mean and standard deviation of the root-mean squared error (RMSE) from 10-fold cross validation<sup>5</sup>. Also presented is the mean training time per fold on a machine with two E5-2680 v3 processors. We use a squared-exponential kernel with automatic relevance determination (SE-ARD) and we compare our test errors to those reported by Yang et al. (2015) using type-II inference on the same train-test splits. Yang et al. (2015) used an exact GP with an SE-ARD kernel for datasets with  $n < 2000$ , and Fastfood expansions were used to approximate the SE-ARD kernel for the larger datasets ( $n > 2000$ ).

Before training, we initialize the base kernel hyperparameters,  $\theta$ , by maximizing the marginal likelihood of an exact GP constructed on  $\min(n, 1000)$  points randomly selected from the dataset. We call our model GP-GRIEF-I which uses the kernel from eq. (5.1) using basis functions  $\Phi$  defined in eq. (4.2), and parameterize  $\mathbf{S}$  as a diagonal matrix. The basis functions are therefore approximate kernel eigenfunctions found using the GRIEF approach of chapter 4, as was discussed in section 5.3.4. The base kernel hyperparameters,  $\theta$ , are fixed to the initialized values such that the basis functions remain constant throughout LML iterations. We marginalize  $\{\text{diag}(\mathbf{S}), \sigma^2\}$  using Metropolis adjusted Langevin dynamics MCMC which uses gradient information (Girolami and Calderhead, 2011). The training time includes MCMC sampling,

<sup>5</sup>90% train, 10% test per fold. We use the same splits as Yang et al. (2015)

which we run for 10000 iterations. We use independent log-normal priors with  $\{\text{mode}, \text{variance}\}$  of  $\{1, 100\}$  for  $\text{diag}(\mathbf{S}^{-1})$ , and  $\{\sigma_0^2, 0.04\}$  for the Gaussian process noise variance  $\sigma^2$ , where  $\sigma_0^2$  is the initialized value. We begin sampling at the prior mode, burning the first 1000 samples and thinning every 50 thereafter. For datasets with  $n > 10^6$  we use the  $\mathcal{O}(m)$  LML computations described in section 5.2.3 and otherwise use the  $\mathcal{O}(m^3)$  approach outlined in section 5.2.1. For all studies, we fix  $m = 1000$  and use the distribution of inducing points described in section 4.4.3.

Results are shown in table 5.1 where results from the GP-GRIEF-II model described in section 4.4.3 are repeated here for comparison. It is firstly evident that both GP-GRIEF-I and GP-GRIEF-II outperform the exact GP presented by Yang et al. (2015) on nearly every small dataset ( $n < 2000$ ). In particular, GP-GRIEF-I performs extremely well on these small datasets as we would expect since it uses a very flexible kernel and is robust to over-fitting as a result of the principled type-I Bayesian approach employed. On larger datasets, where we expect the hyperparameter posterior to be more peaked, we see that the type-II techniques begin to be competitive. On these larger datasets, both GP-GRIEF techniques show comparable test error to Yang et al. (2015) on all datasets but perform considerably better on kin40k and the electric dataset with two-million training points. With respect to time, we note that the GP-GRIEF-I model trained extremely rapidly considering a fully Bayesian approach was taken; only 25 minutes were required for the two million point electric dataset even though this size is prohibitive for most GP models taking a type-II empirical Bayes approach.

## 5.5 Conclusions

The new kernel parameterization technique has been outlined along with promising initial results on large real-world datasets where we demonstrated Gaussian process evidence computations in as little as  $\mathcal{O}(m)$  time with  $\mathcal{O}(m)$  storage. This fast training enables type-I Bayesian inference to remain computationally attractive even for very large datasets as we had shown in our studies. We also provided an asymptotic result showing that any stationary kernel can be recovered when using the developed kernel. This is a potentially promising direction that should be explored further in future work.



## Chapter 6

# Scaling Gaussian Processes using Variational Inference

In this chapter we introduce a stochastic variational inference procedure for training scalable Gaussian process (GP) models whose per-iteration complexity is independent of both the number of training points,  $n$ , and the number of basis functions used in the kernel approximation,  $m$ . Our central contributions include an unbiased stochastic estimator of the evidence lower bound (ELBO) for a Gaussian likelihood, as well as a stochastic estimator that lower bounds the ELBO for several other likelihoods such as Laplace and logistic. Independence of the stochastic optimization update complexity on  $n$  and  $m$  enables inference on huge datasets using large capacity GP models. We demonstrate accurate inference on large classification and regression datasets using GPs and relevance vector machines with up to  $m = 10^7$  basis functions. The following paper includes the contents of this chapter:

T. W. Evans and P. B. Nair (2020). “Quadruply Stochastic Gaussian Processes”.  
In: *arXiv preprint arXiv:2006.03015*

### 6.1 Introduction

As discussed in chapter 2, Gaussian process (GP) modelling is a powerful Bayesian approach for classification and regression; however, it is restricted to modestly sized datasets since training and inference require  $\mathcal{O}(n^3)$  time and  $\mathcal{O}(n^2)$  storage, where  $n$  is the number of training observations. This has motivated the development of sparse GP methods that use a small number of basis functions,  $m$  ( $\ll n$ ), to reduce time and memory requirements to  $\mathcal{O}(m^2n + m^3)$  and  $\mathcal{O}(mn + m^2)$ , respectively (e.g., Lázaro-Gredilla et al. (2010), Smola and Bartlett (2001), Snelson and Ghahramani (2006), and Titsias (2009)). However, such techniques perform poorly if too few inducing points are used, and computational savings are lost on complex datasets that require  $m$  to be large.

Wilson and Nickisch (2015) approached these model capacity concerns by exploiting the structure of inducing points placed on a grid, allowing for  $m > n$  while reducing com-

putational demands over an exact GP. This inducing point structure enables performance gains in low dimensions, however, time and storage complexities scale *exponentially* with the dataset dimensionality, rendering the technique intractable for general learning problems unless a dimensionality reduction procedure is applied. Comparatively, the “variational free energy” (VFE) Gaussian process approximation (Titsias, 2009) is a more general technique that makes efficient use of inducing points by optimizing them through a variational objective in an attempt to more accurately capture the true posterior. Many extensions have been made to this approach including a stochastic training procedure that enables dataset sub-sampling (Hensman et al., 2013), we refer to this approach as SVGP. This allows the technique to be extended to large datasets; however, it is inherently restricted in the capacity of the model since it incurs a cost of  $\mathcal{O}(m^3)$  per iteration.

In this work, we address these concerns by introducing a novel stochastic variational inference technique whose per-iteration complexity is  $\mathcal{O}(1)$  (i.e., independent of both  $n$  and  $m$ ), allowing very powerful models to be constructed on large datasets. Note that by *per-iteration* complexity, we refer to computational and storage demands at each iteration of stochastic gradient descent (SGD) and this should not be confused with the expected complexity to converge to a given accuracy, which we do not discuss. Low per-iteration complexity is extremely valuable from a practical perspective since it does not require limiting model capacity or GP approximation accuracy based upon available resources (e.g., GPU memory constraints). We compare the per-iteration complexity of the proposed approach, QSGP, to prior work in table 6.1. Below is a summary of our main contributions

- For regression, an unbiased SGD scheme is developed for estimating the variational parameters whose per-iteration complexity is *independent* of  $n$  and  $m$ ;
- For classification, we develop a SGD scheme that maximizes a lower bound of the ELBO with a per-iteration complexity that is also *independent* of  $n$  and  $m$ ;
- A novel control variate is introduced to reduce the variance of the stochastic gradient approximations without affecting computational complexity; and
- We demonstrate scalability by training powerful models on large classification and regression datasets, using up to  $m = 10^7$  basis functions.

The first two points comprise novel Monte Carlo estimators for the evidence lower bound (ELBO) with a four-level stochastic sampling approach: we sub-sample the  $n$  training examples once, and sub-sample the  $m$  basis functions three times. Because of these four levels of stochasticity, we call the approach “quadruply stochastic Gaussian processes” (QSGP). Section 6.2 describes QSGP for regression problems, whereas section 6.3 outlines QSGP for other likelihoods, including logistic likelihoods for classification. We conclude with numerical studies in section 6.4, but to begin we provide a brief background on Gaussian processes and outline the matrix notations used in this chapter.

	Per-iteration Computation	Per-iteration Storage
Exact GP	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$
VFE (Titsias, 2009)	$\mathcal{O}(nm^2 + m^3)$	$\mathcal{O}(nm + m^2)$
SVGP (Hensman et al., 2013)	$\mathcal{O}(m^3)$	$\mathcal{O}(m^2)$
QSGP	$\mathcal{O}(1)$	$\mathcal{O}(1)$

Table 6.1: Per-iteration complexities for hyper- or variational-parameter optimization. Storage refers to the working memory requirements per SGD iteration (e.g., GPU memory requirements).

**Notation** We use the notations  $\mathbf{h}_{i,:}$ ,  $\mathbf{h}_i$  and  $h_{ij}$  to denote the  $i$ th row,  $i$ th column and  $ij$ th element of the matrix  $\mathbf{H}$ , respectively. Given the sets of indices  $\mathbf{u}$  and  $\mathbf{v}$ ,  $\mathbf{H}_{\mathbf{u},\mathbf{v}}$  denotes a matrix whose  $ij$ th element is given by  $h_{u_i v_j}$ .

**Background on Gaussian Processes** Gaussian processes (GPs) provide non-parametric prior distributions over the latent function that generated a training dataset. It is typically assumed that the dataset is corrupted by independent Gaussian noise with variance  $\sigma^2 \geq 0$  and that the latent function is drawn from a Gaussian process with zero mean and covariance determined by the kernel  $k: \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ . We consider a regression problem where  $\mathbf{X} = \{\mathbf{x}_i \in \mathbb{R}^d\}_{i=1}^n$  and  $\mathbf{y} \in \mathbb{R}^n$  denote the set of  $n$  training point input locations and responses, respectively. Inference can be carried out at the test point,  $\mathbf{x}_* \in \mathbb{R}^d$ , giving the following posterior distribution of the prediction  $y_* \in \mathbb{R}$

$$\Pr(y_* | \mathbf{X}, \mathbf{x}_*) = \mathcal{N}(y_* \mid \mathbf{k}(\mathbf{x}_*)^T (\mathbf{K}_{\mathbf{X},\mathbf{X}} + \sigma^2 \mathbf{I}_n)^{-1} \mathbf{y}, \quad k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}(\mathbf{x}_*)^T (\mathbf{K}_{\mathbf{X},\mathbf{X}} + \sigma^2 \mathbf{I}_n)^{-1} \mathbf{k}(\mathbf{x}_*)),$$

where  $\mathbf{K}_{\mathbf{X},\mathbf{X}} \in \mathbb{R}^{n \times n}$  is the training dataset kernel covariance matrix whose  $ij$ th element is  $k(\mathbf{x}_i, \mathbf{x}_j)$ , and  $\mathbf{k}(\mathbf{x}_*) \in \mathbb{R}^n$  is the cross-covariance matrix between test point  $\mathbf{x}_*$  and the training dataset such that the  $i$ th element is  $k(\mathbf{x}_i, \mathbf{x}_*)$ . For a more thorough introduction to GPs, please see section 2.2.

## 6.2 Unbiased ELBO Estimator in $\mathcal{O}(1)$ for Regression

In this section we again consider a sparse Gaussian process model using the finite basis function kernel approximation  $k(\mathbf{x}, \mathbf{z}) \approx \boldsymbol{\phi}(\mathbf{x})^T \mathbf{S}^{-1} \boldsymbol{\phi}(\mathbf{z})$  introduced earlier in section 2.4.1. This kernel directly specifies a *function space* prior; however, we can also consider a *weight space* perspective to describe the following *equivalent* model: consider a generalized linear model of the form  $f(\mathbf{x}) = \sum_{i=1}^m w_i \phi_i(\mathbf{x})$ , where the weight space prior is  $\Pr(\mathbf{w}) = \mathcal{N}(\mathbf{w} | \mathbf{0}, \mathbf{S}^{-1})$ , and the likelihood is  $\Pr(\mathbf{y} | \mathbf{X}, \mathbf{w}) = \mathcal{N}(\mathbf{y} | \boldsymbol{\Phi} \mathbf{w}, \sigma^2 \mathbf{I}_n)$ . This connection was discussed in detail in section 2.2.1. Recall that in our notation,  $\mathbf{S} \in \mathbb{R}^{m \times m}$  is an SPD weight prior precision matrix,  $\boldsymbol{\Phi} \in \mathbb{R}^{n \times m}$ , where  $\phi_{ij} = \phi_j(\mathbf{x}_i)$ , contains the evaluations of all  $m$  basis functions at all  $n$  training points, and  $\mathbf{w} \in \mathbb{R}^m$  are the weights. Clearly some choices of  $\mathbf{S}$  and  $\boldsymbol{\Phi}$  will result in better kernel approximations than others but we defer discussion about these attributes until section 6.2.3 and will continue our presentation assuming arbitrary choices.

As a result of conjugacy of the Gaussian prior, the posterior of the discussed model is also Gaussian and can be directly computed in closed form in  $\mathcal{O}(m^2n + m^3)$  (see section 2.2.1). However, in this chapter we choose to instead use variational inference to compute the posterior which we show allows for stochastic inference procedures to scale to larger values of  $m$  and  $n$ . The ability to handle larger  $n$  means we can work with larger datasets, while larger  $m$  means a better kernel approximation and subsequently a better GP approximation. We note that for the remainder of this chapter, we focus on the weight space perspective and tailor the following overview of variational inference to this model structure.

Variational inference is a method that can approximate probability densities in Bayesian statistics (Blei et al., 2017; Hoffman et al., 2013; Jordan et al., 1999; Kucukelbir et al., 2017; Ranganath et al., 2014; Wainwright and Jordan, 2008). Focusing on the generalized linear model described previously, we need to compute the posterior  $\Pr(\mathbf{w}|\mathbf{y}, \mathbf{X})$  which is nominally posed as an integration problem. Variational inference turns this task into an optimization problem. By introducing a family of probability distributions  $q(\mathbf{w})$  parameterized by some variational parameters, we minimize the Kullback-Leibler divergence to the exact posterior. This equates to maximization of the evidence lower bound (ELBO) which we can write as follows:

$$\text{ELBO} = \mathbb{E}_{q(\mathbf{w})} [\log \Pr(\mathbf{w}) + \log \Pr(\mathbf{y}|\mathbf{X}, \mathbf{w}) - \log q(\mathbf{w})] \leq \log \Pr(\mathbf{y}|\mathbf{X}), \quad (6.1)$$

where the equality holds if  $\Pr(\mathbf{w}|\mathbf{y}, \mathbf{X}) = q(\mathbf{w})$ . Computation of the ELBO generally requires analytically intractable computations; however, Challis and Barber (2013) show that all terms of eq. (6.1) can be written in closed form as follows for the previously introduced generalized linear model if we choose the Gaussian variational distribution  $q(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ , where  $\boldsymbol{\mu} \in \mathbb{R}^m$  and  $\boldsymbol{\Sigma} \in \mathbb{R}^{m \times m}$

$$\begin{aligned} \text{ELBO} = & -\frac{1}{2} \left[ \underbrace{\frac{1}{\sigma^2} \left( -2\mathbf{y}^T \boldsymbol{\Phi} \boldsymbol{\mu} + \text{sum} \left( (\boldsymbol{\Phi} \boldsymbol{\mu})^2 \right) \right)}_{\mathcal{L}_{\boldsymbol{\mu}}(\boldsymbol{\mu})} + \boldsymbol{\mu}^T \mathbf{S} \boldsymbol{\mu} \right] \\ & - \frac{1}{2} \left[ \underbrace{\frac{1}{\sigma^2} \text{sum} \left( (\boldsymbol{\Phi} \mathbf{C})^2 \right) + \text{tr}(\mathbf{S} \boldsymbol{\Sigma}) - \log |\boldsymbol{\Sigma}|}_{\mathcal{L}_{\boldsymbol{\Sigma}}(\boldsymbol{\Sigma})} \right] - \frac{1}{2} \underbrace{\left[ \log |2\pi \mathbf{S}^{-1}| - m \log(2\pi) - m + n \log(2\pi \sigma^2) + \frac{\mathbf{y}^T \mathbf{y}}{\sigma^2} \right]}_{\mathcal{L}_{\text{const}}}, \end{aligned} \quad (6.2)$$

where  $(\cdot)^2$  refers to the elementwise square of the argument, and we parameterize the variational covariance using the lower triangular Cholesky factorization  $\mathbf{C} \in \mathbb{R}^{m \times m}$  with positive values on the diagonal such that  $\boldsymbol{\Sigma} = \mathbf{C} \mathbf{C}^T$ . While eq. (6.1) can be maximized with respect to  $\{\boldsymbol{\mu}, \mathbf{C}\}$  using deterministic gradient based optimization at  $\mathcal{O}(m^3 + nm^2)$  computations per iteration, the main contribution of this chapter is a novel stochastic training procedure that reduces this complexity to  $\mathcal{O}(1)$  computations per iteration. That is, we demonstrate how to compute an unbiased estimate of the ELBO (and its gradient) in a time that is *independent* of both the quantity of training data  $n$ , and the number of basis functions  $m$ . Leveraging this estimator allows fast iterations of stochastic gradient descent (SGD), the workhorse of many

large-scale machine learning optimization problems.

We first observe that when the prior is fixed, the ELBO in eq. (6.2) is additively separable in the variational mean parameters ( $\boldsymbol{\mu}$ ) and variational covariance parameters ( $\mathbf{C}$ ). This separability enables these parameters to be estimated by solving two decoupled sub-problems which we analyze separately.

### 6.2.1 Learning the Variational Mean

We begin with the following result which provides a novel estimator for  $\mathcal{L}_\mu(\boldsymbol{\mu})$  from eq. (6.2).

**Theorem 6.1.** *An unbiased estimator whose evaluation has a complexity independent of  $n$  and  $m$  can be written as*

$$\mathcal{L}_\mu(\boldsymbol{\mu}) \approx -\frac{2nm}{\sigma^2\tilde{n}\tilde{m}}\mathbf{y}_\ell^T\boldsymbol{\Phi}_{\tilde{\ell},\tilde{\mathbf{i}}}\boldsymbol{\mu}_{\tilde{\mathbf{i}}} + \frac{nm^2}{\sigma^2\tilde{n}\tilde{m}^2}\boldsymbol{\mu}_{\tilde{\mathbf{j}}}^T\boldsymbol{\Phi}_{\tilde{\ell},\tilde{\mathbf{j}}}\boldsymbol{\Phi}_{\tilde{\ell},\tilde{\mathbf{i}}}\boldsymbol{\mu}_{\tilde{\mathbf{i}}} + \frac{m^2}{\tilde{m}^2}\boldsymbol{\mu}_{\tilde{\mathbf{j}}}^T\mathbf{S}_{\tilde{\mathbf{j}},\tilde{\mathbf{i}}}\boldsymbol{\mu}_{\tilde{\mathbf{i}}},$$

where  $\tilde{\mathbf{i}}, \tilde{\mathbf{j}} \in \mathbb{R}^{\tilde{m}}$  both contain indices sampled uniformly from  $\{1, 2, \dots, m\}$ ,  $\tilde{\ell} \in \mathbb{R}^{\tilde{n}}$  contains indices sampled uniformly from  $\{1, 2, \dots, n\}$ , and  $\tilde{m}$  and  $\tilde{n}$  are the number of Monte Carlo samples.

*Proof.* The main idea of the proof is to interpret matrix operations in  $\mathcal{L}_\mu$  as expectations, allowing us to write Monte Carlo estimators of those expectations to allow mini-batch sampling over the rows and columns of all matrices. We begin by re-writing  $\mathcal{L}_\mu$  in eq. (6.2) purely in terms of matrix products as

$$\mathcal{L}_\mu(\boldsymbol{\mu}) = \frac{1}{\sigma^2} \left( -2\mathbf{y}^T\boldsymbol{\Phi}\boldsymbol{\mu} + \text{sum} \left( (\boldsymbol{\Phi}\boldsymbol{\mu})^2 \right) \right) + \boldsymbol{\mu}^T\mathbf{S}\boldsymbol{\mu} = -\frac{2}{\sigma^2}\mathbf{y}^T\boldsymbol{\Phi}\boldsymbol{\mu} + \frac{1}{\sigma^2}\boldsymbol{\mu}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{\mu} + \boldsymbol{\mu}^T\mathbf{S}\boldsymbol{\mu}.$$

We then expand the matrix products to give a sum over  $m$

$$\mathcal{L}_\mu(\boldsymbol{\mu}) = m \sum_{i=1}^m \frac{1}{m} \left( -\frac{2}{\sigma^2}\mathbf{y}^T\boldsymbol{\phi}_i\boldsymbol{\mu}_i + \frac{1}{\sigma^2}\boldsymbol{\mu}^T\boldsymbol{\Phi}^T\boldsymbol{\phi}_i\boldsymbol{\mu}_i + \boldsymbol{\mu}^T\mathbf{s}_i\boldsymbol{\mu}_i \right),$$

and interpreting this as an expectation, where  $p_m(i) = m^{-1}$ ,  $i \in \{1, 2, \dots, m\}$  is a categorical probability distribution gives

$$\mathcal{L}_\mu(\boldsymbol{\mu}) = m \mathbb{E}_{i \sim p_m} \left( -\frac{2}{\sigma^2}\mathbf{y}^T\boldsymbol{\phi}_i\boldsymbol{\mu}_i + \frac{1}{\sigma^2}\boldsymbol{\mu}^T\boldsymbol{\Phi}^T\boldsymbol{\phi}_i\boldsymbol{\mu}_i + \boldsymbol{\mu}^T\mathbf{s}_i\boldsymbol{\mu}_i \right).$$

Finally, we write a Monte Carlo estimator for this expectation to give the following unbiased estimator

$$\mathcal{L}_\mu(\boldsymbol{\mu}) \approx -\frac{2m}{\tilde{m}\sigma^2}\mathbf{y}^T\boldsymbol{\Phi}_{\cdot,\tilde{\mathbf{i}}}\boldsymbol{\mu}_{\tilde{\mathbf{i}}} + \frac{m}{\tilde{m}\sigma^2}\boldsymbol{\mu}^T\boldsymbol{\Phi}^T\boldsymbol{\Phi}_{\cdot,\tilde{\mathbf{i}}}\boldsymbol{\mu}_{\tilde{\mathbf{i}}} + \frac{m}{\tilde{m}}\boldsymbol{\mu}^T\mathbf{S}_{\cdot,\tilde{\mathbf{i}}}\boldsymbol{\mu}_{\tilde{\mathbf{i}}}.$$

The remainder of the proof follows from repeating these successive steps; (i) *expanding* matrix operations, (ii) interpreting the (expanded) matrix operations as *expectations*, and (iii) writing

Monte Carlo estimators for these expectations. Proceeding in this manner completes the proof,

$$\begin{aligned}
\text{Expanding: } \mathcal{L}_\mu(\boldsymbol{\mu}) &\approx -\frac{2m}{\tilde{m}\sigma^2} \mathbf{y}^T \boldsymbol{\Phi}_{:,i} \boldsymbol{\mu}_i + m \sum_{j=1}^m \frac{1}{m} \left( \frac{m}{\tilde{m}\sigma^2} \mu_j \boldsymbol{\phi}_j^T \boldsymbol{\Phi}_{:,i} \boldsymbol{\mu}_i + \frac{m}{\tilde{m}} \mu_j \mathbf{S}_{j,i} \boldsymbol{\mu}_i \right), \\
\text{Expectations: } \mathcal{L}_\mu(\boldsymbol{\mu}) &\approx -\frac{2m}{\tilde{m}\sigma^2} \mathbf{y}^T \boldsymbol{\Phi}_{:,i} \boldsymbol{\mu}_i + m \mathbb{E}_{j \sim p_m} \left( \frac{m}{\tilde{m}\sigma^2} \mu_j \boldsymbol{\phi}_j^T \boldsymbol{\Phi}_{:,i} \boldsymbol{\mu}_i + \frac{m}{\tilde{m}} \mu_j \mathbf{S}_{j,i} \boldsymbol{\mu}_i \right), \\
\text{Monte Carlo: } \mathcal{L}_\mu(\boldsymbol{\mu}) &\approx -\frac{2m}{\tilde{m}\sigma^2} \mathbf{y}^T \boldsymbol{\Phi}_{:,i} \boldsymbol{\mu}_i + \frac{m^2}{\tilde{m}^2\sigma^2} \boldsymbol{\mu}_j^T \boldsymbol{\Phi}_{:,j}^T \boldsymbol{\Phi}_{:,i} \boldsymbol{\mu}_i + \frac{m^2}{\tilde{m}^2} \boldsymbol{\mu}_j^T \mathbf{S}_{j,i} \boldsymbol{\mu}_i, \\
\text{Expanding: } \mathcal{L}_\mu(\boldsymbol{\mu}) &\approx n \sum_{\ell=1}^n \frac{1}{n} \left( -\frac{2m}{\tilde{m}\sigma^2} y_\ell \boldsymbol{\phi}_{\ell,i} \boldsymbol{\mu}_i + \frac{m^2}{\tilde{m}^2\sigma^2} \boldsymbol{\mu}_j^T \boldsymbol{\phi}_{\ell,j}^T \boldsymbol{\phi}_{\ell,i} \boldsymbol{\mu}_i \right) + \frac{m^2}{\tilde{m}^2} \boldsymbol{\mu}_j^T \mathbf{S}_{j,i} \boldsymbol{\mu}_i, \\
\text{Expectations: } \mathcal{L}_\mu(\boldsymbol{\mu}) &\approx n \mathbb{E}_{\ell \sim p_n} \left( -\frac{2m}{\tilde{m}\sigma^2} y_\ell \boldsymbol{\phi}_{\ell,i} \boldsymbol{\mu}_i + \frac{m^2}{\tilde{m}^2\sigma^2} \boldsymbol{\mu}_j^T \boldsymbol{\phi}_{\ell,j}^T \boldsymbol{\phi}_{\ell,i} \boldsymbol{\mu}_i \right) + \frac{m^2}{\tilde{m}^2} \boldsymbol{\mu}_j^T \mathbf{S}_{j,i} \boldsymbol{\mu}_i, \\
\text{Monte Carlo: } \mathcal{L}_\mu(\boldsymbol{\mu}) &\approx -\frac{2nm}{\tilde{n}\tilde{m}\sigma^2} \mathbf{y}_\ell^T \boldsymbol{\Phi}_{\ell,i} \boldsymbol{\mu}_i + \frac{nm^2}{\tilde{n}\tilde{m}^2\sigma^2} \boldsymbol{\mu}_j^T \boldsymbol{\Phi}_{\ell,j}^T \boldsymbol{\Phi}_{\ell,i} \boldsymbol{\mu}_i + \frac{m^2}{\tilde{m}^2} \boldsymbol{\mu}_j^T \mathbf{S}_{j,i} \boldsymbol{\mu}_i.
\end{aligned}$$

□

To learn  $\boldsymbol{\mu}$ , this estimator can be differentiated to give an (unbiased) gradient estimate of  $\mathcal{L}_\mu$  with respect to  $\boldsymbol{\mu}$ . These gradient estimates can then be used to perform SGD, and since the gradient estimator is unbiased and  $\mathcal{L}_\mu$  is convex in  $\boldsymbol{\mu}$ , the process will converge to the unique minimizer of  $\mathcal{L}_\mu$  provided an appropriate learning rate schedule is used (Robbins and Monro, 1951).

Using the stochastic estimate of  $\mathcal{L}_\mu$  in theorem 6.1 is highly advantageous for SGD since each stochastic gradient evaluation no longer depends on  $n$  or  $m$ . This is a significant achievement for large problems since the complexity of evaluating this loss has decreased from  $\mathcal{O}(nm + m^2) \rightarrow \mathcal{O}(\tilde{n}\tilde{m} + \tilde{m}^2)$ , where  $\tilde{n}$  and  $\tilde{m}$  can be chosen to be arbitrarily small (e.g., small enough to store all matrices in GPU memory).

## 6.2.2 Learning the Variational Covariance

Having shown that the variational mean parameters can be found using an SGD procedure whose per-iteration complexity is independent of  $m$  and  $n$ , the following theorem provides a similar result for the variational covariance parameters  $\mathbf{C}$ .

**Theorem 6.2.** *An unbiased estimator whose evaluation has a complexity independent of  $n$  and  $m$  can be written as follows:*

$$\mathcal{L}_\Sigma(\mathbf{C}) \approx \frac{m}{\tilde{m}} \sum_{r \in \tilde{\mathbf{r}}} \left[ \frac{nm^2}{\sigma^2 \tilde{n} \tilde{m}^2} \mathbf{c}_{j,r}^T \boldsymbol{\Phi}_{\ell,j}^T \boldsymbol{\Phi}_{\ell,i} \mathbf{c}_{i,r} + \frac{m^2}{\tilde{m}^2} \mathbf{c}_{j,r}^T \mathbf{S}_{j,i} \mathbf{c}_{i,r} - 2 \log c_{rr} \right],$$

where  $\tilde{\mathbf{r}} \in \mathbb{R}^{\tilde{m}}$  contains indices sampled uniformly from  $\{1, 2, \dots, m\}$ .

*Proof.* This proof proceeds similarly to that of theorem 6.1 where the main idea is to interpret matrix operations in  $\mathcal{L}_\Sigma$  as expectations, allowing us to write Monte Carlo estimators of those expectations to allow mini-batch sampling over the rows and columns of all matrices. We

begin by re-writing  $\mathcal{L}_\Sigma$  in eq. (6.2) purely in terms of matrix products

$$\mathcal{L}_\Sigma(\Sigma) = \frac{1}{\sigma^2} \text{sum} \left( (\Phi \mathbf{C})^2 \right) + \text{tr}(\mathbf{S}\Sigma) - \log|\Sigma| = \sum_{r=1}^m \frac{1}{\sigma^2} \mathbf{c}_r^T \Phi^T \Phi \mathbf{c}_r + \mathbf{c}_r^T \mathbf{S} \mathbf{c}_r - 2 \log c_{rr},$$

interpreting the summation as an expectation, we have

$$\mathcal{L}_\Sigma(\Sigma) = m \mathbb{E}_{r \sim p_m} \left( \frac{1}{\sigma^2} \mathbf{c}_r^T \Phi^T \Phi \mathbf{c}_r + \mathbf{c}_r^T \mathbf{S} \mathbf{c}_r - 2 \log c_{rr} \right),$$

where  $p_m(i) = m^{-1}$ ,  $i \in \{1, 2, \dots, m\}$  is a categorical probability distribution, and writing a Monte Carlo estimator for this expectation gives the following unbiased estimator

$$\mathcal{L}_\Sigma(\Sigma) \approx \frac{m}{\tilde{m}} \sum_{r \in \tilde{\mathbf{r}}} \frac{1}{\sigma^2} \mathbf{c}_r^T \Phi^T \Phi \mathbf{c}_r + \mathbf{c}_r^T \mathbf{S} \mathbf{c}_r - 2 \log c_{rr}.$$

We now proceed in a manner identically to the proof of theorem 6.1 where we repeat three successive steps; (i) *expanding* matrix operations in  $\mathcal{L}_\Sigma$ , (ii) interpreting these (expanded) matrix operations as *expectations*, and (iii) writing *Monte Carlo* estimators for these expectations. Iterating,

$$\begin{aligned} \text{Expanding: } \mathcal{L}_\Sigma(\Sigma) &\approx \frac{m}{\tilde{m}} \sum_{r \in \tilde{\mathbf{r}}} m \sum_{i=1}^m \frac{1}{m} \left( \frac{1}{\sigma^2} \mathbf{c}_r^T \Phi^T \phi_{i, C_{i,r}} + \mathbf{c}_r^T \mathbf{s}_{i, C_{i,r}} \right) - 2 \log c_{rr}, \\ \text{Expectations: } \mathcal{L}_\Sigma(\Sigma) &\approx \frac{m}{\tilde{m}} \sum_{r \in \tilde{\mathbf{r}}} m \mathbb{E}_{i \sim p_m} \left( \frac{1}{\sigma^2} \mathbf{c}_r^T \Phi^T \phi_{i, C_{i,r}} + \mathbf{c}_r^T \mathbf{s}_{i, C_{i,r}} \right) - 2 \log c_{rr}, \\ \text{Monte Carlo: } \mathcal{L}_\Sigma(\Sigma) &\approx \frac{m}{\tilde{m}} \sum_{r \in \tilde{\mathbf{r}}} \frac{m}{\sigma^2 \tilde{m}} \mathbf{c}_r^T \Phi^T \Phi_{:,i} \mathbf{c}_{i,r} + \frac{m}{\tilde{m}} \mathbf{c}_r^T \mathbf{S}_{:,i} \mathbf{c}_{i,r} - 2 \log c_{rr}, \\ \text{Expanding: } \mathcal{L}_\Sigma(\Sigma) &\approx \frac{m}{\tilde{m}} \sum_{r \in \tilde{\mathbf{r}}} m \sum_{j=1}^m \frac{1}{m} \left( \frac{m}{\sigma^2 \tilde{m}} c_{j,r} \phi_j^T \Phi_{:,i} \mathbf{c}_{i,r} + \frac{m}{\tilde{m}} c_{j,r} \mathbf{s}_{j,i} \mathbf{c}_{i,r} \right) - 2 \log c_{rr}, \\ \text{Expectations: } \mathcal{L}_\Sigma(\Sigma) &\approx \frac{m}{\tilde{m}} \sum_{r \in \tilde{\mathbf{r}}} m \mathbb{E}_{j \sim p_m} \left( \frac{m}{\sigma^2 \tilde{m}} c_{j,r} \phi_j^T \Phi_{:,i} \mathbf{c}_{i,r} + \frac{m}{\tilde{m}} c_{j,r} \mathbf{s}_{j,i} \mathbf{c}_{i,r} \right) - 2 \log c_{rr}, \\ \text{Monte Carlo: } \mathcal{L}_\Sigma(\Sigma) &\approx \frac{m}{\tilde{m}} \sum_{r \in \tilde{\mathbf{r}}} \frac{m^2}{\sigma^2 \tilde{m}^2} \mathbf{c}_{j,r}^T \Phi_{:,j}^T \Phi_{:,i} \mathbf{c}_{i,r} + \frac{m^2}{\tilde{m}^2} \mathbf{c}_{j,r}^T \mathbf{S}_{j,i} \mathbf{c}_{i,r} - 2 \log c_{rr}, \\ \text{Expanding: } \mathcal{L}_\Sigma(\Sigma) &\approx \frac{m}{\tilde{m}} \sum_{r \in \tilde{\mathbf{r}}} n \sum_{\ell=1}^n \frac{1}{n} \left( \frac{m^2}{\sigma^2 \tilde{m}^2} \mathbf{c}_{j,r}^T \phi_{\ell,j}^T \phi_{\ell,i} \mathbf{c}_{i,r} \right) + \frac{m^2}{\tilde{m}^2} \mathbf{c}_{j,r}^T \mathbf{S}_{j,i} \mathbf{c}_{i,r} - 2 \log c_{rr}, \\ \text{Expectations: } \mathcal{L}_\Sigma(\Sigma) &\approx \frac{m}{\tilde{m}} \sum_{r \in \tilde{\mathbf{r}}} n \mathbb{E}_{\ell \sim p_n} \left( \frac{m^2}{\sigma^2 \tilde{m}^2} \mathbf{c}_{j,r}^T \phi_{\ell,j}^T \phi_{\ell,i} \mathbf{c}_{i,r} \right) + \frac{m^2}{\tilde{m}^2} \mathbf{c}_{j,r}^T \mathbf{S}_{j,i} \mathbf{c}_{i,r} - 2 \log c_{rr}, \\ \text{Monte Carlo: } \mathcal{L}_\Sigma(\Sigma) &\approx \frac{m}{\tilde{m}} \sum_{r \in \tilde{\mathbf{r}}} \frac{nm^2}{\sigma^2 \tilde{n} \tilde{m}^2} \mathbf{c}_{j,r}^T \Phi_{\ell,j}^T \Phi_{\ell,i} \mathbf{c}_{i,r} + \frac{m^2}{\tilde{m}^2} \mathbf{c}_{j,r}^T \mathbf{S}_{j,i} \mathbf{c}_{i,r} - 2 \log c_{rr}. \end{aligned}$$

□

This estimator provides significant savings over the exact computation of  $\mathcal{L}_\Sigma$  in eq. (6.2) since the complexity has decreased from  $\mathcal{O}(nm^2 + m^3) \rightarrow \mathcal{O}(\tilde{n}\tilde{m}^2 + \tilde{m}^3)$ . Similarly to the  $\mathcal{L}_\mu$  estimator, this estimator can be differentiated to give an (unbiased) gradient estimate of

$\mathcal{L}_\Sigma$  with respect to  $\mathbf{C}$  that can be used for SGD. This estimator makes use of four stochastic estimates: three over the  $m$  basis functions, and one over the  $n$  training examples. Hence we call this estimator “quadruply stochastic” and the subsequent GP a quadruply stochastic Gaussian process (QSGP). As mentioned in the theorem statement, the cost of evaluating this estimator is independent of  $n$  and  $m$ , allowing for highly flexible models to be trained on huge datasets.

In contrast to the proposed approach, modern stochastic variational inference techniques commonly make use of the same mini-batch sampling procedure over the  $n$  training points that is identified here, and a second stochastic estimator is used which samples the variational distribution (Hoffman et al., 2013). This traditional stochastic variational inference approach does eliminate the per-iteration dependence on  $n$ ; however, even just the act of sampling the variational distribution is at least an  $\mathcal{O}(m)$  operation per-iteration before even estimating the ELBO. In our approach, we perform three levels of basis function sub-sampling that enables the cost per iteration to be independent of  $m$ . To the best of our knowledge, this is the first time in the literature an observation has been made that it is possible to perform mini-batch sampling over basis functions while carrying out stochastic variational inference.

One remaining concern with theorem 6.2 as presented is that there can be as many as  $\mathcal{O}(m^2)$  variational covariance parameters in  $\mathbf{C}$  which can be expensive to store if  $m$  is large. Instead, one could consider the lower triangular  $\mathbf{C}$  matrix to have a sparse “chevron” pattern depicted as  $\mathbf{C} = \begin{bmatrix} \blacksquare & & \\ & \blacksquare & \\ & & \blacksquare \end{bmatrix}$ , where the colour indicates non-zero elements. This parameterization allows important posterior correlations between basis functions to be captured and since only the first few columns of  $\mathbf{C}$  are dense, it ensures that the number of variational parameters scale as  $\mathcal{O}(m)$ . Additionally, the computations in theorem 6.2 can be substantially reduced by exploiting the sparsity patterns of this chevron structure for  $\mathbf{C}$ . While this parameterization is not invariant to permutations of basis function indexing, Challis and Barber (2013) demonstrated that it performs comparably to more complex parameterizations. Also, no matter whether a full or chevron parameterization is employed,  $\mathcal{L}_\Sigma$  remains convex (Challis and Barber, 2013) which again ensures that an SGD procedure will converge to the unique minimizer of  $\mathcal{L}_\Sigma$  provided an appropriate learning rate schedule is used. To further reduce the computational burden, the following result demonstrates that  $c_{rr}$  from the diagonalized columns of the chevron Cholesky structure can be computed in closed form.

**Proposition 6.1.** *Given the parameterization  $\mathbf{c}_r = \mathbf{e}_r c_{rr}$ , where  $\mathbf{e}_i \in \mathbb{R}^m$  is the  $i$ th unit vector, maximizing the ELBO with respect to  $c_{rr}$  gives*

$$c_{rr} = \sqrt{\frac{\sigma^2}{\boldsymbol{\phi}_r^T \boldsymbol{\phi}_r + \sigma^2 s_{rr}}}.$$

*Proof.* The optimal  $\mathbf{c}_r$  minimizes  $\mathcal{L}_\Sigma(\mathbf{C})$ . Therefore, writing the terms of  $\mathcal{L}_\Sigma(\mathbf{C})$  from eq. (6.2)



that depend on  $\mathbf{c}_r$  gives the problem statement

$$\operatorname{argmin}_{\mathbf{c}_r} \mathcal{L}_\Sigma(\mathbf{C}) = \operatorname{argmin}_{\mathbf{c}_r} \frac{1}{\sigma^2} \mathbf{c}_r^T \Phi^T \Phi \mathbf{c}_r + \mathbf{c}_r^T \mathbf{S} \mathbf{c}_r - 2 \log c_{rr}.$$

Assuming  $\mathbf{c}_r = \mathbf{e}_r c_{rr}$ , the solution of the preceding optimization problem can be obtained by solving the one-dimensional minimization problem

$$\min_{c_{rr}} \left( \frac{1}{\sigma^2} \phi_r^T \phi_r + s_{rr} \right) c_{rr}^2 - 2 \log c_{rr},$$

setting the derivative with respect to  $c_{rr}$  to zero and solving for  $c_{rr}$  completes the proof,

$$c_{rr} = \sqrt{\frac{\sigma^2}{\phi_r^T \phi_r + \sigma^2 s_{rr}}}.$$

□

### 6.2.3 Empirical Bayes

We have so far described how to learn the variational mean and covariance parameters  $\boldsymbol{\mu}$  and  $\mathbf{C}$  while keeping the prior constant. Practitioners often choose to modify the prior by maximizing the marginal likelihood (or evidence) with respect to a set of hyperparameters (see section 2.3.3). This is referred to as type-II inference or empirical Bayes and we discuss how this can be performed with the QSGP technique to estimate hyperparameters in the kernel  $k$ , which in turn affect  $\mathbf{S}$ , and  $\Phi$ . We will perform empirical Bayes by maximizing the ELBO in eq. (6.1), which is of course a biased surrogate for (i.e., a lower bound of) the log-marginal likelihood; however, it is a widely used approach that performs well in practice (Titsias, 2009). Referring to the ELBO notation in eq. (6.2), we have already shown how to efficiently estimate  $\mathcal{L}_\mu$  and  $\mathcal{L}_\Sigma$ ; however, the term  $\mathcal{L}_{\text{const}}$  also depends on the GP prior so it must be considered as well. The challenging term in  $\mathcal{L}_{\text{const}}$  from eq. (6.2) is  $\log |\mathbf{S}|$  which can be computed cheaply only in special cases (e.g., if  $\mathbf{S}$  is diagonal) but is expensive to compute in the general case. Some kernels that natively<sup>1</sup> admit a diagonal  $\mathbf{S}$  matrix include random Fourier feature kernel approximations (section 5.3.1), grid-structured eigenfunctions kernel approximations (chapter 4), and kernels used for relevance vector machine (section 5.3.2) which all allow empirical Bayes to be easily performed using the following estimator.

**Proposition 6.2.** *Assuming a diagonal  $\mathbf{S}$ , an unbiased estimator whose evaluation complexity is independent of  $n$  and  $m$  can be written as follows:*

$$\mathcal{L}_{\text{const}} \approx -\frac{m}{\bar{m}} \sum_{i \in \bar{\mathbf{i}}} \log s_{ii} - m + n \log(2\pi\sigma^2) + \frac{n}{\sigma^2 \bar{n}} \mathbf{y}_{\bar{\ell}}^T \mathbf{y}_{\bar{\ell}}.$$

<sup>1</sup>Any kernel can be written to have a diagonal  $\mathbf{S}$  using a linear transformation of features; however, performing this transformation would cost  $\mathcal{O}(m^3)$  in general.

*Proof.* Re-writing  $\mathcal{L}_{\text{const}}$  from eq. (6.2) assuming that  $\mathbf{S}$  is diagonal

$$\begin{aligned}\mathcal{L}_{\text{const}} &= \log|2\pi\mathbf{S}^{-1}| - m \log(2\pi) - m + n \log(2\pi\sigma^2) + \frac{1}{\sigma^2} \mathbf{y}^T \mathbf{y}, \\ &= m \log(2\pi) - \sum_{i=1}^m \log s_{ii} - m \log(2\pi) - m + n \log(2\pi\sigma^2) + \frac{1}{\sigma^2} \sum_{\ell=1}^n y_{\ell}^2.\end{aligned}$$

Cancelling terms and interpreting the sums as expectations allows us to write the following unbiased Monte Carlo estimator to complete the proof

$$\mathcal{L}_{\text{const}} \approx -\frac{m}{\tilde{m}} \sum_{i \in \tilde{\mathbf{i}}} \log s_{ii} - m + n \log(2\pi\sigma^2) + \frac{n}{\sigma^2 \tilde{n}} \mathbf{y}_{\tilde{\ell}}^T \mathbf{y}_{\tilde{\ell}}.$$

□

We now have stochastic estimators for each term of the ELBO in eq. (6.2) and have therefore demonstrated how an unbiased approximator of the ELBO can be performed in  $\mathcal{O}(1)$ . While proposition 6.2 requires a diagonal  $\mathbf{S}$ , this estimator only needs to be included when empirical Bayes is being performed. In the fully Bayesian case where the prior is fixed, only theorems 6.1 and 6.2 need to be considered, and these both assume a general, dense  $\mathbf{S}$ . To extend the result in proposition 6.2 to the case of a general  $\mathbf{S}$ , it may be possible to use a ‘‘Russian roulette’’ estimator following Chen et al. (2019) to provide a stochastic estimate of  $\log|\mathbf{S}|$ ; however, this is left for future work.

#### 6.2.4 Control Variates

We discuss here how to reduce the variance of the Monte Carlo estimators introduced in theorems 6.1 and 6.2 and proposition 6.2. Variance reduction is an important consideration in practice since it affects the rate of convergence of stochastic gradient descent (e.g., Gower et al. (2019)). We focus on techniques that will reduce variance while remaining unbiased. A classic technique to reduce variance of Monte Carlo estimators is to introduce a control variate (Owen, 2013). We consider reducing the variance of the term  $\frac{nm^2}{\sigma^2 \tilde{n} \tilde{m}^2} \boldsymbol{\mu}_{\tilde{\mathbf{j}}}^T \boldsymbol{\Phi}_{\tilde{\ell}, \tilde{\mathbf{j}}}^T \boldsymbol{\Phi}_{\tilde{\ell}, \tilde{\mathbf{i}}} \boldsymbol{\mu}_{\tilde{\mathbf{i}}}$  in theorem 6.1 by adding the following terms to the  $\mathcal{L}_{\mu}$  estimator

$$-\frac{nm^2}{\sigma^2 \tilde{n} \tilde{m}^2} \boldsymbol{\mu}_{\tilde{\mathbf{j}}}^T \boldsymbol{\Phi}_{\mathbf{p}, \tilde{\mathbf{j}}}^T \boldsymbol{\Phi}_{\mathbf{p}, \tilde{\mathbf{i}}} \boldsymbol{\mu}_{\tilde{\mathbf{i}}} + \frac{n}{\sigma^2 \tilde{n}} \boldsymbol{\mu}^T \boldsymbol{\Phi}_{\mathbf{p}, :}^T \boldsymbol{\Phi}_{\mathbf{p}, :} \boldsymbol{\mu}, \quad (6.3)$$

where the negative control variate is the first term (which is stochastic) and the second term is the expectation of the control variate (which is deterministic). The fixed set  $\mathbf{p} \in \mathbb{R}^{\tilde{n}}$  contains indices of  $\tilde{n}$  support points that are randomly sub-sampled from the training set before SGD iterations begin. This control variate can reduce variance of the Monte Carlo estimator if there is correlation between the elements of  $\frac{n}{\tilde{n}} \boldsymbol{\Phi}_{\mathbf{p}, :}^T \boldsymbol{\Phi}_{\mathbf{p}, :}$  and  $\boldsymbol{\Phi}^T \boldsymbol{\Phi}$ . We speculate that there would be correlation since the control variate uses an unbiased low-rank approximation of  $\boldsymbol{\Phi}^T \boldsymbol{\Phi}$ . Additionally, it can be easily seen that the expectation of eq. (6.3) is zero, therefore adding this to the theorem 6.1 estimator does not introduce any bias. Equation (6.3) could

also be scaled by a control variate coefficient to further reduce variance (Owen, 2013). We just consider a control variate coefficient of unity in this work.

We can choose  $\bar{n} \ll n$  such that the first term in eq. (6.3) can be computed cheaply; however, computation of the second term (the control variate expectation) evidently requires  $\mathcal{O}(m)$  computations. The following result demonstrates how this second term can be computed in  $\mathcal{O}(1)$  per SGD iteration.

**Proposition 6.3.** *The expected value of the control variate in eq. (6.3) can be computed at iteration  $t$  with a complexity independent of  $n$  and  $m$  as follows:*

$$\frac{n}{\sigma^2 \bar{n}} \boldsymbol{\mu}^T \boldsymbol{\Phi}_{\mathbf{p},:}^T \boldsymbol{\Phi}_{\mathbf{p},:} \boldsymbol{\mu} = \frac{n}{\sigma^2 \bar{n}} \mathbf{a}^{(t)T} \mathbf{a}^{(t)},$$

where  $\mathbf{a}^{(t)} = \mathbf{a}^{(t-1)} + \boldsymbol{\Phi}_{\mathbf{p},\bar{i} \cup \bar{j}} (\boldsymbol{\mu}_{\bar{i} \cup \bar{j}} - \boldsymbol{\mu}_{\bar{i} \cup \bar{j}}^{(t-1)}) \in \mathbb{R}^{\bar{n}}$ ,  $\boldsymbol{\mu}^{(t-1)} \in \mathbb{R}^m$  is the value of  $\boldsymbol{\mu}$  at the end of iteration  $t-1$ ,  $\mathbf{a}^{(0)} = \boldsymbol{\Phi}_{\mathbf{p},:} \boldsymbol{\mu}^{(0)}$ ,  $\boldsymbol{\mu}_{\bar{i} \cup \bar{j}}$  are all the variables being updated at the current SGD iteration, and  $\mathbf{a}^{(t)}$  is saved once  $\boldsymbol{\mu}_{\bar{i} \cup \bar{j}}$  has been updated at the end of iteration  $t$ .

This result can be easily proven by observing that  $\mathbf{a}^{(t-1)} = \boldsymbol{\Phi}_{\mathbf{p},:} \boldsymbol{\mu}^{(t-1)}$ . Clearly the cost of updating  $\mathbf{a}^{(t)}$  at the end of iteration  $t$  requires just  $\mathcal{O}(\bar{n}\tilde{m})$  time, and evidently if optimization begins at  $\boldsymbol{\mu}^{(0)} = \mathbf{0}$ , then we initialize  $\mathbf{a}^{(0)} = \mathbf{0}$ . The requirement for sparse update directions can be met by simply choosing an appropriate optimizer such as regular gradient descent, or AdaGrad (Duchi et al., 2011). Computation of unbiased, sparse gradients of proposition 6.3 is detailed in appendix D.1.1.

Finally, we note that the control variate in eq. (6.3) can also be directly used in theorem 6.2 by simply replacing  $\boldsymbol{\mu}$  with  $\mathbf{c}_r$ . Additionally, we can derive control variates for the other terms in theorems 6.1 and 6.2 which we discuss in appendix D.1.2.

### 6.3 ELBO Lower Bound Estimator in $\mathcal{O}(1)$ for Classification

Previously we assumed that the likelihood  $\Pr(\mathbf{y}|\mathbf{X}, \mathbf{w})$  is Gaussian, and now we generalize our results for other likelihoods so that different types of learning problems can be addressed (e.g., classification). Specifically, we aim to develop a variational bound that can be estimated in  $\mathcal{O}(1)$  for a wide class of likelihoods. For many likelihoods of interest, the second term in eq. (6.1) can be written as

$$\mathbb{E}_{q(\mathbf{w})} [\log \Pr(\mathbf{y}|\mathbf{X}, \mathbf{w})] = \sum_{\ell=1}^n \mathbb{E}_{q(\mathbf{w})} [\log g_{\ell}(\boldsymbol{\phi}_{\ell,:}, \mathbf{w})], \quad (6.4)$$

where  $g_{\ell} : \mathbb{R} \rightarrow \mathbb{R}$  is referred to as a site projection (Challis and Barber, 2013). Examples of site projections for different likelihoods are provided in appendix D.3. Additionally, since we assume  $q(\mathbf{w})$  is Gaussian, we can re-write eq. (6.4) as a sum of one-dimensional expectations

as follows (Barber and Bishop, 1998; Challis and Barber, 2013; Kuss and Rasmussen, 2005):

$$\mathbb{E}_{q(\mathbf{w})}[\log \Pr(\mathbf{y}|\mathbf{X}, \mathbf{w})] = \sum_{\ell=1}^n \mathbb{E}_{\mathcal{N}(z|0,1)} \left[ \log g_{\ell}(\boldsymbol{\phi}_{\ell,:} \boldsymbol{\mu} + z \boldsymbol{\phi}_{\ell,:} \boldsymbol{\Sigma} \boldsymbol{\phi}_{\ell,:}^T) \right], \quad (6.5)$$

which can be easily approximated using quadrature methods when the integral is not analytically tractable. The following result demonstrates how this likelihood expectation can be estimated in  $\mathcal{O}(1)$ .

**Theorem 6.3.** *Assuming the site projection  $g_{\ell}$  is log-concave, the following inequality holds:*

$$\mathbb{E}_{q(\mathbf{w})}[\log \Pr(\mathbf{y}|\mathbf{X}, \mathbf{w})] \geq \frac{n}{\tilde{n}} \mathbb{E} \left[ \sum_{\ell \in \tilde{\ell}} \log g_{\ell} \left( \frac{m}{\tilde{m}} \boldsymbol{\phi}_{\ell, \tilde{\mathbf{i}}} \boldsymbol{\mu}_{\tilde{\mathbf{i}}} + \frac{m^3}{\tilde{m}^3} z \boldsymbol{\phi}_{\ell, \tilde{\mathbf{j}}} \mathbf{C}_{\tilde{\mathbf{j}}, \tilde{\mathbf{r}}} \mathbf{C}_{\tilde{\mathbf{i}}, \tilde{\mathbf{r}}}^T \boldsymbol{\phi}_{\ell, \tilde{\mathbf{i}}}^T \right) \right],$$

where the expectation on the right-hand side is taken over  $\tilde{\mathbf{i}}, \tilde{\mathbf{j}}, \tilde{\mathbf{r}} \in \mathbb{R}^{\tilde{m}}$ ,  $\tilde{\ell} \in \mathbb{R}^{\tilde{n}}$ , and  $z \sim \mathcal{N}(0,1)$ .

*Proof.* Re-writing eq. (6.5),

$$\mathbb{E}_{q(\mathbf{w})}[\log \Pr(\mathbf{y}|\mathbf{X}, \mathbf{w})] = \sum_{\ell=1}^n \mathbb{E}_{\mathcal{N}(z|0,1)} \left[ \log g_{\ell}(\boldsymbol{\phi}_{\ell,:} \boldsymbol{\mu} + z \boldsymbol{\phi}_{\ell,:} \boldsymbol{\Sigma} \boldsymbol{\phi}_{\ell,:}^T) \right],$$

and writing the expression inside the site projections  $g_{\ell}$  as expectations gives

$$\mathbb{E}_{q(\mathbf{w})}[\log \Pr(\mathbf{y}|\mathbf{X}, \mathbf{w})] = \sum_{\ell=1}^n \mathbb{E}_{\mathcal{N}(z|0,1)} \left[ \log g_{\ell} \left( \mathbb{E}_{i,j,r \sim p_m} [m \phi_{\ell,i} \mu_i + m^3 z \phi_{\ell,j} c_{j,r} c_{i,r} \phi_{\ell,i}] \right) \right],$$

where  $p_a(i) = a^{-1}$ ,  $i \in \{1, 2, \dots, a\}$  is a categorical probability distribution. We can also write the sum over  $n$  as an expectation

$$\mathbb{E}_{q(\mathbf{w})}[\log \Pr(\mathbf{y}|\mathbf{X}, \mathbf{w})] = n \mathbb{E}_{\mathcal{N}(z|0,1), \ell \sim p_n} \left[ \log g_{\ell} \left( \mathbb{E}_{i,j,r \sim p_m} [m \phi_{\ell,i} \mu_i + m^3 z \phi_{\ell,j} c_{j,r} c_{i,r} \phi_{\ell,i}] \right) \right].$$

Writing the expectations over mini-batches of size  $\tilde{m}$ ,  $\tilde{n}$  for the distributions over  $p_m$  and  $p_n$ , respectively, gives

$$\mathbb{E}_{q(\mathbf{w})}[\log \Pr(\mathbf{y}|\mathbf{X}, \mathbf{w})] = \frac{n}{\tilde{n}} \mathbb{E}_{\mathcal{N}(z|0,1), \tilde{\ell} \sim p_{\tilde{n}}} \left[ \sum_{\ell \in \tilde{\ell}} \log g_{\ell} \left( \mathbb{E}_{\tilde{\mathbf{i}}, \tilde{\mathbf{j}}, \tilde{\mathbf{r}} \sim p_{\tilde{m}}} \left[ \frac{m}{\tilde{m}} \boldsymbol{\phi}_{\ell, \tilde{\mathbf{i}}} \boldsymbol{\mu}_{\tilde{\mathbf{i}}} + \frac{m^3}{\tilde{m}^3} z \boldsymbol{\phi}_{\ell, \tilde{\mathbf{j}}} \mathbf{C}_{\tilde{\mathbf{j}}, \tilde{\mathbf{r}}} \mathbf{C}_{\tilde{\mathbf{i}}, \tilde{\mathbf{r}}}^T \boldsymbol{\phi}_{\ell, \tilde{\mathbf{i}}}^T \right] \right) \right],$$

where  $p_a^b$  is an  $b$ -dimensional distribution with each dimension *i.i.d.* according to  $p_a$ . Finally, assuming that  $g_{\ell}$  is log-concave, we apply Jensen's inequality to complete the proof

$$\mathbb{E}_{q(\mathbf{w})}[\log \Pr(\mathbf{y}|\mathbf{X}, \mathbf{w})] \geq \frac{n}{\tilde{n}} \mathbb{E}_{\mathcal{N}(z|0,1), \tilde{\ell} \sim p_{\tilde{n}}, \tilde{\mathbf{i}}, \tilde{\mathbf{j}}, \tilde{\mathbf{r}} \sim p_{\tilde{m}}} \left[ \sum_{\ell \in \tilde{\ell}} \log g_{\ell} \left( \frac{m}{\tilde{m}} \boldsymbol{\phi}_{\ell, \tilde{\mathbf{i}}} \boldsymbol{\mu}_{\tilde{\mathbf{i}}} + \frac{m^3}{\tilde{m}^3} z \boldsymbol{\phi}_{\ell, \tilde{\mathbf{j}}} \mathbf{C}_{\tilde{\mathbf{j}}, \tilde{\mathbf{r}}} \mathbf{C}_{\tilde{\mathbf{i}}, \tilde{\mathbf{r}}}^T \boldsymbol{\phi}_{\ell, \tilde{\mathbf{i}}}^T \right) \right].$$

□

Note that we say  $f(x)$  is log-concave if  $\log f(x)$  is concave in  $x$ . This result is clearly more general than the results presented in section 6.2 which were restricted to a Gaussian likelihood.

In fact, several commonly used likelihoods admit log-concave site projections including Gaussian likelihoods, Laplace likelihoods (commonly used for robust regression), and logistic likelihoods (commonly used for classification). For Gaussian and Laplace likelihoods, the expectation over  $z$  in theorem 6.3 can be computed analytically but for a logistic likelihood it must be approximated numerically, ideally using quadrature methods (Challis and Barber, 2013). Additionally, by extending the observations of Challis and Barber (2013) it can be shown that the ELBO approximation remains concave in the variational parameterizations discussed when the estimator in theorem 6.3 is employed. Therefore we can be sure that an SGD procedure will converge to the unique maximizer of the ELBO lower bound provided an appropriate learning rate schedule is used.

As an interesting side note, maximization of the lower bound in theorem 6.3 with  $\Sigma = \mathbf{0}$  is identical to a maximum likelihood learning procedure using dropout (Srivastava et al., 2014) with a dropout rate of  $\frac{m-\tilde{m}}{m}$ . Therefore, it is evident that the dropout objective lower bounds the log-likelihood when applied to linear models (or the final layer of a neural network) when using a likelihood with log-concave site projections. As a result, dropout can achieve regularization through this biasing of its original objective.

Optimizing the lower bound of the ELBO in theorem 6.3 does introduce bias into the inference procedure, and it can be shown that the bias depends on the variance of the estimator over  $\tilde{\mathbf{i}}, \tilde{\mathbf{j}}$ , and  $\tilde{\mathbf{r}}$ . To see this, consider an extreme example where we set  $\tilde{m} = m$  such that we perform computations with the full batch. In this case, the estimator over  $\tilde{\mathbf{i}}, \tilde{\mathbf{j}}, \tilde{\mathbf{r}}$  has zero variance, and the equality in theorem 6.3 holds. Evidently, the bias decreases as  $\tilde{m}$  increases and is eliminated when  $\tilde{m} = m$ .

Another point of consideration is that the bias of the result in theorem 6.3 reduces as the curvature of the log site projection  $\log g_\ell$  decreases (with the bias being eliminated when  $\log g_\ell$  is linear). This result is a property of Jensen’s inequality (which was used to prove theorem 6.3) and has implications on the choice of likelihood used. For example, the form of  $\log g_\ell$  for Gaussian, Laplacian, and Logistic likelihoods are (shifted and scaled) quadratic, absolute value, and softplus functions, respectively (we provide the specific forms of these in appendix D.3). Empirically we find that the bias is small with the Laplace and Logistic likelihoods which would be expected since both the absolute value and softplus functions are effectively piecewise linear (the absolute value function being exactly so). Conversely, a quadratic function generally behaves linearly nowhere, and we find that the estimator in theorem 6.3 exhibits high bias when applied to a Gaussian likelihood. Of course, the estimators in section 6.2 should instead be applied when using a Gaussian likelihood since they give an unbiased approximation of the ELBO.

## 6.4 Numerical Studies

### 6.4.1 Classification Stress Testing

We consider here a binary classification problem with  $n = 60000$  where we predict whether MNIST digits are odd or even integers. For this problem, we consider random Fourier features to approximate the isotropic exponentiated quadratic kernel shown in eq. (2.28) (Lázaro-Gredilla et al., 2010; Rahimi and Recht, 2007). Random Fourier features (introduced in section 5.3.1) are attractive since they natively admit a diagonal  $\mathbf{S}$  matrix which allows empirical Bayes to be easily performed, and the features are randomly generated rather than data dependent, so the dataset does not need to be stored after training. Further, storing the random features can be extremely cheap since we can just save the random seeds and regenerate them as needed (Yan et al., 2015).

To perform inference on this classification problem, we use the stochastic ELBO lower bound estimator in theorem 6.3 with a logistic likelihood, 101 quadrature points for the integral over  $z$ ,  $m = 10^6$  random Fourier features, and mini-batch sizes of  $\tilde{m} = 20000$  and  $\tilde{n} = 100$ . We learn a mean-field variational distribution (i.e., we choose a diagonal structure for  $\mathbf{C}$ ) while simultaneously performing empirical Bayes to estimate the kernel lengthscale and variance. The inference procedure on this huge model was performed in just 11.1 minutes on a machine with a NVIDIA Quadro M5000 GPU, and considering the predictive posterior median, we achieved a hold-out accuracy of 97.85% and a mean negative-log-probability of 0.068 on the test set<sup>2</sup>. We note that this improves upon the benchmark set by Hensman et al. (2015) when approximating the same kernel using SVGP where they achieved an accuracy and mean negative-log-probability of 97.8% and 0.069, respectively.

To further push the capabilities of the proposed QSGP inference approach, using the same experimental setup but with  $m = 10^7$  further decreased the mean negative-log-probability to 0.063 on the test set. It is promising that accurate inference can be performed while sampling only  $\frac{\tilde{m}}{m} = 0.2\%$  of the basis functions at each SGD iteration. This stress test also demonstrates our observation that once  $\tilde{m}$  is sufficiently large, increasing  $m$  does not appear to effect the bias in theorem 6.3, nor does it seem to greatly effect the gradient variance. Therefore, we find it advisable to choose  $m$  as high as possible, subject to practical constraints. We emphasize that such a recommendation is not possible with existing models which scale poorly in  $m$ . For instance, the cost of each SGD iteration is  $\mathcal{O}(m^3)$  for SVGP compared to  $\mathcal{O}(1)$  for the proposed QSGP approach.

### 6.4.2 Control Variate Studies

In this section we assess the control variate outlined in eq. (6.3) and proposition 6.3. To do this, we compare the variance of the Monte Carlo estimator  $\frac{nm^2}{\sigma^2\tilde{n}\tilde{m}^2}\boldsymbol{\mu}_j^T\boldsymbol{\Phi}_{\tilde{\ell},j}^T\boldsymbol{\Phi}_{\tilde{\ell},i}\boldsymbol{\mu}_i$  from theorem 6.1

<sup>2</sup>To put the scale of this problem into perspective, at each iteration of a deterministic solver, a matrix of size 480 Gb would need to be generated at each iteration (assuming double precision floating point values).

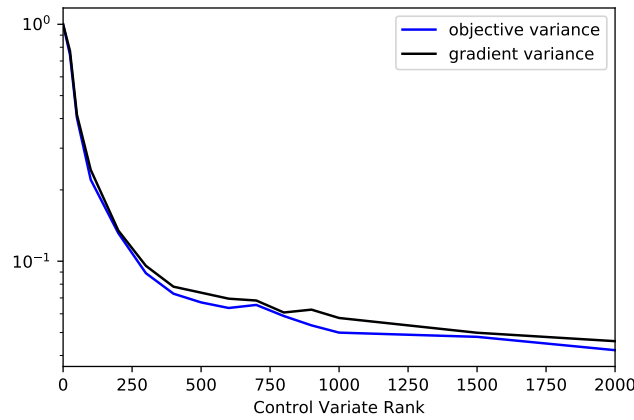


Figure 6.1: Effect of control variate rank  $\bar{n}$  on the Monte Carlo variance of the objective  $\frac{1}{\sigma^2} \boldsymbol{\mu}^T \boldsymbol{\Phi}^T \boldsymbol{\Phi} \boldsymbol{\mu}$ , and the averaged estimator gradient variance with respect to  $\boldsymbol{\mu}$ . Results are normalized.

both with and without the control variate. We compare the variance of both the value of this Monte Carlo objective as well as the gradient of the estimator with respect to  $\boldsymbol{\mu}$ . For the study, we consider the *kin40k* dataset ( $n = 40000$ ,  $d = 8$ ) from the UCI repository, with  $m = 10000$  random Fourier features to approximate the same kernel used at initialization of the regression studies of the following section. We also set  $\boldsymbol{\mu}$  to be a sample from the prior, i.e.,  $\boldsymbol{\mu} \sim \mathcal{N}(\mathbf{0}, \mathbf{S}^{-1})$ , and we use  $\tilde{m} = \tilde{n} = 500$ . Figure 6.1 plots both the variance of the objective, as well as the variance of its gradient with respect to  $\boldsymbol{\mu}$ , averaged over all elements. These variance values are plotted with respect to the rank of the control variate used in the Monte Carlo estimator,  $\bar{n}$ . For each  $\bar{n}$ , the estimator is evaluated 1000 times to obtain the variance. Both the objective and gradient variance are normalized with respect to the variance at  $\bar{n} = 0$  where no control variate is used. The results in fig. 6.1 show a rapid decrease in both objective and gradient variance, even with a small control variate rank. It is promising that even though  $n = 40000$ , when the control variate rank is just  $\bar{n} \approx 300$  the variance has already decreased by an order of magnitude. This reduced gradient variance accelerates the convergence of SGD (e.g., Gower et al. (2019)). We will assess the SGD convergence benefits of the control variate in section 6.4.4.

### 6.4.3 Regression Studies with Type-II Inference

We now consider large regression datasets from the UCI repository using the proposed QSGP inference procedure. We use a chevron Cholesky variational covariance parameterization, use random Fourier features to approximate the exponentiated quadratic kernel (eq. (2.28)) with automatic relevance determination, and we perform empirical Bayes to estimate kernel hyperparameters and the likelihood noise variance  $\sigma^2$  simultaneously with the variational parameters. The results in table 6.2 report the mean and standard deviation of the root mean squared error (RMSE) over five test-train splits (90% train, 10% test per split). Also presented is the mean negative log probability (MNLP) of the predictive posterior on the test data in the first split. The best MNLP value of each row is in boldface whereas the best RMSE value is only in boldface if the difference is statistically significant (if the means differ

by more than three standard deviations). QSGP- $\#$  denotes a quadruply stochastic Gaussian process model where the first  $\#$  columns of  $\mathbf{C}$  have a dense lower triangle (note that QSGP-0 is a mean-field model with a diagonal  $\mathbf{C}$ ). For all QSGP models, we consider  $m = 10^5$  basis functions and mini-batch sizes of  $\tilde{m} = 10000$  and  $\tilde{n} = 500$ . We used the control variate outlined in proposition 6.3 with a control variate rank of  $\bar{n} = 500$  for both the  $\frac{nm^2}{\sigma^2\tilde{n}\tilde{m}^2}\boldsymbol{\mu}_j^T\boldsymbol{\Phi}_{\ell,j}^T\boldsymbol{\Phi}_{\ell,\bar{i}}\boldsymbol{\mu}_{\bar{i}}$  term in theorem 6.1, and the  $\frac{nm^2}{\sigma^2\tilde{n}\tilde{m}^2}\mathbf{c}_{j,r}^T\boldsymbol{\Phi}_{\ell,j}^T\boldsymbol{\Phi}_{\ell,\bar{i}}\mathbf{c}_{\bar{i},r}$  terms in theorem 6.2 for each dense lower triangular column in  $\mathbf{C}$ . Therefore, the QSGP-100 model used control variates for 101 terms in its ELBO estimator, for example.

For the inference procedure we also used an AdaGrad optimizer (Duchi et al., 2011) for the variational parameters with an initial learning rate of 0.1, an Adam optimizer (Kingma and Ba, 2015) for the hyperparameters with an initial learning rate of  $10^{-5}$ , and we decay both these learning rates exponentially over iterations. We run these optimization procedures for  $10^5$  iterations in total and following Hensman et al. (2015) we find it helpful to freeze the hyperparameters for the first  $10^4$  iterations until the variational parameters find a tighter ELBO. We also compare to stochastic variational Gaussian processes (SVGP) using GPFlow (Hensman et al., 2015; Matthews et al., 2017) with 512 inducing points whose locations are learned along with model hyperparameters. For all models,  $\sigma^2$  and all kernel hyperparameters are initialized to the same values found by performing empirical Bayes with an exact GP constructed on 1000 points randomly selected from the dataset. All models were trained on a machine with one GeForce GTX 980 Ti GPU where the maximum training time was 2.1 hours per train-test split for the QSGP-100 model trained on the *ctslice* dataset.

Comparing the MNLP values between the SVGP and QSGP models, we see that the values are comparable across all datasets while the QSGP models performed noticeably better on *kegg* and *ctslice* caused by overconfident predictions made by SVGP. Comparing the RMSE values between the SVGP and QSGP models, we also see that the values are comparable across all datasets with a few exceptions. The largest two deviances include the *ctslice* dataset where the average RMSE of the SVGP model was nearly twice that of the QSGP models, as well as the *kin40k* dataset where the QSGP models also performed significantly better.

Comparing the mean-field model QSGP-0 with a diagonal  $\mathbf{C}$  to QSGP-100 with a chevron  $\mathbf{C}$  structure, we find that QSGP-100 generally admits lower MNLP values, suggesting a more accurate predictive posterior variance. On some datasets such as *ctslice*, this effect is quite dramatic. This effect is expected since the chevron Cholesky parameterization allows important posterior correlations between basis functions to be captured, unlike the mean-field (diagonal  $\mathbf{C}$ ) parameterization which allows no posterior correlations to be captured and tends to give over-confident predictions.

#### 6.4.4 Optimization Trace Studies

In this study we analyze the effect of control variate rank ( $\bar{n}$ ) and basis function minibatch size ( $\tilde{m}$ ) on the optimization trajectory. Figure 6.2 plots the value of the (exact) ELBO verses



Dataset	$n$	$d$	QSGP-0		QSGP-100		SVGP	
			RMSE	MNLP	RMSE	MNLP	RMSE	MNLP
kin40k	40000	8	<b>0.174±0.004</b>	0.135	<b>0.175±0.004</b>	0.139	0.247±0.004	<b>0.055</b>
protein	45730	9	0.602±0.007	1.34	0.602±0.007	1.34	<b>0.542±0.006</b>	<b>1.06</b>
kegg	48827	20	0.135±0.004	-0.821	0.135±0.004	-0.794	0.124±0.005	<b>-0.953</b>
ctslice	53500	385	<b>2.484±0.107</b>	0.377	<b>2.488±0.107</b>	<b>-0.541</b>	4.746±0.087	1.08
keggg	63608	27	0.120±0.004	0.783	0.120±0.004	<b>-0.268</b>	0.120±0.003	36.1
song	515345	90	0.492±0.002	1.26	0.492±0.002	1.26	0.488±0.002	1.26

Table 6.2: Large UCI regression dataset results. Mean and standard deviation of RMS test error over five test-train splits, as well as MNLP of the test set from the first split.

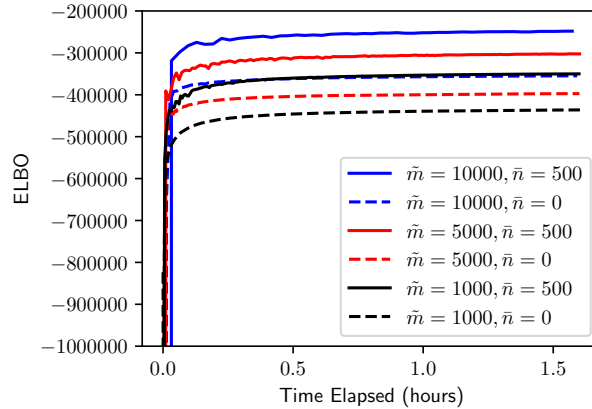


Figure 6.2: Value of the exact ELBO versus wall-clock time for various values of  $\tilde{m}$  and  $\bar{n}$  on the first split of the *kin40k* dataset.

wall-clock time for various values of  $\tilde{m}$  and  $\bar{n}$  on the first split of the *kin40k* dataset (note that the plot is zoomed-in since the initial ELBO magnitude was extremely large). The experimental setup is otherwise identical to that of the QSGP-100 model in section 6.4.3 such that the model at the end of the  $\tilde{m} = 10000, \bar{n} = 500$  curve is exactly the model (i.e., the same RMSE and MNLP) presented in table 6.2. Notice that where the curve flattens, the gradient variance dominates convergence for the chosen learning rate schedule. Here the benefits of the control variate are evident where the performance of the  $\tilde{m} = 10^3, \bar{n} = 500$  curve gives approximately the same ELBO value as the  $\tilde{m} = 10^4, \bar{n} = 0$  curve. In other words, including the control variate allowed reducing the basis function minibatch size by an order of magnitude without degradation of optimization performance. The minibatch size  $\tilde{m}$  also has a noticeable effect on performance, although this effect is less than the inclusion of the control variate. We would also like to emphasize that gradient variance is lesser for the theorem 6.3 estimator for certain site projections. For instance, in the MNIST classification studies of section 6.4.1, fast convergence was observed even though no control variate was used and a relative minibatch size of only  $\frac{\tilde{m}}{m} = 0.002$  was considered.

### 6.4.5 Regression Studies with Inducing Point Kernel Approximations

This section considers additional regression studies with an inducing point kernel approximation rather than the random Fourier feature approximation used in the regression studies of section 6.4.3. We again consider inference on large regression datasets from the UCI repository; however, for this study, we choose to use an inducing point approximation

Dataset	$n$	$d$	Time	QSGP-0		SVGP		Yang et al. (2015)
				RMSE	MNLP	RMSE	MNLP	RMSE
kin40k	40000	8	1.40hrs	<b>0.154±0.007</b>	<b>-0.348</b>	0.171±0.003	-0.089	0.28±0.01
protein	45730	9	2.13hrs	0.598±0.007	<b>1.157</b>	0.598±0.007	1.159	<b>0.53±0.01</b>
kegg	48827	20	1.70hrs	0.124±0.005	-0.554	0.124±0.004	<b>-0.883</b>	0.12±0.01
ctslice	53500	385	5.18hrs	2.574±0.263	0.149	2.938±0.098	<b>-0.440</b>	4.00±0.12
keggu	63608	27	1.46hrs	0.123±0.004	<b>0.848</b>	0.118±0.004	64.014	0.12±0.00
song	515345	90	3.84hrs	0.491±0.002	1.374	0.491±0.002	<b>1.321</b>	0.49±0.00

Table 6.3: Large UCI regression dataset results. Mean and standard deviation of RMS test error and average training time over five test-train splits, as well as MNLP of the test set from the first split. These results differ from those of table 6.2 in the kernel approximation used for QSGP in addition to the fact that empirical Bayes is not performed.

with  $n = m$  inducing points centred on each training point. The inducing point kernel approximation results in  $\Phi = \mathbf{S} = \mathbf{K}_{X,X}$ , the exact kernel covariance matrix between training points. Clearly  $\mathbf{S} = \mathbf{K}_{X,X}$  will be dense in general, therefore we do not consider empirical Bayes in the studies but keep the kernel hyperparameters fixed. This is a natural kernel approximation choice in many ways since it can be shown that the approximation recovers the exact GP prior (i.e.,  $\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_n)$ ), and the learning procedure recovers the exact GP predictive posterior mean. While the exact GP predictive posterior variance is not recovered, we use an augmentation strategy that incurs negligible additional cost to improve the quality of the predictive variance. This augmentation procedure is outlined in proposition D.1.

For the regression studies here, the proposed quadruply stochastic Gaussian process (QSGP) model approximates a squared-exponential kernel with automatic relevance determination and we compare our test errors to those reported by Yang et al. (2015) on the same train-test splits where the same kernel type was approximated using Fastfood expansions. We also compare to stochastic variational Gaussian processes (SVGP) using GPFlow (Hensman et al., 2015; Matthews et al., 2017) with 512 inducing points whose locations are learned<sup>3</sup>. For the QSGP model, a mean-field (diagonal) covariance parameterization is assumed and therefore following the notations of section 6.4.3, we denote this model as QSGP-0. Mini-batch sizes of  $\tilde{n} = \tilde{m} = 3000$  and a rank  $\bar{n} = 200$  control variate was used for all datasets except for *ctslice* which used  $\tilde{n} = \tilde{m} = 1000$ . Optimization was performed with AdaGrad (Duchi et al., 2011). For both QSGP and SVGP,  $\sigma^2$  and all kernel hyperparameters were initialized by performing empirical Bayes with an exact GP constructed on 1000 points randomly selected from the dataset.

Results are presented in table 6.3 where we report the mean and standard deviation of the RMSE over five test-train splits (90% train, 10% test per split). Also presented is the MNLP of the predictive posterior on the test data in the first split, and mean training time per split on a machine with one GeForce GTX 980 Ti graphics card. The best MNLP value of each row is in boldface whereas the best RMSE value is only in boldface if the difference is statistically significant (if the means differ by more than three standard deviations). The RMSE results in table 6.3 demonstrate that QSGP performs well on these large datasets which is not surprising considering that the model has the capacity to recover the exact GP

<sup>3</sup>Note that the SVGP model in this study is identical to the SVGP model considered in section 6.4.3 except here the model hyperparameters are fixed to initial values, as is done for the QSGP model.

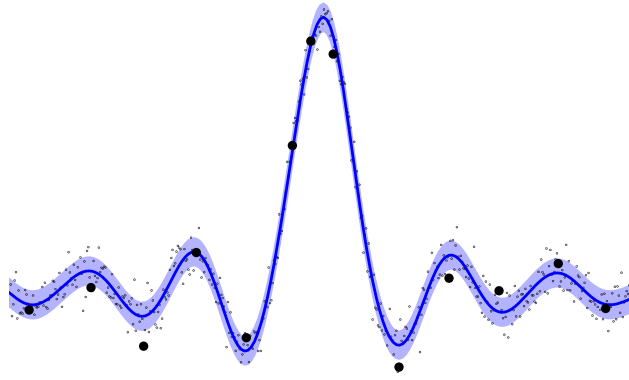


Figure 6.3: QSRVM fit to a noisy sinc dataset with  $n = 500$ . Training points are the small black dots, and the relevance vectors are indicated by large black dots. The shaded blue region is one standard deviation from the predictive posterior mean.

posterior mean. Interestingly, QSGP performs nearly identically to the other sparse GP models for the *kegg*, *keggu*, and *song* datasets. This could indicate that the exact GP mean can be approximated well by lower capacity models for these problems.

Considering the MNLP results, QSGP performed comparably to the SVGP model while performing noticeably better on *keggu*. While a mean-field assumption was made for these studies, we explore going beyond mean-field through the use of the chevron Cholesky structure in the regression results of section 6.4.3.

#### 6.4.6 Relevance Vector Machines

In this section we apply the proposed quadruply stochastic inference procedure to relevance vector machines (also discussed in section 5.3.2), and we refer to this model as the quadruply stochastic relevance vector machine (QSRVM). QSRVMs are identical to the QSGP model except that we parameterize  $\mathbf{S} = \text{diag}(\mathbf{s})$ , where  $\mathbf{s} \in (0, \infty)^m$  are separate prior precision hyperparameters for each basis function. When we maximize the evidence with respect to  $\mathbf{s}$  by empirical Bayes, a significant fraction of them will tend to infinity and the posterior distribution over the corresponding weight parameters will be concentrated at zero, thus achieving model sparsity. For the QSRVM model we consider  $m = n$  basis functions that are kernel evaluations at all  $n$  training points such that  $\Phi = \mathbf{K}_{\mathbf{X}, \mathbf{X}}$ , the exact kernel covariance matrix between training points. After training, the vectors  $\mathbf{x}^{(i)}$  with  $s_i$  finite are referred to as “relevance vectors”. In practice, we consider  $s_i < 10^4$  to be finite.

Figure 6.3 demonstrates the QSRVM model trained on a noisy sinc dataset with  $n = m = 500$  training points (and basis functions). For this model, we also use a chevron Cholesky variational covariance parameterization where the first ten columns of  $\mathbf{C}$  have a dense lower triangle. We also use the control variate outlined in proposition 6.3 for both the variational mean term and the terms corresponding to all ten dense columns of  $\mathbf{C}$  with control variate rank  $\bar{n} = 250$ . For inference, we use mini-batch sizes of  $\tilde{m} = 250$ , and  $\tilde{n} = 100$ .

Figure 6.3 plots both the  $m = n = 500$  training points (and basis function centres), along

with the locations of the discovered relevance vectors that are present in the final model. In the plot, it can be seen that just 13 relevance vectors remained in the model after training. Evidently this model is extremely sparse.

The quadruply stochastic approach to training relevance vector machines introduces a novel strategy that can allow for sparse models to be learned on huge datasets since the per-iteration complexity does not depend on the number of training points or the number of basis functions in the original dictionary. This can be a limitation for alternative RVM training techniques.

## 6.5 Conclusion

The proposed QSGP method demonstrates how stochastic variational inference can be applied to sparse Gaussian processes to give a per-iteration complexity that is independent of both the number of training points and the number of basis functions that define the kernel. The technique therefore enables Gaussian process inference to be performed on huge datasets (large  $n$ ) with highly accurate kernel approximations (since  $m$  can be made large). A novel variance reduction strategy was also developed to accelerate SGD convergence, and the QSGP approach was demonstrated on large regression and classification problems with up to  $m = 10^7$  basis functions to show the scaling capabilities with respect to both dataset size and model capacity.

## Chapter 7

# Variational Inference with Discrete Variable Models

In this chapter we explore a new research direction in Bayesian variational inference with discrete latent variable priors where we exploit Kronecker matrix algebra for efficient and exact computations of the evidence lower bound (ELBO). The proposed DIRECT (DIScrete RELaxation of ConTInuous variables) approach has several advantages over its predecessors: (i) it can exactly compute ELBO gradients (i.e., unbiased, zero-variance gradient estimates), eliminating the need for high-variance stochastic gradient estimators and enabling the use of quasi-Newton optimization methods; (ii) its training complexity is *independent* of the number of training points, permitting inference on large datasets; and (iii) its posterior samples consist of sparse and low-precision quantized integers which permit fast inference on hardware limited devices. In addition, our DIRECT models can exactly compute statistical moments of the parameterized predictive posterior without relying on Monte Carlo sampling. While the DIRECT approach is not practical for all likelihoods, we identify a popular model structure which is practical, and demonstrate accurate inference using latent variables discretized as low-precision 4-bit quantized integers. While the ELBO computations considered in the numerical studies require over  $10^{2352}$  log-likelihood evaluations, we train on datasets with over two-million points in just seconds. A particular application of the methods introduced in this chapter is the discrete relaxation of a Gaussian process prior which we consider in section 7.5.3. The following paper was published from the contents of this chapter:

T. W. Evans and P. B. Nair (2018a). “Discretely Relaxing Continuous Variables for tractable Variational Inference”. In: *Advances in Neural Information Processing Systems*, pp. 10487–10497, spotlight paper.

### 7.1 Introduction

Hardware restrictions posed by mobile devices make Bayesian inference particularly ill-suited for on-board machine learning. This is unfortunate since the safety afforded by Bayesian

statistics is valuable in many prominent mobile applications. The robustness and uncertainty quantification provided by Bayesian inference is therefore very valuable for these applications provided inference can be performed on-board in real-time (Bradshaw et al., 2017; Thrun et al., 2005).

Outside of mobile applications, resource efficiency is still an important concern. For example, deployed models making billions of predictions per day can incur substantial energy costs, making energy efficiency an important consideration in modern machine learning architectures (Louizos et al., 2017).

We approach the problem of efficient Bayesian inference by considering discrete latent variable models such that posterior samples of the variables will be quantized and sparse, leading to efficient inference computations with respect to energy, memory, and computational requirements. Training a model with a discrete prior is typically very slow and expensive, requiring the use of high variance Monte Carlo gradient estimators to learn the variational distribution. The main contribution of this work is the development of a method to rapidly learn the variational distribution for such a model without the use of any stochastic estimators; the objective function will be computed exactly at each iteration. To our knowledge, such an approach has not been taken for variational inference of large-scale probabilistic models.

In this chapter, we compare our work not only to competing stochastic variational inference (SVI) methods for discrete latent variables, but also to the more general SVI methods for continuous latent variables. We make this comparison with continuous variables by discretely relaxing continuous priors using a discrete prior with a finite support set that contains much of the structure and information as its continuous analogue. Using this discretized prior we show that we can make use of Kronecker matrix algebra for efficient and exact ELBO computations. We will call our technique DIRECT (DIScrete RELaxation of ConTInuous variables). We summarize our main contributions below:

- We efficiently and exactly compute the ELBO using a discrete prior even when this computation requires more likelihood evaluations than the number of atoms in the known universe. This achieves unbiased, zero-variance gradients which we show outperforms competing Monte Carlo sampling alternatives that give high-variance gradient estimates while learning.
- Complexity of our ELBO computations are *independent* of the quantity of training data using the DIRECT method, making the proposed approach amenable to big data applications.
- At inference time, we can exactly compute the statistical moments of the parameterized predictive posterior distribution, unlike competing techniques which rely on Monte Carlo sampling.
- Using a discrete prior, our models admit sparse posterior samples that can be represented as quantized integer values to enable efficient inference, particularly on hardware limited

devices.


- We present the DIRECT approach for generalized linear models and deep Bayesian neural networks for regression, and discuss approximations that allow extensions to many other models.
- Our empirical studies demonstrate superior performance relative to competing SVI methods on problems with as many as 2 million training points.

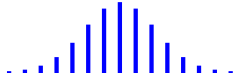
This chapter will proceed as follows; section 7.2 contains a background on variational inference and poses the learning problem to be addressed while section 7.3 outlines the central ideas of the DIRECT method, demonstrating the approach on several popular probabilistic models. Section 7.4 discusses limitations of the proposed approach and outlines some workarounds, for instance, we discuss how to go beyond mean-field variational inference. We empirically demonstrate our approaches in section 7.5, and conclude in section 7.6. Full code for the methods introduced in this chapter is available at <https://github.com/treforevans/direct>.

## 7.2 Variational Inference Background

We begin with a review of variational inference, a method for approximating probability densities in Bayesian statistics (Blei et al., 2017; Hoffman et al., 2013; Jordan et al., 1999; Kucukelbir et al., 2017; Ranganath et al., 2014; Wainwright and Jordan, 2008). We introduce a regression problem for motivation; given  $\mathbf{X} \in \mathbb{R}^{d \times n}$ ,  $\mathbf{y} \in \mathbb{R}^n$ , a  $d$ -dimensional dataset of size  $n$ , we wish to evaluate  $y_*$  at an untried point  $\mathbf{x}_* \in \mathbb{R}^d$  by constructing a statistical model that depends on the  $b$  latent variables in the vector  $\mathbf{w} \in \mathbb{R}^b$ . After specifying a prior over the latent variables,  $\Pr(\mathbf{w})$ , and selecting a probabilistic model structure that admits the likelihood  $\Pr(\mathbf{y}|\mathbf{w})$ , we may proceed with Bayesian inference to determine the posterior  $\Pr(\mathbf{w}|\mathbf{y})$  which generally requires analytically intractable computations.

Variational inference turns the task of computing a posterior into an optimization problem. By introducing a family of probability distributions  $q_{\theta}(\mathbf{w})$  parameterized by  $\theta$ , we minimize the Kullback-Leibler divergence to the exact posterior (Blei et al., 2017). This equates to maximization of the evidence lower bound (ELBO) which we can write as follows for a continuous or discrete prior, respectively:

Prior	ELBO
	$\text{ELBO}(\theta) = \int q_{\theta}(\mathbf{w}) \left( \log \Pr(\mathbf{y} \mathbf{w}) + \log \Pr(\mathbf{w}) - \log q_{\theta}(\mathbf{w}) \right) d\mathbf{w}, \quad (7.1)$

	$\text{ELBO}(\theta) = \mathbf{q}^T \left( \log \boldsymbol{\ell} + \log \mathbf{p} - \log \mathbf{q} \right), \quad (7.2)$
---	---

where  $\log \boldsymbol{\ell} = \{\log \Pr(\mathbf{y}|\mathbf{w}_i)\}_{i=1}^m$ ,  $\log \mathbf{p} = \{\log \Pr(\mathbf{w}_i)\}_{i=1}^m$ ,  $\mathbf{q} = \{q_{\theta}(\mathbf{w}_i)\}_{i=1}^m$ , and  $\{\mathbf{w}_i\}_{i=1}^m = \mathbf{W} \in \mathbb{R}^{b \times m}$  is the entire support set of the discrete prior.

It is immediately evident that computing the ELBO is challenging when  $b$  is large, since in the continuous case eq. (7.1) is a  $b$ -dimensional integral, and in the discrete case the size of the sum in eq. (7.2) generally increases exponentially with respect to  $b$ . Typically, the ELBO is not explicitly computed and instead, a Monte Carlo estimate of the gradient of the ELBO with respect to the variational parameters  $\theta$  is found, allowing stochastic gradient descent to be performed. We will outline some existing techniques to estimate ELBO gradients with respect to the variational parameters,  $\theta$ .

For continuous priors, the reparameterization trick (Kingma and Welling, 2013) can be used to perform variational inference. The technique uses Monte Carlo estimates of the gradient of the evidence lower bound (ELBO) which is maximized during the training procedure. While this approach has been employed successfully for many large-scale models, we find that discretely relaxing continuous latent variable priors can improve training and inference performance when using our proposed DIRECT technique which computes the ELBO (and its gradients) exactly.

When the latent variable priors are discrete, reparameterization cannot be applied; however, the REINFORCE (Williams, 1992) estimator may be used to provide an unbiased estimate of the ELBO gradient during training (alternatively called the score function estimator (Fu, 2006), or likelihood ratio estimator (Glynn, 1990)). Empirically, the REINFORCE gradient estimator is found to give a high variance when compared with reparameterization, leading to a slow learning process. Unsurprisingly, we find that our proposed DIRECT technique trains significantly faster than a model trained using a REINFORCE estimator.

Recent work in variational inference with discrete latent variables has largely focused on continuous relaxations of discrete variables such that reparameterization can be applied to reduce gradient variance compared to REINFORCE. One example is CONCRETE (Jang et al., 2016; Maddison et al., 2016) and its extensions (Grathwohl et al., 2017; Tucker et al., 2017). We consider an opposing direction by identifying how the ELBO (eq. (7.2)) can be computed exactly for a class of discretely relaxed probabilistic models such that the discrete latent variable model can be trained more easily than its continuous counterpart. We outline this approach in the following section.

### 7.3 DIRECT: Efficient ELBO computations

We outline the central ideas of the DIRECT method and illustrate its application on several probabilistic models. The DIRECT method allows us to efficiently and exactly compute the ELBO which has several advantages over existing SVI techniques for discrete latent variable models such as zero-variance gradient estimates (i.e., exact gradient computations), the ability to use a super-linearly convergent quasi-Newton optimizer (since our objective is deterministic), and the per-iteration complexity is independent of training set size. We will also discuss advantages at inference time such as the ability to exactly compute predictive



posterior statistical moments, and to exploit sparse and low-precision posterior samples.

To begin, we consider a discrete prior over our latent variables whose support set  $\mathbf{W}$  forms a Cartesian tensor product grid as most discrete priors do (e.g., any prior that factorizes between variables) so that we can write

$$\mathbf{W} = \begin{pmatrix} \bar{\mathbf{w}}_1^T & \otimes & \mathbf{1}_{\bar{m}}^T & \otimes & \cdots & \otimes & \mathbf{1}_{\bar{m}}^T \\ \mathbf{1}_{\bar{m}}^T & \otimes & \bar{\mathbf{w}}_2^T & \otimes & \cdots & \otimes & \mathbf{1}_{\bar{m}}^T \\ \vdots & & \vdots & & \ddots & & \vdots \\ \mathbf{1}_{\bar{m}}^T & \otimes & \mathbf{1}_{\bar{m}}^T & \otimes & \cdots & \otimes & \bar{\mathbf{w}}_b^T \end{pmatrix}, \quad (7.3)$$

where  $\mathbf{1}_{\bar{m}} \in \mathbb{R}^{\bar{m}}$  denotes a vector of ones,  $\bar{\mathbf{w}}_i \in \mathbb{R}^{\bar{m}}$  contains the  $\bar{m}$  discrete values that the  $i$ th latent variable  $w_i$  can take<sup>1</sup>,  $m = \bar{m}^b$ , and  $\otimes$  denotes the Kronecker product (Van Loan, 2000). Since the number of columns of  $\mathbf{W} \in \mathbb{R}^{b \times \bar{m}^b}$  increases exponentially with respect to  $b$ , it is evident that computing the ELBO in eq. (7.2) is typically intractable when  $b$  is large. For instance, forming and storing the matrices involved naively require exponential time and memory. We can alleviate this concern if  $\mathbf{q}$ ,  $\log \mathbf{p}$ ,  $\log \ell$ , and  $\log \mathbf{q}$  can be written as a sum of Kronecker product vectors (i.e.,  $\sum_i \otimes_{j=1}^b \mathbf{f}_j^{(i)}$ , where  $\mathbf{f}_j^{(i)} \in \mathbb{R}^{\bar{m}}$ ). If we find this structure, then we never need to explicitly compute or store a vector of length  $m$ . This is because eq. (7.2) would simply require multiple inner products between Kronecker product vectors which the following result demonstrates can be computed extremely efficiently.

**Proposition 7.1.** *The inner product between two Kronecker product vectors  $\mathbf{k} = \otimes_{i=1}^b \mathbf{k}^{(i)}$ , and  $\mathbf{a} = \otimes_{i=1}^b \mathbf{a}^{(i)}$  can be computed as follows (Van Loan, 2000):*

$$\mathbf{a}^T \mathbf{k} = \prod_{i=1}^b \mathbf{a}^{(i)T} \mathbf{k}^{(i)}, \quad (7.4)$$

where  $\mathbf{a}^{(i)} \in \mathbb{R}^{\bar{m}}$ ,  $\mathbf{a} \in \mathbb{R}^{\bar{m}^b}$ ,  $\mathbf{k}^{(i)} \in \mathbb{R}^{\bar{m}}$ , and  $\mathbf{k} \in \mathbb{R}^{\bar{m}^b}$ .

This result enables substantial savings in the computation of the ELBO since each inner product computation is reduced from the naive *exponential*  $\mathcal{O}(\bar{m}^b)$  cost to a *linear*  $\mathcal{O}(b\bar{m})$  cost.

We now discuss how the Kronecker product structure of the variables in eq. (7.2) can be achieved. Firstly, if the prior is chosen to factorize between latent variables, as it often is, (i.e., if  $\Pr(\mathbf{w}) = \prod_{i=1}^b \Pr(w_i)$ ) then  $\mathbf{p} = \otimes_{i=1}^b \mathbf{p}_i$  admits a Kronecker product structure where  $\mathbf{p}_i = \{\Pr(w_i = \bar{w}_{ij})\}_{j=1}^{\bar{m}} \in (0, 1)^{\bar{m}}$ . The following result demonstrates how this structure for  $\mathbf{p}$  enables  $\log \mathbf{p}$  to be written as a sum of  $b$  Kronecker product vectors.

**Proposition 7.2.** *The element-wise logarithm of the Kronecker product vector  $\mathbf{k} = \otimes_{i=1}^b \mathbf{k}^{(i)}$*

<sup>1</sup>The discrete values that the  $i$ th latent variable can take,  $\bar{\mathbf{w}}_i$ , may be chosen a priori or learned during ELBO maximization (may be helpful for coarse discretizations). For the sake of simplicity, we focus on the former.

can be written as a sum of  $b$  Kronecker product vectors as follows:

$$\log \mathbf{k} = \bigoplus_{i=1}^b \log \mathbf{k}^{(i)}, \quad (7.5)$$

where  $\mathbf{k}^{(i)} \in \mathbb{R}^{\bar{m}}$ ,  $\mathbf{k} \in \mathbb{R}^{\bar{m}^b}$  contain positive values, and  $\bigoplus$  is a generalization of the Kronecker sum (Horn and Johnson, 1994) for vectors which we define as follows:

$$\bigoplus_{i=1}^b \log \mathbf{k}^{(i)} = \sum_{i=1}^b \left( \bigotimes_{j=1}^{i-1} \mathbf{1}_{\bar{m}} \right) \otimes \log \mathbf{k}^{(i)} \otimes \left( \bigotimes_{j=i+1}^b \mathbf{1}_{\bar{m}} \right). \quad (7.6)$$

This result is easy to verify by analyzing the structure of the log of the expanded vector  $\mathbf{k}$ . Proceeding, we first consider a mean-field variational distribution that factorizes over latent variables such that both  $\mathbf{q} = \bigotimes_{i=1}^b \mathbf{q}_i$  and  $\log \mathbf{q} = \bigoplus_{i=1}^b \log \mathbf{q}_i$  can be written as a sum of Kronecker product vectors, where  $\mathbf{q}_j = \{\Pr(w_j = \bar{w}_{ji})\}_{i=1}^{\bar{m}} \in (0,1)^{\bar{m}}$  are used as the variational parameters,  $\boldsymbol{\theta}$ , with the use of the softmax function. For the mean-field case we can rewrite eq. (7.2) as

$$\text{ELBO}(\boldsymbol{\theta}) = \mathbf{q}^T \log \boldsymbol{\ell} + \sum_{i=1}^b \mathbf{q}_i^T \log \mathbf{p}_i - \sum_{i=1}^b \mathbf{q}_i^T \log \mathbf{q}_i, \quad (7.7)$$

where we use the fact that  $\mathbf{q}_i$  defines a valid probability distribution for the  $i$ th latent variable such that  $\mathbf{q}_i^T \mathbf{1}_{\bar{m}} = 1$ . We extend results to unfactorized prior and variational distributions later in section 7.4.2.

The structure of  $\log \boldsymbol{\ell}$  depends on the probabilistic model used; in the worst case,  $\log \boldsymbol{\ell}$  can always be represented as a sum of  $m$  Kronecker product vectors. However, many models admit a far more compact structure where dramatic savings can be realized as we demonstrate in the following sections.

### 7.3.1 Generalized Linear Regression

We first focus on the popular class of Bayesian generalized linear models (GLMs) for regression. While the Bayesian integrals that arise in GLMs can be easily computed in the case of conjugate priors, for general priors inference is challenging.

This highly general model architecture has been applied in a vast array of application areas. Perhaps most notably, Gaussian processes can be seen as generalized linear models with Gaussian priors on the weights, see section 2.2.1. Therefore as one application of the developed approaches, the DIRECT method can be used to discretely relax Gaussian process priors.

Consider the generalized linear regression model  $\mathbf{y} = \Phi \mathbf{w} + \boldsymbol{\epsilon}$ , where  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_n)$ , and  $\Phi = \{\phi_j(\mathbf{x}_i)\}_{i,j} \in \mathbb{R}^{n \times b}$  contains the evaluations of the basis functions on the training data. The following result demonstrates how the ELBO can be exactly and efficiently computed, assuming the factorized prior and variational distributions over  $\mathbf{w}$  discussed earlier. Note that we also consider a prior over  $\sigma^2$ .

**Theorem 7.1.** *The ELBO can be exactly computed for a discretely relaxed regression GLM as follows:*

$$\begin{aligned} \text{ELBO}(\boldsymbol{\theta}) = & -\frac{n}{2} \mathbf{q}_\sigma^T \log \sigma^2 - \frac{1}{2} (\mathbf{q}_\sigma^T \sigma^{-2}) \left( \mathbf{y}^T \mathbf{y} - 2\mathbf{s}^T (\boldsymbol{\Phi}^T \mathbf{y}) + \mathbf{s}^T \boldsymbol{\Phi}^T \boldsymbol{\Phi} \mathbf{s} - \text{diag}(\boldsymbol{\Phi}^T \boldsymbol{\Phi})^T \mathbf{s}^2 + \right. \\ & \left. \sum_{j=1}^b \mathbf{q}_j^T \mathbf{h}_j \right) + \sum_{i=1}^b (\mathbf{q}_i^T \log \mathbf{p}_i - \mathbf{q}_i^T \log \mathbf{q}_i) + \mathbf{q}_\sigma^T \log \mathbf{p}_\sigma - \mathbf{q}_\sigma^T \log \mathbf{q}_\sigma, \end{aligned} \quad (7.8)$$

where  $\mathbf{q}_\sigma, \mathbf{p}_\sigma \in \mathbb{R}^{\bar{m}}$  are factorized variational and prior distributions over the Gaussian noise variance  $\sigma^2$  for which we consider the discrete positive values  $\sigma^2 \in \mathbb{R}^{\bar{m}}$ , respectively. Also, we use the shorthand notation  $\mathbf{H} = \{\bar{\mathbf{w}}_j^T \sum_{i=1}^n \phi_{ij}^2\}_{j=1}^b \in \mathbb{R}^{\bar{m} \times b}$  (whose  $j$ th column is  $\mathbf{h}_j \in \mathbb{R}^{\bar{m}}$ ), and  $\mathbf{s} = \{\mathbf{q}_j^T \bar{\mathbf{w}}_j\}_{j=1}^b \in \mathbb{R}^b$ .

*Proof.* For a generalized linear regression model with a prior over  $\sigma^2$ , we can re-write eq. (7.7) as follows:

$$\text{ELBO}(\boldsymbol{\theta}) = (\mathbf{q}_\sigma \otimes \mathbf{q})^T \log \ell + \sum_{i=1}^b \mathbf{q}_i^T \log \mathbf{p}_i - \sum_{i=1}^b \mathbf{q}_i^T \log \mathbf{q}_i + \mathbf{q}_\sigma^T \log \mathbf{p}_\sigma - \mathbf{q}_\sigma^T \log \mathbf{q}_\sigma, \quad (7.9)$$

where we have simply expanded the factorized variational distribution to include  $\sigma^2$ , resulting in the two extra terms. To complete the ELBO in eq. (7.9), we need to take the inner product between the variational distribution and log-likelihood for each point in the hypothesis space,  $(\mathbf{q}_\sigma \otimes \mathbf{q})^T \log \ell$ . We can write this relation as follows for our generalized linear regression model (see e.g., Bishop (2006)):

$$(\mathbf{q}_\sigma \otimes \mathbf{q})^T \log \ell = -\frac{n}{2} \mathbf{q}_\sigma^T \log \sigma^2 - \frac{1}{2} (\mathbf{q}_\sigma^T \sigma^{-2}) (\mathbf{q}^T \{(\mathbf{y} - \boldsymbol{\Phi} \mathbf{w}_i)^T (\mathbf{y} - \boldsymbol{\Phi} \mathbf{w}_i)\}_{i=1}^m}), \quad (7.10)$$

whose computation would be prohibitively expensive when  $m = \bar{m}^b$  is large. We will now focus on computing the inner product involving the variational distribution over the  $\mathbf{w}$  variables,  $\mathbf{q}$ , which we can break into three terms as follows:

$$\mathbf{q}^T \{(\mathbf{y} - \boldsymbol{\Phi} \mathbf{w}_i)^T (\mathbf{y} - \boldsymbol{\Phi} \mathbf{w}_i)\}_{i=1}^m = \mathbf{y}^T \mathbf{y} - 2\mathbf{q}^T \{\mathbf{y}^T \boldsymbol{\Phi} \mathbf{w}_i\}_{i=1}^m + \mathbf{q}^T \{\mathbf{w}_i^T \boldsymbol{\Phi}^T \boldsymbol{\Phi} \mathbf{w}_i\}_{i=1}^m, \quad (7.11)$$

for which the first term is trivial to compute as written since it does not depend on  $\mathbf{w}$ . We now demonstrate how the second and third terms can be computed, recalling that we have assumed that  $\mathbf{q}$  is a mean-field variational distribution. Firstly, define  $\mathbf{Z} = (\boldsymbol{\Phi} \mathbf{W})^T = \{\oplus_{j=1}^b \phi_{ij} \bar{\mathbf{w}}_j\}_{i=1}^n \in \mathbb{R}^{m \times n}$  whose columns contain the model prediction for a single training point at every possible set of latent variable values in the hypothesis space. Observe that each column is represented as a sum of  $b$  Kronecker product vectors. We can then write the second term of eq. (7.11) as

$$\mathbf{q}^T \{\mathbf{y}^T \boldsymbol{\Phi} \mathbf{w}_i\}_{i=1}^m = \sum_{i=1}^n y_i \mathbf{q}^T \mathbf{z}_i = \sum_{i=1}^n y_i \sum_{j=1}^b \phi_{ij} \mathbf{q}_j^T \bar{\mathbf{w}}_j = \sum_{j=1}^b \mathbf{q}_j^T \bar{\mathbf{w}}_j \left( \sum_{i=1}^n y_i \phi_{ij} \right) = \mathbf{s}^T (\boldsymbol{\Phi}^T \mathbf{y}), \quad (7.12)$$

where  $\mathbf{s} = \{\mathbf{q}_j^T \bar{\mathbf{w}}_j\}_{j=1}^b \in \mathbb{R}^b$ . Finally, considering the third term of eq. (7.11), observe that we can write  $\{\mathbf{w}_i^T \Phi^T \Phi \mathbf{w}_i\}_{i=1}^m = \sum_{i=1}^m \mathbf{z}_i^2$ , and since each  $\mathbf{z}_i$  is a sum of  $b$  Kronecker product vectors,  $\mathbf{z}_i^2$  a sum of  $(b+b^2)/2$  Kronecker product vectors. We can then write the third term of eq. (7.11) as follows:

$$\mathbf{q}^T \{\mathbf{w}_i^T \Phi^T \Phi \mathbf{w}_i\}_{i=1}^m = \sum_{i=1}^n \sum_{j=1}^b \mathbf{q}_j^T \bar{\mathbf{w}}_j^2 \phi_{ij}^2 + 2 \sum_{k=j+1}^b \phi_{ij} \phi_{ik} (\mathbf{q}_j^T \bar{\mathbf{w}}_j) (\mathbf{q}_k^T \bar{\mathbf{w}}_k), \quad (7.13)$$

$$= \sum_{j=1}^b \mathbf{q}_j^T \left( \bar{\mathbf{w}}_j^2 \sum_{i=1}^n \phi_{ij}^2 \right) + 2 \sum_{k=j+1}^b s_j s_k \left( \sum_{i=1}^n \phi_{ij} \phi_{ik} \right), \quad (7.14)$$

$$= \mathbf{s}^T \Phi^T \Phi \mathbf{s} - \text{diag}(\Phi^T \Phi)^T \mathbf{s}^2 + \sum_{j=1}^b \mathbf{q}_j^T \mathbf{h}_j, \quad (7.15)$$

where we have used the short-hand notation  $\mathbf{H} = \{\bar{\mathbf{w}}_j^2 \sum_{i=1}^n \phi_{ij}^2\}_{j=1}^b \in \mathbb{R}^{\bar{m} \times b}$ . Substituting eq. (7.12) and eq. (7.15) into eq. (7.11), we can re-write the inner product between the variational distribution and the log-likelihood in eq. (7.10) as follows:

$$\begin{aligned} (\mathbf{q}_\sigma \otimes \mathbf{q})^T \log \ell = & -\frac{n}{2} \mathbf{q}_\sigma^T \log \sigma^2 - \frac{1}{2} (\mathbf{q}_\sigma^T \sigma^{-2}) \left( \mathbf{y}^T \mathbf{y} - 2 \mathbf{s}^T (\Phi^T \mathbf{y}) + \mathbf{s}^T \Phi^T \Phi \mathbf{s} - \right. \\ & \left. \text{diag}(\Phi^T \Phi)^T \mathbf{s}^2 + \sum_{j=1}^b \mathbf{q}_j^T \mathbf{h}_j \right), \end{aligned} \quad (7.16)$$

and substituting this into eq. (7.9) completes the proof.  $\square$

We can pre-compute the terms  $\mathbf{y}^T \mathbf{y}$ ,  $\Phi^T \mathbf{y}$ ,  $\mathbf{H}$ , and  $\Phi^T \Phi$  before training begins (since these do not depend on the variational parameters) such that the final complexity of the proposed DIRECT method outlined in theorem 7.1 is only  $\mathcal{O}(b\bar{m} + b^2)$ . This complexity is *independent* of the number of training points, making the proposed technique ideal for massive datasets. Also, each of the pre-computed terms can easily be updated as more data is observed making the techniques amenable to online learning applications.

### Predictive Posterior Computations

Typically, the predictive posterior distribution is found by sampling the variational distribution at a large number of points and running the model forward for each sample. To exactly compute the statistical moments, a model would have to be run forward at every point in the hypothesis space which is typically intractable; however, we can exploit Kronecker matrix algebra to efficiently compute these moments exactly. For example, the exact predictive posterior mean for our generalized linear regression model is computed as follows:

$$\mathbb{E}(y_*) = \sum_{i=1}^m q(\mathbf{w}_i) \int y_* \Pr(y_* | \mathbf{w}_i) dy_*, = \Phi_* \mathbf{W} \mathbf{q} = \Phi_* \mathbf{s}, \quad (7.17)$$

where  $\mathbf{s} = \{\mathbf{q}_j^T \bar{\mathbf{w}}_j\}_{j=1}^b \in \mathbb{R}^b$ , and  $\Phi_* \in \mathbb{R}^{1 \times b}$  contains the basis functions evaluated at  $\mathbf{x}_*$ . This computation is highly efficient, requiring just  $\mathcal{O}(b)$  time per test point. It can be shown that a similar scheme can be derived to exactly compute higher order statistical moments, such as the predictive posterior variance, for generalized linear regression models and other DIRECT models.

We have shown how to exactly compute statistical moments, and now we show how to exploit our discrete prior to compute predictive posterior samples efficiently. This sampling approach may be preferable to the exact computation of statistical moments on hardware limited devices where we need to perform inference with extreme memory, energy, and computational efficiency. The latent variable posterior samples  $\widetilde{\mathbf{W}} \in \mathbb{R}^{b \times \text{num. samples}}$  will of course be represented as a low-precision quantized integer array because of the discrete support of the prior which enables extremely compact storage in memory. Much work has been done elsewhere in the machine learning community to quantize variables for storage compression purposes since memory is a very restrictive constraint on mobile devices (Chen et al., 2015; Gong et al., 2014; Han et al., 2015; Zhou et al., 2017). However, we can go beyond this to additionally reduce computational and energy demands for the evaluation of  $\Phi_* \widetilde{\mathbf{W}}$ . One approach is to constrain the elements of  $\bar{\mathbf{w}}$  to be 0 or a power of 2 so that multiplication operations simply become efficient bit-shift operations (Hubara et al., 2016; Li et al., 2016b; Rastegari et al., 2016). An even more efficient approach is to employ basis functions with discrete outputs so that  $\Phi_*$  can also be represented as a low-precision quantized integer array. For example, a rounding operation could be applied to continuous basis functions. Provided that the quantization schemes are an affine mapping of integers to real numbers (i.e., the quantized values are evenly spaced), then inference can be conducted using efficient integer arithmetic (Jacob et al., 2017). Either of these approaches enable efficient on-device inference.

### 7.3.2 Deep Neural Networks for Regression

We consider the hierarchical model structure of a Bayesian deep neural network for regression. Considering a DIRECT approach for this architecture is not conceptually challenging so long as an appropriate neuron activation function is selected. We would like a non-linear activation that maintains a compact representation of the log-likelihood evaluated at every point in the hypothesis space, i.e., we would like  $\log \ell$  to be represented as a sum of as few Kronecker product vectors as possible. Using a power function for the activation can maintain a compact representation; the natural choice being a quadratic activation function (i.e., output  $x^2$  for input  $x$ ).

It can be shown that the ELBO can be exactly computed in  $\mathcal{O}(\ell \bar{m} (b/\ell)^{4\ell})$  for a deep Bayesian neural network with  $\ell$  layers, where we assume a quadratic activation function and an equal distribution of discrete latent variables between network layers. This complexity evidently enables scalable Bayesian inference for models of moderate depth, and like we found for the regression GLM model of section 7.3.1, computational complexity is *independent* of the quantity of training data, making this approach ideal for large datasets. We outline this model and the computation of its ELBO in appendix E.

## 7.4 Limitations & Extensions

In general, when the support of the prior is on a Cartesian grid, any prior, likelihood, or variational distribution (or log-distribution) can be expressed using the proposed Kronecker matrix representation; however, this representation will not always be compact enough to be practical. We can see this by viewing these probability distributions over the hypothesis space as high-dimensional tensors. In section 7.3, we exploited some popular models whose variational probability tensors and whose prior, likelihood, and variational log-probability tensors all admit a low-rank structure; however, other models may not admit this structure, in which case their representation will not be so compact. In the interest of generalizing the technique, we outline a likelihood, a prior, and a variational distribution that does not admit a compact representation of the ELBO and discuss several ways the DIRECT method can still be used to efficiently compute, or lower bound the ELBO. We hope that these extensions inspire future research directions in approximate Bayesian inference.

### 7.4.1 Generalized Linear Logistic Regression

Logistic regression models do not easily admit a compact representation for exact ELBO computations; however, we will demonstrate that we can efficiently compute a lower-bound of the ELBO by leveraging developed algebraic techniques. To demonstrate, we will consider a generalized linear logistic regression model which is commonly employed for classification problems. Such a model could easily be extended to a deep architecture following Bradshaw et al. (2017), if desired. All terms in the ELBO in eq. (7.7) can be computed exactly for this model except the term involving the log-likelihood, for which the following result demonstrates an efficient computation of the lower bound.

**Theorem 7.2.** *For a generalized linear logistic regression model with classification training labels  $\mathbf{y} \in \{0,1\}^n$ , the class-conditional probability  $Pr(y_i = 0 | \mathbf{w}) = (1 + \exp(-\Phi[i, :] \mathbf{w}))^{-1}$ , and with the assumption that training examples are sampled independently, the following inequality holds*

$$\mathbf{q}^T \log \ell \geq -\mathbf{s}^T (\Phi^T \mathbf{y}) - \sum_{i=1}^n \begin{cases} \prod_{j=1}^b \mathbf{q}_j^T \exp(-\phi_{ij} \bar{\mathbf{w}}_j) & \text{if } y_i = 0 \\ \prod_{j=1}^b \mathbf{q}_j^T \exp(\phi_{ij} \bar{\mathbf{w}}_j) - \sum_{j=1}^b \mathbf{q}_i^T \phi_{ij} \bar{\mathbf{w}}_j & \text{if } y_i = 1 \end{cases} \quad (7.18)$$

*Proof.* For the generalized linear logistic regression model considered, we can write the log likelihood as follows (see e.g., Bishop (2006)):

$$\log \ell = \sum_{i=1}^n -y_i \mathbf{z}_i - \log(1 + \exp(-\mathbf{z}_i)), \quad (7.19)$$

where  $\mathbf{Z} = (\Phi \mathbf{W})^T = \{\oplus_{j=1}^b \phi_{ij} \bar{\mathbf{w}}_j\}_{i=1}^n \in \mathbb{R}^{m \times n}$  is a matrix whose columns contain the logit values for a single training point at every possible set of latent variables in the hypothesis space. It is evident that the first term is identical to that discussed in eq. (7.12); however,

computation of the second term requires more development. We can write

$$\mathbf{q}^T \log \ell = -\mathbf{s}^T (\Phi^T \mathbf{y}) - \sum_{i=1}^n \mathbf{q}^T \log(1 + \exp(-\mathbf{z}_i)). \quad (7.20)$$

Since  $\mathbf{z}_i = \bigoplus_{j=1}^b \phi_{ij} \bar{\mathbf{w}}_j \in \mathbb{R}^m$  is a sum of  $b$  Kronecker product vectors, each with one unique sub-matrix that is not unity,  $\exp(-\mathbf{z}_i)$  is a single Kronecker product vector. This follows from Proposition 7.2. We can then take a Taylor series explanation of  $\log(1 + \exp(-\mathbf{z}_i))$  as follows:

$$\log(1 + \exp(-\mathbf{z}_i)) = - \sum_{k=1}^{\infty} \frac{(-1)^k \exp(-k\mathbf{z}_i)}{k} \quad \text{for } |\exp(-\mathbf{z}_i)| < 1 \rightarrow \mathbf{z}_i > 0, \quad (7.21)$$

$$\log(1 + \exp(-\mathbf{z}_i)) = -\mathbf{z}_i - \sum_{k=1}^{\infty} \frac{(-1)^k \exp(k\mathbf{z}_i)}{k} \quad \text{for } |\exp(-\mathbf{z}_i)| > 1 \rightarrow \mathbf{z}_i < 0, \quad (7.22)$$

and although the use of either choice would result in an ELBO lower bound, we choose the approximation based on the training label as follows; if  $y_i = 0$  or  $1$  then we would choose the  $(\mathbf{z}_i > 0)$  or  $(\mathbf{z}_i < 0)$  approximation, respectively. We choose this because  $z_i > 0$  gives a higher class conditional probability to class 0 than class 1 so this approximation would yield a tight lower bound when the training examples are correctly classified. These approximations are plotted in fig. 7.1 with a first-order expansion where it is evident that the computation lower-bounds the exact computation. Using this first-order Taylor series approximation, we can write our lower bound for the inner product between the variational distribution and the log-likelihood as follows which completes the proof,

$$\mathbf{q}^T \log \ell \geq -\mathbf{s}^T (\Phi^T \mathbf{y}) - \sum_{i=1}^n \begin{cases} \mathbf{q}^T \exp(-\mathbf{z}_i) & \text{if } y_i = 0 \\ \mathbf{q}^T \exp(\mathbf{z}_i) - \mathbf{q}^T \mathbf{z}_i & \text{if } y_i = 1 \end{cases}, \quad (7.23)$$

$$= -\mathbf{s}^T (\Phi^T \mathbf{y}) - \sum_{i=1}^n \begin{cases} \prod_{j=1}^b \mathbf{q}_j^T \exp(-\phi_{ij} \bar{\mathbf{w}}_j) & \text{if } y_i = 0 \\ \prod_{j=1}^b \mathbf{q}_j^T \exp(\phi_{ij} \bar{\mathbf{w}}_j) - \sum_{j=1}^b \mathbf{q}_i^T \phi_{ij} \bar{\mathbf{w}}_j & \text{if } y_i = 1 \end{cases}. \quad (7.24)$$

□

The computations in the lower bound of theorem 7.2 can be performed in  $\mathcal{O}(\bar{m}bn)$  time, where dependence on  $n$  is evident unlike in the case of the exact computations described in section 7.3. As a result, stochastic optimization techniques should be considered.

Using the lower bound in theorem 7.2, the log-likelihood is accurately approximated for hypotheses that correctly classify the training data; however, hypotheses that confidently misclassify training labels may be over-penalized. This is because we expect the Taylor series approximations (used in the proof) to admit a tight bound only within and just outside of their logit domains (i.e., where a label is correctly classified) as we can see in fig. 7.1. Far outside of the approximation domain (i.e., where a label is incorrectly classified) it can be observed in fig. 7.1 how the approximation may significantly underestimate the exact solution. As a result, this lower bound has the effect of erroneously decreasing the likelihood of hypotheses

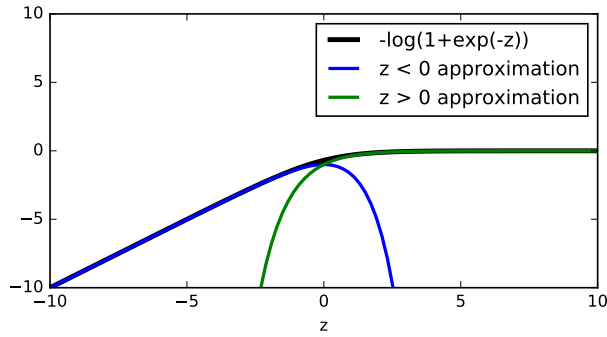


Figure 7.1: First-order Taylor series approximation of  $-\log(1 + \exp(-z))$ . The approximations evidently lower-bound the exact computation.

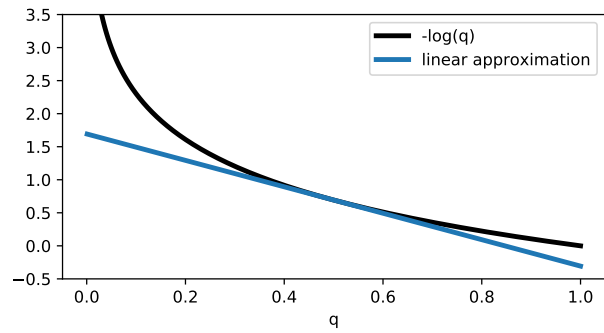


Figure 7.2: Taylor series approximation of  $-\log(q)$  about  $a = 0.5$ . The approximations evidently lower-bound the exact computation.

that misclassify examples, thus resulting in less posterior probability mass on these hypothesis verses exact inference.

As a final remark, the products over  $b$  terms in theorem 7.2 may result in overflow or loss of precision; however, computations can be performed in a stable manner in logit space, and the LogSumExp trick (Nielsen and Sun, 2016) can be used to avoid precision loss in the summations.

#### 7.4.2 Unfactorized Variational Distributions

We now consider going beyond a mean-field variational distribution to account for correlations between latent variables. Considering a finite mixture of factorized categorical distributions as is used in latent structure analysis (Goodman, 1974; Lazarsfeld and Henry, 1968), we can write  $\mathbf{q} = \sum_{i=1}^r \alpha_i \otimes_{j=1}^b \mathbf{q}_j^{(i)}$ , where  $\boldsymbol{\alpha} \in (0, 1)^r$  is a vector of mixture probabilities for  $r$  components, and  $\mathbf{q}_j^{(i)} = \{\Pr(w_j = \bar{w}_{jk} | i)\}_{k=1}^{\bar{m}} \in (0, 1)^{\bar{m}}$ .

While  $\mathbf{q}$  can evidently be expressed as a compact sum of Kronecker product vectors,  $\log \mathbf{q}$  is more challenging to compute than in the mean-field case; however, the following result demonstrates how we can lower-bound the term involving  $\log \mathbf{q}$  in the ELBO (eq. (7.7)).

**Theorem 7.3.** *The following inequality holds when we consider a finite mixture of factorized categorical distributions for  $q_{\boldsymbol{\theta}}(\mathbf{w})$ ,*

$$-\mathbf{q}^T \log \mathbf{q} \geq \max_{\{\mathbf{a}_i \in (0, 1)^{\bar{m}}\}_{i=1}^b} 1 - \sum_{j=1}^r \alpha_j \left( \sum_{i=1}^b \mathbf{q}_i^{(j)T} \log \mathbf{a}_i + \alpha_j \prod_{i=1}^b \mathbf{q}_i^{(j)T} \frac{\mathbf{q}_i^{(j)}}{\mathbf{a}_i} + 2 \sum_{k=j+1}^r \alpha_k \prod_{i=1}^b \mathbf{q}_i^{(j)T} \frac{\mathbf{q}_i^{(k)}}{\mathbf{a}_i} \right),$$

where  $\mathbf{a} = \otimes_{i=1}^b \mathbf{a}_i$ ,  $\mathbf{a}_i \in (0, 1)^{\bar{m}}$  is the center of the Taylor series approximation of  $\log \mathbf{q}$ .

*Proof.* We begin by taking a Taylor series approximation of  $\log \mathbf{q}$  about  $\mathbf{a} = \otimes_{i=1}^b \mathbf{a}_i$ ,  $\mathbf{a}_i \in (0, 1)^{\bar{m}}$  as follows:

$$\log \mathbf{q} = \log \mathbf{a} + \sum_{k=1}^{\infty} \frac{(-1)^{(k+1)}}{k \mathbf{a}^k} (\mathbf{q} - \mathbf{a})^k, \quad (7.25)$$



which can be represented as a sum of Kronecker product vectors once the exponents are computed explicitly. However, the number of terms in this sum will grow quickly with respect to the order of the Taylor series approximation. When a first order Taylor series expansion is considered, the approximation will give a strict lower bound of  $-\log \mathbf{q}$  and consequently a lower bound of the ELBO (eq. (7.7)) will be achieved. The approximation for a linear Taylor series expansion is plotted in fig. 7.2 where it is apparent that the approximation lower bounds the exact computation. We consider this linear approximation for the result in theorem 7.3. Note that the exact computation will always be lower bounded irrespective of the location that the Taylor series is taken about, therefore, we may select the values of  $\{\mathbf{a}_i \in (0,1)^{\bar{m}}\}_{i=1}^b$  that maximize this lower bound, as written in the theorem statement. We can then write our approximation of the third term from the ELBO (eq. (7.7)) to complete the proof as follows:

$$-\mathbf{q}^T \log \mathbf{q} \geq 1 - \sum_{j=1}^r \alpha_j \left( \sum_{i=1}^b \mathbf{q}_i^{(j)T} \log \mathbf{a}_i + \alpha_j \prod_{i=1}^b \mathbf{q}_i^{(j)T} \frac{\mathbf{q}_i^{(j)}}{\mathbf{a}_i} + 2 \sum_{k=j+1}^r \alpha_k \prod_{i=1}^b \mathbf{q}_i^{(j)T} \frac{\mathbf{q}_i^{(k)}}{\mathbf{a}_i} \right). \quad (7.26)$$

□

Note that if the mixture variational distribution  $\mathbf{q}$  degenerates to a mean-field distribution equal to  $\mathbf{a}$ , then the equality in the theorem holds such that the ELBO will be computed exactly. Conversely, as  $\mathbf{q}$  moves away from  $\mathbf{a}$ , the ELBO will be underestimated.

In theorem 7.3, the products over  $b$  terms might seem problematic; however, we do not expect the final results to be too large to pose an overflow concern. To avoid precision loss, we compute the log of the products, which can be done stably, and then exponentiate.

### 7.4.3 Unfactorized Prior Distributions

To consider an unfactorized prior, we assume a prior mixture distribution given by  $\mathbf{p} = \sum_{i=1}^r \alpha_i \otimes_{j=1}^b \mathbf{p}_j^{(i)}$ . When we use this mixture distribution for the prior,  $\mathbf{p}$  can evidently be expressed as a compact sum of Kronecker product vectors but  $\log \mathbf{p}$  cannot. The following result demonstrates how we can still lower-bound the term involving  $\log \mathbf{p}$  in the ELBO (eq. (7.2)). For simplicity, we assume that the variational distribution factorizes; however, the result could easily be extended to the case of a mixture variational distribution.

**Theorem 7.4.** *The following inequality holds when we consider a finite mixture of factorized categorical distributions for  $p_{\theta}(\mathbf{w})$ ,*

$$\mathbf{q}^T \log \mathbf{p} \geq \sum_{i=1}^r \alpha_i \sum_{j=1}^b \mathbf{q}_j^T \log \mathbf{p}_j^{(i)}$$

This result is easily verified by application of Jensen's inequality. Note that the equality only holds when the prior mixture degenerates to a factorized distribution with all mixture components equivalent.

### 7.4.4 Unbiased Entropy and Prior Expectation Gradients

We previously showed how to lower bound the ELBO terms  $\mathbf{q}^T \log \mathbf{p}$  and  $-\mathbf{q}^T \log \mathbf{q}$  when the variational and/or prior distributions do not factor; however, optimizing this bound introduces bias and does not guarantee convergence to a local optimum of the true ELBO. Here we reintroduce REINFORCE to deliver unbiased gradient estimates for these terms. The REINFORCE estimator typically has high variance; however, since gradient estimates for these terms are so cheap, a massive number of samples can be used per stochastic gradient descent (SGD) iteration to decrease variance. Since we can still compute the expensive  $\mathbf{q}^T \log \ell$  term *exactly* when  $\mathbf{q}$  is an unfactorized mixture distribution, its gradient can be computed exactly. The unbiased gradient estimator of  $\mathbf{q}^T \log \mathbf{q}$  is expressed as follows<sup>2</sup>:

$$\frac{\partial}{\partial \theta} \mathbf{q}^T \log \mathbf{q} = \frac{1}{2} \mathbf{q}^T \left( \frac{\partial}{\partial \theta} (\log \mathbf{q} + 1)^2 \right) \approx \frac{\partial}{\partial \theta} \frac{1}{2t} \sum_{i=1}^t (\log q(\mathbf{s}_i) + 1)^2, \quad (7.27)$$

where  $\mathbf{s}_i \in \mathbb{R}^b$  is the  $i$ th of  $t$  samples from the variational distribution used in the Monte Carlo gradient estimator. It is evident that this surrogate loss can be easily optimized using automatic differentiation, and the per-sample computations are extremely cheap.

## 7.5 Numerical Studies

### 7.5.1 Comparison with REINFORCE

As discussed in section 7.2, we cannot reparameterize because of the discrete latent variable priors considered; however, we can directly compare the optimization performance of the proposed techniques with the REINFORCE gradient estimator (Williams, 1992). In fig. 7.3, we compare ELBO maximization performance between the proposed DIRECT, and the REINFORCE methods. For this study we generated a dataset from a random weighting of  $b = 20$  random Fourier features (section 5.3.1) of an exponentiated quadratic kernel (eq. (2.28)) and corrupted by independent Gaussian noise. We use a generalized linear regression model as described in section 7.3.1 which uses the same features with  $\bar{m} = 3$ . We consider a prior over  $\sigma^2$ , and a mean-field variational distribution giving  $\bar{m}(b + 1) = 63$  variational parameters which we initialize to be the same as the prior; a uniform categorical distribution. For DIRECT, an L-BFGS optimizer is used (Byrd et al., 1995) and stochastic gradient descent is used for REINFORCE with a varying number of samples used for the Monte Carlo gradient estimator. Both methods use full batch training and are implemented using TensorFlow (Abadi et al., 2016). It can be seen that DIRECT greatly outperforms REINFORCE both in the number of iterations and computational time. As we move to a large  $n$  or a larger  $b$ , the difference between the proposed DIRECT technique and REINFORCE becomes more profound. The superior scaling with respect to  $n$  was expected since we had shown in section 7.3.1 that the DIRECT computational

<sup>2</sup>We used the identity  $(\log \mathbf{q} + 1) \odot \frac{\partial \log \mathbf{q}}{\partial \theta} = \frac{1}{2} \frac{\partial}{\partial \theta} (\log \mathbf{q} + 1)^2$ , where  $\odot$  denotes an elementwise product.

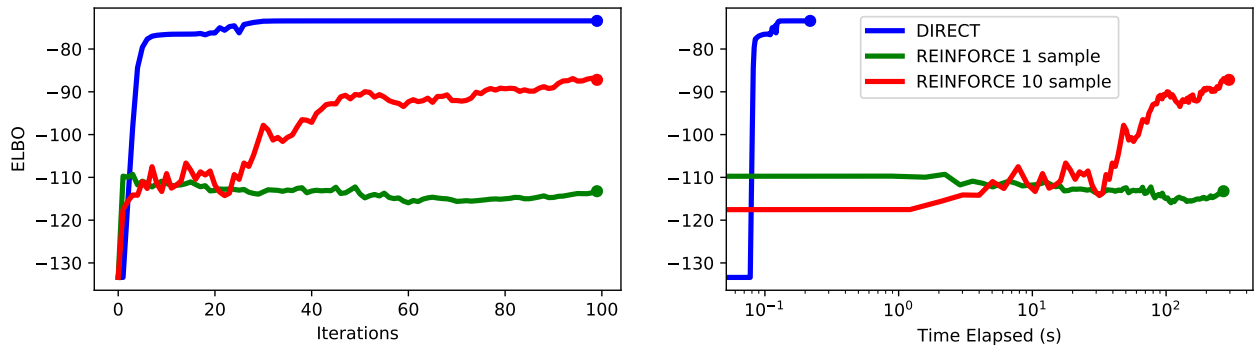


Figure 7.3: Convergence rates of a GLM trained with REINFORCE versus the proposed DIRECT method. The DIRECT method greatly outperforms REINFORCE in iterations and wall-clock time.

runtime is independent of  $n$ . However, the improved scaling with respect to  $b$  is an interesting result and may be attributed to the fact that as the dimension of the variational parameter space increases, there is more value in having low (or zero) variance estimates of the gradient.

## 7.5.2 One-Dimensional Regression Visualization

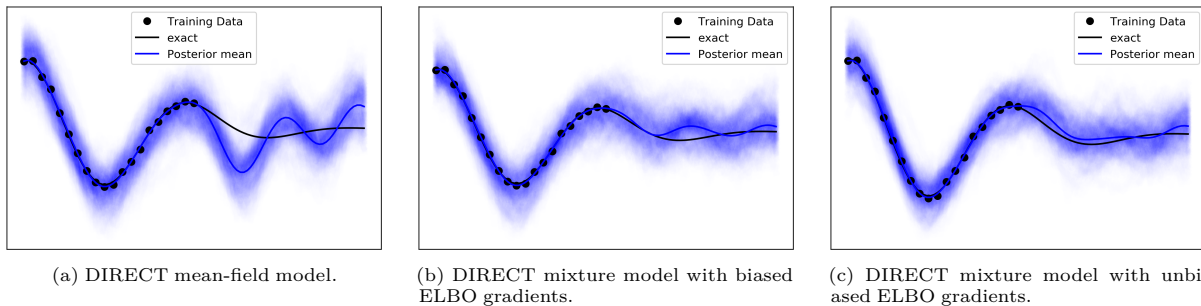


Figure 7.4: Comparison of predictive posterior distributions found with various variational distributions and variational objectives.

In this study we will consider a one-dimensional DIRECT generalized linear model with  $b = 100$  basis functions and each latent variables will be able to take  $\bar{m} = 15$  values so that posterior samples can be expressed as a vector of low-precision 4-bit quantized integers. In this case, computing the exact posterior is intractable, since it would require evaluating the model likelihood at  $\bar{m}^b = 15^{100} \approx 10^{117}$  possible permutations of latent variables. However, using the DIRECT method we can train this model extremely quickly.

The black dots in fig. 7.4 show a dataset of  $n = 20$  points generated from an exact underlying function which shown as a black line and given by the relation  $\cos(2\pi x)\exp(-x^2)$ . The training observations are also corrupted with independent Gaussian noise. The basis functions we will use are random Fourier features (section 5.3.1) of a squared exponential kernel (eq. (2.28)). The latent variables have support at  $\bar{m} = 15$  equally spaced points between  $-1$  and  $1$  inclusive (i.e.,  $w \in \text{linspace}(-1, 1, \bar{m} = 15)$ ) and we also specify  $\bar{m} = 15$  discrete values of independent Gaussian noise variance  $\sigma^2$  with support at equally spaced points on a log scale between  $10^{-5}$  and  $10^{-2}$  inclusive (i.e.,  $\text{logspace}(-5, -2, \bar{m} = 15)$ ). We place an

independent uniform prior over both the latent variable and noise variance support.

In fig. 7.4a, we show that posterior mean (blue line) and samples from the posterior (shaded blue region) where a factorizing variational distribution is considered. For this mean-field model, we compute the ELBO as described in section 7.3.1 and optimize the variational parameters using an L-BFGS optimizer. Looking at the plot of the predictive posterior, it is evident that the model fits the data well; however, the posterior distribution behaves poorly away from the data.

In fig. 7.4b, we consider a mixture variational distribution that does not factorize. We use a mixture distribution with  $r = 5$  components and maximize the ELBO lower bound derived in theorem 7.3. Since the ELBO gradients are still deterministic in this case, we again use an L-BFGS optimizer for training; however, since ELBO gradient estimates are biased, we are not guaranteed to converge to a local optima of the ELBO. In the plot, it is evident that the predictive posterior is more reasonable than that of the mean-field model.

In fig. 7.4c, we again consider a mixture variational distribution with  $r = 5$  components but this time use the unbiased gradient estimator described in section 7.4.4 with  $t = 100$  Monte Carlo samples for the entropy gradient estimator. Since the gradient estimates are now stochastic, we optimize the variational parameters with stochastic gradient descent. Observing the figure, it is clear that the predictive posterior extrapolates the data better than the mixture model trained by maximizing the ELBO lower bound. It would be expected that optimizing an unbiased estimate of the ELBO would perform better; however, this comes at the cost of introducing stochasticity into our optimization objective which ultimately requires more optimization iterations. While the unbiased method takes slightly longer to train due to its stochastic gradients, it is a more robust approach than using biased gradients. Still, a mixture model with biased gradients often outperforms the (unbiased) mean-field model as a result of a more flexible variational distribution.

### 7.5.3 Relaxing Gaussian Priors on UCI Regression Datasets

In this section, we consider discretely relaxing a continuous Gaussian prior on the weights of a generalized linear regression model. This allows us to compare performance between a reparameterization gradient estimator for a continuous prior (REPARAM) and our DIRECT method for a relaxed, discrete prior.

Considering regression datasets from the UCI repository, we report the mean and standard deviation of the root mean squared error (RMSE) from 10-fold cross validation (90% train, 10% test per fold). Also presented is the mean training time per fold on a machine with two E5-2680 v3 processors and 128Gb of RAM, and the expected sparsity (percentage of zeros) within a posterior sample. Using a generalized linear model, we consider  $b = 2000$  random Fourier features (section 5.3.1) of a squared-exponential kernel (eq. (2.28)) with automatic relevance determination. Before generating the features, we initialize the kernel hyperparameters including the prior variance  $\sigma_w^2$  and the Gaussian noise variance  $\sigma^2$  by maximizing

the marginal likelihood of an exact Gaussian process constructed on  $\min(n, 1000)$  points randomly selected from the dataset. All discretely relaxed models (containing “DIRECT”), only have support at  $\bar{m} = 15$  equally spaced points between  $-3\sigma_w$  and  $3\sigma_w$  inclusive (i.e.,  $w \in \text{linspace}(-3\sigma_w, 3\sigma_w, \bar{m} = 15)$ ), allowing  $\mathbf{w}$  to be stored as 4-bit quantized integers. We fix  $\sigma^2$  to be a single value since a prior over this value cannot be computed analytically for the models we will compare against (although section 7.3.1 discussed how we can easily place a prior distribution over  $\sigma^2$  for our discrete model in practice).

For REPARAM we perform doubly stochastic optimization using a mini-batch size of 100 and using 10 Monte Carlo samples for the gradient estimates at each iteration. For datasets with  $n < 3000$  we optimize for 1000 iterations and we optimize for 10000 iterations for all larger datasets. This model was implemented in Edward (Tran et al., 2016). For the DIRECT mean-field model we use an L-BFGS optimizer (Byrd et al., 1995) and run until convergence, or 1000 iterations are reached. For the DIRECT 5-mixture model we perform stochastic gradient descent using  $t = 3000$  Monte Carlo samples for the entropy gradient estimator in eq. (7.27).

For the DIRECT mean-field model we initialize the variational distribution to the prior. For the DIRECT mixture models, we first run the mean-field model and then initialize each mixture component to be randomly perturbed from the mean-field solution, and we initialize  $\mathbf{a}$  to the mean-field solution. We initialize the mixture probabilities to be constant.

For predictive posterior mean computations, we use the exact computation presented in eq. (7.17) for both the DIRECT and mixture models. For REPARAM, we approximate the mean by sampling the variational distribution using 1000 samples.

In table 7.1, we see the results of our studies across several model-types. In the left column, the “REPARAM Mean-Field” model uses a (continuous) Gaussian prior, an uncorrelated Gaussian variational distribution and reparameterization gradients. The right two models use a discrete relaxation of a Gaussian prior (DIRECT) with support at 15 discrete values, allowing storage of each latent variable sample as a vector of 4-bit quantized integers. Therefore, each ELBO evaluation requires  $15^{2000} > 10^{2352}$  log-likelihood evaluations; however, these computations can be done quickly by exploiting Kronecker matrix algebra. We compute the ELBO as described in section 7.3.1 for the “DIRECT Mean-Field” model, and use the unbiased gradient estimator described in section 7.4.4 for the “DIRECT 5-Mixture SGD” model which uses a mixture distribution with  $r = 5$  components, and  $t = 3000$  Monte Carlo samples for the entropy gradient estimator.

The boldface entries indicate top performance on each dataset, where it is evident that the DIRECT method not only outperformed REPARAM on most datasets but also trained much faster, particularly on the large datasets due to the independence of dataset size on computational complexity. The DIRECT mean-field model contains  $\bar{m}b = 30,000$  variational parameters; however, training took just seconds on all datasets, including *electric* with over 2 million points. The DIRECT mixture model contains  $\bar{m}br = 150,000$  variational parameters, and since the gradient estimates are stochastic, average training times are on the order of

Dataset	$n$	$d$	Continuous Prior			Discrete 4-bit Prior				
			REPARAM	Mean-Field	Sparsity	DIRECT	Mean-Field	DIRECT 5-Mixture SGD		
			Time	RMSE		Time	RMSE	Sparsity	RMSE	Sparsity
challenger	23	4	8	<b>0.515 ± 0.284</b>	0%	1	0.523 ± 0.248	17%	0.525 ± 0.246	17%
fertility	100	9	8	0.161 ± 0.043	0%	2	<b>0.159 ± 0.041</b>	17%	0.16 ± 0.041	17%
automobile	159	25	5	0.425 ± 0.2	0%	10	0.129 ± 0.063	51%	<b>0.122 ± 0.056</b>	51%
servo	167	4	5	0.524 ± 0.184	0%	10	<b>0.271 ± 0.08</b>	35%	0.274 ± 0.077	35%
cancer	194	33	5	27.488 ± 5.45	0%	4	22.954 ± 3.09	19%	<b>22.937 ± 3.135</b>	19%
hardware	209	7	5	1.796 ± 1.537	0%	11	<b>0.401 ± 0.048</b>	51%	<b>0.401 ± 0.046</b>	51%
yacht	308	6	5	0.815 ± 0.17	0%	1	0.234 ± 0.07	96%	<b>0.225 ± 0.082</b>	96%
autompkg	392	7	5	4.05 ± 0.739	0%	10	2.564 ± 0.363	31%	<b>2.543 ± 0.362</b>	31%
housing	506	13	5	3.014 ± 0.567	0%	10	<b>2.752 ± 0.405</b>	40%	<b>2.699 ± 0.361</b>	39%
forest	517	12	5	1.378 ± 0.148	0%	2	1.363 ± 0.15	17%	<b>1.357 ± 0.155</b>	17%
stock	536	11	5	0.751 ± 0.338	0%	8	0.011 ± 0.003	98%	<b>0.008 ± 0.001</b>	98%
pendulum	630	9	5	1.465 ± 0.26	0%	1	1.329 ± 0.282	68%	<b>1.312 ± 0.253</b>	63%
energy	768	8	5	78.852 ± 21.73	0%	1	3.272 ± 0.332	99%	<b>2.911 ± 0.309</b>	99%
concrete	1030	8	5	10.347 ± 2.847	0%	10	<b>5.316 ± 0.716</b>	82%	5.477 ± 0.632	82%
solar	1066	10	5	0.902 ± 0.171	0%	10	<b>0.787 ± 0.192</b>	23%	0.788 ± 0.189	23%
airfoil	1503	5	5	<b>2.071 ± 0.271</b>	0%	11	2.175 ± 0.349	48%	2.156 ± 0.316	45%
wine	1599	11	5	0.939 ± 0.33	0%	11	0.472 ± 0.044	54%	<b>0.469 ± 0.042</b>	54%
gas	2565	128	5	0.27 ± 0.052	0%	1	0.211 ± 0.058	84%	<b>0.184 ± 0.063</b>	76%
skillcraft	3338	19	46	0.273 ± 0.029	0%	7	<b>0.253 ± 0.016</b>	97%	<b>0.253 ± 0.016</b>	97%
sml	4137	26	47	<b>0.327 ± 0.013</b>	0%	1	0.677 ± 0.044	57%	0.671 ± 0.047	57%
parkinsons	5875	20	48	<b>0.158 ± 0.009</b>	0%	1	0.651 ± 0.034	13%	0.613 ± 0.083	13%
poletele	15000	26	50	<b>12.487 ± 0.363</b>	0%	10	13.65 ± 0.348	16%	13.369 ± 0.431	17%
elevators	16599	18	51	0.247 ± 0.156	0%	1	<b>0.124 ± 0.003</b>	99%	<b>0.124 ± 0.003</b>	99%
protein	45730	9	58	0.642 ± 0.006	0%	11	0.619 ± 0.007	76%	<b>0.618 ± 0.007</b>	60%
kegg	48827	20	58	<b>0.178 ± 0.012</b>	0%	1	0.222 ± 0.009	96%	0.205 ± 0.004	95%
ctslice	53500	385	61	<b>4.415 ± 0.113</b>	0%	2	6.063 ± 0.122	19%	5.478 ± 0.137	42%
keggu	63608	27	61	<b>0.122 ± 0.004</b>	0%	1	0.139 ± 0.004	87%	0.136 ± 0.006	87%
3droad	434874	3	141	11.057 ± 0.091	0%	2	10.493 ± 0.105	40%	<b>10.354 ± 0.077</b>	33%
song	515345	90	158	0.537 ± 0.002	0%	2	0.501 ± 0.002	32%	<b>0.498 ± 0.002</b>	28%
buzz	583250	77	169	<b>0.94 ± 0.006</b>	0%	1	1.007 ± 0.007	82%	0.959 ± 0.004	80%
electric	2049280	11	500	9.26 ± 4.47	0%	1	0.575 ± 0.032	99.6%	<b>0.557 ± 0.055</b>	99.6%

Table 7.1: Mean and standard deviation of test error, average training time (in seconds), and average expected sparsity of a posterior sample from 10-fold cross validation on UCI regression datasets.

hundreds of seconds across all datasets. While the time for precomputations does depend on dataset size, its contribution to the overall timings are negligible, being well under one second for the largest dataset, *electric*. Additionally, it is evident that posterior samples from the DIRECT model tend to be very sparse. For example, the DIRECT models on the *gas* dataset admit posterior samples that are over 84% sparse on average, meaning that over 1680 weights are expected to be zero in a posterior sample with  $b = 2000$  elements. This would yield massive computational savings on hardware limited devices. Samples from the DIRECT models on the *electric* dataset are over 99.6% sparse.

Comparing the DIRECT mean-field model to the mixture model, we observe gains in the RMSE performance on many datasets, as we would expect with the increased flexibility of the variational distribution. While we only showed the posterior mean in our results, we would expect an even larger disparity in the quality of the predictive uncertainty which was not analyzed.

In table 7.2 we consider again an unfactorized mixture variational distribution; however, we maximize the ELBO lower bound derived in theorem 7.3. Since the ELBO gradients are deterministic, we again use an L-BFGS optimizer for training. In addition to the 150,000 variational parameters used by the DIRECT 5-Mixture SGD model in table 7.1, computing the ELBO lower bound involves the simultaneous optimization of  $\mathbf{a}$ , adding 30,000 additional optimization parameters. Observing the values in table 7.2, this model does not perform as well as the DIRECT mixture model trained using an unbiased SGD approach, as would be expected; however, it does train faster since its objective is evaluated deterministically, and its RMSE

Dataset	$n$	$d$	Discrete 4-bit Prior		
			DIRECT Time	5-Mixture RMSE	ELBO-LB Sparsity
challenger	23	4	15	$0.528 \pm 0.243$	16%
fertility	100	9	15	$0.16 \pm 0.04$	16%
automobile	159	25	24	$0.137 \pm 0.053$	47%
servo	167	4	24	$0.282 \pm 0.067$	32%
cancer	194	33	17	$23.344 \pm 3.414$	18%
hardware	209	7	24	$0.492 \pm 0.117$	46%
yacht	308	6	5	$0.23 \pm 0.077$	96%
autopmg	392	7	24	$2.624 \pm 0.339$	29%
housing	506	13	24	$2.782 \pm 0.324$	37%
forest	517	12	15	$1.361 \pm 0.159$	16%
stock	536	11	233	$0.011 \pm 0.002$	98%
pendulum	630	9	6	$1.36 \pm 0.227$	68%
energy	768	8	5	$3.116 \pm 0.218$	99%
concrete	1030	8	19	$5.571 \pm 0.665$	81%
solar	1066	10	24	$0.799 \pm 0.192$	22%
airfoil	1503	5	16	$2.175 \pm 0.32$	46%
wine	1599	11	24	$0.486 \pm 0.047$	50%
gas	2565	128	5	$0.204 \pm 0.053$	84%
skillcraft	3338	19	78	$0.253 \pm 0.017$	97%
sml	4137	26	7	$0.675 \pm 0.044$	57%
parkinsons	5875	20	8	$0.642 \pm 0.06$	13%
poletele	15000	26	24	$13.728 \pm 0.447$	16%
elevators	16599	18	5	$0.124 \pm 0.003$	99%
protein	45730	9	16	$0.62 \pm 0.007$	76%
kegg	48827	20	6	$0.222 \pm 0.01$	95%
ctslice	53500	385	6	$6.036 \pm 0.163$	19%
keggu	63608	27	6	$0.139 \pm 0.004$	87%
3droad	434874	3	14	$10.487 \pm 0.075$	40%
song	515345	90	8	$0.502 \pm 0.002$	31%
buzz	583250	77	9	$1.009 \pm 0.004$	82%
electric	2049280	11	5	$0.593 \pm 0.036$	99.6%

Table 7.2: Using a mixture variational distribution along with the ELBO lower bound presented in theorem 7.3, we present the mean and standard deviation of test error, average training time (in seconds), and average expected sparsity of a posterior sample from 10-fold cross validation on UCI regression datasets.

performance is still marginally better than the DIRECT mean-field model on many datasets.

## 7.6 Conclusions

We showed that by discretely relaxing continuous priors, variational inference can be performed accurately and efficiently using our DIRECT method. Using Kronecker matrix algebra, the ELBO of a discretely relaxed model was computed exactly even when this computation required significantly more log-likelihood evaluations than the number of atoms in the known universe. Through this ability to exactly perform ELBO computations we achieve unbiased, zero-variance gradient estimates using automatic differentiation which we show significantly outperforms competing Monte Carlo alternatives that admit high-variance gradient estimates. We also demonstrate that the computational complexity of ELBO computations is *independent* of the quantity of training data using the DIRECT method, making the proposed approaches amenable to big data applications. At inference time, we show that we can again use Kronecker matrix algebra to exactly compute the statistical moments of the parameterized predictive posterior distribution, unlike competing techniques which rely on Monte Carlo sampling. Finally, we discuss and demonstrate how posterior samples can be sparse and can be represented as quantized integer values to enable efficient inference, which is particularly powerful on hardware limited devices, or if energy efficiency is a major concern.

We illustrate the DIRECT approach on several popular models such as mean-field varia-

tional inference for generalized linear models and deep Bayesian neural networks for regression. We also discuss some models which do not admit a compact representation for exact ELBO computations. For these cases, we discuss and demonstrate novel extensions to the DIRECT method that allow efficient computation of a lower bound of the ELBO, and we demonstrate how an unfactorized variational distribution can be used by introducing a manageable level of stochasticity into the gradients. We hope that these new approaches for ELBO computations will inspire new model structures and research directions in approximate Bayesian inference.



## Chapter 8

# Concluding Remarks

Learning is the ability to generalize beyond training examples. In practice, many hypotheses might be consistent with a set of training observations, and each will generalize differently. Bayesian inference provides a principled approach to select some hypotheses over others based on prior knowledge. In this thesis, we focused on particular modelling choices used for Bayesian inference, namely Gaussian processes and discrete latent variable models. Both of these modelling choices are ideal for many modern machine learning tasks. For instance, Gaussian processes are non-parametric models whose capacity naturally adapts to the quantity of training data, GPs are highly interpretable, and they offer powerful opportunities to incorporate prior knowledge. Unfortunately, both Gaussian processes and discrete variable approaches struggle to scale to complex models and to large quantities of observed data. This thesis outlined numerous approaches to scale these methods for large scale Bayesian inference in modern machine learning applications. This chapter summarizes the contributions of the thesis and discusses possibilities for future research.

### 8.1 Overview of Main Results

First we showed that exact Gaussian process inference can be efficiently performed on problems where training examples are (at least partially) structured on a Cartesian product grid in input space by leveraging Kronecker matrix algebra (Chapter 3). This contribution allows Gaussian processes to be constructed on massive image, video, spatiotemporal, or multi-output datasets without the introduction of any approximation. We demonstrated exact GP modelling on a spatiotemporal climate modelling problem with 3.7 million observations, a PIV fluid-flow reconstruction problem with nearly 20 million observations, and a video reconstruction problem with 1 billion observations; all of which are far beyond the scale of traditional GP modelling techniques which are typically limited to tens of thousands of observations.

The use of Kronecker matrix algebra was then applied to more general, unstructured problems using an efficient and compact Nyström approximation (Chapter 4). Compared to existing works that employ a Nyström approximation for Gaussian processes, the complexity

of the introduced approach is independent of the quantity of inducing points, allowing a huge number to be used to obtain a globally accurate approximation. Specifically, we demonstrated the use of  $10^{33}$  inducing points whereas prior techniques are limited to the use of a few thousand inducing points.

To deal with uncertainty over Gaussian process kernel hyperparameters, we showed that a particular class of kernels allow hyperparameter marginalization using Monte Carlo methods whose per-iteration complexity is independent of the quantity of training data (Chapter 5). Specifically we demonstrated a dramatic reduction in the computational complexity of GP marginal likelihood computations from  $\mathcal{O}(nm^2 + m^3)$  to as low as  $\mathcal{O}(m)$ , where  $n$  is the number of training observations, and  $m$  is the number of kernel basis functions. We also showed that this class of kernels is highly general; for example, we showed that it has the potential to asymptotically recover any stationary kernel.

We also introduced a variational inference procedure for approximate Gaussian processes that admits a stochastic training procedure with a per-iteration complexity that is independent of both the number of training examples and the number of kernel basis functions (Chapter 6). Specifically, the evaluation cost of the variational objective is reduced to  $\mathcal{O}(1)$  compared to leading prior approaches that scale as  $\mathcal{O}(m^3)$ . This resulted in a method that supports accurate and high-capacity Gaussian process approximations for large datasets while being amenable to modern computational hardware such as GPUs. We demonstrated state-of-the-art empirical results with the use of  $m = 10$  million basis functions while prior approaches are limited to just a few thousand basis functions.

Finally, we considered a class of discrete variable models that can approximate a broad class of continuous priors, such as Gaussian process priors, and has numerous runtime advantages since posterior samples are sparse and low-precision quantized integers. While previous techniques to construct such models rely on high-variance gradient estimators that converge slowly, by exploiting Kronecker matrix algebra, we presented a variational inference procedure that admits deterministic gradients with a computational complexity that is independent of the size of the training dataset (Chapter 7). We demonstrated that this enables Bayesian discrete variable models to be rapidly constructed on large datasets.

## 8.2 Opportunities for Future Research

There are many open leads for future research in the areas presented in this thesis.

**Exploiting Kronecker matrix algebra.** One of the recurring themes throughout the thesis was the exploitation of Kronecker matrix algebra to accelerate computations. As discussed throughout the various chapters, Kronecker matrix algebra can potentially accelerate computations for algorithms beyond the presented contributions. For instance, in chapters 3 and 4, Kronecker matrix algebra could be exploited for more general kernels than the class of product correlation kernels, such as additive kernels. This would allow the

techniques to facilitate much more general priors. Additionally, in chapter 7 it was discussed that the variational bound computations (and indeed the marginal likelihood computations) for any discrete variable model can be written in a Kronecker product structure but this structure may not always be compact. The proposed model did admit a compact structure, but other modelling choices may admit compact computations as well.

**Physics-inspired priors.** Gaussian processes present powerful opportunities to incorporate high-level prior knowledge about a function. Modelling of physical phenomena is an instance where a practitioner may have strong prior information about a problem from first-principles, or physics. There are opportunities to combine the techniques developed in this thesis with powerful, physics-inspired priors to both improve predictive performance, and to enforce physical constraints that may be needed for downstream analysis. To give an example, the large-scale PIV reconstruction problem from section 3.5.4 considered a low-speed fluid velocity field that can be accurately assumed to be an incompressible fluid flow. Mass is conserved in an incompressible flow only if the flow velocity field is divergence-free; a linear operator constraint which can be imposed on a GP prior (Narcowich and Ward, 1994). To scale exact GP modelling using a physics-inspired prior of this sort cannot directly leverage the techniques outlined in chapter 3; however, it can be shown that by exploiting the structure of divergence-free kernels, an extension of the ignore-gaps approach of section 3.3.2 can be leveraged to permit exact GP inference for large-scale PIV problems, such as the problem considered with nearly 20 million observations. Use of physics-inspired priors such as this would be expected to further improve reconstruction performance, and it would guarantee that mass is conserved within the flow-field which is likely to be valuable for downstream analysis.

**Extension to non-Gaussian Likelihoods.** Much of the thesis develops techniques for regression; however, many of these techniques may be extended to other learning problems. For instance, the developed techniques may be extended to the case of supervised learning with non-Gaussian likelihoods, which was explored in only some of the chapters.

**Stochastic training with basis function subsampling.** In chapter 6, a stochastic training method was developed that only required working with a subset of basis functions at each SGD iteration. While this technique was applied to variational inference for approximate Gaussian process models, the same technique could be directly applied to other inference algorithms or large-scale linear algebra problems. For instance, the subsampling techniques could be applied to achieve  $\mathcal{O}(1)$  unbiased gradient estimates for any quadratic optimization problem. To our knowledge, no approach with these properties have been discussed in the literature.

### 8.3 Conclusion

Gaussian processes provide many advantages that modern machine learning methods can benefit from. Computational scaling has been an obstacle that has prevented the large-scale uptake of Gaussian processes in some sectors. I hope that this thesis provides additional

direction and inspiration towards the numerous open challenges in scalable Gaussian process inference, and Bayesian inference in general. In particular, I hope that the contributions made in the application of Kronecker matrix algebra and stochastic training methods will inspire new algorithms for scalable Bayesian inference.

# Bibliography

- Abadi, M. et al. (2016). “TensorFlow: A System for Large-Scale Machine Learning.” In: *proceedings of the USENIX Symposium on Operating Systems Design and Implementation*. Vol. 16, pp. 265–283.
- Álvarez, M. A., L. Rosasco, and N. D. Lawrence (2012). “Kernels for Vector-Valued Functions: A Review”. In: *Foundations and Trends in Machine Learning* 4.3, pp. 195–266.
- Arndt, C. M., M. Severin, C. Dem, M. Stöhr, A. M. Steinberg, and W. Meier (2015). “Experimental analysis of thermo-acoustic instabilities in a generic gas turbine combustor by phase-correlated PIV, chemiluminescence, and laser Raman scattering measurements”. In: *Experiments in Fluids* 56.4, p. 69.
- Arora, S., S. S. Du, W. Hu, Z. Li, R. R. Salakhutdinov, and R. Wang (2019). “On exact computation with an infinitely wide neural net”. In: *Advances in Neural Information Processing Systems*, pp. 8141–8150.
- Atkinson, K. E. (2008). *An Introduction to Numerical Analysis*. John Wiley & Sons.
- Baker, C. T. H. (1977). *The numerical treatment of integral equations*. Oxford: Clarendon press.
- Barber, D. and C. M. Bishop (1998). “Ensemble learning in Bayesian neural networks”. In: *Neural Networks and Machine Learning*, pp. 215–237.
- Belabbas, M.-A. and P. J. Wolfe (2009). “Spectral methods in machine learning: New strategies for very large datasets”. In: *proceedings of the National Academy of Sciences of the USA*. 106, pp. 369–374.
- Bell, G., T. Hey, and A. Szalay (2009). “Beyond the data deluge”. In: *Science* 323.5919, pp. 1297–1298.
- Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.
- Blei, D. M., A. Kucukelbir, and J. D. McAuliffe (2017). “Variational inference: A review for statisticians”. In: *Journal of the American Statistical Association* 112.518, pp. 859–877.
- Boxx, I., C. Slabaugh, P. Kutne, R. P. Lucht, and W. Meier (2015). “3kHz PIV/OH-PLIF measurements in a gas turbine combustor at elevated pressure”. In: *Proceedings of the Combustion Institute* 35.3, pp. 3793–3802.
- Bradshaw, J., A. G. d. G. Matthews, and Z. Ghahramani (2017). “Adversarial Examples, Uncertainty, and Transfer Testing Robustness in Gaussian Process Hybrid Deep Networks”. In: *arXiv preprint arXiv:1707.02476*.
- Buhmann, M. D. (2003). *Radial basis functions: theory and implementations*. Cambridge university press.
- Byrd, R. H., P. Lu, J. Nocedal, and C. Zhu (1995). “A limited memory algorithm for bound constrained optimization”. In: *SIAM Journal on Scientific Computing* 16.5, pp. 1190–1208.
- Caux-Brisebois, V., A. M. Steinberg, C. M. Arndt, and W. Meier (2014). “Thermo-acoustic velocity coupling in a swirl stabilized gas turbine model combustor”. In: *Combustion and Flame* 161.12, pp. 3166–3180.
- Challis, E. and D. Barber (2013). “Gaussian Kullback-Leibler Approximate Inference”. In: *The Journal of Machine Learning Research* 14.1, pp. 2239–2286.
- Chen, H., R. H. Chiang, and V. C. Storey (2012). “Business Intelligence and Analytics: From Big Data to Big Impact”. In: *Management Information Systems quarterly* 36.4, pp. 1165–1188.
- Chen, T. Q., J. Behrmann, D. K. Duvenaud, and J.-H. Jacobsen (2019). “Residual Flows for Invertible Generative Modeling”. In: *Advances in Neural Information Processing Systems*, pp. 9913–9923.

- Chen, W., J. Wilson, S. Tyree, K. Weinberger, and Y. Chen (2015). “Compressing neural networks with the hashing trick”. In: *International Conference on Machine Learning*, pp. 2285–2294.
- Ćosić, B., S. Terhaar, J. P. Moeck, and C. O. Paschereit (2015). “Response of a swirl-stabilized flame to simultaneous perturbations in equivalence ratio and velocity at high oscillation amplitudes”. In: *Combustion and Flame* 162.4, pp. 1046–1062.
- Cutajar, K., M. A. Osborne, J. P. Cunningham, and M. Filippone (2016). “Preconditioning kernel matrices”. In: *International Conference on Machine Learning*, pp. 2529–2538.
- Dong, K., D. Eriksson, H. Nickisch, D. Bindel, and A. G. Wilson (2017). “Scalable log determinants for Gaussian process kernel learning”. In: *Advances in Neural Information Processing Systems*, pp. 6327–6337.
- Drineas, P. and M. W. Mahoney (2005). “On the Nyström method for approximating a Gram matrix for improved kernel-based learning”. In: *Journal of Machine Learning Research* 6, pp. 2153–2175.
- Duchi, J., E. Hazan, and Y. Singer (2011). “Adaptive subgradient methods for online learning and stochastic optimization”. In: *Journal of Machine Learning Research* 12, pp. 2121–2159.
- Evans, T. W. and P. B. Nair (2018a). “Discretely Relaxing Continuous Variables for tractable Variational Inference”. In: *Advances in Neural Information Processing Systems*, pp. 10487–10497.
- (2018b). “Exploiting Structure for Fast Kernel Learning”. In: *SIAM International Conference on Data Mining*. SIAM, pp. 414–422.
- (2018c). “Scalable Gaussian Processes with Grid-Structured Eigenfunctions (GP-GRIEF)”. In: *International Conference on Machine Learning*, pp. 1416–1425.
- (2020). “Quadruply Stochastic Gaussian Processes”. In: *arXiv preprint arXiv:2006.03015*.
- Everson, R. and L. Sirovich (1995). “Karhunen–Loeve procedure for gappy data”. In: *Journal of the Optical Society of America. Series A* 12.8, pp. 1657–1664.
- Fu, M. C. (2006). “Gradient estimation”. In: *Handbooks in operations research and management science* 13, pp. 575–616.
- Fureby, C., F. F. Grinstein, G. Li, and E. J. Gutmark (2007). “An experimental and computational study of a multi-swirl gas turbine combustor”. In: *Proceedings of the Combustion Institute* 31.2, pp. 3107–3114.
- G. Matthews, A. G. de, J. Hron, M. Rowland, R. E. Turner, and Z. Ghahramani (2018). “Gaussian Process Behaviour in Wide Deep Neural Networks”. In: *International Conference on Learning Representations*.
- Ghahramani, Z. (2015). “Probabilistic machine learning and artificial intelligence”. In: *Nature* 521.7553, pp. 452–459.
- Gibbs, M. N. and D. J. MacKay (2000). “Variational Gaussian process classifiers”. In: *Institute of Electrical and Electronics Engineers (IEEE) Transactions on Neural Networks* 11.6, pp. 1458–1464.
- Girolami, M. and B. Calderhead (2011). “Riemann manifold Langevin and Hamiltonian Monte Carlo methods”. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 73.2, pp. 123–214.
- Gittens, A. and M. W. Mahoney (2013). “Revisiting the Nyström method for improved large-scale machine learning”. In: *International Conference on Machine Learning*.
- Glynn, P. W. (1990). “Likelihood ratio gradient estimation for stochastic systems”. In: *Communications of the Association for Computing Machinery (ACM)* 33.10, pp. 75–84.
- Golub, G. H., M. Heath, and G. Wahba (1979). “Generalized cross-validation as a method for choosing a good ridge parameter”. In: *Technometrics* 21.2, pp. 215–223.
- Gong, Y., L. Liu, M. Yang, and L. Bourdev (2014). “Compressing deep convolutional networks using vector quantization”. In: *arXiv preprint arXiv:1412.6115*.
- Goodman, L. A. (1974). “Exploratory latent structure analysis using both identifiable and unidentifiable models”. In: *Biometrika* 61.2, pp. 215–231.
- Gower, R. M., N. Loizou, X. Qian, A. Sailanbayev, E. Shulgin, and P. Richtárik (2019). “SGD: General analysis and improved rates”. In: *International Conference on Machine Learning*.

- GPy (since 2012). *GPy: A Gaussian process framework in python*. <http://github.com/SheffieldML/GPy>.
- Grathwohl, W., D. Choi, Y. Wu, G. Roeder, and D. Duvenaud (2017). “Backpropagation through the void: Optimizing control variates for black-box gradient estimation”. In: *International Conference on Learning Representations*.
- Gunes, H., S. Sirisup, and G. E. Karniadakis (2006). “Gappy data: To Krig or not to Krig?” In: *Journal of Computational Physics* 212.1, pp. 358–382.
- Han, S., H. Mao, and W. J. Dally (2015). “Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding”. In: *arXiv preprint arXiv:1510.00149*.
- Harville, D. A. (2006). *Matrix Algebra From a Statistician’s Perspective*. Springer Science & Business Media.
- Henderson, H. V., F. Pukelsheim, and S. R. Searle (1983). “On the history of the Kronecker product”. In: *Linear and Multilinear Algebra* 14.2, pp. 113–120.
- Hensman, J., N. Fusi, and N. D. Lawrence (2013). “Gaussian processes for big data”. In: *Uncertainty in Artificial Intelligence*.
- Hensman, J., A. G. d. G. Matthews, and Z. Ghahramani (2015). “Scalable Variational Gaussian Process Classification”. In: *Proceedings of AISTATS*.
- Hoffman, M. D., D. M. Blei, C. Wang, and J. Paisley (2013). “Stochastic variational inference”. In: *The Journal of Machine Learning Research* 14.1, pp. 1303–1347.
- Horn, R. A. and C. R. Johnson (1994). *Topics in Matrix analysis*. Cambridge university press, p. 208.
- Hubara, I., M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio (2016). “Binarized neural networks”. In: *Advances in neural information processing systems*, pp. 4107–4115.
- Hutchinson, M. F., D. W. McKenney, K. Lawrence, J. H. Pedlar, R. F. Hopkinson, E. Milewska, and P. Papadopol (2009). “Development and testing of Canada-wide interpolated spatial models of daily minimum–maximum temperature and precipitation for 1961–2003”. In: *Journal of Applied Meteorology and Climatology* 48.4, pp. 725–741.
- Jacob, B., S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko (2017). “Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference”. In: *arXiv preprint arXiv:1712.05877*.
- Jacot, A., F. Gabriel, and C. Hongler (2018). “Neural tangent kernel: Convergence and generalization in neural networks”. In: *Advances in neural information processing systems*, pp. 8571–8580.
- Jang, E., S. Gu, and B. Poole (2016). “Categorical reparameterization with Gumbel-softmax”. In: *arXiv preprint arXiv:1611.01144*.
- Jaynes, E. T. (2003). *Probability theory: The logic of science*. Cambridge university press.
- Jordan, M. I., Z. Ghahramani, T. S. Jaakkola, and L. K. Saul (1999). “An introduction to variational methods for graphical models”. In: *Machine learning* 37.2, pp. 183–233.
- Kim, G.-H., S. Trimi, and J.-H. Chung (2014). “Big-data Applications in the Government Sector”. In: *Communications of the Association for Computing Machinery (ACM)* 57.3, pp. 78–85.
- Kingma, D. P. and J. Ba (2015). “Adam: A method for stochastic optimization”. In: *International Conference on Learning Representations*.
- Kingma, D. P. and M. Welling (2013). “Auto-encoding variational Bayes”. In: *arXiv preprint arXiv:1312.6114*.
- Kucukelbir, A., D. Tran, R. Ranganath, A. Gelman, and D. M. Blei (2017). “Automatic differentiation variational inference”. In: *The Journal of Machine Learning Research* 18.1, pp. 430–474.
- Kumar, S., M. Mohri, and A. Talwalkar (2012). “Sampling methods for the Nyström method”. In: *Journal of Machine Learning Research* 13, pp. 981–1006.
- Kuss, M. and C. E. Rasmussen (2005). “Assessing approximate inference for binary Gaussian process classification”. In: *Journal of machine learning research* 6, pp. 1679–1704.
- Lázaro-Gredilla, M., J. Quiñonero-Candela, C. E. Rasmussen, and A. Figueiras-Vidal (2010). “Sparse

- spectrum Gaussian process regression”. In: *Journal of Machine Learning Research* 11.6, pp. 1865–1881.
- Lazarsfeld, P. and N. Henry (1968). *Latent structure analysis*. Houghton Mifflin Company, Boston, Massachusetts.
- Lee, J., J. Sohl-dickstein, J. Pennington, R. Novak, S. Schoenholz, and Y. Bahri (2018). “Deep Neural Networks as Gaussian Processes”. In: *International Conference on Learning Representations*.
- Li, C., S. Jegelka, and S. Sra (2016a). “Fast DPP sampling for Nyström with application to kernel methods”. In: *International Conference on Machine Learning*.
- Li, F., B. Zhang, and B. Liu (2016b). “Ternary weight networks”. In: *arXiv preprint arXiv:1605.04711*.
- Liang, Y., H. Lee, S. Lim, W. Lin, K. Lee, and C. Wu (2002). “Proper orthogonal decomposition and its applications—Part I: Theory”. In: *Journal of Sound and vibration* 252.3, pp. 527–544.
- Liu, S. and G. Trenkler (2008). “Hadamard, Khatri-Rao, Kronecker and other matrix products”. In: *International Journal of Information and Systems Sciences* 4.1, pp. 160–177.
- Louizos, C., K. Ullrich, and M. Welling (2017). “Bayesian compression for deep learning”. In: *Advances in Neural Information Processing Systems*, pp. 3288–3298.
- Maddison, C. J., A. Mnih, and Y. W. Teh (2016). “The concrete distribution: A continuous relaxation of discrete random variables”. In: *arXiv preprint arXiv:1611.00712*.
- Matthews, A. G. d. G., M. van der Wilk, T. Nickson, K. Fujii, A. Boukouvalas, P. León-Villagrà, Z. Ghahramani, and J. Hensman (Apr. 2017). “GPflow: A Gaussian process library using TensorFlow”. In: *Journal of Machine Learning Research* 18.40, pp. 1–6.
- Minka, T. P. (2001). “A family of algorithms for approximate Bayesian inference”. PhD thesis. Massachusetts Institute of Technology.
- Mühlenbein, H., M. Schomisch, and J. Born (1991). “The parallel genetic algorithm as function optimizer”. In: *Parallel computing* 17.6-7, pp. 619–632.
- Musco, C. and C. Musco (2017). “Recursive Sampling for the Nyström Method”. In: *Advances in Neural Information Processing Systems*.
- Narcowich, F. J. and J. D. Ward (1994). “Generalized Hermite interpolation via matrix-valued conditionally positive definite functions”. In: *Mathematics of Computation* 63.208, pp. 661–687.
- Natural Resources and Forestry (2008). *Ontario Daily In-filled Climate Data*.  
<https://www.ontario.ca/data/filled-climate-data>. Contains information licensed under the Open Government Licence – Ontario.
- Neal, R. M. (1995). “Bayesian Learning for Neural Networks”. PhD thesis. University of Toronto.
- (1997). *Monte Carlo implementation of Gaussian process models for Bayesian regression and classification*. Tech. rep. 9702. University of Toronto.
- (1998). “Regression and classification using Gaussian process priors”. In: *Bayesian Statistics* 6. Ed. by J. Bernardo, J. Berger, A. Dawid, and A. Smith, p. 475.
- Nickson, T., T. Gunter, C. Lloyd, M. A. Osborne, and S. Roberts (2015). “Blitzkriging: Kronecker-structured Stochastic Gaussian Processes”. In: *arXiv preprint arXiv:1510.07965*.
- Nielsen, F. and K. Sun (2016). “Guaranteed bounds on the Kullback-Leibler divergence of univariate mixtures using piecewise log-sum-exp inequalities”. In: *arXiv:1606.05850*.
- Osborne, M. A., S. J. Roberts, A. Rogers, S. D. Ramchurn, and N. R. Jennings (2008). “Towards Real-Time Information Processing of Sensor Network Data Using Computationally Efficient Multi-output Gaussian Processes”. In: *International Conference on Information Processing in Sensor Networks*, pp. 109–120.
- Owen, A. B. (2013). *Monte Carlo theory, methods and examples*.
- Peng, H. and Y. Qi (2015). “EigenGP: Gaussian Process Models with Adaptive Eigenfunctions.” In: *International Joint Conference on Artificial Intelligence*, pp. 3763–3769.
- Pinheiro, J. C. and D. M. Bates (1996). “Unconstrained parametrizations for variance-covariance matrices”.



- In: *Statistics and computing* 6.3, pp. 289–296.
- Poggio, T. and F. Girosi (1990). “Networks for approximation and learning”. In: *proceedings of the Institute of Electrical and Electronics Engineers (IEEE)* 78.9, pp. 1481–1497.
- Quiñonero-Candela, J. and C. E. Rasmussen (2005). “A unifying view of sparse approximate Gaussian process regression”. In: *Journal of Machine Learning Research* 6.12, pp. 1939–1959.
- Raben, S. G., J. J. Charonko, and P. P. Vlachos (2012). “Adaptive gappy proper orthogonal decomposition for particle image velocimetry data reconstruction”. In: *Measurement Science and Technology* 23.2.
- Rahimi, A. and B. Recht (2007). “Random features for large-scale kernel machines”. In: *Advances in Neural Information Processing Systems*, pp. 1177–1184.
- Ranganath, R., S. Gerrish, and D. Blei (2014). “Black box variational inference”. In: *Artificial Intelligence and Statistics*, pp. 814–822.
- Rasmussen, C. E. and C. K. I. Williams (2006). *Gaussian Processes for Machine Learning*. MIT Press.
- Rastegari, M., V. Ordonez, J. Redmon, and A. Farhadi (2016). “XNOR-Net: Imagenet classification using binary convolutional neural networks”. In: *European Conference on Computer Vision*. Springer, pp. 525–542.
- Remes, S., M. Heinonen, and S. Kaski (2017). “Non-stationary spectral kernels”. In: *Advances in Neural Information Processing Systems*, pp. 4645–4654.
- Robbins, H. and S. Monro (1951). “A stochastic approximation method”. In: *The annals of mathematical statistics*, pp. 400–407.
- Roberts, S., M. Osborne, M. Ebden, S. Reece, N. Gibson, and S. Aigrain (2013). “Gaussian processes for time-series modelling”. In: *Philosophical Transactions of the Royal Society A* 371.1984.
- Saatçi, Y. (2011). “Scalable inference for structured Gaussian process models”. PhD thesis. University of Cambridge.
- Saini, P., C. M. Arndt, and A. M. Steinberg (2016). “Development and evaluation of gappy-POD as a data reconstruction technique for noisy PIV measurements in gas turbine combustors”. In: *Experiments in Fluids* 57.7, pp. 1–15.
- Schölkopf, B. and A. J. Smola (2002). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press.
- Shepard, D. (1968). “A two-dimensional interpolation function for irregularly-spaced data”. In: *Proceedings of the Association for Computing Machinery (ACM) national conference*, pp. 517–524.
- Silverman, B. W. (1985). “Some aspects of the spline smoothing approach to non-parametric regression curve fitting”. In: *Journal of the Royal Statistical Society B* 47.1, pp. 1–52.
- Smola, A. J. and P. Bartlett (2001). “Sparse greedy Gaussian process regression”. In: *Advances in Neural Information Processing Systems*, pp. 619–625.
- Smola, A. J. and B. Schölkopf (2000). “Sparse greedy matrix approximation for machine learning”. In: *International Conference on Machine Learning*, pp. 911–918.
- Snelson, E. and Z. Ghahramani (2006). “Sparse Gaussian processes using pseudo-inputs”. In: *Advances in Neural Information Processing Systems*, pp. 1257–1264.
- Snoek, J., K. Swersky, R. Zemel, and R. Adams (2014). “Input warping for Bayesian optimization of non-stationary functions”. In: *International Conference on Machine Learning*, pp. 1674–1682.
- Solin, A. and S. Särkkä (2020). “Hilbert space methods for reduced-rank Gaussian process regression”. In: *Statistics and Computing* 30.2, pp. 419–446.
- Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). “Dropout: a simple way to prevent neural networks from overfitting”. In: *The journal of machine learning research* 15.1, pp. 1929–1958.
- Stein, M. L. (1999). *Interpolation of spatial data*. Springer Verlag.

- Steinberg, A. M., C. M. Arndt, and W. Meier (2013). “Parametric study of vortex structures and their dynamics in swirl-stabilized combustion”. In: *Combustion Institute* 34.2, pp. 3117–3125.
- Stopper, U., W. Meier, R. Sadanandan, M. Stöhr, M. Aigner, and G. Bulat (2013). “Experimental study of industrial gas turbine flames including quantification of pressure influence on flow field, fuel/air premixing and flame shape”. In: *Combustion and Flame* 160.10, pp. 2103–2118.
- Temme, J. E., P. M. Allison, and J. F. Driscoll (2014). “Combustion instability of a lean premixed prevaporized gas turbine combustor studied using phase-averaged PIV”. In: *Combustion and Flame* 161.4, pp. 958–970.
- Thrun, S., W. Burgard, and D. Fox (2005). *Probabilistic robotics*. MIT press.
- Tipping, M. E. (2000). “The relevance vector machine”. In: *Advances in neural information processing systems*, pp. 652–658.
- (2001). “Sparse Bayesian learning and the relevance vector machine”. In: *Journal of machine learning research* 1. Jun, pp. 211–244.
- Titsias, M. (2009). “Variational learning of inducing variables in sparse Gaussian processes”. In: *Artificial Intelligence and Statistics*, pp. 567–574.
- Titsias, M. K., N. Lawrence, and M. Rattray (2008). “Markov Chain Monte Carlo Algorithms for Gaussian Processes”. In: *Inference and Estimation in Probabilistic Time-Series Models*, p. 9.
- Tomczak, J. M. and M. Welling (2016). “Improving variational auto-encoders using Householder flow”. In: *arXiv preprint arXiv:1611.09630*.
- Ton, J.-F., S. Flaxman, D. Sejdinovic, and S. Bhatt (2018). “Spatial mapping with Gaussian processes and nonstationary Fourier features”. In: *Spatial statistics* 28, pp. 59–78.
- Tran, D., A. Kucukelbir, A. B. Dieng, M. Rudolph, D. Liang, and D. M. Blei (2016). “Edward: A library for probabilistic modeling, inference, and criticism”. In: *arXiv preprint arXiv:1610.09787*.
- Tucker, G., A. Mnih, C. J. Maddison, J. Lawson, and J. Sohl-Dickstein (2017). “REBAR: Low-variance, unbiased gradient estimates for discrete latent variable models”. In: *Advances in Neural Information Processing Systems*, pp. 2627–2636.
- van der Wilk, M. (2019). “Sparse Gaussian Process Approximations and Applications”. PhD thesis. University of Cambridge.
- Van Loan, C. F. (2000). “The ubiquitous Kronecker product”. In: *Journal of Computational and Applied Mathematics* 123.1, pp. 85–100.
- Vapnik, V. (1998). *Statistical Learning Theory*. John Wiley & Sons.
- Wahba, G. (1990). *Spline models for observational data*. SIAM.
- Wainwright, M. J. and M. I. Jordan (2008). “Graphical models, exponential families, and variational inference”. In: *Foundations and Trends in Machine Learning* 1.1–2, pp. 1–305.
- Wang, S. and Z. Zhang (2013). “Improving CUR matrix decomposition and the Nyström approximation via adaptive sampling”. In: *Journal of Machine Learning Research* 14, pp. 2729–2769.
- Williams, C. K. I. and M. Seeger (2001). “Using the Nyström method to speed up kernel machines”. In: *Advances in Neural Information Processing Systems*, pp. 682–688.
- Williams, C. K. and D. Barber (1998). “Bayesian classification with Gaussian processes”. In: *Institute of Electrical and Electronics Engineers (IEEE) Transactions on Pattern Analysis and Machine Intelligence* 20.12, pp. 1342–1351.
- Williams, R. J. (1992). “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Reinforcement Learning*, pp. 5–32.
- Wilson, A. G., E. Gilboa, J. P. Cunningham, and A. Nehorai (2014). “Fast kernel learning for multidimensional pattern extrapolation”. In: *Advances in Neural Information Processing Systems*, pp. 3626–3634.

- Wilson, A. G., Z. Hu, R. Salakhutdinov, and E. P. Xing (2016). “Deep kernel learning”. In: *Artificial Intelligence and Statistics*, pp. 370–378.
- Wilson, A. G. and H. Nickisch (2015). “Kernel Interpolation for Scalable Structured Gaussian Processes (KISS-GP)”. In: *International Conference on Machine Learning*, pp. 1775–1784.
- Yan, X., B. Xie, L. Song, and B. Boots (2015). “Large-scale Gaussian process regression via doubly stochastic gradient descent”. In: *The ICML Workshop on Large-Scale Kernel Learning*.
- Yang, Z., A. J. Smola, L. Song, and A. G. Wilson (2015). “À la Carte – Learning Fast Kernels”. In: *Artificial Intelligence and Statistics*, pp. 1098–1106.
- Zhang, K., I. W. Tsang, and J. T. Kwok (2008). “Improved Nyström low-rank approximation and error analysis”. In: *International Conference on Machine Learning*, pp. 1232–1239.
- Zhou, A., A. Yao, Y. Guo, L. Xu, and Y. Chen (2017). “Incremental network quantization: Towards lossless CNNs with low-precision weights”. In: *arXiv preprint arXiv:1702.03044*.

## Appendix A

# Kronecker Matrix Algebra

This section will review useful algebra properties involving Kronecker product matrices which are used extensively throughout the thesis. As will be evident in the below identities, many algebraic operations can be cheaply performed with great computational efficiency when matrices have a Kronecker product structure. In the numerical studies throughout the thesis, we showed that Kronecker matrix algebra enables algebraic operations to be performed with large and dense matrices.

Let  $\mathbf{A} \in \mathbb{R}^{k \times \ell}$  and let  $\mathbf{B} \in \mathbb{R}^{m \times n}$ . The Kronecker product of  $\mathbf{A}$  and  $\mathbf{B}$  is the  $km \times \ell n$  matrix:

$$\mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} a_{1,1}\mathbf{B} & a_{1,2}\mathbf{B} & \cdots & a_{1,\ell}\mathbf{B} \\ a_{2,1}\mathbf{B} & a_{2,2}\mathbf{B} & \cdots & a_{2,\ell}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k,1}\mathbf{B} & a_{k,2}\mathbf{B} & \cdots & a_{k,\ell}\mathbf{B} \end{pmatrix},$$

where  $a_{i,j}$  denotes the element of  $\mathbf{A}$  in row  $i$  and column  $j$ . This tensor product is also known as the direct product or the Zehfuss product, which is perhaps a more appropriate name based upon its original inventor (Henderson et al., 1983).

Below we list some general and useful properties of the Kronecker product (Van Loan, 2000). The important aspect to note is that none of these identities require the explicit evaluation of the Kronecker product as described in the preceding equation. This is valuable since the explicit evaluation of the Kronecker product can be very memory intensive for large matrices.

**Multiplication with a scalar:**

$$\mathbf{A} \otimes (\alpha \mathbf{B}) = \alpha (\mathbf{A} \otimes \mathbf{B}) = (\alpha \mathbf{A}) \otimes \mathbf{B}.$$

**Distributive and Associative properties:**

$$(\mathbf{A} + \mathbf{B}) \otimes \mathbf{C} = (\mathbf{A} \otimes \mathbf{C}) + (\mathbf{B} \otimes \mathbf{C}),$$

$$\mathbf{A} \otimes (\mathbf{B} + \mathbf{C}) = (\mathbf{A} \otimes \mathbf{B}) + (\mathbf{A} \otimes \mathbf{C}),$$

$$(\mathbf{A} \otimes \mathbf{B}) \otimes \mathbf{C} = \mathbf{A} \otimes (\mathbf{B} \otimes \mathbf{C}).$$

**Transpose:**

$$(\mathbf{A} \otimes \mathbf{B})^T = \mathbf{A}^T \otimes \mathbf{B}^T.$$

**Multiplication with another Kronecker Product Matrix:** Assuming that the number of columns of  $\mathbf{A}$  equals the number of rows of  $\mathbf{C}$  and similarly for  $\mathbf{B}$  and  $\mathbf{D}$ , then

$$(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = \mathbf{AC} \otimes \mathbf{BD}.$$

**Matrix Inverse:** Assume that  $\mathbf{A}$  and  $\mathbf{B}$  are square and invertible. Then

$$(\mathbf{A} \otimes \mathbf{B})^{-1} = \mathbf{A}^{-1} \otimes \mathbf{B}^{-1}.$$

**Cholesky Factorization:** Assume that  $\mathbf{A}$  and  $\mathbf{B}$  are Hermitian positive-definite. Then

$$\mathbf{A} \otimes \mathbf{B} = (\mathbf{U}_A \mathbf{U}_A^T) \otimes (\mathbf{U}_B \mathbf{U}_B^T) = (\mathbf{U}_A \otimes \mathbf{U}_B)(\mathbf{U}_A \otimes \mathbf{U}_B)^T,$$

where  $\mathbf{U}_A, \mathbf{U}_B$  are lower triangular matrices, each of the same size as  $\mathbf{A}, \mathbf{B}$ , respectively. Consequently,  $\mathbf{U}_A \otimes \mathbf{U}_B$  is also lower triangular.

**Schur Decomposition:** Assume that  $\mathbf{A}$  and  $\mathbf{B}$  are square. Then

$$\mathbf{A} \otimes \mathbf{B} = (\mathbf{Q}_A \mathbf{T}_A \mathbf{Q}_A^T) \otimes (\mathbf{Q}_B \mathbf{T}_B \mathbf{Q}_B^T) = (\mathbf{Q}_A \otimes \mathbf{Q}_B)(\mathbf{T}_A \otimes \mathbf{T}_B)(\mathbf{Q}_A \otimes \mathbf{Q}_B)^T,$$

where  $\mathbf{Q}_A, \mathbf{Q}_B$  and  $\mathbf{T}_A, \mathbf{T}_B$  are unitary matrices and diagonal matrices, respectively, each of the same size as  $\mathbf{A}, \mathbf{B}$ , respectively. Consequently,  $\mathbf{Q}_A \otimes \mathbf{Q}_B$  and  $\mathbf{T}_A \otimes \mathbf{T}_B$  are also unitary and diagonal, respectively.

**Matrix Vector Multiplication:** Assume  $\mathbf{A} \in \mathbb{R}^{k \times \ell}$ ,  $\mathbf{B} \in \mathbb{R}^{m \times n}$  and  $\mathbf{x} \in \mathbb{R}^{\ell n}$ . Then

$$(\mathbf{A} \otimes \mathbf{B})\mathbf{x} = \text{vec}(\mathbf{A} \text{ reshape}(\mathbf{B} \text{ reshape}(\mathbf{x}, n, \ell), \ell, m)) = \mathbf{y} \in \mathbb{R}^{km},$$

where  $\text{vec}(\cdot)$  “stacks” the rows of the matrix passed to it to turn a matrix into a vector assuming column-major ordering and  $\text{reshape}(\cdot, \alpha, \beta)$  reshapes the matrix passed to it to size  $\alpha \times \beta$  using column-major ordering. Please see (Saatçi, 2011, algorithm 15) for an efficient matrix-vector product algorithm that can easily take advantage of BLAS level 3 functionality.

## Appendix B

# Regularization Kernel Method for Multi-dimensional Grids

This appendix discusses interesting connections and extensions of the work in chapter 3 for a regularization (frequentist) perspective, as opposed to a Bayesian perspective. This content is placed in the appendix since the focus of the thesis is on a Bayesian perspective; however, we believe there is merit in bridging the connections between these multiple lines of research. In this appendix we re-use notations and problem setup from chapter 3 unless otherwise specified. Specifically, we are considering the scenario when the inputs of a regression problem are structured on a multi-dimensional Cartesian product grid, with some missing responses (or “gaps”).

We discuss here an alternative regularization (frequentist) perspective where the function of interest is assumed to belong to a reproducing kernel Hilbert space (RKHS) (e.g., Schölkopf and Smola (2002), Vapnik (1998), and Wahba (1990)). The predictive model is of the form

$$f(\mathbf{x}) \approx \hat{f}(\mathbf{x}) = \sum_{i=1}^N \alpha_i k_i(\mathbf{x}), \quad (\text{B.1})$$

where  $k_i(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}_i)$ ,  $k(\cdot, \cdot)$  is the radial basis function (RBF) kernel, and  $\alpha_i$  for  $i = 1, 2, \dots, N$  denotes the RBF weights.

Considering ridge regression, the model is derived as the minimizer of the regularized function

$$\sum_{i=1}^N \left( \hat{f}(\mathbf{x}_i) - y_i \right)^2 + \sigma^2 \alpha_i^2, \quad (\text{B.2})$$

which we can formulate as

$$\boldsymbol{\alpha} = \arg \min_{\boldsymbol{\xi} \in \mathbb{R}^N} \|\mathbf{K}\boldsymbol{\xi} - \mathbf{y}\|_2^2 + \sigma^2 \|\boldsymbol{\xi}\|_2^2, \quad (\text{B.3})$$

where  $\boldsymbol{\alpha} \in \mathbb{R}^N$  is the vector of basis function weights,  $\mathbf{K} \in \mathbb{R}^{N \times N}$ , is the symmetric and positive definite Gram matrix formed by the kernel evaluations such that  $k_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ ,  $\mathbf{x}_i$  is the  $i$ th training point inputs,  $\mathbf{y} \in \mathbb{R}^N$  is the vector of training responses, and  $\sigma^2 \geq 0$  is the ridge regular-

ization factor which prevents overfitting and accounts for any noise which corrupts the training data. It can be shown that the preceding optimization problem has the analytic solution

$$(\mathbf{K} + \sigma^2 \mathbf{I}_N)^{-1} \mathbf{y} = \boldsymbol{\alpha}, \quad (\text{B.4})$$

and since the Gram matrix  $\mathbf{K}$  is generally dense, it is evident that training nominally requires  $\mathcal{O}(N^3)$  time, a prohibitive expense for large  $N$ . If; however, the data is structured on a Cartesian product grid and the kernel has a product correlation structure, i.e.,  $k(\mathbf{x}_i, \mathbf{x}_j) = \prod_{l=1}^d k_l(x_{il}, x_{jl})$ , then  $\mathbf{K} = \bigotimes_{j=1}^d \mathbf{K}_j$  admits the Kronecker product decomposition along with its algebraic benefits as discussed in section 3.2. Further, using the ‘‘gappy’’ notation from section 3.2, we can rewrite our non-gappy training problem of eq. (B.4) as

$$(\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_N)^{-1} \mathbf{y}_X = \boldsymbol{\alpha}_X, \quad (\text{B.5})$$

where we once again observe that  $\mathbf{K}_{X,X} \in \mathbb{R}^{N \times N}$  *does not* contain a Kronecker product structure due to the presence of the gaps in the data. Note that this problem is identically structured to the linear system of equations addressed in sections 3.3.1 to 3.3.4 and thus the techniques developed therein can be directly applied.

It is evident that the present approach is closely related to the fast Gaussian process inference outlined in chapter 3 where the same reproducing kernel is used to model the process covariance. In-fact the model structure of the regularization model and the GP model posterior mean are mathematically identical, a more detailed discussion of the connections between these models can be found elsewhere (Poggio and Girosi, 1990; Rasmussen and Williams, 2006; Wahba, 1990). Due to this connection, many of the techniques developed in chapter 3 can easily be applied to the regularization model presented here, the difference lies in how kernel hyperparameters,  $\boldsymbol{\theta}$ , and the regularization factor,  $\sigma^2$  are selected. While for the Gaussian process model of chapter 3 this was done by maximizing the marginal likelihood, here consider an alternative generalization metric based on cross validation. Specifically, we minimize the commonly employed generalized cross validation (GCV) error metric (Golub et al., 1979) which can be computed for a full problem (without gaps) as

$$\ell(\boldsymbol{\theta}, \sigma^2) = \frac{\frac{1}{M} \sum_{i=1}^N \left( \frac{\sigma^2}{\mathbf{T}_{ii} + \sigma^2} \right)^2 \mathbf{z}_i^2}{\left[ \frac{1}{M} \sum_{i=1}^N \frac{\sigma^2}{\mathbf{T}_{ii} + \sigma^2} \right]^2}, \quad (\text{B.6})$$

where the square is computed elementwise,  $\mathbf{z} = (\mathbf{Q}^T \mathbf{y}) \in \mathbb{R}^M$ ,  $\mathbf{T} = \bigotimes_{j=1}^d \mathbf{T}_j$ ,  $\mathbf{Q} = \bigotimes_{j=1}^d \mathbf{Q}_j \in \mathbb{R}^{M \times M}$  are the diagonal and unitary Kronecker product matrices defined in eq. (3.5), and once again,  $\boldsymbol{\theta}, \sigma^2$  are the kernel shape parameters and the ridge regression parameter, respectively. We choose the hyperparameters  $\{\boldsymbol{\theta}, \sigma^2\}$  to be a minimizer of our generalization error metric,  $\ell$ .

It can be shown that eq. (B.6) can be rewritten as

$$\ell(\boldsymbol{\theta}, \sigma^2) = \sigma^2 N \frac{\mathbf{y}^T (\mathbf{K} + \sigma^2 \mathbf{I}_M)^{-1} \mathbf{y}}{\text{Tr}[(\mathbf{K} + \sigma^2 \mathbf{I}_M)^{-1}]^2}, \quad (\text{B.7})$$

where  $\text{Tr}[\cdot]$  computes the matrix trace of the argument. Rewriting this for the gappy case gives

$$\ell(\boldsymbol{\theta}, \sigma^2) = \sigma^2 N \frac{\mathbf{y}_X^T (\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_N)^{-1} \mathbf{y}_X}{\text{Tr}[(\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_N)^{-1}]^2}, \quad (\text{B.8})$$

which can be efficiently computed using the tools outlined in chapter 3. Firstly, observe that the numerator contains the solution of a linear system identical to eq. (B.5) and can therefore be quickly solved using the efficient techniques of sections 3.3.1 to 3.3.4. Secondly, note that using the Nyström approximation of the eigenvalues of  $\mathbf{K}_{X,X}$  (as discussed in section 3.4), we can rewrite the denominator as

$$\text{Tr}[(\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_N)^{-1}] = \sum_{i=1}^N \frac{1}{\lambda_i^N + \sigma^2} \approx \sum_{i=1}^N \frac{1}{\frac{N}{M} \lambda_i^M + \sigma^2}, \quad (\text{B.9})$$

where  $\lambda_i^N$ ,  $i=1, \dots, N$  are the eigenvalues of  $\mathbf{K}_{X,X}$ , and  $\lambda_i^M$  is the  $i^{\text{th}}$  largest eigenvalue of  $\mathbf{K} = \bigotimes_{j=1}^d \mathbf{K}_j$  which can be computed very rapidly in  $\mathcal{O}(dM^{\frac{3}{d}})$  time using Kronecker matrix algebra. In this way, a regularization model can be efficiently constructed by re-applying the efficient techniques from chapter 3.



## Appendix C

# Re-Weighted Basis Kernel Log-Marginal Likelihood Derivatives

This section derives the log marginal likelihood (LML) derivatives presented in eq. (5.5). Calculating the  $m+1$  derivatives of the log marginal likelihood (LML) with respect to  $\{\gamma, \sigma^2\}$  using finite difference approximations would require  $\mathcal{O}(m^4)$  time. We show how all these derivatives can be analytically computed in  $\mathcal{O}(m^3)$  time. We also discuss how the use of transformed basis functions (i.e., replacing  $\Phi$  with  $\tilde{\Phi}$ ) allows derivative computations in  $\mathcal{O}(m)$ .

The LML can be written as follows:

$$\log\Pr(\mathbf{y}|\boldsymbol{\theta}, \sigma^2, \mathbf{X}) = -\frac{n}{2}\log(2\pi) - \underbrace{\frac{1}{2}\log|\mathbf{K}_{X,X} + \sigma^2\mathbf{I}_n|}_{\text{Complexity}} - \underbrace{\frac{1}{2}\mathbf{y}^T\boldsymbol{\alpha}}_{\text{Data Fit}},$$

where we use the shorthand  $\boldsymbol{\alpha} = (\mathbf{K}_{X,X} + \sigma^2\mathbf{I}_n)^{-1}\mathbf{y} \in \mathbb{R}^n$ . For clarity, we will derive the gradients of the complexity and data-fit terms separately (interpretations of these terms are discussed in section 2.3).

**Data-Fit  $\gamma$  Derivatives** First we show how the derivative of the data-fit term can be computed within  $\mathcal{O}(m^3)$  time

$$\frac{\partial \mathbf{y}^T \boldsymbol{\alpha}}{\partial \gamma_i} = -\boldsymbol{\alpha}^T \frac{\partial \mathbf{K}_{X,X}}{\partial \gamma_i} \boldsymbol{\alpha} = -(\phi_i^T \boldsymbol{\alpha})^2$$

where  $\boldsymbol{\alpha} = (\mathbf{K}_{X,X} + \sigma^2\mathbf{I}_n)^{-1}\mathbf{y} \in \mathbb{R}^n$ , we make the observation that  $\frac{\partial \mathbf{K}_{X,X}}{\partial \gamma_i} = \phi_i \phi_i^T$ , and the square,  $(\cdot)^2$ , is taken element-wise. We can vectorize this to compute all data-fit derivatives

$$\frac{\partial \mathbf{y}^T \boldsymbol{\alpha}}{\partial \boldsymbol{\gamma}} = -(\Phi^T \boldsymbol{\alpha})^2 = -\sigma^{-4}(\Phi^T \mathbf{y} - \mathbf{A}\mathbf{P}^{-1}\Phi^T \mathbf{y})^2,$$

where  $\mathbf{A} = \Phi^T \Phi \in \mathbb{R}^{m \times m}$  and  $\Phi^T \mathbf{y} \in \mathbb{R}^m$  are both precomputed before LML iterations begin, and  $\mathbf{P} = \sigma^2 \mathbf{S} + \mathbf{A} \in \mathbb{R}^{m \times m}$  is also required to compute the LML (see eq. (5.4)) so it is already computed and factorized. Evidently, the data-fit term derivatives can be computed in  $\mathcal{O}(m^3)$

at each LML iteration.

**Complexity Term  $\gamma$  Derivatives** Now we derive the complexity term gradient which is written as follows:

$$\frac{\partial \log |\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_n|}{\partial \gamma_i} = \text{Tr} \left[ (\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_n)^{-1} \frac{\partial \mathbf{K}_{X,X}}{\partial \gamma_i} \right].$$

Using  $\frac{\partial \mathbf{K}_{X,X}}{\partial \gamma_i} = \boldsymbol{\phi}_i \boldsymbol{\phi}_i^T$  and the cyclic permutation invariance of the trace operation, we get

$$\frac{\partial \log |\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_n|}{\partial \gamma_i} = \boldsymbol{\phi}_i^T (\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_n)^{-1} \boldsymbol{\phi}_i.$$

Using the matrix inversion lemma, the preceding equation becomes

$$\frac{\partial \log |\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_n|}{\partial \gamma_i} = \sigma^{-2} (\boldsymbol{\phi}_i^T \boldsymbol{\phi}_i - \boldsymbol{\phi}_i^T \boldsymbol{\Phi} \mathbf{P}^{-1} \boldsymbol{\Phi}^T \boldsymbol{\phi}_i) = \sigma^{-2} (a_{ii} - \mathbf{a}_i^T \mathbf{P}^{-1} \mathbf{a}_i).$$

Evidently, the complexity term derivatives each require  $\mathcal{O}(m^2)$  time so all  $m$  derivatives can be computed in  $\mathcal{O}(m^3)$ . We can write the vectorized computation as

$$\frac{\partial \log |\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_n|}{\partial \boldsymbol{\gamma}} = \sigma^{-2} [\text{diag}(\mathbf{A}) - (\mathbf{A} \odot \mathbf{P}^{-1} \mathbf{A})^T \mathbf{1}_p],$$

where it is evident that the dominating expense  $\mathbf{P}^{-1} \mathbf{A}$  is also required for the data-fit derivatives.

**Noise Variance Derivatives** We show here how the derivatives of the LML with respect to  $\sigma^2$  can be computed in  $\mathcal{O}(n+m)$ , as follows:

$$\frac{\partial \mathbf{y}^T \boldsymbol{\alpha}}{\partial \sigma^2} = -\boldsymbol{\alpha}^T \frac{\partial (\sigma^2 \mathbf{I}_n)}{\partial \sigma^2} \boldsymbol{\alpha} = -\boldsymbol{\alpha}^T \boldsymbol{\alpha} = -\sigma^{-4} (\mathbf{y}^T \mathbf{y} - 2\mathbf{y}^T \boldsymbol{\Phi} \mathbf{P}^{-1} \boldsymbol{\Phi}^T \mathbf{y} + \mathbf{y}^T \boldsymbol{\Phi} \mathbf{P}^{-1} \mathbf{A} \mathbf{P}^{-1} \boldsymbol{\Phi}^T \mathbf{y}),$$

and

$$\frac{\partial \log |\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_n|}{\partial \sigma^2} = \text{Tr} \left( (\mathbf{K}_{X,X} + \sigma^2 \mathbf{I}_n)^{-1} \frac{\partial (\sigma^2 \mathbf{I}_n)}{\partial \sigma^2} \right) = \text{Tr} \left( \sigma^{-2} [\mathbf{I}_n - \boldsymbol{\Phi} \mathbf{P}^{-1} \boldsymbol{\Phi}^T] \right) = \sigma^{-2} [n - \text{Tr}(\mathbf{P}^{-1} \mathbf{A})].$$

Evidently the first relation can be computed in  $\mathcal{O}(m^3)$  if  $\mathbf{y}^T \mathbf{y}$ ,  $\boldsymbol{\Phi}^T \mathbf{y}$ , and  $\mathbf{A}$  are precomputed, and the second relation can be computed in  $\mathcal{O}(m)$  since the matrix product  $\mathbf{P}^{-1} \mathbf{A}$  has already been explicitly computed to compute the derivatives with respect to  $\boldsymbol{\gamma}$ .

**Final Expressions** Combining the derived expressions for the derivatives of the LML with respect to the  $m+1$  hyperparameters  $\{\boldsymbol{\gamma}, \sigma^2\}$ , it is evident that all computations can be

performed in  $\mathcal{O}(m^3)$ . We can write the final expressions as follows:

$$\begin{aligned}\frac{\partial \text{LML}}{\partial \gamma} &= \frac{(\mathbf{r} - \mathbf{A}\mathbf{P}^{-1}\mathbf{r})^2}{2\sigma^4} - \frac{\text{diag}(\mathbf{A}) - (\mathbf{A} \odot \mathbf{P}^{-1}\mathbf{A})^T \mathbf{1}_p}{2\sigma^2}, \\ \frac{\partial \text{LML}}{\partial \sigma^2} &= \frac{\mathbf{y}^T \mathbf{y} - 2\mathbf{r}^T \mathbf{P}^{-1}\mathbf{r} + \mathbf{r}^T \mathbf{P}^{-1}\mathbf{A}\mathbf{P}^{-1}\mathbf{r}}{2\sigma^4} - \frac{n - \text{Tr}(\mathbf{P}^{-1}\mathbf{A})}{2\sigma^2},\end{aligned}$$

where  $\mathbf{r} = \Phi^T \mathbf{y} \in \mathbb{R}^m$ . If we transform the basis functions by replacing  $\Phi$  with  $\tilde{\Phi}$  then it is evident that the derivative computations can be made in  $\mathcal{O}(m)$  since both  $\mathbf{A} = \tilde{\Phi}^T \tilde{\Phi} = \mathbf{I}_{\tilde{m}}$  and  $\mathbf{P} = \sigma^2 \mathbf{S} + \mathbf{A}$  will be diagonal.

## Appendix D

# Additional Content: Quadruply Stochastic Gaussian Processes

This appendix contains additional content for the quadruply stochastic Gaussian processes (QSGP) approach presented in chapter 6.

### D.1 Control Variates: Additional Details

#### D.1.1 Sparse Unbiased Gradient Computation

The gradient of the expectation of the control variate outlined in proposition 6.3 can be directly differentiated to give the exact (dense) gradient; however, we would like sparse unbiased gradients with respect to the parameters  $\boldsymbol{\mu}_{\tilde{\mathbf{i}}\tilde{\mathbf{j}}}$  that are updated at an SGD iteration. Simply taking the relevant terms from the dense gradient and ignoring the others would introduce bias but we can scale this sparsified gradient appropriately to give unbiased gradients. Because the gradient is sparse with just  $|\tilde{\mathbf{i}}\tilde{\mathbf{j}}| \leq 2\bar{m}$  non-zeros, this sparsified gradient needs to be scaled by  $\frac{m}{|\tilde{\mathbf{i}}\tilde{\mathbf{j}}|}$ . This can be implemented in automatic differentiation software to give the correct zeroth derivative and an unbiased estimate of the first derivative as follows:

$$\frac{m}{|\tilde{\mathbf{i}}\tilde{\mathbf{j}}|} \frac{n}{\sigma^2 \bar{n}} \mathbf{a}^{(t)T} \mathbf{a}^{(t)} + \text{stop\_gradient} \left( \left( 1 - \frac{m}{|\tilde{\mathbf{i}}\tilde{\mathbf{j}}|} \right) \frac{n}{\sigma^2 \bar{n}} \mathbf{a}^{(t)T} \mathbf{a}^{(t)} \right),$$

where the argument of `stop_gradient` does not contribute to the gradient during its computation.

#### D.1.2 Additional Control Variates

In this section we outline additional control variates that can be used for the various terms in theorems 6.1 and 6.2. Note that all these control variates can be used to reduce gradient variance without introducing bias and without introducing dependence on  $n$  or  $m$  in the cost of an SGD iteration.

We begin by presenting a control variate for the term  $\frac{m^2}{\tilde{m}^2} \boldsymbol{\mu}_j^T \mathbf{S}_{j,i} \boldsymbol{\mu}_i$  in theorem 6.1. The goal is ultimately to find a low-rank approximation to  $\mathbf{S}$  which depends on the form of the matrix. As a useful example, the popular class of inducing point kernel approximations give a prior precision matrix  $\mathbf{S}$  that is a kernel covariance (or Gram) matrix evaluated on the set of inducing points. For notational convenience, consider the set of inducing points to be identical to the training set points in which case  $n = m$  and  $\mathbf{S} = \mathbf{K}(\mathbf{X}, \mathbf{X})$ . In the following expression, we present the negative control variate (first term) plus the control variate's expectation (term two) that can be simply added to the  $\mathcal{L}_\mu$  estimator in theorem 6.1

$$-\frac{m^2}{\tilde{m}^2} \boldsymbol{\mu}_i^T \mathbf{K}(\mathbf{X}_i, \mathbf{U}) \mathbf{K}(\mathbf{U}, \mathbf{U})^{-1} \mathbf{K}(\mathbf{U}, \mathbf{X}_j) \boldsymbol{\mu}_j + \boldsymbol{\mu}^T \mathbf{K}(\mathbf{X}, \mathbf{U}) \mathbf{K}(\mathbf{U}, \mathbf{U})^{-1} \mathbf{K}(\mathbf{U}, \mathbf{X}) \boldsymbol{\mu},$$

where  $\mathbf{U}$  is a set of  $\bar{n}$  support points in the  $d$ -dimensional input space, and the notation  $\mathbf{K}(\mathbf{X}, \mathbf{U}) \in \mathbb{R}^{m \times \bar{n}}$  denotes the kernel cross-covariance matrix between the sets  $\mathbf{X}$  and  $\mathbf{U}$  such that the  $ij$ th element of this matrix is  $k(\mathbf{x}_i, \mathbf{u}_j)$ . Clearly this expression has an expectation of zero and so simply adding it to the stochastic estimator in theorem 6.1 does not introduce any bias. This control variate has the capacity to reduce variance of the  $\mathcal{L}_\mu$  estimator provided there is correlation between the elements of  $\mathbf{S} = \mathbf{K}(\mathbf{X}, \mathbf{X})$  and  $\mathbf{K}(\mathbf{X}, \mathbf{U}) \mathbf{K}(\mathbf{U}, \mathbf{U})^{-1} \mathbf{K}(\mathbf{U}, \mathbf{X}) \in \mathbb{R}^{m \times m}$ . This matrix is a Nyström approximation of  $\mathbf{K}(\mathbf{X}, \mathbf{X})$ , a low rank kernel approximation that is widely used in the sparse GP community for kernel approximation, and preconditioning (Cutajar et al., 2016; Evans and Nair, 2018c; Peng and Qi, 2015; Snelson and Ghahramani, 2006; Williams and Seeger, 2001). The size of the set  $\mathbf{U}$  should be much smaller than  $n$  and is often selected randomly from the training set  $\mathbf{X}$ ; however, a wealth of other selection strategies have been developed (Belabbas and Wolfe, 2009; Drineas and Mahoney, 2005; Gittens and Mahoney, 2013; Kumar et al., 2012; Li et al., 2016a; Musco and Musco, 2017; Smola and Schölkopf, 2000; Wang and Zhang, 2013; Zhang et al., 2008). Empirically, it has been found that this matrix approximation is quite accurate (Evans and Nair, 2018c; Musco and Musco, 2017), and we find that this control variate dramatically reduces variance in practice.

Unfortunately, naive evaluation of the expectation of this control variate (the second term in the previous equation) requires  $\mathcal{O}(n) = \mathcal{O}(m)$  computations at each iteration. However, when sparse updates are performed at each SGD iteration, an approach similar to that outlined in proposition 6.3 can be used to decrease the per-iteration computations to  $\mathcal{O}(1)$  when using this control variate. This control variate was used in the empirical studies of table 6.3.

We additionally note that a nearly identical control variate to the one outlined in the previous equation can be derived for the terms  $\frac{m^2}{\tilde{m}^2} \mathbf{c}_{j,r}^T \mathbf{S}_{j,i} \mathbf{c}_{i,r}$  in theorem 6.2 by simply replacing  $\boldsymbol{\mu}$  with  $\mathbf{c}_{i,r}$ .

We can also easily develop a control variate for the term  $-\frac{2nm}{\sigma^2 \tilde{m}} \mathbf{y}_\ell^T \boldsymbol{\Phi}_{\ell,i} \boldsymbol{\mu}_i$  from theorem 6.1 as follows:

$$\frac{2m}{\sigma^2 \tilde{m}} \mathbf{b}_i^T \boldsymbol{\mu}_i - \frac{2}{\sigma^2} \mathbf{b}^T \boldsymbol{\mu},$$

where the negative control variate is the first term, the expectation of the control variate is the second term, and  $\mathbf{b} = \Phi^T \mathbf{y} \in \mathbb{R}^m$  is a vector that is precomputed before SGD iterations begin. Similarly to proposition 6.3 it is possible to make the cost of SGD iterations independent of  $n$  and  $m$  with this control variate; however, the precomputation of  $\mathbf{b}$  costs  $\mathcal{O}(nm)$  which could be prohibitive. To reduce this cost the vector  $\mathbf{b}$  could be approximated. We did not explore the use of this control variate in our experiments.

### D.1.3 Control Variates with Empirical Bayes

If empirical Bayes is being performed, then  $\Phi$  is likely to change during optimization and so the basis functions used in eq. (6.3) and in proposition 6.3 should be fixed to the basis functions at initialization. This same approach can be extended to the additional control variates discussed above.

## D.2 Predictive Posterior Augmentation

Although the QSGP inference procedure allows  $m$  to be very large, it is still finite, and this degeneracy can negatively impact the quality of the predictive posterior variance. In this section we discuss *augmentation* to address this where a radial basis function is added to the QSGP model at each test point when evaluating the predictive variance (Quiñonero-Candela and Rasmussen, 2005).

The results of this section are not presented in the main body of the thesis since the proposed augmentation requires a specific choice of kernel approximation, i.e., a specific choice of  $\Phi$  and  $\mathbf{S}$ . Namely, the augmentation results assume that an inducing point approximation is made such that  $\phi_{i,j} = k(\mathbf{x}_i, \mathbf{z}_j)$  and  $s_{i,j} = k(\mathbf{z}_i, \mathbf{z}_j)$ , where  $\mathbf{z}_i \in \mathbb{R}^d$  for  $i = 1, \dots, m$  are the inducing point locations. For ease of notation in our presentation, we assume that  $n = m$  inducing points are centred on all training points such that  $\mathbf{x}_i = \mathbf{z}_i$  for all  $i = 1, \dots, n$ . We will see that this is a sensible choice since augmentation requires the storage of the training dataset at test time, and predictive posterior variance computations using augmentation require  $\mathcal{O}(n)$  time anyway, regardless of  $m$ . This approach can be easily generalized for arbitrary inducing point locations.

The following result demonstrates how augmentation can be implemented affordably.

**Proposition D.1.** *The proposed model with augmentation gives the following predictive variance*

$$\mathbb{V}[y_*] = \mathbf{k}(\mathbf{x}_*)^T \Sigma \mathbf{k}(\mathbf{x}_*) + \frac{\sigma^2 k(\mathbf{x}_*, \mathbf{x}_*)^2}{\mathbf{k}(\mathbf{x}_*)^T \mathbf{k}(\mathbf{x}_*) + \sigma^2 k(\mathbf{x}_*, \mathbf{x}_*)},$$

where it is assumed that there is no posterior correlation between the augmented basis function and the basis functions in  $\mathbf{K}$ .

*Proof.* Let  $\mathbf{X}_*$  denote the set of  $n$  training points with an additional test point  $\mathbf{x}_* = \mathbf{x}_{n+1}$

appended. We will use  $\mathbf{X}_*$  as inducing points at test time so that

$$\Phi = \begin{bmatrix} \mathbf{K}(X, X) & \mathbf{k}(\mathbf{x}_*) \end{bmatrix}, \quad \text{and} \quad \mathbf{S} = \begin{bmatrix} \mathbf{K}(X, X) & \mathbf{k}(\mathbf{x}_*) \\ \mathbf{k}(\mathbf{x}_*)^T & k(\mathbf{x}_*, \mathbf{x}_*) \end{bmatrix}.$$

We will also denote  $\Sigma', \mathbf{C}' \in \mathbb{R}^{n+1 \times n+1}$  as the posterior covariance, and its respective Cholesky parameterization. We can modify  $\mathcal{L}_\Sigma$  from eq. (6.2) to give  $\mathcal{L}'_\Sigma$  as for this augmented model as follows:

$$\mathcal{L}'_\Sigma = \frac{1}{\sigma^2} \text{sum} \left( (\Phi \mathbf{C}')^2 \right) - \log |\Sigma'| + \text{tr}(\mathbf{S} \Sigma') = \sum_{r=1}^{n+1} \frac{1}{\sigma^2} \mathbf{c}'_r{}^T (\Phi^T \Phi + \sigma^2 \mathbf{S}) \mathbf{c}'_r - 2 \log c'_{rr}.$$

The assumption of no posterior correlation between the augmented basis function and the basis functions in the columns of  $\mathbf{K}$  means that  $c'_{n+1,i} = 0$  for  $i = 1, 2, \dots, n$ , giving

$$\mathcal{L}'_\Sigma = \mathcal{L}_\Sigma(\mathbf{C}'_{1:n,1:n}) - 2 \log c'_{n+1,n+1} + \frac{1}{\sigma^2} \left( \mathbf{k}(\mathbf{x}_*)^T \mathbf{k}(\mathbf{x}_*) + \sigma^2 k_{x_*, x_*} \right) c'^2_{n+1,n+1},$$

and consequently the first  $n$  rows and columns of  $\mathbf{C}'$  should be selected to minimize  $\mathcal{L}_\Sigma$  from eq. (6.2) (and also theorem 6.2) without any influence from the test point  $\mathbf{x}_*$ . Therefore, the augmented and non-augmented predictive posterior variance differ only by the contribution of  $c'_{n+1,n+1}$ . Setting the derivative of  $\mathcal{L}'_\Sigma$  with respect to  $c'_{n+1,n+1}$  to zero and solving gives

$$\frac{2}{\sigma^2} \left( \mathbf{k}(\mathbf{x}_*)^T \mathbf{k}(\mathbf{x}_*) + \sigma^2 k_{x_*, x_*} \right) c'_{n+1,n+1} = \frac{2}{c'_{n+1,n+1}},$$

$$c'_{n+1,n+1} = \sqrt{\frac{\sigma^2}{\mathbf{k}(\mathbf{x}_*)^T \mathbf{k}(\mathbf{x}_*) + \sigma^2 k_{x_*, x_*}}},$$

and using the square of this value as the posterior variance for the augmented basis function completes the proof.  $\square$

We make the following additional observations about the use of this augmentation strategy.

**Remark D.1.** *Augmentation cannot decrease the predictive posterior variance.*

**Remark D.2.** *Augmentation will cause the predictive posterior variance to revert to the prior variance far from the training data (where  $\mathbf{k}(\mathbf{x}_*)$  approaches  $\mathbf{0}$ ).*

These follow from the facts that the second term in proposition D.1 must be non-negative, and that the value of the second term approaches the prior variance  $k(\mathbf{x}_*, \mathbf{x}_*)$  as  $\mathbf{k}(\mathbf{x}_*)$  approaches  $\mathbf{0}$ , respectively.

Further, the use of augmentation requires negligible additional work at test time since  $\mathbf{k}(\mathbf{x}_*)$  is already required for both the predictive mean, and the non-augmented predictive variance. The following study demonstrates how augmentation can improve the quality of the predictive variance far from the training data.

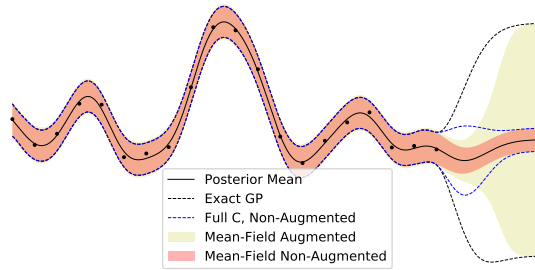


Figure D.1: Comparison of predictive variance between an exact GP, and a QSGP model with  $\Phi = \mathbf{S} = \mathbf{K}(\mathbf{X}, \mathbf{X})$  and both with and without augmentation.

### Augmentation Visualization

Figure D.1 plots the predictive posterior mean and standard deviation of an exact GP, as well as a QSGP model with  $\Phi = \mathbf{S} = \mathbf{K}(\mathbf{X}, \mathbf{X})$  and both with and without augmentation. The dataset was generated using the sinc function with  $n = 20$  points, corrupted with independent Gaussian noise. Note that all models have an identical posterior mean, and we use the exponentiated quadratic kernel outlined in eq. (2.28). Around the training data the augmented and non-augmented models give a nearly identical predictive variance that agrees well with the variance of the exact GP. As the non-augmented model extrapolates, its predictive variance shrinks which is a highly undesirable trait of the degeneracy of the approximate GP. However, the predictive variance of the augmented model grows as it extrapolates, and while it does not grow as fast as the exact GP, it similarly returns to the prior variance, as expected from remark D.2. Figure D.1 also shows the predictive variance where a dense lower triangular (or “full”)  $\mathbf{C}$  is considered. Note that due to the conjugacy of the Gaussian prior, this model gives exact inference and evidently the results are better than the models constructed with a mean-field assumption where the training data ends. However, far from the training data, the posterior variance shrinks back to that of the mean-field model. With augmentation, the full  $\mathbf{C}$  model would be the closest to the exact GP; however, this model is not practical for large  $m$ . A mean-field or chevron Cholesky structure with augmentation could provide a compromise for large models.

### D.3 Site Projection Notes

We can re-write eq. (6.5) as

$$\mathbb{E}_{q(\mathbf{w})}[\log \Pr(\mathbf{y} | \mathbf{X}, \mathbf{w})] = \sum_{\ell=1}^n \mathbb{E}_{\mathcal{N}(z|0,1)} \left[ \log g_{\ell}(\phi_{\ell,:} \boldsymbol{\mu} + z \phi_{\ell,:} \boldsymbol{\Sigma} \phi_{\ell,:}^T) \right],$$

and in this section we will outline what the form of  $\log g_{\ell}$  is for various likelihoods.



### D.3.1 Logistic Likelihood

Here we first consider Bayesian logistic regression and model the class conditional distribution using

$$\Pr(y = 1 | \mathbf{w}, \mathbf{x}) = \text{sig}(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}))$$

where  $\text{sig}(x) = \frac{1}{1 + \exp(-x)}$  is the sigmoid function, and we consider  $\mathbf{y} \in \{-1, 1\}^n$  to be the binary training labels. Using the symmetry property of the logistic sigmoid  $\Pr(y = -1 | \mathbf{w}, \mathbf{x}) = 1 - \Pr(y = 1 | \mathbf{w}, \mathbf{x}) = \text{sig}(-\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}))$  and assuming the data are *i.i.d.*, we can write the log-likelihood of the training set as

$$\log \Pr(\mathbf{y} | \mathbf{X}, \mathbf{w}) = \sum_{\ell=1}^n \log \left( \text{sig}(y_{\ell} \boldsymbol{\phi}_{\ell, \cdot} \mathbf{w}) \right),$$

where it is immediately clear from eq. (6.4) that  $g_{\ell}(x) = \text{sig}(y_{\ell} x)$ . It is therefore evident that  $\log g_{\ell}(x) = -\log(1 + \exp(-y_{\ell} x))$  which is a vertically flipped (and potentially horizontally flipped) softplus function, a function is commonly used as a continuous relaxation of a rectified linear unit (ReLU). Additionally, it is clear by inspection that  $-\log(1 + \exp(-y_{\ell} x))$  is a concave function in  $x$  such that the result in theorem 6.3 holds for this likelihood.

### D.3.2 Laplace Likelihood

We now consider Bayesian regression using a Laplace likelihood and assume the data is *i.i.d.* to give the training set log-likelihood

$$\log \Pr(\mathbf{y} | \mathbf{w}, \mathbf{X}) = \sum_{\ell=1}^n \log \mathcal{L}(y_{\ell} | \boldsymbol{\phi}_{\ell, \cdot} \mathbf{w}, b) = \sum_{\ell=1}^n -\log(2b) - \frac{1}{b} |y_{\ell} - \boldsymbol{\phi}_{\ell, \cdot} \mathbf{w}|,$$

where  $b > 0$  is the scale parameter. It is immediately clear from eq. (6.4) that  $\log g_{\ell}(x) = -\log(2b) - \frac{1}{b} |y_{\ell} - x|$  which is a shifted absolute value function. Additionally, it is clear by inspection that  $\log g_{\ell}(x)$  is a concave function in  $x$  such that the result in theorem 6.3 holds for this likelihood.

The expectation over  $z$  in eq. (6.5) can be computed analytically for this site projection, please refer to (Challis and Barber, 2013) for details.

### D.3.3 Gaussian Likelihood

We now consider Bayesian regression using a Gaussian likelihood and assume the data is distributed *i.i.d.* to give the training set log-likelihood

$$\log \Pr(\mathbf{y} | \mathbf{w}, \mathbf{X}) = \sum_{\ell=1}^n \log \mathcal{N}(y_{\ell} | \boldsymbol{\phi}_{\ell, \cdot} \mathbf{w}, \sigma^2) = \sum_{\ell=1}^n -\frac{1}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} (y_{\ell} - \boldsymbol{\phi}_{\ell, \cdot} \mathbf{w})^2,$$

where it is immediately clear from eq. (6.4) that  $\log g_\ell(x) = -\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}(y_\ell - x)^2$  which a quadratic. Additionally, it is clear by inspection that  $\log g_\ell(x)$  is a concave function in  $x$  such that the result in theorem 6.3 holds for this likelihood.

The expectation over  $z$  in eq. (6.5) can be computed analytically for this site projection, please refer to (Challis and Barber, 2013) for details.

## Appendix E

# DIRECT Bayesian Neural Networks

This appendix extends the DIRECT approach of chapter 7 to the case of a Bayesian neural network. To demonstrate DIRECT computation of the log-likelihood for a Bayesian neural network we will first perform a forward-pass through the neural network from top to bottom; however, unlike how a forward-pass is conventionally conducted in literature where the network is fixed at a specific location in the hypothesis space, we will simultaneously evaluate the neural network at *all* locations in entire hypothesis space. Consequently, a forward-pass through the neural network with our  $n$ -point training set will give us  $\bar{m}^b \times n$  values.

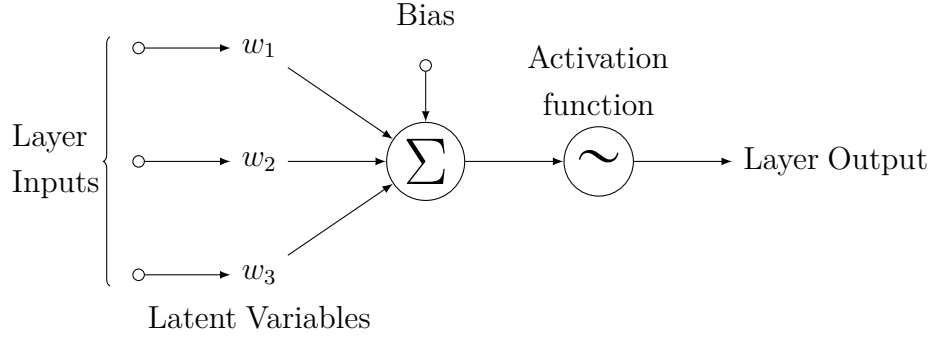
### Nomenclature and Neuron Structure

At all points in the forward-pass we can represent the internal (or final) state of the neural network with a special structure which is a sum of Kronecker product vectors as follows for  $i = 1, \dots, (\text{number of neurons in the layer})$ , and  $l = 1, \dots, n$ ,

$$\mathbf{u}_l^{(i)} = \sum_{j=1}^h c_{jl} \bigotimes_{k=1}^b \mathbf{g}_{jk}^{(i)}, \quad (\text{E.1})$$

where  $\mathbf{U}^{(i)} = \{\mathbf{u}_l^{(i)}\}_{l=1}^n \in \mathbb{R}^{\bar{m}^b \times n}$ ,  $\mathbf{u}_l^{(i)} \in \mathbb{R}^{\bar{m}^b}$  denotes the internal state of the  $i$ th neuron of the current layer, and both  $\mathbf{G}^{(i)} = \{\{\mathbf{g}_{jk}^{(i)}\}_{j=1}^h\}_{k=1}^b \in \mathbb{R}^{h \times b \times \bar{m}}$ ,  $\mathbf{g}_{jk}^{(i)} \in \mathbb{R}^{\bar{m}}$  and  $\mathbf{C} \in \mathbb{R}^{h \times n}$  change as we move from one layer to the next.  $h$  depends on the network architecture and it is constant throughout a layer but grows as we observe deeper layers. Using this nomenclature it is evident that we can compactly represent the internal state of any location within the neural network while we compute our forward pass.

The following image denotes the structure of a neuron that we will use in our neural network.



For clarity of illustration, we will not discuss the bias term although this can be easily added by associating a latent variable with a layer input that is fixed to unity. In our discussion, we will break the computation of the neuron into two stages; the first will involve multiplication of the layer inputs with the latent variables as well as the summation, and the second stage will involve passing this summation through a non-linear activation function.

### Multiplication with Latent Variables and Summation

Our computational neurons begin by multiplying the layer inputs with a specific latent variable and then summing up these values. Assuming we are conducting a forward-pass moving deeper into the network and are currently at the “layer inputs” location in our computational neuron figure, the internal state for the  $i$ th input is denoted by  $\mathbf{U}^{(i)} \in \mathbb{R}^{\bar{m}^b \times n}$  whose structure is defined in eq. (E.1). We must multiply this state by all possible values of the corresponding latent variable, which we will assume is indexed as the  $p$ th of our  $b$  latent variables. We can easily perform this multiplication as follows for  $l = 1, \dots, n$

$$\mathbf{u}_l'^{(i)} = \left( \sum_{j=1}^h c_{jl} \bigotimes_{k=1}^b \mathbf{g}_{jk}^{(i)} \right) \odot \mathbf{W}[p, :]^T, \quad (\text{E.2})$$

$$= \sum_{j=1}^h c_{jl} \left( \bigotimes_{k=1}^{p-1} \mathbf{g}_{jk}^{(i)} \right) \otimes (\mathbf{g}_{jp}^{(i)} \odot \bar{\mathbf{w}}_p) \otimes \left( \bigotimes_{k=p+1}^b \mathbf{g}_{jk}^{(i)} \right) = \sum_{j=1}^h c_{jl} \bigotimes_{k=1}^b \mathbf{g}_{jk}^{(i)}, \quad (\text{E.3})$$

where  $\odot$  denotes element-wise multiplication, and we have taken advantage of the Kronecker product structure of the rows of  $\mathbf{W}$  as depicted in eq. (7.3). Finally, the summing operation is straightforward for our computational neuron. It simply involves summing the multiplied inputs from each layer input as follows:

$$\sum_{i=1}^{\text{num. inputs}} \sum_{j=1}^h c_{jl} \bigotimes_{k=1}^b \mathbf{g}_{jk}^{(i)}. \quad (\text{E.4})$$

At this point we would update  $h$ ,  $\mathbf{G}$  and  $\mathbf{C}$  to convert this double summation into a single summation before passing through the non-linear activation function, as we will discuss next.

### Quadratic Activation

We will use a quadratic activation function for our neural network. Any other non-linear activation could be used; however, we choose the quadratic since it allows a more compact representation of internal state of the network to be maintained, i.e., it allows for a small  $h$  versus other non-linear activations. Again assuming that the current state at the  $i$ th neuron is defined by  $\mathbf{U}^{(i)}$ , the output for the activation function for the  $i$ th neuron is as follows for  $l = 1, \dots, n$

$$\mathbf{u}_l^{(i)} = \mathbf{u}_l^{(i)} \odot \mathbf{u}_l^{(i)} = \left( \sum_{j=1}^h c_{jl} \bigotimes_{k=1}^b \mathbf{g}_{jk}^{(i)} \right) \odot \left( \sum_{j=1}^h c_{jl} \bigotimes_{k=1}^b \mathbf{g}_{jk}^{(i)} \right), \quad (\text{E.5})$$

$$= \sum_{j=1}^h c_{jl}^2 \bigotimes_{k=1}^b \mathbf{g}_{jk}^{(i)} \odot \mathbf{g}_{jk}^{(i)} + 2 \sum_{j=1}^h \sum_{p=1}^{j-1} c_{jl} c_{pl} \mathbf{g}_{jk}^{(i)} \odot \mathbf{g}_{pk}^{(i)}, \quad (\text{E.6})$$

and at this point we would update  $h$ ,  $\mathbf{G}$  and  $\mathbf{C}$  to convert this double summation into a single summation to represent the internal state compactly before moving deeper.

### Forward-Pass Algorithm

Using the previously defined operations, we can summarize the forward-pass procedure in algorithm `forward_pass`. Note that algorithm `forward_pass` is simplified for clarity of presentation. The computations involved could be performed far more efficiently and in a more stable manner. For example, the vast majority of entries in the  $\mathbf{G}$  matrices are unity, so identifying this could massively decrease storage and computational requirements. Additionally,  $\tilde{\mathbf{C}}$  evidently has a Kronecker product structure which could be carefully exploited to yield benefits for very wide neural networks. For stability, all matrices could be represented by storing both the sign and logarithm of all entries. For deep networks, this could be advantageous to avoid precision loss. Nonetheless, we will proceed with the algorithm as presented, for purposes of clarity.

### ELBO Computation

Computation of the ELBO will proceed similarly to the GLM regression model in section 7.3.1; however, there are several differences since we no longer have constant basis functions, so our state representation is more complicated. We will again assume a Gaussian noise model for the observed responses and will again place a prior over the Gaussian variance. We can then modify eq. (7.10) which focuses on the ELBO term related to the log-likelihood as follows:

$$(\mathbf{q}_\sigma \otimes \mathbf{q})^T \log \ell = -\frac{n}{2} \mathbf{q}_\sigma^T \log \sigma^2 - \frac{1}{2} (\mathbf{q}_\sigma^T \sigma^{-2}) (\mathbf{q}^T \{(\mathbf{y} - \mathbf{U}[i, :]^T)^T (\mathbf{y} - \mathbf{U}[i, :]^T)\}_{i=1}^m}), \quad (\text{E.7})$$

where we assume that we have already conducted algorithm `forward_pass` such that the state  $\mathbf{U}$  represents the output of the neural network. We will now focus on computing the inner product involving the variational distribution over the  $\mathbf{w}$  variables,  $\mathbf{q}$ , which we can

---

**Algorithm forward\_pass** Perform a forward pass for through the neural network using the entire training set and simultaneously computing the outputs for all  $m = \bar{m}^b$  points in the hypothesis space. `mult_var` multiplies the current state with the appropriate latent variable as is done in eq. (E.3), `neuron_sum` computes the neuron sum as is done in eq. (E.4), and `activation` computes the non-linear activation function as is done in eq. (E.6). All the pseudo-functions defined take  $\mathbf{G}$  and/or  $\mathbf{C}$  and perform the necessary computations with those inputs. We omit latent-variable indexing values for clarity of presentation.

---

**Input:**  $\mathbf{X} \in \mathbb{R}^{d \times n}$

**Output:**  $\mathbf{C} \in \mathbb{R}^{h \times n}$  &  $\mathbf{G} \in \mathbb{R}^{h \times b \times \bar{m}}$  which define state  $\mathbf{U} \in \mathbb{R}^{\bar{m}^b \times n}$  in eq. (E.1)

$\mathbf{C} = \mathbf{X}$ ,  $\mathbf{G}^{(i)} = \text{ones}(1 \times b \times \bar{m})$ ,  $i = 1, \dots, d$

**for** each layer **do**

$\tilde{\mathbf{C}} = \text{neuron\_sum}(\{\mathbf{C}\}_1^{\text{num. inputs}}) = \mathbf{1}_{\text{num. inputs}} \otimes \mathbf{C}$

**for**  $j = 1$  **to** num. neurons in layer **do**

**for**  $i = 1$  **to** num. inputs to layer **do**

$\mathbf{G}'^{(i)} = \text{mult\_var}(\mathbf{G}^{(i)})$

multiplication with the appropriate row of  $\mathbf{W}$

**end for**

$\tilde{\mathbf{G}}^{(j)} = \text{neuron\_sum}(\mathbf{G}'^{(1)}, \dots, \mathbf{G}'^{(\text{num. inputs})})$

sum operation for the current ( $j$ th) neuron

**if not** last layer **then**

$\tilde{\mathbf{G}}^{(j)}$ ,  $\tilde{\mathbf{C}} = \text{activation}(\tilde{\mathbf{G}}^{(j)}, \tilde{\mathbf{C}})$

**end if**

**end for**

$\mathbf{C} = \tilde{\mathbf{C}}$ ,  $\mathbf{G}^{(j)} = \tilde{\mathbf{G}}^{(j)}$ ,  $j = 1, \dots, \text{num. neurons in layer}$

update variables

**end for**

$\mathbf{G} = \mathbf{G}^{(1)}$

only one neuron in the last (output) layer, so remove indexing

---

break into three terms as follows:

$$\mathbf{q}^T \{(\mathbf{y} - \mathbf{U}[i,:])^T (\mathbf{y} - \mathbf{U}[i,:])\}_{i=1}^m = \mathbf{y}^T \mathbf{y} - 2\mathbf{q}^T \{\mathbf{y}^T \mathbf{U}[i,:]\}_{i=1}^m + \mathbf{q}^T \{\mathbf{U}[i,:]\mathbf{U}[i,:]\}_{i=1}^m, \quad (\text{E.8})$$

for which the first term is trivial to compute as written since it does not depend on the latent variables. We now demonstrate how the second and third terms can be computed, recalling we assume  $\mathbf{q}$  is a mean-field variational distribution (although we can extend beyond mean-field using the techniques discussed in section 7.4). Considering the second term in eq. (E.8), we can write

$$\begin{aligned} \mathbf{q}^T \{\mathbf{y}^T \mathbf{U}[i,:]\}_{i=1}^m &= \mathbf{q}^T \left( \sum_{k=1}^n y_k \sum_{j=1}^h c_{jk} \bigotimes_{i=1}^b \mathbf{g}_{ij} \right) = \sum_{j=1}^h \left( \sum_{k=1}^n y_k c_{jk} \right) \prod_{i=1}^b \mathbf{q}_i^T \mathbf{g}_{ij}, \\ &= \sum_{j=1}^h p_j \prod_{i=1}^b \mathbf{q}_i^T \mathbf{g}_{ij}, \end{aligned} \quad (\text{E.9})$$

where we have used the short-hand notation  $\mathbf{p} = \{\sum_{k=1}^n y_k c_{jk}\}_j \in \mathbb{R}^h$ . Finally, considering the third term in eq. (E.8), we can write

$$\mathbf{q}^T \{\mathbf{U}[i, :] \mathbf{U}[i, :]^T\}_{i=1}^m = \mathbf{q}^T \sum_{i=1}^n \mathbf{u}_i \odot \mathbf{u}_i = \mathbf{q}^T \sum_{i=1}^n \sum_{j=1}^h \sum_{k=1}^h c_{ji} c_{ki} \bigotimes_{l=1}^b (\mathbf{g}_{jl} \odot \mathbf{g}_{kl}), \quad (\text{E.10})$$

$$= \sum_{j=1}^h \sum_{k=1}^h \left( \sum_{i=1}^n c_{ji} c_{ki} \right) \prod_{l=1}^b \mathbf{q}_l^T (\mathbf{g}_{jl} \odot \mathbf{g}_{kl}), \quad (\text{E.11})$$

$$= \sum_{j=1}^h \sum_{k=1}^h v_{jk} \prod_{l=1}^b \mathbf{q}_l^T (\mathbf{g}_{jl} \odot \mathbf{g}_{kl}), \quad (\text{E.12})$$

where we define  $\mathbf{V} = \{\sum_{i=1}^n c_{ji} c_{ki}\}_{j,k} \in \mathbb{R}^{h \times h}$ . Substituting eq. (E.9) and eq. (E.12) into eq. (E.8), we can now compute the inner product between the variational distribution and the log-likelihood in eq. (E.7). The other terms required to compute the ELBO can be seen in eq. (7.9), and the computation of these other terms do not differ from the case of the generalized linear regression model. So, we can now tractably compute the ELBO for our DIRECT Bayesian neural network.

We can pre-compute the terms  $\mathbf{y}^T \mathbf{y}$ ,  $\mathbf{p}$ , and  $\mathbf{V}$  before training begins (since these do not depend on the variational parameters) such that the final complexity of the DIRECT method is *independent* of the number of training points, making the proposed techniques ideal for massive datasets. Also, it is evident that each of these pre-computed terms can easily be updated as more data is observed making the techniques amenable to online learning applications. If we assume a neural network with  $\ell$  hidden layers and an equal distribution of latent variables between layers, the computational complexity of the ELBO computations are  $\mathcal{O}(\ell \bar{m} (b/\ell)^{4\ell})$ . This can be seen by observing eq. (E.12) and noting that  $h = \mathcal{O}((b/\ell)^{2\ell})$ , and that only  $\mathcal{O}(\ell)$  of the vectors in  $\{\mathbf{g}_{jl}\}_{l=1}^b$  are not unity for any value of  $j = 1, \dots, h$ , allowing computations to be saved.