Flexible and efficient Gaussian process models for machine learning

Edward Lloyd Snelson

M.A., M.Sci., Physics, University of Cambridge, UK (2001)

Gatsby Computational Neuroscience Unit
University College London
17 Queen Square
London WC1N 3AR, United Kingdom

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy
of the
University of London.

2007

I, Edward Snelson, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Abstract

Gaussian process (GP) models are widely used to perform Bayesian nonlinear regression and classification — tasks that are central to many machine learning problems. A GP is nonparametric, meaning that the complexity of the model grows as more data points are received. Another attractive feature is the behaviour of the error bars. They naturally grow in regions away from training data where we have high uncertainty about the interpolating function.

In their standard form GPs have several limitations, which can be divided into two broad categories: computational difficulties for large data sets, and restrictive modelling assumptions for complex data sets. This thesis addresses various aspects of both of these problems.

The training cost for a GP has $\mathcal{O}(N^3)$ complexity, where N is the number of training data points. This is due to an inversion of the $N \times N$ covariance matrix. In this thesis we develop several new techniques to reduce this complexity to $\mathcal{O}(NM^2)$, where M is a user chosen number much smaller than N. The sparse approximation we use is based on a set of M 'pseudo-inputs' which are optimised together with hyperparameters at training time. We develop a further approximation based on clustering inputs that can be seen as a mixture of local and global approximations.

Standard GPs assume a uniform noise variance. We use our sparse approximation described above as a way of relaxing this assumption. By making a modification of the sparse covariance function, we can model input dependent noise. To handle high dimensional data sets we use supervised linear dimensionality reduction. As another extension of the standard GP, we relax the Gaussianity assumption of the process by learning a nonlinear transformation of the output space. All these techniques further increase the applicability of GPs to real complex data sets.

We present empirical comparisons of our algorithms with various competing techniques, and suggest problem dependent strategies to follow in practice.

Acknowledgements

I would like especially to thank my supervisor Zoubin Ghahramani, who has always been inspirational and enthusiastic in guiding my research, as well as a good friend. I would also like to acknowledge the Gaussian process community as a whole; everyone has been extremely receptive, supportive and helpful to me over the course of my PhD. For many useful discussions, I would like to mention these people in particular : Carl Rasmussen, Chris Williams, Joaquin Quiñonero Candela, Neil Lawrence, Matthias Seeger, Peter Sollich and Anton Schwaighofer.

I would like to thank my office mates Iain Murray and Katherine Heller for many hours of stimulating discussions and for their friendship. I have also enjoyed interaction with all members of the Gatsby Unit during my time here. I would like also to thank David MacKay, Tom Minka, and Sam Roweis for helpful discussions at various times during my PhD.

For financial support I thank the UCL Graduate Research Scholarships, the Gatsby Charitable Foundation, and the The PASCAL Network of Excellence.

I thank my family for their caring support, and above all I would like to thank my wife Sarah for her endless support, love and patience.

Contents

List of Figures

List of Tables

Outline

Gaussian processes (GPs) define prior distributions on functions. When combined with suitable noise models or likelihoods, Gaussian process models allow one to perform Bayesian nonparametric regression, classification, and other more complex machine learning tasks. Being Bayesian probabilistic models, GPs handle the uncertainty inherent in function modelling from noisy data, and give full probabilistic predictions or 'error bars'. Chapter 1 gives an overview of Gaussian process modelling, with a focus on regression, but also briefly discussing classification and other tasks.

One of the main problems with applying GPs successfully in machine learning is the computational difficulties for large data sets. GP training scales cubically with the number of training data points N, and predictions are quadratic in N. Given the plethora of large data sets being currently collected, especially for web-based applications, this is a big problem for GP methods. GPs in their standard form have therefore been limited in their applicability to data sets of only several thousand data points.

Chapter 2 discusses methods to deal with this problem. In this chapter we give a review of previous approximation techniques, and we introduce the sparse pseudo-input Gaussian process (SPGP) which we proposed in [Snelson and Ghahramani, 2006a]. The SPGP consists of an approximation based on a small set of M pseudo-inputs, which are then optimised to find the best solution. The training and prediction costs are then reduced respectively to $\mathcal{O}(NM^2)$ and $\mathcal{O}(M^2)$. We explain the relationship between this new method and the many similar previous approaches, both from a theoretical and empirical point of view. From a theoretical perspective we present minimal KL divergence arguments for the SPGP approximation to the GP, and we highlight some of the deficiencies of previous approximations. Empirically we demonstrate strategies for obtaining the best performance when we care most about reducing training time or reducing prediction time.

All the approximation methods we discuss in chapter 2 can be considered global approximations, in the sense that they use a small number of support points to summarise all N training points. In chapter 3 we examine the use of local GP approximations for dealing with the same computational problem. The basic idea is an old one: to break the input space down into smaller regions and use local predictors for each region, thereby saving computation by only utilising nearby data points. We then propose a new method that is a combination of the local type of approximation and the global type of chapter 2, and we show how it can be naturally derived as an extension of the theoretical framework for GP approximations presented by Quiñonero Candela and Rasmussen [2005]. We examine the types of data that are best dealt with by a local approach or by a global approach. The new approximation we present subsumes both types of method. This chapter is based on [Snelson and Ghahramani, 2007].

In chapter 4 we increase the range and complexity of data sets that can be handled by the SPGP that we presented in chapter 2. One of the problems with the SPGP in its initial form is that the optimisation of the pseudo-inputs becomes unfeasible for very high dimensional input spaces. We address this problem by learning a low dimensional projection of the input space within the model. The pseudo-inputs then live in this low-dimensional space. This can be seen as a particular supervised dimensionality reduction for regression. We then examine the capabilities of the SPGP for modelling input dependent noise, or heteroscedasticity. The SPGP covariance function, as well as being computationally efficient, is inherently more flexible than the underlying GP in this regard. We also develop a simple extension of the SPGP that further improves its ability to model heteroscedasticity, and we test the methods in an environmental modelling competition. This chapter is based on [Snelson and Ghahramani, 2006b].

In chapter 5 we make another extension of the GP to increase its modelling flexibility. One limitation with GP regression is the Gaussianity assumption of the process itself. There are many types of data that are not well modelled under this assumption. One simple example is data naturally specified in positive quantities. Standard practice is to apply some sort of preprocessing transformation before modelling. We present a simple extension of the GP that learns such a transformation as part of the probabilistic model itself, transforming to a space where the data is modelled well by a GP. This gives rise to a process which is non-Gaussian in the original data space, and further increases the applicability of GPs to real world modelling and machine learning tasks. This chapter is based on [Snelson et al.,

2004].

Finally we conclude and suggest some ideas for future work in chapter 6. Appendix A explains some notation used throughout the thesis, and appendix B provides some mathematical background. Both may be useful to those unfamiliar with GP models.

Chapter 1

Introduction to Gaussian processes

1.1 A brief history of Gaussian processes

The Gaussian process (GP) is a simple and general class of probability distributions on functions. When viewed in this general setting, Gaussian processes of many types have been studied and used for centuries. The well known Wiener process [e.g. Papoulis, 1991] is a particular type of Gaussian process. However this thesis is concerned with the more specific use of Gaussian processes for prediction. The setting we will be considering is that of *regression* — prediction of a continuous quantity, dependent on a set of continuous inputs, from noisy measurements.

The history of the use of Gaussian processes for prediction is still a long one. Probably due to the most widely studied Gaussian processes being stochastic processes in time (e.g. the Wiener process), Gaussian processes were first used for time series prediction. Work in this area dates back to the 1940's [Wiener, 1949, Kolmogorov, 1941].

Gaussian processes have been widely used since the 1970's in the fields of geostatistics and meteorology. In geostatistics, prediction with Gaussian processes is termed *kriging*, named after the South African mining engineer D. G. Krige by Matheron [1973]. Naturally in spatial statistics the inputs to the process are the two or three space dimensions.

GPs were then applied by statisticians to the slightly more general multivariate input regression problem [e.g. O'Hagan, 1978]. It is fair to say however that their widespread use in statistics is still largely confined to the spatial statistics sub-field.

This thesis is concerned with the use of GPs for prediction in a machine learning context. Whilst the core problem of regression is a standard statistical one, machine learning problems have directed researchers to use and develop GPs along slightly different lines. Williams and Rasmussen [1996] first described Gaussian process regression in a machine learning context. At that time neural networks [e.g. Bishop, 1995] were in vogue as general purpose function approximators, and Williams and Rasmussen [1996] were partly inspired by the link shown by Neal [1996] between GPs and neural networks.

Over the past decade or so, there has been much work on Gaussian processes in the machine learning community. The definitive book on GPs in the context of machine learning is the recent one by Rasmussen and Williams [2006], and most of the material covered in this introductory chapter, and a lot more, can be found there.

1.2 The regression problem

Machine learning problems can be broadly divided into three fundamental classes: supervised learning, unsupervised learning, and reinforcement learning. See [e.g. Hastie et al., 2001, MacKay, 2003] for textbooks describing the field as a whole. The most widely studied and easiest of these tasks is supervised learning, which concerns learning a relationship from inputs to targets (or outputs). Supervised learning may be further subdivided into two primary tasks: classification and regression. In classification the outputs are discrete labels, whereas in regression the outputs are continuous variables.

In their simplest form Gaussian process models are used for regression, and this thesis will concentrate on the regression task. Although the regression problem is one of the simplest and most general statistical problems, it is at the core of many machine learning tasks. Reliable general methods for regression are therefore of prime importance to the field as a whole, and can be utilised inside more complex and specific learning tasks. Gaussian processes have been developed beyond the basic regression model to be used in classification [e.g. Williams and Barber, 1998, Kuss and Rasmussen, 2005], and even unsupervised learning [e.g. Lawrence, 2005] and reinforcement learning [e.g. Engel et al., 2003] contexts. Although this thesis will not specifically cover these extended models, much of what we will develop can be applied to these more complicated tasks too.

In a regression task we have a data set $\mathcal{D}$ consisting of N input vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ (of dimension D) and corresponding continuous outputs $y_1, y_2, \ldots, y_N$. We assume the outputs are noisily observed from an underlying functional mapping $f(\mathbf{x})$. The object of the regression task is to estimate $f(\mathbf{x})$ from the data $\mathcal{D}$. Clearly without further assumptions this problem is ill-defined, and moreover we are dealing with noisy data. Therefore any approach we might take is an exercise in reasoning under uncertainty. For this reason, we really want not just a single estimate of $f(\mathbf{x})$, but a probability distribution over likely functions. A Gaussian process regression model is a fully probabilistic Bayesian model, and so it allows us to do just this. This is in contrast to many other commonly used regression techniques, which only provide a best estimate of $f(\mathbf{x})$.

A Gaussian process defines a probability distribution on functions $p(f)$. This can be used as a Bayesian prior for the regression, and Bayesian inference can be used to make predictions from data:

$$p(f|\mathcal{D}) = \frac{p(\mathcal{D}|f)p(f)}{p(\mathcal{D})} \ . \tag{1.1}$$

This is a high level description of how a GP solves the regression problem outlined above, giving us probabilistic predictions of possible interpolating functions f.

1.3 Gaussian process definition

A Gaussian process is a type of continuous stochastic process, i.e. it defines a probability distribution for functions [e.g. Papoulis, 1991]. Another way to think of this is as a set of random variables indexed by a continuous variable: $f(\mathbf{x})$. Suppose we choose a particular finite subset of these random function variables $\mathbf{f} = \{f_1, f_2, \ldots, f_N\}$, with corresponding inputs (indices) $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$. In a GP, *any* such set of random function variables are distributed multivariate Gaussian:

$$p(\mathbf{f}|\mathbf{X}) = \mathcal{N}(\boldsymbol{\mu}, \mathbf{K}) \ , \tag{1.2}$$

where $\mathcal{N}(\boldsymbol{\mu}, \mathbf{K})$ denotes a Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance $\mathbf{K}$. The final requirement is that these Gaussian distributions are *consistent*; that is the usual rules of probability apply to the collection of random variables, e.g. marginalisation:

$$p(f_1) = \int df_2 \ p(f_1, f_2) \ . \tag{1.3}$$

A GP is a *conditional* probabilistic model. This means that the distribution on the inputs $p(\mathbf{x})$ is not specified, and only the conditional distribution $p(\mathbf{f}|\mathbf{X})$ is modelled. For this reason, throughout the thesis we will often drop the explicit notational conditioning on the inputs, with the understanding that the appropriate inputs are being conditioned on: $p(\mathbf{f}) \equiv p(\mathbf{f}|\mathbf{X})$. For further notational devices used throughout the thesis see appendix A.

1.4 Covariances

To specify a particular GP prior, we need to define the mean $\boldsymbol{\mu}$ and covariance $\mathbf{K}$ of equation (1.2). The GPs we will use as *priors* will have a zero mean. Although this sounds restrictive, offsets and simple trends can be subtracted out before modelling, and so in practice it is not. It is worth noting however, that the *posterior* GP $p(f|\mathcal{D})$ that arises from the regression is not a zero mean process (see section 1.5.1).

The important quantity is the covariance matrix $\mathbf{K}$. We construct this from a covariance function $K(\mathbf{x}, \mathbf{x}')$:

$$\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j) \ . \tag{1.4}$$

This function characterises the correlations between different points in the process:

$$K(\mathbf{x}, \mathbf{x}') = \mathcal{E}[f(\mathbf{x})f(\mathbf{x}')] \ , \tag{1.5}$$

where $\mathcal{E}$ denotes expectation and we have assumed a zero mean. We are free in our choice of covariance function, so long as the covariance matrices produced are always symmetric and positive semidefinite ($\mathbf{v}^\top \mathbf{K} \mathbf{v} \geq 0, \quad \forall \mathbf{v}$). Specifying the covariance matrix $\mathbf{K}$ via a covariance function guarantees the consistency requirement of section 1.3.

The particular choice of covariance function determines the properties of sample functions drawn from the GP prior (e.g. smoothness, lengthscales, amplitude etc). Therefore it is an important part of GP modelling to select an appropriate covariance function for a particular problem. An important aspect of GP research is the development of different more flexible covariance functions. In fact it is one of the goals of this thesis to develop a particular class of flexible and computationally efficient covariance functions. However as a starting point there are a number of well known and widely used covariances which we discuss in the next sections. The class of covariances we will develop in this thesis are built from one of these

simpler covariances.

1.4.1 The squared exponential (SE) covariance

The SE covariance function is the most widely used in machine learning. It provides very smooth sample functions, that are infinitely differentiable:

$$K_{\text{SE}}(r) = a^2 \exp\left(-\frac{r^2}{2\lambda^2}\right) , \tag{1.6}$$

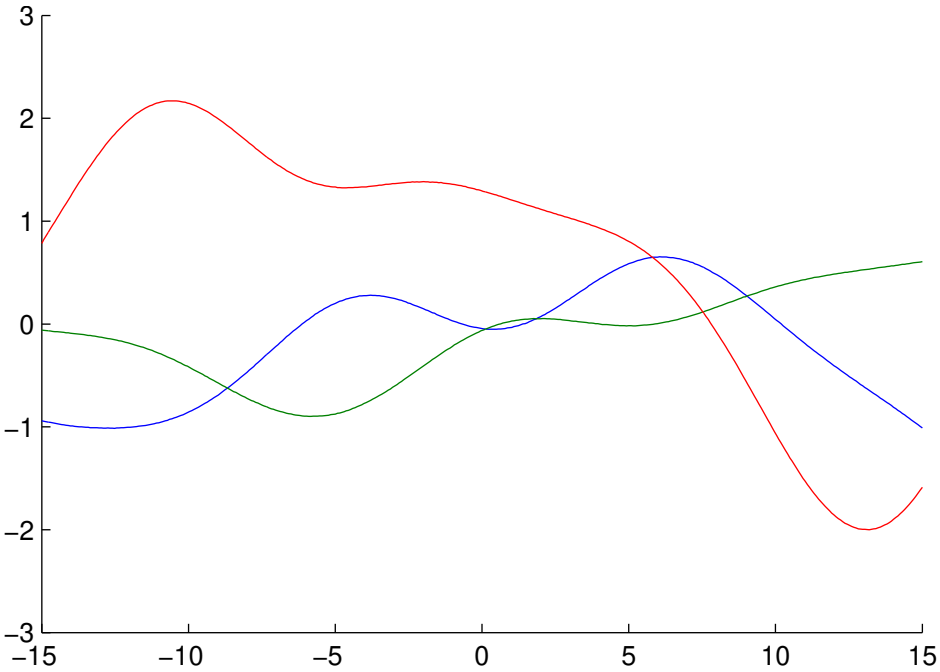
where $r = \|\mathbf{x}-\mathbf{x}'\|$. The SE covariance is stationary (a function of $\mathbf{x}-\mathbf{x}'$ — invariant to translations) and more specifically isotropic (a function of r — invariant to both translation and rotation).

The two *hyperparameters* a and λ govern properties of sample functions. a controls the typical amplitude and λ controls the typical lengthscale of variation. We refer to any hyperparameters of a covariance function collectively as the vector $\boldsymbol{\theta}$. Figure 1.1 shows three sample GP functions using the SE covariance, with (a) and (b) using different lengthscale and amplitude hyperparameters. A function is created in practice by finely discretising the input space and drawing a sample from the multivariate Gaussian equation (1.2) with zero mean $\boldsymbol{\mu} = \mathbf{0}$ (see section B.2 for implementation details). We see that the samples are very smooth functions which vary around the $f = 0$ axis due to the zero mean.

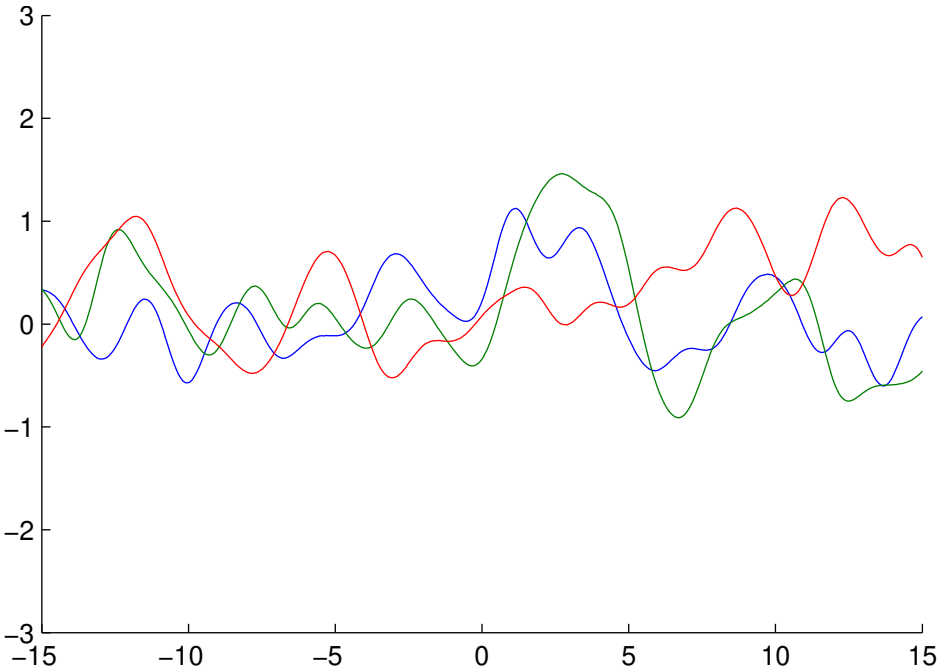
The smoothness of the sample functions arises from the form of the covariance equation (1.6). Function variables close in input space are highly correlated, whereas function variables far apart relative to the lengthscale λ are uncorrelated. A criticism of the SE covariance is that it may be unreasonably smooth for some realistic regression tasks. For this reason some practitioners prefer the Matérn class of covariance (see section 1.4.2).

The structure of the covariance can also be understood visually by plotting the values in the covariance matrix as colours in an image. This is done in figure 1.2. A 1D input space has been discretised equally, and the covariance matrix constructed from this ordered list of inputs is plotted.[1] The region of high covariance appears as a diagonal constant width band, reflecting the local stationary nature of the SE covariance. Increasing the lengthscale λ increases the width of the diagonal band, as points further away from each other become correlated.

[1] The ordering is necessary for visualising the structure, and so this demonstration is only possible in 1D.



(a) Lengthscale $\lambda = 5$, amplitude $a = 1$



(b) Lengthscale $\lambda = 1$, amplitude $a = 0.5$

Figure 1.1: Three sample GP functions with the SE covariance

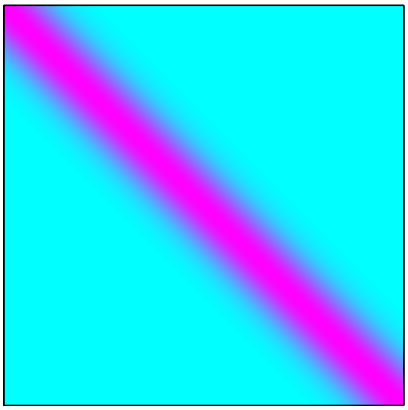


Figure 1.2: The SE prior covariance matrix for a set of equally spaced ordered points. Magenta indicates high covariance, and cyan low.

1.4.2 The Matérn covariance

$$K_{\text{Mat}}(r) = a^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu} r}{\lambda} \right)^{\nu} K_{\nu} \left(\frac{\sqrt{2\nu} r}{\lambda} \right) , \tag{1.7}$$

where K_ν is a modified Bessel function, $\Gamma(\cdot)$ is the Gamma function, and a, ν, and λ are hyperparameters.

The Matérn covariance is isotropic, and again hyperparameters a and λ control amplitude and lengthscale respectively. ν controls how rough the sample functions are. In fact for $\nu \to \infty$ we regain the SE covariance, but for finite ν the sample functions are significantly rougher than those from the SE.

The special case of $\nu = 1/2$ gives the exponential covariance, and the well known Ornstein-Uhlenbeck process [Uhlenbeck and Ornstein, 1930]. Figure 1.3 shows sample functions from the OU process, with lengthscale and amplitude parameters the same as in figure 1.1a. Comparing the two, we can see how much rougher functions the Matérn covariance produces.

1.4.3 Nonstationary covariances

We briefly list a few examples of common nonstationary covariances. The linear covariance produces straight line sample functions, and using it in GP regression is therefore equivalent to doing Bayesian linear regression (see section 1.6):

$$K_{\text{Lin}}(\mathbf{x}, \mathbf{x}') = \sigma_0^2 + \sigma_1^2 \mathbf{x}\mathbf{x}'^{\top} . \tag{1.8}$$

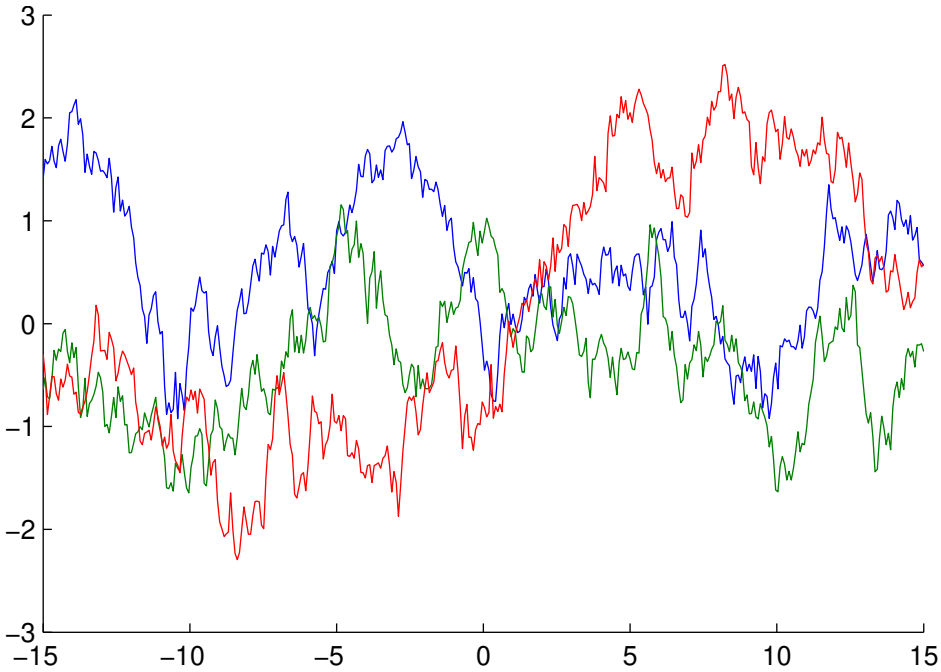


Figure 1.3: Three sample GP functions with the Matérn covariance, $\nu = 1/2$, length-scale $\lambda = 5$, amplitude $a = 1$

The periodic covariance can be used to generate periodic random functions (1D):

$$K_{\text{Per}}(x, x') = \exp\Big(-\frac{2\sin^2\big(\frac{x-x'}{2}\big)}{\lambda^2}\Big) \tag{1.9}$$

The Wiener process, or continuous time Brownian motion, is a one-dimensional nonstationary GP:

$$K_{\text{Wien}}(x, x') = \min(x, x') \;, \qquad x, x' \geq 0 \;. \tag{1.10}$$

A nonstationary neural network covariance function can be constructed as a limit of a particular type of neural network with an infinite number of hidden units [Williams, 1998].

There are many other examples of covariance functions, both stationary and non-stationary. It is also possible to combine covariances in sums, products and convolutions to obtain more flexible and complex processes. See [Rasmussen and Williams, 2006] for further details.

1.5 Gaussian process regression

So far we have defined a Gaussian process, and we have seen how the choice of covariance function leads to different typical sample functions from the GP. Within a class of covariance function, a small number of hyperparameters control global properties such as lengthscale and amplitude. Our task is to use GPs to do regression. To do this we use the GP to express our prior belief about the underlying function we are modelling. We define a noise model that links the observed data to the function, and then regression is simply a matter of Bayesian inference.

We start with the zero mean GP prior on the function variables:

$$p(\mathbf{f}) = \mathcal{N}(\mathbf{0}, \mathbf{K}) \; . \tag{1.11}$$

We assume that the observed data y is generated with Gaussian white noise around the underlying function f:

$$y = f + \epsilon \; , \qquad \mathcal{E}[\epsilon(\mathbf{x})\epsilon(\mathbf{x}')] = \sigma^2 \delta_{\mathbf{x}\mathbf{x}'} \; , \tag{1.12}$$

where σ^2 is the variance of the noise, and δ is the Kronecker delta.[2] Equivalently, the noise model, or *likelihood* is:

$$p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{f}, \sigma^2 \mathbf{I}) \; , \tag{1.13}$$

where $\mathbf{I}$ is the identity matrix. Integrating over the unobserved function variables $\mathbf{f}$ gives the *marginal likelihood* (or evidence):

$$\begin{aligned} p(\mathbf{y}) &= \int \mathrm{d}\mathbf{f} \, p(\mathbf{y}|\mathbf{f}) p(\mathbf{f}) \\ &= \mathcal{N}(\mathbf{0}, \mathbf{K} + \sigma^2 \mathbf{I}) \; . \end{aligned} \tag{1.14}$$

1.5.1 Prediction

For this section refer to section A.2 and section A.3 for a more detailed explanation of data point and covariance matrix notation used. Suppose we have our N training input and output pairs $(\mathbf{X}, \mathbf{y})$, together with a set of T input locations $\mathbf{X}_T$

[2] We have used a slight abuse of notation when using the Kronecker delta with continuous indices $\mathbf{x}$ and $\mathbf{x}'$. We mean that $\delta = 1$ only when $\mathbf{x}$ and $\mathbf{x}'$ refer to the same data point. Otherwise $\delta = 0$, even if we happened to sample two data points at exactly the same $\mathbf{x}$ location. This reflects the fact that the noise is independent for each data point.

at which we want to make predictions — the test inputs. These variables are all observed. We also have some sets of unobserved variables: the latent function variables at both training and test locations, and the test outputs themselves. We refer collectively to groups of these points as: $(\mathbf{X}, \mathbf{f}, \mathbf{y}) = (\{\mathbf{x}_n\}, \{f_n\}, \{y_n\})_{n=1}^N$, and $(\mathbf{X}_T, \mathbf{f}_T, \mathbf{y}_T) = (\{\mathbf{x}_t\}, \{f_t\}, \{y_t\})_{t=1}^T$. We first note the joint training and test prior:

$$p(\mathbf{f}, \mathbf{f}_T) = \mathcal{N}(\mathbf{0}, \mathbf{K}_{N+T}) \,, \tag{1.15a}$$

$$\mathbf{K}_{N+T} = \begin{bmatrix} \mathbf{K}_N & \mathbf{K}_{NT} \\ \mathbf{K}_{TN} & \mathbf{K}_T \end{bmatrix} \,. \tag{1.15b}$$

Inclusion of the noise model gives the joint distribution on training and test outputs (from equation (1.14)):

$$p(\mathbf{y}, \mathbf{y}_T) = \mathcal{N}(\mathbf{0}, \mathbf{K}_{N+T} + \sigma^2 \mathbf{I}) \,. \tag{1.16}$$

The predictive distribution is obtained by conditioning on the observed training outputs (and using the inverse of a partitioned matrix identity of section B.1.2):

$$p(\mathbf{y}_T|\mathbf{y}) = \mathcal{N}(\boldsymbol{\mu}_T, \boldsymbol{\Sigma}_T) \,, \tag{1.17a}$$

$$\begin{aligned} \boldsymbol{\mu}_T &= \mathbf{K}_{TN}[\mathbf{K}_N + \sigma^2 \mathbf{I}]^{-1} \mathbf{y} \\ \boldsymbol{\Sigma}_T &= \mathbf{K}_T - \mathbf{K}_{TN}[\mathbf{K}_N + \sigma^2 \mathbf{I}]^{-1} \mathbf{K}_{NT} + \sigma^2 \mathbf{I} \,. \end{aligned} \tag{1.17b}$$

Notice how the predictive mean and variance $(\boldsymbol{\mu}_T, \boldsymbol{\Sigma}_T)$ in equation (1.17b) are constructed from the blocks of the GP prior covariance $\mathbf{K}_{N+T}$ of equation (1.15b). This construction will be useful to remember later on when forming the predictive distribution for various GP approximations.

It is important to appreciate that equation (1.17) is a correlated prediction. As well as giving you the mean and marginal variance at each test point, it tells you the predictive correlations between any pair of test outputs. In fact, the GP predictive distribution is actually a full Gaussian *process* in its own right, with a particular non-zero mean function and covariance function:[3]

$$\begin{aligned} \mu(\mathbf{x}) &= \mathbf{K}_{\mathbf{x}N}[\mathbf{K}_N + \sigma^2 \mathbf{I}]^{-1} \mathbf{y} \\ \Sigma(\mathbf{x}, \mathbf{x}') &= K(\mathbf{x}, \mathbf{x}') - \mathbf{K}_{\mathbf{x}N}[\mathbf{K}_N + \sigma^2 \mathbf{I}]^{-1} \mathbf{K}_{N\mathbf{x}'} \,. \end{aligned} \tag{1.18}$$

Refer to section A.3 to explain the slight abuse of covariance notation in equa-

[3] Actually, as written here without the noise, this is the *posterior process* on f rather than y: $p(f(\mathbf{x})|\mathbf{y})$.

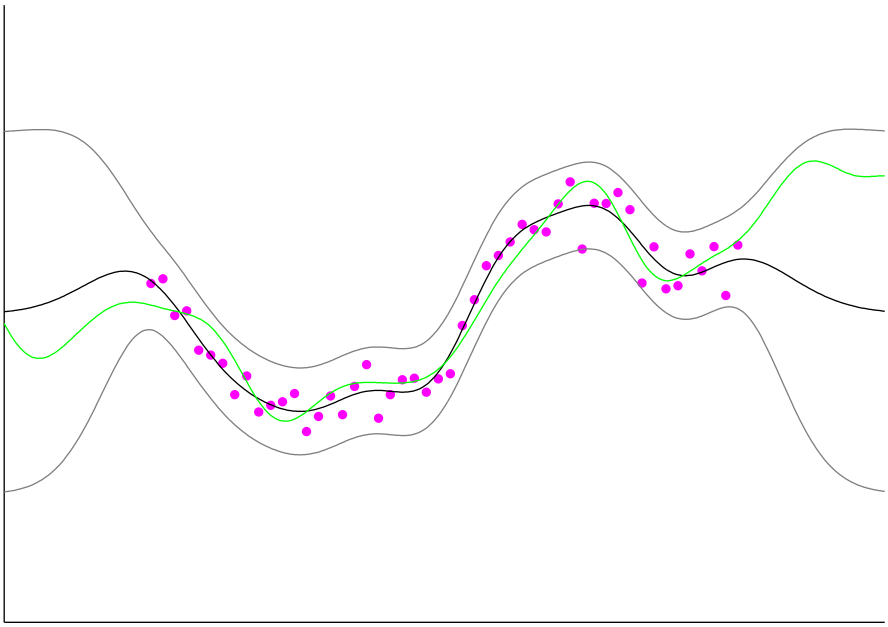


Figure 1.4: The GP predictive distribution. The single test point predictive distribution (1.19) is shown as mean and two standard deviation lines in black and grey respectively. A sample function from the full posterior process (1.18) is shown in green.

tion (1.18).

Whilst these predictive correlations are potentially useful for some applications, it is often just the marginal variances ($\mathrm{diag}\,\boldsymbol{\Sigma}_T$) that are computed and used as measures of predictive uncertainty. In this case it is sufficient to consider a single general test input $\mathbf{x}_*$, at which the predictive mean and variance is:

$$\begin{aligned} \mu_* &= \mathbf{K}_{*N}[\mathbf{K}_N + \sigma^2\mathbf{I}]^{-1}\mathbf{y} \\ \sigma_*^2 &= K_* - \mathbf{K}_{*N}[\mathbf{K}_N + \sigma^2\mathbf{I}]^{-1}\mathbf{K}_{N*} + \sigma^2 \,. \end{aligned} \tag{1.19}$$

Computing the inverse of $\mathbf{K}_N + \sigma^2\mathbf{I}$ costs $\mathcal{O}(N^3)$. We see that the mean prediction is simply a weighted sum of N basis functions: $\mu_* = \mathbf{K}_{*N}\boldsymbol{\alpha}$, where $\boldsymbol{\alpha} = [\mathbf{K}_N + \sigma^2\mathbf{I}]^{-1}\mathbf{y}$. Therefore, if we precompute $\boldsymbol{\alpha}$ the mean prediction per test case costs only $\mathcal{O}(N)$. For the variance no such precomputation can be done, and the cost is $\mathcal{O}(N^2)$ per test case. The most stable and efficient method of implementation of equation (1.19) is via Cholesky decomposition of the covariance matrix (see section B.2 for details).

Figure 1.4 shows a plot of the GP predictive distribution for a simple 1D data set

using SE covariance. The hyperparameters are assumed known and given. Mean and two standard deviation lines (equation (1.19)) are plotted in black and grey, and this will be the standard way of showing GP predictions throughout the thesis. Note that this prediction includes the noise variance ($p(y_*|\mathbf{y})$). A sample function is also plotted from the posterior process (equation (1.18)) in green. This is noise free ($p(f(\mathbf{x})|\mathbf{y})$). Notice how the sample function varies around the mean function, and we can imagine how averaging many such sample functions would give rise to the mean function. The plot shows how the predictive variance grows in regions away from training data until it reaches $K(\mathbf{x},\mathbf{x})+\sigma^2$ (in the case of a stationary covariance such as SE, $K(\mathbf{x},\mathbf{x})$ is a constant, a^2). It is evident by looking at equation (1.19) how the local nature of the covariance function causes this to happen. The property of increasing predictive uncertainty away from training data is an intuitive and desirable property of GPs.

1.5.2 Hyperparameter learning

One of the major advantages of GPs over other methods is the ability to select covariance hyperparameters from the training data directly, rather than use a scheme such as cross validation. The reason we can do this is because the GP is a full probabilistic model. Ideally we would like to place a prior and compute a Bayesian posterior $p(\boldsymbol{\theta}|\mathbf{y})$ on hyperparameters. However this is not analytically tractable in general. Instead we can use the marginal likelihood (equation (1.14)) as an appropriate cost function. More specifically we minimise the negative $\log$ marginal likelihood $\mathcal{L}$ with respect to the hyperparameters of the covariance $\boldsymbol{\theta}$ (including the noise variance σ^2):

$$\begin{aligned}\mathcal{L} &= -\log p(\mathbf{y}|\boldsymbol{\theta}) \\ &= \frac{1}{2}\log\det\mathbf{C}(\boldsymbol{\theta}) + \frac{1}{2}\mathbf{y}^\top\mathbf{C}^{-1}(\boldsymbol{\theta})\mathbf{y} + \frac{N}{2}\log(2\pi)\ ,\end{aligned} \tag{1.20}$$

where $\mathbf{C} = \mathbf{K}_N + \sigma^2\mathbf{I}$.

As with any other procedure where we are optimising parameters using the training data we have to worry about overfitting. As we have seen in section 1.4, covariance functions tend to have a small number of hyperparameters controlling broad aspects of the fitted function, and therefore overfitting tends not to be a problem. Secondly, we are not optimising the function variables $\mathbf{f}$ themselves, but rather integrating over their uncertainty. The GP hyperparameter optimisation takes place at a higher hierarchical level, and hence is sometimes referred to as type II maximum

likelihood.

The minimisation of the negative log marginal likelihood of equation (1.20) is a non-convex optimisation task. However gradients are easily obtained and therefore standard gradient optimisers can be used, such as conjugate gradient (CG) techniques or quasi-Newton methods. To compute the derivatives a few matrix identities from section B.1.3 are useful, and for implementation section B.2 is again relevant. The precise details will of course depend on the choice of covariance function. The cost of computing the log marginal likelihood and gradients is again dominated by the inversion of the covariance matrix $\mathbf{C}$.

Local minima can be a problem, particularly when there is a small amount of data, and hence the solution ambiguous. In this situation local minima can correspond to alternative credible explanations for the data (such as low noise level and short lengthscale vs. high noise level and long lengthscale). For this reason it is often worth making several optimisations from random starting points and investigating the different minima.

1.5.3 Automatic relevance determination

One can make an anisotropic (but still stationary) version of the SE covariance (section 1.4.1) by allowing an independent lengthscale hyperparameter λ_d for each input dimension:

$$K_{\text{SE-ARD}}(\mathbf{x}, \mathbf{x}') = a^2 \exp\left[-\frac{1}{2}\sum_{d=1}^{D}\left(\frac{x_d - x'_d}{\lambda_d}\right)^2\right] . \tag{1.21}$$

When combined with the techniques of section 1.5.2 for determining hyperparameters, this covariance is said to implement *automatic relevance determination* [Neal, 1996]. If a particular input dimension d has no relevance for the regression, then the appropriate lengthscale λ_d will increase to essentially filter that feature out. This is because the evidence will suggest that the underlying function is very slowly varying in the direction of that feature. This means that alternative ad-hoc feature selection methods are not necessary.

We will use this ARD version of the SE covariance throughout the thesis to illustrate our examples and in experiments. However most of the methods we develop apply generally with any choice of covariance function, such as those discussed in section 1.4.

1.6 Relation to Bayesian linear regression

In the first part of this chapter we have described GPs as priors directly on the space of functions, specified with a suitable covariance function. This viewpoint is sometimes referred to as the *function space viewpoint*, and is arguably the most intuitive way to understand GPs. However, there is an alternative way of viewing GPs called the *weight space viewpoint*, which is closely related to the well known Bayesian generalised linear regression model.

Suppose we define our function to be a weighted sum of a set of M basis functions:

$$f(\mathbf{x}) = \sum_{m=1}^{M} w_m \phi_m(\mathbf{x}) = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}) \ . \tag{1.22}$$

If we now place a Gaussian prior on the weights:

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_w) \ , \tag{1.23}$$

then we have equivalently defined a Gaussian process prior on the function f with covariance function:

$$\begin{aligned} K(\mathbf{x}, \mathbf{x}') &= \mathcal{E}[f(\mathbf{x}) f(\mathbf{x}')] \\ &= \boldsymbol{\phi}^\top(\mathbf{x}) \boldsymbol{\Sigma}_w \boldsymbol{\phi}(\mathbf{x}') \ . \end{aligned} \tag{1.24}$$

This type of GP is called *degenerate* because it can always be converted back to the weight space viewpoint and represented with a finite set of basis functions. The covariance function (1.24) is of rank M, meaning that any covariance matrix formed from this covariance function will have maximum rank M. The linear covariance function mentioned in section 1.4.3 is exactly of this type.

As we have seen, the function space viewpoint allows us to specify a covariance function directly, rather than in terms of basis functions. However Mercer's theorem [e.g. Schölkopf and Smola, 2002] tells us that we can still decompose the covariance function in terms of its eigenfunctions and eigenvalues, theoretically at least:

$$K(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{\infty} \lambda_i \psi_i(\mathbf{x}) \psi_i(\mathbf{x}') \ . \tag{1.25}$$

In this way we convert back to the weight space viewpoint, in the sense that we now have a description in terms of basis functions like in equation (1.24). If the sum of equation (1.25) has a finite number of elements, then we have a degenerate GP,

but for general covariance functions (e.g. SE) the sum will have an infinite number of elements with possibly no analytical expressions for the eigenfunctions, and the GP is said to be non-degenerate. It is this type of GP that we are most interested in, since they are nonparametric flexible models for functions.

From a practical computational point of view, let us suppose we have a degenerate GP with M basis functions, and N training data points. We know that to make predictions we must invert the $N \times N$ covariance matrix $\mathbf{K}_N + \sigma^2\mathbf{I}$. If $N < M$ then it is most efficient to invert this directly at a cost of $\mathcal{O}(N^3)$.[4] However, if $N > M$, it is best to do the regression in the weight space viewpoint. The covariance matrix formed from equation (1.24) can be written $\mathbf{K}_N = \mathbf{V}\mathbf{V}^\top$ where $\mathbf{V}$ is the $N \times M$ matrix $\mathbf{\Phi}\mathbf{\Sigma}_w^{\frac{1}{2}}$, and the matrix inversion lemma (section B.1.1) provides the link:

$$(\mathbf{V}\mathbf{V}^\top + \sigma^2\mathbf{I}_N)^{-1} = \sigma^{-2}\mathbf{I}_N - \sigma^{-2}\mathbf{V}(\sigma^2\mathbf{I}_M + \mathbf{V}^\top\mathbf{V})^{-1}\mathbf{V}^\top . \tag{1.26}$$

Note that the RHS of equation (1.26) only involves inverting an $M \times M$ matrix rather than $N \times N$. Any evaluation needed for prediction is then dominated by the cost of constructing $\mathbf{V}^\top\mathbf{V}$ which is $\mathcal{O}(NM^2)$. Put simply, if $N > M$ linear regression should be solved in the traditional way, not via the GP formulation.

Of course, for the non-degenerate case that we are most interested in we do not have a choice of practical solution method. If we tried to convert to the weight space viewpoint then we would be trying to invert an infinite dimensional matrix! In fact in this way we see the power of GPs (and kernel methods in general)[5] — we are effectively utilising an infinite set of basis functions but with finite computation. This type of model is called a *nonparametric* model because the complexity of the solution is increasing as we receive more data (we can see this by looking for example at the GP predictive mean in equation (1.18) which is a weighted sum of N kernel evaluations). This nonparametric property is desirable because it means that we are not limited in function complexity as we receive more data. However the price we pay is a cubic scaling in N rather than linear if we were to stick to a fixed finite set of basis functions.

[4] Construction of the covariance matrix would be more expensive at $\mathcal{O}(MN^2)$.

[5] The GP is a particular type of probabilistic kernel method. The GP covariance matrix can also be called a kernel matrix. The general GP can be regarded as 'kernelised' Bayesian linear regression, by the process described in this section. See [e.g. Schölkopf and Smola, 2002] for more details on kernel methods.

1.7 Gaussian process classification

In this thesis we focus on Gaussian process regression, but GPs have been applied successfully to classification too, and for completeness we briefly discuss this here. In a classification task we have discrete outputs, e.g. binary $y = \pm 1$, rather than continuous. A GP classification model uses a Gaussian process prior $p(f)$ for an underlying function $f(\mathbf{x})$, just as for regression. However, instead of the Gaussian regression noise model equation (1.13), a discrete distribution is obtained by squashing the function f through e.g. a sigmoid:

$$p(y = +1|f(\mathbf{x})) = \sigma(f(\mathbf{x})) \; . \tag{1.27}$$

The non-Gaussian likelihood of equation (1.27) causes a problem: we cannot integrate out the unknown function variables $\mathbf{f}$ analytically as we did to compute the marginal likelihood in equation (1.14). The usual method for dealing with this problem is to make a Gaussian approximation to the posterior $p(\mathbf{f}|\mathbf{y})$. For example, Williams and Barber [1998] use the Laplace approximation that approximates $p(\mathbf{f}|\mathbf{y})$ around its mode. Kuss and Rasmussen [2005] show empirically that the EP approximation [Minka, 2001] is more accurate. The EP approximation is an iterative algorithm that approximates each data point likelihood term $p(y_n|f_n)$ by a scaled Gaussian in f, thereby giving an overall Gaussian approximation to $p(\mathbf{f}|\mathbf{y})$. Other approaches include the variational methods of Gibbs and MacKay [2000] and Girolami and Rogers [2006].

The need for approximation is not specific to classification, and applies equally to any non-Gaussian likelihood, e.g. a heavy tailed regression noise model.

Chapter 2

Computationally efficient Gaussian processes

One of the fundamental problems we try to address in this thesis is the computational difficulties of applying Gaussian processes to large data sets. This is a problem particularly apparent in the machine learning field where large data sets are commonplace. The full procedure of GP regression consists of three stages, which we will call hyperparameter training, precomputation, and testing. Hyperparameter training is usually performed by the optimisation of the marginal likelihood as outlined in section 1.5.2. Each gradient computation requires inversion of the covariance matrix $\mathbf{K}_N + \sigma^2\mathbf{I}$, and hence cost will be $\mathcal{O}(N^3\times$ number of gradient evaluations in the optimisation$)$.

We define the precomputation stage to include any computations that can be done given a set of fixed hyperparameters but before seeing any test data. The $\mathcal{O}(N^3)$ inversion of the covariance matrix is again the dominant part of this stage. If the hyperparameter learning has been done as described above, then these two stages may as well be amalgamated and called 'training'. However it is sometimes useful to make computational comparisons where the hyperparameters have perhaps been obtained in a different way, or are even given a priori. In this case then the precomputation stage alone is the relevant one. In fact some authors, particularly from the kernel machines literature [e.g. Smola and Bartlett, 2001], call this stage training.

Finally we have the testing stage, where we make predictions at a set of T test points. As explained in section 1.5.1, after the precomputation stage, each predic-

tion costs $\mathcal{O}(N^2)$ leading to an overall test cost of $\mathcal{O}(N^2 T)$.

For a large training set both the training and test times become prohibitively expensive due to the poor scaling with N. As we saw in section 1.6 the reason for the poor scaling is the desirable nonparametric nature of the GP itself. Therefore we seek methods that overcome the computational difficulties whilst still preserving the desirable properties of the GP.

The cost time of relevance depends very much on the particular application. The user may be concerned with the total training and test time. Alternatively the user may be able to afford a very large offline training time, but requires fast online predictions. As we shall see different methods will be suitable for these different requirements.

2.1 Summary of previous approaches

There has been a lot of work done over the past two decades that attempts to address the computational problem, by developing efficient approximations to the full Gaussian process. Although the approaches have been developed from different angles, many are closely related to each other. We will examine some of the approximations in much more detail later, but here we give a high level overview.

2.1.1 Subset of data (SD)

The most obvious and simplest technique to reduce computation is to ignore a large part of the training data! A GP prediction is made based only on a subset of size M of the training data. We call this technique *subset of data* (SD). Whilst it seems too obvious and simple to be worthy of much mention, it is after all the benchmark against which other techniques must be compared. In fact, in some cases it may be the best one can do, because it has no extra overhead compared to some of the more sophisticated approximations, both in terms of computation and memory. For highly redundant data sets, where extra data points bring very little extra information about the function, then there is no point wasting computation on a sophisticated approximation for little gain in performance.

SD also serves as a prototype for most of the other techniques as it highlights the different options that are available. The reason for this is that most of the more sophisticated approximations are also based on a subset of the training data just

like SD; the difference being that these approximations also include some information from the rest of the training data. We can therefore use SD to outline three important steps relevant to all the GP approximations:

Subset selection

When using SD we clearly have a choice of selecting the subset of M data points from the total N training points. The simplest scheme is to choose randomly, and due to its negligible overhead it is often very effective. However more sophisticated schemes can be used based on various information criteria that score how much one learns about the function by including a point into the subset. The *informative vector machine* [Lawrence et al., 2003] is an example of the SD method using a differential entropy score for subset selection.[1]

As explained above, for more sophisticated approximations that we will discuss the same choice exists. In this context the subset is variously known as the active set, the support set, or the inducing set. Again we may choose randomly, or use other information based selection criteria.

Hyperparameter selection

For SD we will generally obtain hyperparameters by maximising the SD marginal likelihood, or in other words the GP marginal likelihood on the subset:

$$p_{\text{SD}}(\mathbf{y}_M) = \mathcal{N}(\mathbf{0}, \mathbf{K}_M + \sigma^2\mathbf{I}) \; . \tag{2.1}$$

We could however choose to use more sophisticated GP marginal likelihood approximations, that take into account all of the training data.

Prediction

Having obtained hyperparameters and made a choice of subset, we can choose an approximation to make predictions. For SD this is simply the GP predictive

[1] Although not a GP model, an SVM can be similarly seen as a method for selecting a subset of data on which a predictor is based.

distribution based solely on the subset (from equation (1.19)):

$$\begin{aligned} \mu_*^{\text{SD}} &= \mathbf{K}_{*M}[\mathbf{K}_M + \sigma^2\mathbf{I}]^{-1}\mathbf{y}_M \\ (\sigma_*^2)^{\text{SD}} &= K_* - \mathbf{K}_{*M}[\mathbf{K}_M + \sigma^2\mathbf{I}]^{-1}\mathbf{K}_{M*} + \sigma^2 \ . \end{aligned} \tag{2.2}$$

We can see that even for something as trivial as SD, various different strategies exist to balance predictive performance against computation. Which strategy will be best depends a lot on the requirements of the user in terms of training versus testing time as discussed at the beginning of section 2.1. For example, a more sophisticated subset selection method could be chosen at the expense of greater training time but improved test performance for given test time. We do not even have to choose the same subset for hyperparameter learning as for prediction. To save on training time a smaller subset could be used to determine hyperparameters, with extra points being included for the prediction stage. If using a subset selection method, we also have the tricky issue of whether and how to interleave hyperparameter learning with the subset selection until some sort of convergence.

The reason we exhaustively list such options for SD is that all of the above applies to more sophisticated approximations too. In practice the best choices depend very much on the particular data and priorities of the user, and individual empirical trials are probably the best way to sort this out. The purpose of this thesis is not to attempt an exhaustive empirical comparison with all approximations and all strategies. Rather we will present new approximations and options, and suggest in which regimes they will be most useful.

2.1.2 Low rank approximations

At a high level, most of the more sophisticated approximations that have been developed can be understood as some kind of *reduced rank* approximation to the covariance matrix:

$$\mathbf{K}_N \approx \mathbf{V}\mathbf{V}^\top \ , \tag{2.3}$$

where $\mathbf{V}$ is an $N \times M$ matrix. We know from the discussion of the relationship between GPs and linear models in section 1.6 that this is equivalent to approximating the full non-degenerate GP with a finite (M dimensional) degenerate linear model. From this discussion, and via the matrix inversion lemma equation (B.1), we also know how the computational saving is made. The training cost is reduced from $\mathcal{O}(N^3)$ computation to $\mathcal{O}(NM^2)$. Similarly prediction time is reduced to $\mathcal{O}(M^2)$

per test point after precomputation.

The success of such a low rank type approximation depends on the fact that for typical covariances (e.g. SE, section 1.4.1), the eigenvalue spectrum decays fairly rapidly (reflecting the smoothness of the prior). One might think of trying to directly find a low rank approximation to $\mathbf{K}_N$ by doing an eigendecomposition and keeping the M leading eigenvalues. However this does not really help us because the eigendecomposition itself is $\mathcal{O}(N^3)$ in general. Instead a subset of the input points (just like in SD) is used as a basis for a low rank construction, with these M points determining the regions in input space where the approximation is good. One particular construction which forms the heart of many of these GP approximation methods is the *Nyström* construction:

$$\mathbf{K}_N \approx \mathbf{K}_{NM}\mathbf{K}_M^{-1}\mathbf{K}_{MN} \ , \tag{2.4}$$

where $\mathbf{K}_{NM}$ is the covariance between the training points and the subset (see section A.3). This construction can be derived from various different angles [Smola and Schölkopf, 2000, Williams and Seeger, 2001], but one way of understanding it is as approximating the eigendecomposition discussed above. In fact we will use this construction so much that we use a separate symbol for this low rank covariance:

$$Q(\mathbf{x}, \mathbf{x}') = \mathbf{K}_{\mathbf{x}M}\mathbf{K}_M^{-1}\mathbf{K}_{M\mathbf{x}'} \ . \tag{2.5}$$

Here we have used a small abuse of notation (see section A.3) to refer to the covariance *function* $Q(\mathbf{x}, \mathbf{x}')$ rather than the covariance matrix of equation (2.4).

The *subset of regressors* (SR) approximation consists of the degenerate GP using Q as its covariance function. The fact that this approximation can be derived in various ways, and that there exist different strategies for choosing the subset, means that it appears in various guises in the literature [e.g. Wahba, 1990, Poggio and Girosi, 1990]. Smola and Bartlett [2001] use SR in conjunction with a greedy forward selection method to choose the subset.

From the discussion of linear models in section 1.6 it should be clear that there is something unsatisfactory about approximating the full non-degenerate GP with a degenerate one as SR does. We lose some of the power of full nonparametric modelling, where the complexity of the function grows with more data. See [Quiñonero Candela, 2004] for more details. As we shall see in section 2.3.2, this is a big problem for SR because it gives rise to far too small predictive variances in regions away from subset points.

The *projected process* (PP) approximation is a fix to this problem: whilst based on the same low rank Nyström construction of equation (2.4), it is not strictly a degenerate process. It does still however have some technical problems which we discuss in section 2.3.8. Seeger et al. [2003] use PP with a greedy forward selection method based on information gain, and Keerthi and Chu [2006] use PP with a matching pursuit algorithm for selecting the subset.

A general term for these type of approximations is *sparse* Gaussian processes. This is slightly misleading, as the matrices involved are not sparse but low rank. However, it does convey the idea that the approximations are based on a small subset of the training data points.

2.1.3 Other approaches

The *sparse online GP* methods of Csató and Opper [2002], Csató [2002] are closely related methods to those of section 2.1.2, which process data sequentially, and are designed to deal with non-Gaussian likelihoods such as for classification. We will discuss the relationships in more detail in section 2.3.7, but we note that the approximation presented in [Csató, 2002], when applied to regression, is a sequential version of PP.

The *Bayesian committee machine* (BCM) [Tresp, 2000] is another approach to speed up GP regression based on partitioning the data set. It is related to the low rank approximations of section 2.1.2, but has a somewhat different flavour. It is *transductive* rather than inductive, meaning that the approximation used is dependent on the test inputs, therefore requiring one to know the test inputs in advance of training/precomputation. The principal problem with the method is a high test cost of $\mathcal{O}(NM)$ per test case, rather than the $\mathcal{O}(M^2)$ of the approaches of section 2.1.2. However the BCM can be modified to be used in an inductive way, and we will discuss this further in chapter 3.

Gibbs [1997] used an approach based on early stopping in an iterative conjugate gradient (CG) solution to the equation $(\mathbf{K}_N + \sigma^2\mathbf{I})\mathbf{a} = \mathbf{y}$. The problem with this is that it still scales quadratically in N. Recently Yang et al. [2005] have proposed the *improved fast Gauss transform* (IFGT) to speed up these CG iterations to give a linear scaling with N. Other recent methods that also rely on speeding up kernel matrix-vector products within CG include [Gray, 2004, Shen et al., 2006, de Freitas et al., 2006]. This type of method shows some promise, but at present it is not feasible for more than a few input dimensions, and the speedups seem very sensitive to par-

ticular lengthscale regimes. Also hyperparameter optimisation is complicated and predictive variances expensive to obtain, when using CG iteration based methods.

2.2 The sparse pseudo-input Gaussian process (SPGP)

It should be clear from section 2.1 that for the purpose of analysis it makes sense to distinguish between the actual approximation and the other aspects of strategy that go with it, such as determination of the subset. This is because often we can mix and match strategies with approximations depending on what suits. However in the literature this has not been done in general, with individual papers tending to recommend both a particular approximation and a particular technique for subset selection. This has made the literature somewhat confusing to read as a whole. In [Snelson and Ghahramani, 2006a] we contributed to this confusion by doing exactly the same! We suggested a method that consisted both of a particular approximation and a new way of obtaining a subset. These two parts of the SPGP can be considered independently for the most part. However there is a lot of intuition to be gained by considering the original derivation of the SPGP as a whole as presented in [Snelson and Ghahramani, 2006a]. In this chapter we will therefore first present the SPGP as a complete model, as it was originally derived. We will then analyse and compare the purely technical aspects of the approximation itself with others such as SR and PP, independently of the subset selection. To do this we will make use of the framework suggested by Quiñonero Candela and Rasmussen [2005] for comparing the theoretical aspects of the different approximations.

In order to derive a model that is computationally tractable for large data sets, but which still preserves the desirable properties of the full GP, we examine in detail the single test point GP predictive distribution equation (1.19):

$$\mu_* = \mathbf{K}_{*N}[\mathbf{K}_N + \sigma^2\mathbf{I}]^{-1}\mathbf{y}$$
$$\sigma_*^2 = K_* - \mathbf{K}_{*N}[\mathbf{K}_N + \sigma^2\mathbf{I}]^{-1}\mathbf{K}_{N*} + \sigma^2 \ .$$

Consider the mean and variance of this distribution as functions of $\mathbf{x}_*$, the new input. Regarding the hyperparameters as known and fixed for now, these functions are effectively parameterised by the locations of the N training input and output pairs, $\mathbf{X}$ and $\mathbf{y}$. The intuition behind the SPGP is that we can replace the real data set $\mathcal{D}$ by a *pseudo data set* $\bar{\mathcal{D}}$ of parameters, and use the GP predictive distribution from this pseudo data set as a parameterised model likelihood. The computational

efficiency of the model will arise because we will use a pseudo data set of size $M \ll N$: pseudo-inputs $\bar{\mathbf{X}} = \{\bar{\mathbf{x}}_m\}_{m=1}^M$ and pseudo-outputs $\bar{\mathbf{f}} = \{\bar{f}_m\}_{m=1}^M$. We have denoted the pseudo-outputs $\bar{\mathbf{f}}$ instead of $\bar{\mathbf{y}}$ because as they are not real observations, it does not make sense to include a noise variance for them. They are therefore equivalent to the latent function variables $\mathbf{f}$. The bar notation just serves to emphasise that these variables are not observed data, but nevertheless live in the same spaces as their equivalents $\mathbf{f}$ and $\mathbf{X}$. The actual observed output value will of course be assumed noisy as before. These assumptions therefore lead to the following single data point likelihood:

$$p(y|\mathbf{x}, \bar{\mathbf{X}}, \bar{\mathbf{f}}) = \mathcal{N}\left(\mathbf{K}_{\mathbf{x}M}\mathbf{K}_M^{-1}\bar{\mathbf{f}},\ K_{\mathbf{xx}} - \mathbf{K}_{\mathbf{x}M}\mathbf{K}_M^{-1}\mathbf{K}_{M\mathbf{x}} + \sigma^2\right) , \tag{2.6}$$

where $\mathbf{K}_{\mathbf{x}M}$ is the covariance between the input $\mathbf{x}$ and the pseudo-inputs $\bar{\mathbf{X}}$, and $\mathbf{K}_M$ is the self covariance of the pseudo-inputs (see section A.3 for details on covariance notation). Notice this likelihood is just the GP predictive distribution equation (1.19) with the real data set replaced by the pseudo-data set and with no noise on the pseudo-outputs.

This can be viewed as a standard regression model with a particular form of parameterised mean function and input-dependent noise model. We assume the output data are generated i.i.d. given the inputs, giving the complete data likelihood:

$$\begin{aligned} p(\mathbf{y}|\bar{\mathbf{f}}) &= \prod_{n=1}^N p(y_n|\bar{\mathbf{f}}) \\ &= \mathcal{N}\left(\mathbf{K}_{NM}\mathbf{K}_M^{-1}\bar{\mathbf{f}},\ \mathrm{diag}(\mathbf{K}_N - \mathbf{Q}_N) + \sigma^2\mathbf{I}\right) , \end{aligned} \tag{2.7}$$

where we have dropped the explicit conditioning on the relevant inputs from the notation. We also see the first instance of the low rank covariance $\mathbf{Q}_N \equiv \mathbf{K}_{NM}\mathbf{K}_M^{-1}\mathbf{K}_{MN}$, discussed in section 2.1.2, arising naturally in the derivation of the SPGP.

Learning in the model involves finding a suitable setting of the parameters — in this case an appropriate pseudo-data set that explains the real data well. However rather than simply maximise the likelihood with respect to $\bar{\mathbf{X}}$ and $\bar{\mathbf{f}}$ it turns out that we can integrate out the pseudo-outputs $\bar{\mathbf{f}}$. We place a Gaussian prior on these pseudo outputs:

$$p(\bar{\mathbf{f}}) = \mathcal{N}(\mathbf{0}, \mathbf{K}_M) . \tag{2.8}$$

This is a very reasonable prior because we expect the pseudo-data to be distributed in a very similar manner to the real data, if they are to model them well. However, we could consider different priors here to obtain more complex models. For exam-

ple, we could use a different kernel or different hyperparameters. It is not easy to place a prior on the pseudo-inputs and still remain with a tractable model, so we will find these by maximum likelihood (ML). For the moment though, consider the pseudo-inputs as known.

Now that the model is fully specified we can apply Bayesian reasoning to fill in the rest of the details. Since all the distributions involved are Gaussian this ends up being simple matrix algebra. First we find the SPGP *marginal likelihood* by integrating equations (2.7) and (2.8) over the pseudo-outputs:

$$\begin{aligned} p(\mathbf{y}) &= \int \mathrm{d}\bar{\mathbf{f}}\; p(\mathbf{y}|\bar{\mathbf{f}})p(\bar{\mathbf{f}}) \\ &= \mathcal{N}\big(\mathbf{0},\; \mathbf{Q}_N + \mathrm{diag}(\mathbf{K}_N - \mathbf{Q}_N) + \sigma^2\mathbf{I}\big)\;, \end{aligned} \tag{2.9}$$

where we have used the Gaussian identity (B.11). This should be compared to the full GP marginal likelihood of equation (1.14). We can see that the SPGP marginal likelihood can be obtained by replacing the full GP covariance matrix $\mathbf{K}_N$ by the low rank covariance $\mathbf{Q}_N$ everywhere except on the diagonal. On the diagonal the SPGP marginal likelihood covariance matches exactly.

To obtain the predictive distribution we first find the joint $p(y_*, \mathbf{y})$ and then condition on the observed outputs $\mathbf{y}$. The joint is exactly the marginal likelihood equation (2.9) extended to one new point $(\mathbf{x}_*, y_*)$. When we condition on the outputs (and use equation (B.4)), we get the SPGP predictive distribution:

$$p(y_*|\mathbf{y}) = \mathcal{N}(\mu_*, \sigma_*^2) \tag{2.10a}$$

$$\begin{aligned} \mu_* &= \mathbf{Q}_{*N}[\mathbf{Q}_N + \mathrm{diag}(\mathbf{K}_N - \mathbf{Q}_N) + \sigma^2\mathbf{I}]^{-1}\mathbf{y} \\ \sigma_*^2 &= K_* - \mathbf{Q}_{*N}[\mathbf{Q}_N + \mathrm{diag}(\mathbf{K}_N - \mathbf{Q}_N) + \sigma^2\mathbf{I}]^{-1}\mathbf{Q}_{N*} + \sigma^2\;. \end{aligned} \tag{2.10b}$$

The forms of equations (2.9) and (2.10b) are written in such a way as to make it easy to compare to the equivalent equations (1.14) and (1.19) for the full GP. However, for implementation these need converting with the matrix inversion lemma because as written they still involve the inversion of $N \times N$ matrices. The way this works is very similar to the method of converting degenerate GPs as discussed in section 1.6. The difference is that here the covariance matrix is not simply low rank, but rather low rank plus a diagonal: $\mathbf{Q}_N + \mathbf{\Lambda} + \sigma^2\mathbf{I}$, where $\mathbf{\Lambda} = \mathrm{diag}(\mathbf{K}_N - \mathbf{Q}_N)$.

The matrix inversion lemma (B.1) applied here is therefore:

$$\left[\mathbf{K}_{NM}\mathbf{K}_M^{-1}\mathbf{K}_{MN} + (\mathbf{\Lambda} + \sigma^2\mathbf{I})\right]^{-1} = (\mathbf{\Lambda} + \sigma^2\mathbf{I})^{-1} - (\mathbf{\Lambda} + \sigma^2\mathbf{I})^{-1}\mathbf{K}_{NM}\mathbf{B}^{-1}\mathbf{K}_{MN}(\mathbf{\Lambda} + \sigma^2\mathbf{I})^{-1} , \quad (2.11)$$

where $\mathbf{B} = \mathbf{K}_M + \mathbf{K}_{MN}(\mathbf{\Lambda} + \sigma^2\mathbf{I})^{-1}\mathbf{K}_{NM}$. The reason that this remains computationally feasible is that the inversion of $\mathbf{\Lambda} + \sigma^2\mathbf{I}$ is only $\mathcal{O}(N)$ because it is diagonal. To obtain the modified form of the SPGP predictive distribution we substitute equation (2.11) into equation (2.10), and make various simplifications using the substitution $\mathbf{K}_{MN}(\mathbf{\Lambda} + \sigma^2\mathbf{I})^{-1}\mathbf{K}_{NM} = \mathbf{B} - \mathbf{K}_M$. This finally gives:

$$\begin{aligned} \mu_* &= \mathbf{K}_{*M}\mathbf{B}^{-1}\mathbf{K}_{MN}(\mathbf{\Lambda} + \sigma^2\mathbf{I})^{-1}\mathbf{y} \\ \sigma_*^2 &= K_* - \mathbf{K}_{*M}(\mathbf{K}_M^{-1} - \mathbf{B}^{-1})\mathbf{K}_{M*} + \sigma^2 . \end{aligned} \quad (2.12)$$

Computational cost is dominated by the matrix multiplication $\mathbf{K}_{MN}(\mathbf{\Lambda}+\sigma^2\mathbf{I})^{-1}\mathbf{K}_{NM}$ in the calculation of $\mathbf{B}$ which is $\mathcal{O}(NM^2)$. Just as for the full GP we observe that the mean predictor of equation (2.12) is just a weighted sum of basis functions. However, here there are only M basis functions in the sum, rather than N: $\mu_* = \mathbf{K}_{*M}\boldsymbol{\alpha}$. Comparing this mean predictor to the SD mean predictor of equation (2.2), we see that they both consist of weighted sums of M basis functions. The difference is that the SPGP weights $\boldsymbol{\alpha}$ take into account information from *all* data points $\mathbf{y}$, rather than a subset. Once precomputations have been done, the mean prediction per test case is only $\mathcal{O}(M)$, for both SD and SPGP. Similar reasoning shows that the variance predictions cost $\mathcal{O}(M^2)$ per test case. Implementation of equation (2.12) can be done using Cholesky decompositions throughout (section B.2).

2.2.1 Determination of pseudo-inputs and hyperparameters

The reason we chose the term pseudo-inputs for the support set $\bar{\mathbf{X}}$ on which the SPGP approximation is built, is to emphasise that they are to be simply viewed as parameters of the model to be learnt. This is in contrast to previous similar approaches discussed in section 2.1.2, which built approximations based on a strict subset of the data set. With this viewpoint in mind, a natural way to learn these parameters is to maximise the SPGP marginal likelihood equation (2.9), just like we do to learn hyperparameters in a GP. In fact the pseudo-inputs can be considered as extra hyperparameters, especially when we note that the SPGP is in fact equivalent

to a GP with a particular covariance function:

$$\tilde{K}^{\text{SPGP}}(\mathbf{x}, \mathbf{x}') = Q(\mathbf{x}, \mathbf{x}') + \delta_{\mathbf{x}\mathbf{x}'}[K(\mathbf{x}, \mathbf{x}) - Q(\mathbf{x}, \mathbf{x})] \,, \tag{2.13}$$

where δ is the Kronecker delta as used in equation (1.12). We could have simply posited this specially designed covariance function, and then derived the marginal likelihood equation (2.9) and the predictive distribution equation (2.10) from the usual GP equations. The SPGP covariance function $\tilde{K}^{\text{SPGP}}$ is constructed from an underlying covariance function K (quite possibly stationary), but it is heavily parameterised by the locations of the pseudo-inputs $\bar{\mathbf{X}}$ (via Q) and is itself nonstationary. It is therefore a flexible covariance function, with computational efficiency built in. The SPGP is not however simply a degenerate GP of the type discussed in section 1.6, due to the extra delta function part of the covariance of equation (2.13).

We can therefore jointly maximise the marginal likelihood with respect to both the pseudo-inputs and hyperparameters $(\bar{\mathbf{X}}, \boldsymbol{\theta})$ by gradient ascent. The details of the gradient calculations are long and tedious and are presented in appendix C. The exact form of the gradients will of course depend on the functional form of the covariance function chosen, but our method will apply to any covariance that is differentiable with respect to the input points. The derivatives with respect to *all* pseudo-inputs can be computed in $\mathcal{O}(NM^2 + NMD)$ time.

Since we now have $MD + |\boldsymbol{\theta}|$ parameters to fit, instead of just $|\boldsymbol{\theta}|$ for the full GP, one may be worried about overfitting. However, consider for now fixing the hyperparameters, and look at the case where we let $M = N$ and $\bar{\mathbf{X}} = \mathbf{X}$ — the pseudo-inputs coincide with the real training inputs. At this point the SPGP marginal likelihood is equal to that of a full GP. This is because at this point $\mathbf{K}_{MN} = \mathbf{K}_M = \mathbf{K}_N$, and therefore $\mathbf{Q}_N = \mathbf{K}_N$. Similarly the predictive distribution equation (2.10) also collapses to the full GP predictive distribution equation (1.19). In fact it is easy to show that if we then keep on adding more pseudo-inputs, i.e. let $\bar{\mathbf{X}} = \{\mathbf{X}, \mathbf{X}_+\}$ for any extra points $\mathbf{X}_+$, then still $\mathbf{Q}_N = \mathbf{K}_N$. The extra points are entirely redundant if we already have pseudo-inputs covering all training inputs. These facts help us intuit that placing a pseudo-input *anywhere* is always beneficial in terms of getting closer to the full GP solution. It never hurts to have too many pseudo-inputs, other than via a wastage of resources if they are overlapping (approximately to within a lengthscale of each other). One of the major advantages of the SPGP is that by allowing the pseudo-inputs to vary continuously away from the training data points, we try to make the best use of the limited resources when $M \ll N$, to obtain better solutions than the more restrictive strict subset based methods.

A second major advantage of the SPGP is the ability to smoothly optimise both hyperparameters and pseudo-inputs together. With other subset selection techniques, you are forced to try to interleave subset selection with gradient based hyperparameter learning. Each time the subset is updated, the optimisation landscape for hyperparameter learning is altered discontinuously. Seeger et al. [2003] discusses problems getting this technique to reliably converge to suitable hyperparameter values due to these discontinuities in the learning. The SPGP is a neat way to avoid these problems.

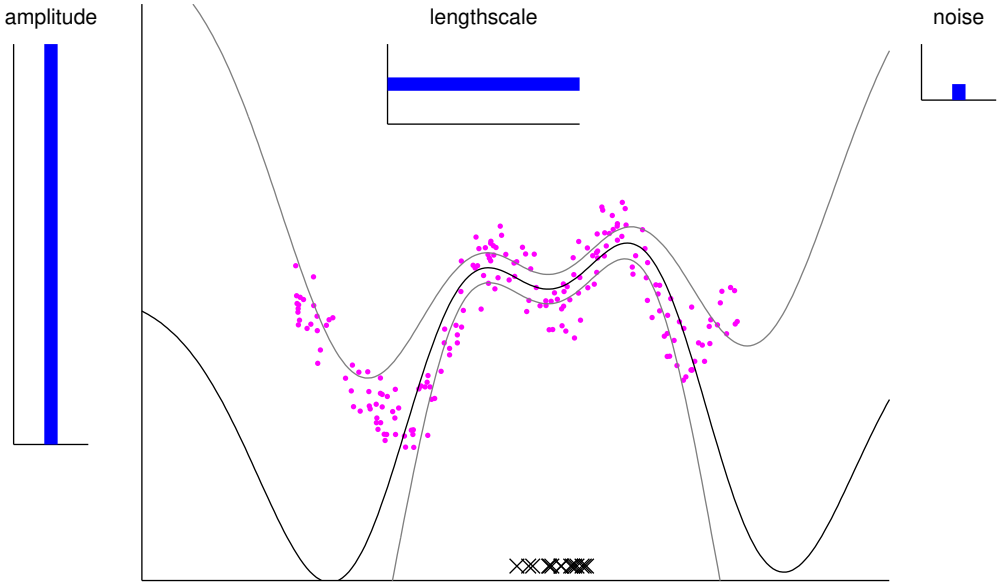
However, the interplay between pseudo-input and hyperparameter learning is a complicated one. The SPGP covariance is fundamentally more flexible than the full GP. In the discussion above we argued that you can never have too many pseudo-inputs. However, when we consider learning *both* hyperparameters and pseudo-inputs, overfitting effects can sometimes occur, due to the interplay between them. On the other hand this extra flexibility can afford advantages too, with some data sets being better modelled by the SPGP than the original full GP. We will discuss these issues further in chapter 4.

A disadvantage of this type of pseudo-input learning is that if either the number of pseudo-inputs M or input dimension D is very large then the optimisation space may be impractically big for gradient methods. We present some solutions to this problem in chapter 4. The optimisation task is of course non-convex, and so we have local optima problems. However, as we shall show by experiment, the obvious sensible initialisation of pseudo-inputs randomly to data inputs gets around this problem to a large degree.

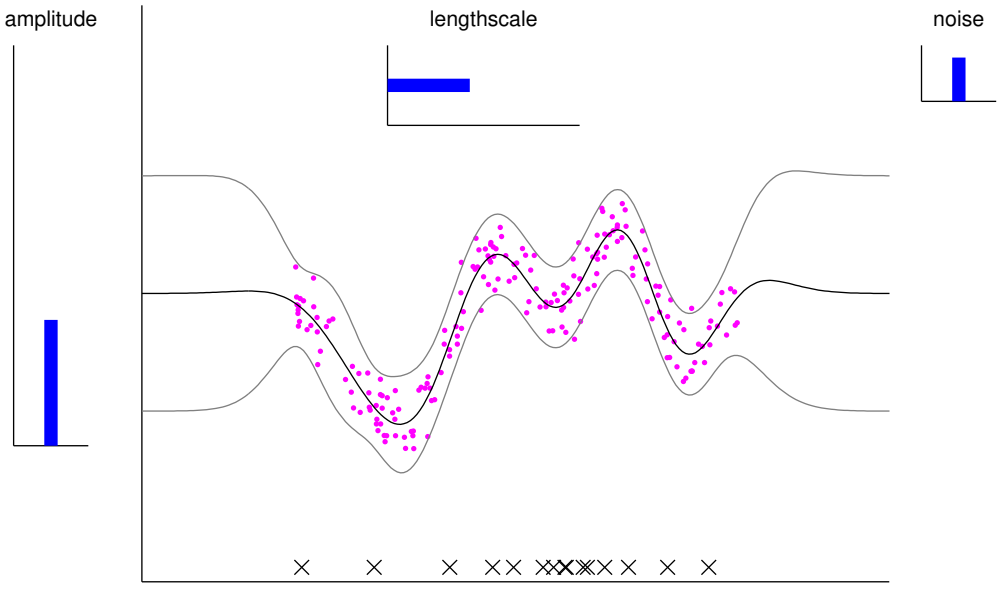
An example of pseudo-input and hyperparameter optimisation

Figure 2.1 shows the SPGP predictive distribution for a simple 1D example, both before and after pseudo-input and hyperparameter optimisation. In figure 2.1a, the pseudo-inputs have all been initialised close to the centre of the range in an adversarial position for demonstration. The pseudo-inputs determine the regions in which the SPGP predictions are good, and hence it is only in the centre of the range that the predictions are reasonable. The predictions are made poorer because the hyperparameters have also been initialised to unsuitable values: the amplitude and lengthscale are too big and the noise level is too small.

Figure 2.1b shows the SPGP prediction after joint optimisation of pseudo-inputs and hyperparameters. The hyperparameters have adjusted to suitable values, and



(a) SPGP predictions before optimisation. The pseudo-inputs are initialised adversarially towards the centre of the range.



(b) SPGP predictions after pseudo-input and hyperparameter optimisation.

Figure 2.1: SPGP predictions before and after pseudo-input and hyperparameter optimisation. The locations of pseudo-inputs are marked as back crosses; their y positions are irrelevant. The values of hyperparameters are displayed as the lengths of the blue bars.

the pseudo-inputs have spread themselves along the range of the data, so that predictions are good everywhere. The predictions are close to those of a full GP with only 15 pseudo-inputs used. The desirable characteristics of the full GP, such as the growing predictive variances away from data, are preserved by the SPGP predictive distribution.

Of course in this simple 1D example it would be very easy to spread 15 pseudo-inputs roughly evenly along the input range beforehand, with no need for any optimisation. However, the cases we are really interested in have higher dimensional input spaces where it would require far too many pseudo-inputs to tile the space densely and evenly. In this case the optimisation is a way of finding a best configuration under the limited resources.

2.2.2 Relation to RBF networks

The idea of basis functions with movable centres (like pseudo-inputs) dates back to radial basis function (RBF) networks [Moody, 1989]. Using our notation, an RBF predictor is of the form:

$$f(\mathbf{x}_*) = \sum_m K(\mathbf{x}_*, \bar{\mathbf{x}}_m)\alpha_m \ , \tag{2.14}$$

for some weights $\boldsymbol{\alpha}$ and SE kernel K. In an adaptive RBF one could move the basis function centres $\bar{\mathbf{x}}_m$ continuously perhaps by a least squares cost function. In this respect one could regard the SPGP *mean* predictor equation (2.12) as a certain type of adaptive RBF, with $\boldsymbol{\alpha} = \mathbf{B}^{-1}\mathbf{K}_{MN}(\boldsymbol{\Lambda} + \sigma^2\mathbf{I})^{-1}\mathbf{y}$.

However, the fundamental difference is that the SPGP has sensible predictive variances akin to a GP, and because it is a full probabilistic model, it has a principled method for selecting pseudo-inputs and hyperparameters. A Bayesian RBF network could be made by placing a Gaussian prior on the weights $\boldsymbol{\alpha}$. However this would lead to exactly the type of degenerate GP discussed in section 1.6. In fact with a particular Gaussian prior this would generate exactly the subset of regressors (SR) approximation discussed in section 2.1.2. Unfortunately as we see in section 2.3.2 SR does not have the type of predictive variances we want, because of this degeneracy. We could perhaps regard the SPGP as a Bayesian adaptive RBF network with sensible variances!

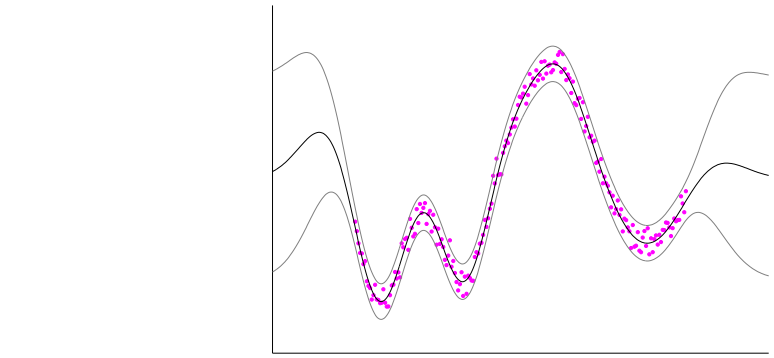
2.3 Theoretical comparison of approximations

In section 2.2 we presented a description of the SPGP, based on the intuitive idea of a pseudo-dataset parameterising a GP based model, and the subsequent learning of these parameters. However, as we alluded to, we can also view the SPGP as two rather separate methods. The first is the particular form of the SPGP approximation as summarised by the SPGP covariance function equation (2.13). The second is the determination of pseudo-inputs as a continuous optimisation task, based on the SPGP marginal likelihood. These two ideas can be used independently if required. For example, the SPGP approximation can be used in a standalone way by choosing the pseudo-inputs as a random subset of the training data, or even with a greedy subset selection technique. Equally the method of gradient based pseudo-input optimisation could be applied to some previous approximations such as SR or PP.

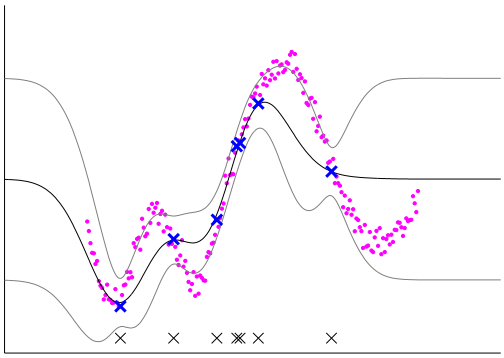
In this section, we examine the properties of the approximation itself, independent of how the pseudo-inputs are found, and we compare the approximation to others in the literature such as SR and PP. To do this we will utilise the framework of Quiñonero Candela and Rasmussen [2005], which brings together these approximations into one theoretical formalism. Whilst the derivation of the SPGP approximation under this framework is essentially the same as presented in section 2.2, the emphasis is different. Under this framework the emphasis is on how close the approximation is to the full GP.

The starting point to any of the approximations is a set of *inducing inputs* $\bar{\mathbf{X}} = \{\bar{\mathbf{x}}_m\}_{m=1}^M$. Quiñonero Candela and Rasmussen [2005] introduced this blanket term to refer either to a subset of the training inputs ('active set' or 'support set'), or to continuous 'pseudo-inputs' as we use in the SPGP. We will also adopt this terminology as then it is quite clear that we could be talking about either type. The point is that in terms of the mathematical forms of the approximations, the locations of the inducing inputs are irrelevant.

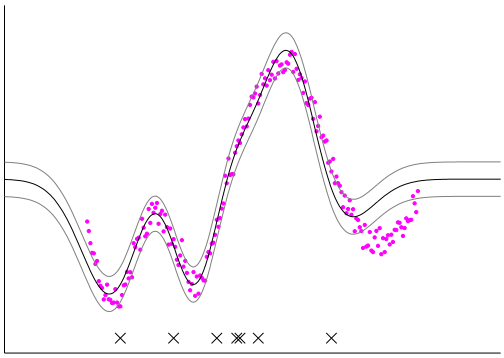
Throughout this section we will illustrate the different approximations by plotting their predictions on the same simple 1D dataset in figure 2.2. Only a very small number of inducing inputs have been chosen, randomly from the training inputs, so that some of the failure modes of the approximations are highlighted. Suitable hyperparameters are assumed known, and the same hyperparameters and inducing inputs used for all the approximations. Figure 2.2a shows the full GP solution for comparison.



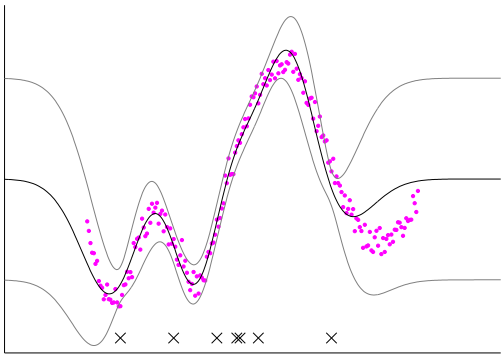
(a) GP



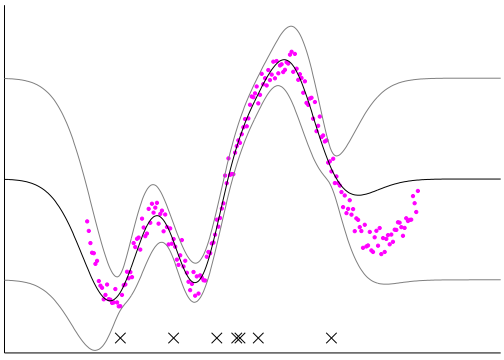
(b) SD



(c) SR



(d) PP



(e) FIC

Figure 2.2: Comparison of predictions of different GP approximations. Predictions are plotted with inducing inputs marked as black crosses.

The simplest way of using the inducing points is to use them as the training data in SD as described in section 2.1.1. The SD solution is shown in figure 2.2b, with the data points upon which it is based highlighted with blue crosses. Based on such a small number of points the SD prediction is very poor. The solution is clearly very affected by the x locations of the chosen points. However not only that: in a noisy regime, at a given x location the choice of individual points can skew the mean prediction up or down by a large amount. In summary, SD cannot gain advantage of the averaging effect of all the available data. For the same reason the predictive variances are generally too large. Of course the SD solution could be substantially improved by a better choice of points, but the point here is to highlight what happens in less than optimal situations, which is generally the case when we have limited resources.

2.3.1 A unifying framework

The framework of Quiñonero Candela and Rasmussen [2005] looks at how the set of inducing points can be used to approximate the full GP prior $p(\mathbf{f}, \mathbf{f}_T)$ over training and test function variables. Given a set of inducing inputs, the GP prior can be split into two parts:

$$p(\mathbf{f}, \mathbf{f}_T) = \int \mathrm{d}\bar{\mathbf{f}}\, p(\mathbf{f}, \mathbf{f}_T|\bar{\mathbf{f}})\, p(\bar{\mathbf{f}}) \;, \tag{2.15}$$

where the *inducing variables* $\bar{\mathbf{f}}$ are marginalised out. No approximation has been made in this step. In the first stage to all the approximations, Quiñonero Candela and Rasmussen [2005] make the approximation that $\mathbf{f}$ and $\mathbf{f}_T$ are conditionally independent given $\bar{\mathbf{f}}$:

$$p(\mathbf{f}, \mathbf{f}_T) \approx q(\mathbf{f}, \mathbf{f}_T) = \int \mathrm{d}\bar{\mathbf{f}}\, q(\mathbf{f}_T|\bar{\mathbf{f}})\, q(\mathbf{f}|\bar{\mathbf{f}})\, p(\bar{\mathbf{f}}) \;. \tag{2.16}$$

We will examine this assumption in more detail in chapter 3. Here the prior on the inducing variables is exact: $p(\bar{\mathbf{f}}) = \mathcal{N}(\mathbf{0}, \mathbf{K}_M)$. The various different sparse approximations are then derived by making additional approximations to the training and test conditionals $q(\mathbf{f}|\bar{\mathbf{f}})$ and $q(\mathbf{f}_T|\bar{\mathbf{f}})$. First we should note that the exact forms of these two conditionals are simply the noiseless correlated GP predictors (equation (1.17b)):

$$p(\mathbf{f}|\bar{\mathbf{f}}) = \mathcal{N}\big(\mathbf{K}_{NM}\mathbf{K}_M^{-1}\bar{\mathbf{f}},\; \mathbf{K}_N - \mathbf{Q}_N\big) \tag{2.17a}$$

$$p(\mathbf{f}_T|\bar{\mathbf{f}}) = \mathcal{N}\big(\mathbf{K}_{TM}\mathbf{K}_M^{-1}\bar{\mathbf{f}},\; \mathbf{K}_T - \mathbf{Q}_T\big) \;. \tag{2.17b}$$

2.3.2 Subset of regressors (SR)

The subset of regressors (SR) approximation is derived in this framework by using deterministic approximations (delta functions) for the conditionals:

$$q(\mathbf{f}|\bar{\mathbf{f}}) = \mathcal{N}\left(\mathbf{K}_{NM}\mathbf{K}_M^{-1}\bar{\mathbf{f}},\ \mathbf{0}\right) \tag{2.18a}$$

$$q(\mathbf{f}_T|\bar{\mathbf{f}}) = \mathcal{N}\left(\mathbf{K}_{TM}\mathbf{K}_M^{-1}\bar{\mathbf{f}},\ \mathbf{0}\right) . \tag{2.18b}$$

The exact means from equation (2.17) are retained, but the covariances are entirely dropped. Whilst it seems strange in this framework to use deterministic approximations, SR was not originally derived in this way. However it is exactly the dropping of these covariances that causes problems for SR.

When we integrate out the inducing variables $\bar{\mathbf{f}}$ in equation (2.16), we arrive at the SR approximate prior:

$$q_{\text{SR}}(\mathbf{f}, \mathbf{f}_T) = \mathcal{N}(\mathbf{0}, \tilde{\mathbf{K}}^{\text{SR}}_{N+T}) \tag{2.19a}$$

$$\tilde{\mathbf{K}}^{\text{SR}}_{N+T} = \begin{bmatrix} \mathbf{Q}_N & \mathbf{Q}_{NT} \\ \mathbf{Q}_{TN} & \mathbf{Q}_T \end{bmatrix} . \tag{2.19b}$$

As we stated in section 2.1.2, this also corresponds to a degenerate GP with covariance function $Q(\mathbf{x}, \mathbf{x}')$.

Figure 2.2c shows the SR prediction for the same 1D data set as discussed earlier for SD. The SR mean predictor is for the most part very good in comparison to SD, because it takes advantage of the averaging effect over all the training data. The approximation is still governed by the location of the inducing points, so in areas away from them, the mean prediction can be poor (tends back to zero mean). The big problem with SR is that the predictive variance is small away from inducing inputs (tends to the noise level σ^2). This is disastrous, because the regions away from inducing inputs are precisely the regions where the approximation is poor, and hence uncertainty should be high. Figure 2.2c shows data points falling well outside the two standard deviation predictive envelopes. This problem is caused by the deterministic nature of the approximations of equation (2.18), because uncertainty that should be present is not included. It is also essentially the same problem as faced by any degenerate GP, or finite linear model, as compared to the full non-parametric GP.

Quiñonero Candela and Rasmussen [2005] suggest renaming these different ap-

proximations with more descriptive names relating to their approximation framework. Under this scheme SR becomes the *Deterministic Inducing Conditional* approximation (DIC).

2.3.3 Projected process (PP)

The PP approximation solves some of the problems of SR, by reinstating the *test* conditional covariance. The training conditional remains deterministic like SR:

$$q(\mathbf{f}|\bar{\mathbf{f}}) = \mathcal{N}\left(\mathbf{K}_{NM}\mathbf{K}_M^{-1}\bar{\mathbf{f}},\ \mathbf{0}\right) \tag{2.20a}$$

$$q(\mathbf{f}_T|\bar{\mathbf{f}}) = p(\mathbf{f}_T|\bar{\mathbf{f}})\ . \tag{2.20b}$$

Again, whilst it seems strange to leave the training conditional deterministic, PP was not originally derived in this way (see section 2.3.6).

When we integrate out the inducing variables $\bar{\mathbf{f}}$ (equation (2.16)), we arrive at the PP approximate prior:

$$q_{\text{PP}}(\mathbf{f}, \mathbf{f}_T) = \mathcal{N}(\mathbf{0}, \tilde{\mathbf{K}}^{\text{PP}}_{N+T}) \tag{2.21a}$$

$$\tilde{\mathbf{K}}^{\text{PP}}_{N+T} = \begin{bmatrix} \mathbf{Q}_N & \mathbf{Q}_{NT} \\ \mathbf{Q}_{TN} & \mathbf{K}_T \end{bmatrix}\ . \tag{2.21b}$$

Notice how the test self covariance block of the PP prior covariance is now exact: $\mathbf{K}_T$. This ensures that the PP predictive distribution has full variance away from inducing inputs. Figure 2.2d demonstrates this effect showing that PP has much more desirable predictive variances than SR. The predictive means of SR and PP are exactly the same. However the remaining deterministic training approximation equation (2.20a) causes PP to 'break' in the low noise regime (see section 2.3.8).

PP has exactly the same marginal likelihood as SR:

$$q_{\text{PP/SR}}(\mathbf{y}) = \mathcal{N}(\mathbf{0}, \mathbf{Q}_N + \sigma^2\mathbf{I})\ . \tag{2.22}$$

Since the training and test variables are treated differently under PP, unlike SR it does not correspond to a GP model with a particular covariance function. Under Quiñonero Candela and Rasmussen [2005]'s naming scheme PP becomes *Deterministic Training Conditional* (DTC).

2.3.4 The fully independent (training) conditional (FI(T)C) approximation

FIC is the approximation we developed for the SPGP as described in section 2.2, renamed according to Quiñonero Candela and Rasmussen [2005]'s naming scheme. To derive FIC within this framework, we make the approximation that the function variables in both the training and test conditionals of equation (2.17) are fully independent:

$$q(\mathbf{f}|\bar{\mathbf{f}}) = \prod_n p(f_n|\bar{\mathbf{f}}) \tag{2.23a}$$

$$q(\mathbf{f}_T|\bar{\mathbf{f}}) = \prod_t p(f_t|\bar{\mathbf{f}}) \; . \tag{2.23b}$$

Note that the independence approximation is a way of reinstating part of the lost training conditional covariance of PP, whilst keeping the model tractable. If no independence approximation were made we would be back to inverting the full training covariance $\mathbf{K}_N + \sigma^2\mathbf{I}$. We are retaining the *marginal* variances in the conditionals, but dropping any correlations (replacing the full conditional covariance with a diagonal one). This independence approximation is equivalent to the i.i.d. assumption we made in the earlier derivation of the SPGP in section 2.2.

With these approximations we compute the integral of equation (2.16) to obtain the FIC approximate prior distribution:

$$q_{\text{FIC}}(\mathbf{f}, \mathbf{f}_T) = \mathcal{N}(\mathbf{0}, \tilde{\mathbf{K}}^{\text{FIC}}_{N+T}) \tag{2.24a}$$

$$\tilde{\mathbf{K}}^{\text{FIC}}_{N+T} = \begin{bmatrix} \mathbf{Q}_N + \text{diag}[\mathbf{K}_N - \mathbf{Q}_N] & \mathbf{Q}_{NT} \\ \mathbf{Q}_{TN} & \mathbf{Q}_T + \text{diag}[\mathbf{K}_T - \mathbf{Q}_T] \end{bmatrix} \; . \tag{2.24b}$$

Notice that since the training and test variables are treated in exactly the same way, there was no need in this case to first separate them out as in equation (2.16). *All* function variables are conditionally independent given $\bar{\mathbf{f}}$. Therefore, as we noted in the derivation of the SPGP (equation (2.13)), the FIC approximation corresponds to a standard GP model with a particular covariance function: $\tilde{K}^{\text{FIC}}(\mathbf{x}, \mathbf{x}') = Q(\mathbf{x}, \mathbf{x}') + \delta_{\mathbf{x}\mathbf{x}'}[K(\mathbf{x}, \mathbf{x}) - Q(\mathbf{x}, \mathbf{x})]$. In contrast to PP, the delta correction means that all the marginal variances of the FIC prior match the full GP exactly: $K(\mathbf{x}, \mathbf{x})$.

The FIC predictive distribution is formed from the blocks of equation (2.24b) in the

same way as equation (1.17) for the full GP:

$$p(\mathbf{y}_T|\mathbf{y}) = \mathcal{N}(\boldsymbol{\mu}_T^{\text{FIC}}, \boldsymbol{\Sigma}_T^{\text{FIC}}) \,, \tag{2.25a}$$

$$\begin{aligned} \boldsymbol{\mu}_T^{\text{FIC}} &= \mathbf{Q}_{TN}[\tilde{\mathbf{K}}_N^{\text{FIC}} + \sigma^2\mathbf{I}]^{-1}\mathbf{y} \\ \boldsymbol{\Sigma}_T^{\text{FIC}} &= \tilde{\mathbf{K}}_T^{\text{FIC}} - \mathbf{Q}_{TN}[\tilde{\mathbf{K}}_N^{\text{FIC}} + \sigma^2\mathbf{I}]^{-1}\mathbf{Q}_{NT} + \sigma^2\mathbf{I} \,. \end{aligned} \tag{2.25b}$$

The FITC approximation differs slightly from FIC in that only the *training* conditional is factorised. The test conditional remains exact:

$$q(\mathbf{f}|\bar{\mathbf{f}}) = \prod_n p(f_n|\bar{\mathbf{f}}) \tag{2.26a}$$

$$q(\mathbf{f}_T|\bar{\mathbf{f}}) = p(\mathbf{f}_T|\bar{\mathbf{f}}) \,. \tag{2.26b}$$

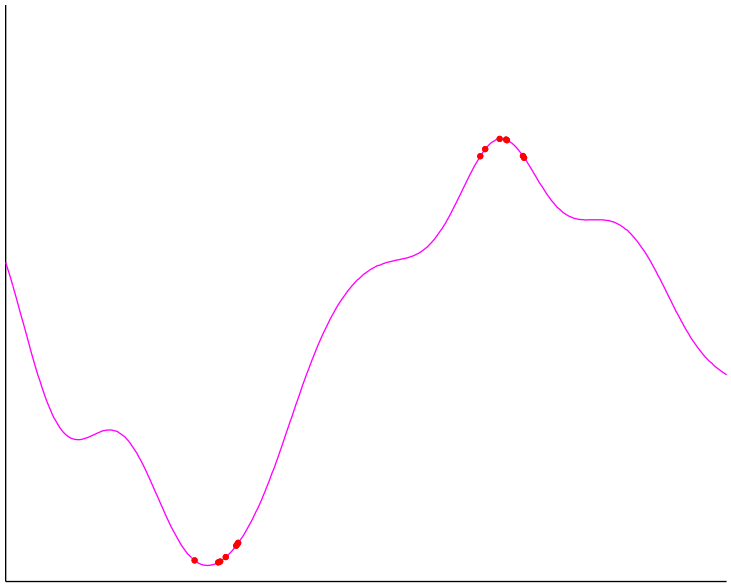
The FITC predictive distribution is therefore identical to FIC apart from the approximate $\tilde{\mathbf{K}}_T^{\text{FIC}}$ being replaced with the exact $\mathbf{K}_T$ in the predictive covariance. However the difference is only apparent if you want to make *correlated* predictions. Since the diagonal of $\tilde{\mathbf{K}}_T^{\text{FIC}}$ is exact ($\operatorname{diag}\tilde{\mathbf{K}}_T^{\text{FIC}} = \operatorname{diag}\mathbf{K}_T$), the *marginal* predictive variances of FITC and FIC are exactly the same. In either case therefore the FI(T)C single test case predictive distribution is exactly as reported in equation (2.10) for the SPGP:

$$\mu_*^{\text{FIC}} = \mathbf{Q}_{*N}[\tilde{\mathbf{K}}_N^{\text{FIC}} + \sigma^2\mathbf{I}]^{-1}\mathbf{y} \tag{2.27a}$$

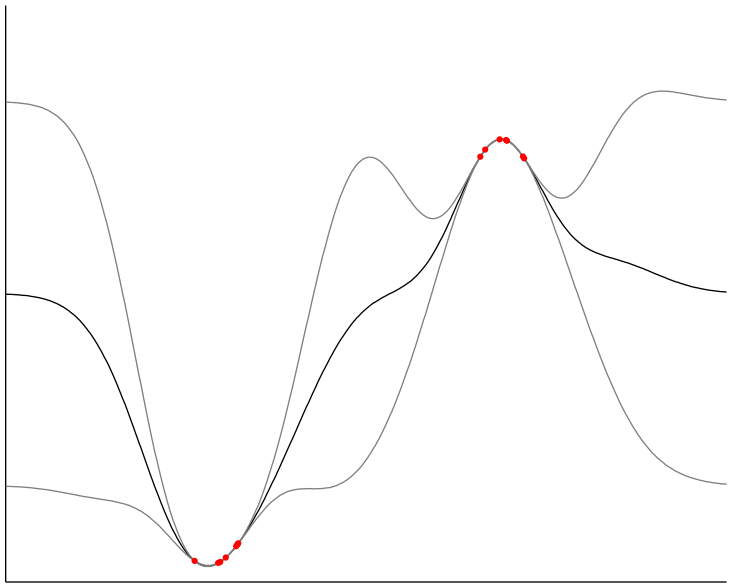
$$(\sigma_*^2)^{\text{FIC}} = K_* - \mathbf{Q}_{*N}[\tilde{\mathbf{K}}_N^{\text{FIC}} + \sigma^2\mathbf{I}]^{-1}\mathbf{Q}_{N*} + \sigma^2 \,. \tag{2.27b}$$

Figure 2.2e shows the FI(T)C predictions on the 1D example. In this case the predictions are very similar to PP, but this is not always the case (see section 2.3.8).

Whilst it is perhaps a little tedious to have gone through two derivations with essentially the same content in this section and in section 2.2, the different emphases have their own merits. The original SPGP derivation inspired the treatment of the pseudo-inputs as parameters. This framework of Quiñonero Candela and Rasmussen [2005] is the more useful for theoretical comparison to other approximations. We suggest that the term FIC or FITC is adopted when referring purely to the approximation, and the term SPGP is reserved for the entire process of the FIC approximation together with gradient based pseudo-input learning.

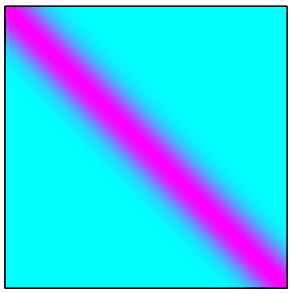


(a) A sample from a GP prior (magenta line) generated in two stages. First the inducing variables are sampled (red dots), and then the rest of the sample is generated from the conditional.

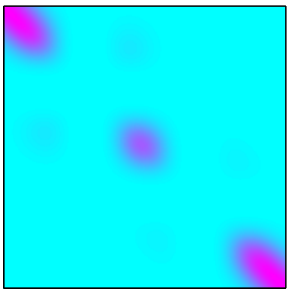


(b) The FIC factorised conditional approximation is represented by the mean and two standard deviation envelopes.

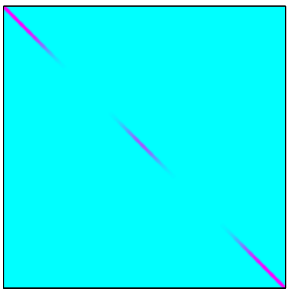
Figure 2.3: Two stage sampling methods for a full GP and FIC



(a) Covariance of the GP prior



(b) Covariance of the conditional $p(f(\mathbf{x})|\bar{\mathbf{f}})$



(c) FIC diagonal approximation to the conditional

(d) FIC approximate covariance, as constructed from the low rank and diagonal parts. This is a graphical representation of: $\tilde{K}^{\text{FIC}}(\mathbf{x},\mathbf{x}') = Q(\mathbf{x},\mathbf{x}') + \delta_{\mathbf{x}\mathbf{x}'}(K(\mathbf{x},\mathbf{x}) - Q(\mathbf{x},\mathbf{x}))$.

Figure 2.4: Graphical representation of various covariances from the FIC derivation

2.3.5 A graphical derivation of FIC

For intuition and understanding it is useful to view some of the covariances involved in the derivation of FIC in a graphical form, just as we showed the full GP prior covariance in figure 1.2. To start with consider the following two stage process for generating a sample function from a GP prior. We first pick a set of inducing inputs $\bar{\mathbf{X}}$ and generate corresponding inducing variables $\bar{\mathbf{f}}$ from the prior $p(\bar{\mathbf{f}})$. Secondly we generate the rest of the sample function from the posterior process $p(f(\mathbf{x})|\bar{\mathbf{f}})$. This is shown in figure 2.3a, where we have chosen inducing inputs in two clumps for the purposes of illustration. This is a graphical representation of equation (2.15), and of course the sample is a true sample from the original GP prior.

Figure 2.4b shows a plot of the covariance of the conditional $p(f(\mathbf{x})|\bar{\mathbf{f}})$. We can see that in comparison to the prior GP covariance figure 2.4a, the two clumps of inducing points remove two chunks of covariance, corresponding to these regions of high certainty. In the derivation of FIC it is this covariance matrix that is approximated by its diagonal in equation (2.23). This covariance approximation is displayed in

figure 2.4c, and represented in figure 2.3b by the mean and two standard deviation lines.

In the final stage of the derivation we integrate over the unknown inducing variables $\bar{\mathbf{f}}$ to get the FIC approximate prior of equation (2.24). The FIC prior covariance is represented graphically in figure 2.4d, showing how it is broken down into a very low rank part and a diagonal part. Notice how the low rank part (Q) contributes two chunks of covariance in the regions of the inducing points, and the diagonal part adds a 'correction' to ensure that the diagonal of the final approximation is exact. This plot enables us to graphically understand how the positions of the inducing points control the regions of correlations in the prior.

2.3.6 Optimal KL derivations

Within the framework discussed in the previous sections it seems fairly clear that the FIC approximation is 'closer' to the full GP than PP, in the sense that PP throws away relevant uncertainty whereas FIC partly retains it. Can this be justified a little better theoretically?

The Kullback-Leibler (KL) divergence [Kullback and Leibler, 1951] is a natural measure of 'closeness' for probability distributions (see section B.4). Therefore when analysing approximations it is natural to ask which approximation is closer under KL to the exact. In the framework discussed in the previous sections we have the exact GP prior $p(\mathbf{f}, \bar{\mathbf{f}}) = p(\mathbf{f}|\bar{\mathbf{f}})p(\bar{\mathbf{f}})$, where here we are changing notation slightly using $\mathbf{f}$ to refer to *any* function variables, training or test. The approximations made are all of the form $q(\mathbf{f}, \bar{\mathbf{f}}) = q(\mathbf{f}|\bar{\mathbf{f}})p(\bar{\mathbf{f}})$. In other words the prior on the inducing variables is kept exact, and some form of approximation made to the conditional.

The FIC approximation can be derived as minimising $\mathrm{KL}[p(\mathbf{f}, \bar{\mathbf{f}})\|q(\mathbf{f}, \bar{\mathbf{f}})]$ subject to the independence constraint $q(\mathbf{f}|\bar{\mathbf{f}}) = \prod_i q_i(f_i|\bar{\mathbf{f}})$, where the minimisation is over all q_i.[2] The solution is simply $q_i(f_i|\bar{\mathbf{f}}) = p(f_i|\bar{\mathbf{f}})$, i.e. approximate the full conditional with a product of single variable conditionals, as in equation (2.23). If no factorisation constraint were applied, then of course the solution would just be the exact GP. Therefore under this KL measure FIC is strictly closer to the full GP than PP (or SR).

It turns out that the PP approximation can also be derived using an optimal KL argument [see Seeger et al., 2003]. Firstly, PP was derived as a *likelihood* approxima-

[2] We thank Peter Sollich for pointing out this fact.

tion, rather than as an approximation to the prior. Therefore to follow the argument we need to also consider the outputs $\mathbf{y}$, and the full joint $p(\mathbf{f}, \bar{\mathbf{f}}, \mathbf{y})$. This joint can be rewritten: $p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|\bar{\mathbf{f}})p(\bar{\mathbf{f}})$. The exact likelihood $p(\mathbf{y}|\mathbf{f})$ is simply the noise around the function: $\mathcal{N}(\mathbf{f}, \sigma^2\mathbf{I})$. To get PP we approximate $p(\mathbf{y}|\mathbf{f})$ by a distribution that is restricted only to depend on $\bar{\mathbf{f}}$: $q(\mathbf{y}|\bar{\mathbf{f}})$. Then if we minimise the KL the 'wrong' way around, $\mathrm{KL}[q(\mathbf{f}, \bar{\mathbf{f}}, \mathbf{y})\|p(\mathbf{f}, \bar{\mathbf{f}}, \mathbf{y})]$, we get the solution $q(\mathbf{y}|\bar{\mathbf{f}}) = \mathcal{N}(\mathcal{E}[\mathbf{f}|\bar{\mathbf{f}}], \sigma^2\mathbf{I})$. We can see how this is essentially equation (2.20a) of PP with the noise included. In this case no factorisation constraint is necessary — the fact that the exact likelihood $p(\mathbf{y}|\mathbf{f})$ is already factorised (white noise) ensures that the approximation will be too.

To summarise: the exact joint may be written $p(\mathbf{y}|\mathbf{f})\, p(\mathbf{f}|\bar{\mathbf{f}})\, p(\bar{\mathbf{f}})$. The approximations are then:

Method	Approximation	KL minimisation
FIC	$p(\mathbf{y}|\mathbf{f}) \prod_i q_i(f_i|\bar{\mathbf{f}})\, p(\bar{\mathbf{f}})$	(p to q)
PP	$q(\mathbf{y}|\bar{\mathbf{f}})\, p(\mathbf{f}|\bar{\mathbf{f}})\, p(\bar{\mathbf{f}})$	(q to p)

We feel that the optimal KL argument for PP is unnatural. Firstly the KL divergence is the 'wrong' or variational way around, which means that the approximate distribution is being averaged over, rather than the exact. Secondly, the likelihood seems the wrong place to be making the approximation, as it is simply the addition of white noise. It is not this term which makes GP regression computationally intensive, but rather the full correlations in the GP prior. The conditional independence approximation in FIC breaks some of these correlations and seems a cleaner way to achieve computational tractability.

2.3.7 Sparse online Gaussian processes

Csató and Opper [2002], Csató [2002] presented sparse GP methods that process data sequentially (online) and that were particularly designed to approximate non-Gaussian likelihoods. These methods are also based on a subset of data points, and so are naturally related to the ones we have discussed. The methods process one data point at a time and consist of a Gaussian projection to deal with the non-Gaussianity of the likelihood, and a projection onto a sparse model to limit computation. Clearly the first projection is exact in the case of regression with Gaussian noise. Csató [2002] rarely discusses the batch regression case as opposed to the sequential classification case, however the method is equivalent to PP in the batch

regression case. This can be seen by referring to section 4.5 of Csató [2002]'s thesis, which is the one place where he makes a direct comparison with batch regression, and where the equations given are the PP equations. The sparsity projection presented in Csató [2002] uses essentially the same KL derivation as presented in section 2.3.6 to derive PP.

It has only very recently become clear that FITC is in fact a batch version of the online scheme presented in [Csató and Opper, 2002]. Superficially [Csató, 2002] appears to be an extended version of [Csató and Opper, 2002], and hence [Csató and Opper, 2002] had been previously taken as being equivalent to PP also. However the sparsity projection in [Csató and Opper, 2002] is actually slightly different to that in [Csató, 2002], and is in fact equivalent to a KL (p to q) projection not (q to p). Unfortunately it has taken a long while for the equivalence to FITC to be realised, partly due to the presentation and derivation styles being so different between [Csató and Opper, 2002] and [Snelson and Ghahramani, 2006a, Quiñonero Candela and Rasmussen, 2005]. It is clear now that Csató [2002], Csató and Opper [2002] are the originators of both the PP and FITC approximations, albeit in sequential form. Although Csató and Opper [2002] use the FITC approximation, they do not seem to consider the FITC marginal likelihood for adjustment of hyperparameters.

We briefly try to give a high level correspondence between Csató and Opper [2002]'s online scheme and the KL derivation of FITC presented in section 2.3.6. We do this by interpreting Csató and Opper [2002]'s online scheme in the language of this thesis. Csató and Opper [2002] consider projections onto a sparse *posterior process*:

$$q(\mathbf{f}_T|\mathbf{y}_n) = \int \mathrm{d}\bar{\mathbf{f}}\, p(\mathbf{f}_T|\bar{\mathbf{f}}) q_n(\bar{\mathbf{f}}) \;, \tag{2.28}$$

where $q_n(\bar{\mathbf{f}})$ is a Gaussian approximate posterior at the inducing points, having currently observed n training points. Notice that this essentially corresponds to the conditional independence assumption between training and test points of equation (2.16). Starting with the exact prior $q_0(\bar{\mathbf{f}}) = p(\bar{\mathbf{f}})$, at each step of the online scheme a new training point is included with the current approximation: $\hat{p}(\bar{\mathbf{f}}|\mathbf{y}_{n+1}) \propto p(y_{n+1}|\bar{\mathbf{f}}) q_n(\bar{\mathbf{f}})$. Then moments are matched at the inducing points between $\hat{p}(\bar{\mathbf{f}}|\mathbf{y}_{n+1})$ and the new approximate posterior $q_{n+1}(\bar{\mathbf{f}})$.[3] In the regression case of course this moment matching is exact, but the important point is that the sequential processing has induced a factorisation assumption equivalent to the one in section 2.3.6, and

[3] This type of online posterior update is known as *assumed density filtering* (ADF), and is the single sweep precursor to the expectation propagation (EP) algorithm discussed in section 1.7.

the moment matching is equivalent to minimising KL (p to q). In the regression case therefore, after all N data points have been processed, the approximate posterior is:

$$q_N(\bar{\mathbf{f}}) \propto \prod_{n=1}^{N} p(y_n|\bar{\mathbf{f}})p(\bar{\mathbf{f}}) \ . \tag{2.29}$$

This leads to the posterior process (from equation (2.28)):

$$q(\mathbf{f}_T|\mathbf{y}) \propto \int \mathrm{d}\bar{\mathbf{f}}\, p(\mathbf{f}_T|\bar{\mathbf{f}}) \prod_{n=1}^{N} p(y_n|\bar{\mathbf{f}})p(\bar{\mathbf{f}}) \ , \tag{2.30}$$

which is exactly the FITC approximation. Although the correspondence seems clear with hindsight, the presentation we have just given is quite different to that in [Csató and Opper, 2002].

2.3.8 Low noise problems for PP

The fact that PP was derived in section section 2.3.6 as a likelihood approximation, i.e. an approximation to the noise distribution, may suggest that things might go wrong in low noise situations. There are two different aspects to look at: the marginal likelihood approximation, and the predictive distribution. Consider figure 2.5 which shows some data drawn from a low noise GP. Also plotted are the two standard deviation lines of marginal likelihood variance,[4] for the full GP, SR, PP, and FIC, with some centrally placed inducing points. We see that the full GP and FIC have lines of constant variance ($K(\mathbf{x}, \mathbf{x}) + \sigma^2$), whereas the PP (and SR) lines decay to the noise level away from the inducing points. Therefore in the low noise regime, the data shown has a tiny probability of being drawn under the PP marginal likelihood equation (2.22). Although the FIC marginal likelihood equation (2.9) lacks correlations away from inducing points, at least the variances are correct, and therefore the probability of the data being drawn is still reasonably high. Empirically we can observe that the value of the FIC marginal likelihood is much closer to the full GP for the low noise data of figure 2.5:

Approximation	log likelihood
GP	534
FIC	142
PP (SR)	-1.14 $\times 10^5$

[4] The marginal likelihood variance is the diagonal of the training covariance matrix plus noise.

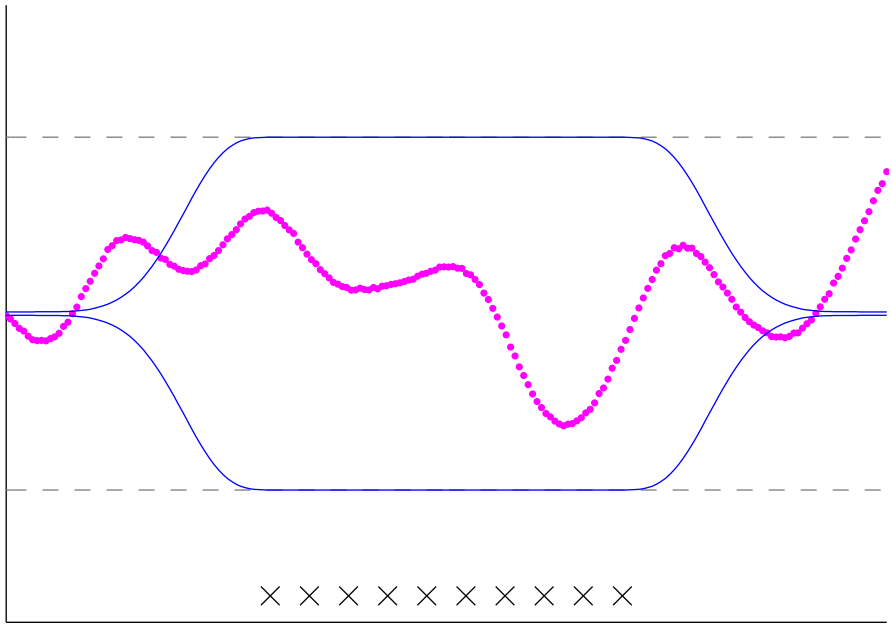
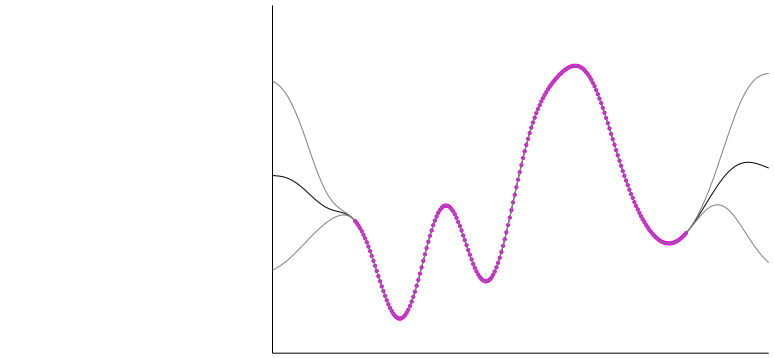


Figure 2.5: Comparison of marginal likelihood approximations in low noise regime. Plotted are the two standard deviation lines of the marginal likelihood variance. Grey, dashed = full GP and FIC. Blue, solid = PP and SR.

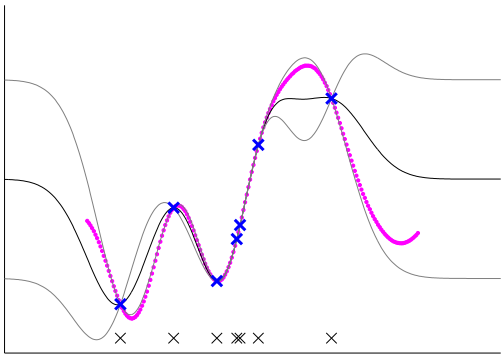
Figure 2.5 shows the inducing inputs clustered in the centre of the input range. Of course with enough inducing inputs covering the entire input space then both FIC and PP marginal likelihoods tend to the GP marginal likelihood, for reasons discussed in section 2.2.1. However in a realistic situation this is unlikely to be the case due to limited resources.

The second aspect of the approximation is the predictive distribution. Figure 2.2 shows the FIC and PP predictive distributions extremely similar to one another. However, again in the low noise regime the situation changes. Figure 2.6 is essentially the same plot with close to zero noise data. We see that PP catastrophically 'breaks' because at the locations of the inducing points, it makes predictions with almost zero variance 'off' the data. The problem is not the zero variance — the places with inducing points are exactly the locations where the function should be known with certainty (since there are also data points here in this example). The problem is that the PP mean function does not pass through these data points. Incidentally the SR prediction is even worse as it predicts with zero variance everywhere.

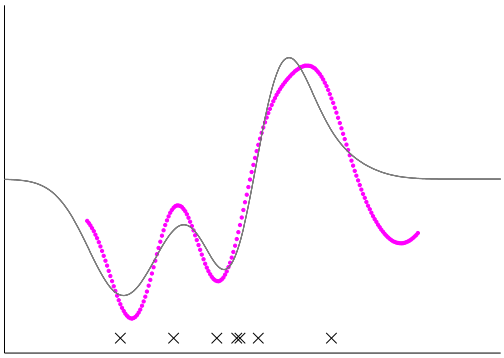
A further surprise when comparing figures (b) and (e) is that the FIC solution is exactly the same as SD in this regime, although further theoretical analysis below



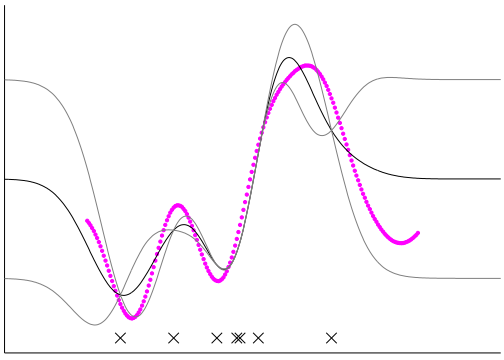
(a) GP



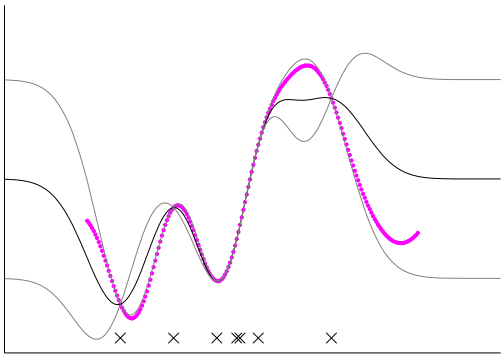
(b) SD



(c) SR



(d) PP



(e) FIC

Figure 2.6: Comparison of predictions of different GP approximations in a low noise regime. Predictions are plotted with inducing inputs marked as black crosses.

shows that in fact this must be the case. The FIC/SD solution also has zero variance predictions at the inducing points, but the data points at these positions are all exactly interpolated by the mean function (as we know must be the case for SD). What is happening with FIC is that the extra uncertainty in the model allows it to completely ignore the data away from the inducing inputs, and the solution collapses to SD. With PP the mean solution continues to be influenced by other data around the inducing inputs, and so the solution breaks. Clearly in this zero noise situation using FIC is an expensive way of doing SD! However, in more intermediate situations we know that FIC is a safe approximation that does not break, unlike PP.

For the reasons given here concerning low noise problems with PP, and for the fact that FIC can be considered a closer approximation to the full GP as outlined in section 2.3.6, we suggest that the FIC approximation is used in preference to PP in all cases. There is very little difference in cost of implementation, so as far as we can see there is no good reason to continue to use PP. The decision of whether to use the full SPGP method of choosing pseudo-inputs is a further one to make, and we shall empirically evaluate this in the experiments. However, as we stated before one can always choose to use the FIC approximation standalone from the SPGP, using whatever inducing point selection method is preferred.

Collapse of FIC to SD in zero noise

In this section we show that under a few reasonable assumptions all 'non-broken' approximations must collapse to SD in the zero noise limit. Suppose we have a GP approximation that is based on a set of M inducing inputs. We assume the predictive mean function will take on the following form:

$$\mu(\mathbf{x}) = \mathbf{K}_{\mathbf{x}M}\mathbf{T}_{MN}\mathbf{y}_N \ , \tag{2.31}$$

for some $\mathbf{T}_{MN}$ that is only a function of the data inputs and inducing inputs, not a function of the outputs $\mathbf{y}_N$. This type of mean function is common to all the different sparse approximations.

Consider the case where the inducing inputs are a subset of the training inputs. Suppose that the predictive variance reduces to the noise level at the inducing inputs: $\sigma^2_M = \sigma^2$. If we are in a noiseless regime, then this assumption implies we predict with total confidence at a point where we have an observed data point *and*

an inducing input. Again, this property is common to all the sparse approximations.

For the predictive distribution not to be broken in the noiseless regime, we must therefore require that the mean function passes exactly through the data points at the inducing inputs:

$$\begin{aligned} & \boldsymbol{\mu}_M \equiv \mathbf{y}_M \\ \Rightarrow \quad & \mathbf{K}_M \mathbf{T}_{MN} \mathbf{y}_N = \mathbf{y}_M \\ \Rightarrow \quad & \mathbf{K}_M \mathbf{T}_{MM} \mathbf{y}_M + \mathbf{K}_M \mathbf{T}_{M(N\setminus M)} \mathbf{y}_{(N\setminus M)} = \mathbf{y}_M \\ \Rightarrow \quad & (\mathbf{K}_M \mathbf{T}_{MM} - \mathbf{I}) \mathbf{y}_M + \mathbf{K}_M \mathbf{T}_{M(N\setminus M)} \mathbf{y}_{(N\setminus M)} = \mathbf{0} \ . \end{aligned} \tag{2.32}$$

Here we have split the data points into those with inducing inputs, and those without. Since this expression must hold for *any* setting of $\mathbf{y}_N$, and since $\mathbf{T}_{MN}$ is not a function of $\mathbf{y}_N$, the above matrix coefficients must be zero:

$$\begin{aligned} & \mathbf{K}_M \mathbf{T}_{MM} - \mathbf{I} \equiv \mathbf{0} \\ \Rightarrow \quad & \mathbf{T}_{MM} = \mathbf{K}_M^{-1} \end{aligned} \tag{2.33}$$

and

$$\begin{aligned} & \mathbf{K}_M \mathbf{T}_{M(N\setminus M)} \equiv \mathbf{0} \\ \Rightarrow \quad & \mathbf{T}_{M(N\setminus M)} = \mathbf{0} \ . \end{aligned} \tag{2.34}$$

Plugging these results back into equation (2.31) gives the predictive mean:

$$\mu(\mathbf{x}) = \mathbf{K}_{\mathbf{x}M} \mathbf{K}_M^{-1} \mathbf{y}_M \ , \tag{2.35}$$

which is exactly the noiseless subset of data (SD) predictive mean.

The same type of argument can be made for the predictive (co)variance. We assume that the predictive (co)variance takes the following form:

$$\Sigma(\mathbf{x}, \mathbf{x}') = K(\mathbf{x}, \mathbf{x}') - \mathbf{K}_{\mathbf{x}M} \mathbf{U}_{MM} \mathbf{K}_{M\mathbf{x}'} + \sigma^2 \delta_{\mathbf{x}\mathbf{x}'} \ , \tag{2.36}$$

for some $\mathbf{U}_{MM}$ which could depend on all N training inputs. This is the case for PP/FIC. When $\sigma^2 = 0$, again making the assumption that the predictive (co)variance

must vanish at the inducing inputs ($\mathbf{\Sigma}_M \equiv \mathbf{0}$) implies:

$$\begin{aligned} & \mathbf{K}_M - \mathbf{K}_M \mathbf{U}_{MM} \mathbf{K}_M \equiv \mathbf{0} \\ \Rightarrow \quad & \mathbf{I} - \mathbf{U}_{MM} \mathbf{K}_M = \mathbf{0} \\ \Rightarrow \quad & \mathbf{U}_{MM} = \mathbf{K}_M^{-1} \ . \end{aligned} \tag{2.37}$$

Plugging this back into equation (2.36) gives exactly the noiseless SD predictive (co)variance. Essentially the noiseless regime constrains the systems of equations so much so that the only solution is to ignore all data points apart from the ones at the inducing points, and to interpolate these data points exactly as SD does.

In the noisy regime ($\sigma^2 \neq 0$), then the predictive mean need not pass exactly through the data points at the inducing inputs i.e. equation (2.32) does not hold. Similarly equation (2.37) does not hold because $\mathbf{\Sigma}_M \neq \sigma^2 \mathbf{I}$ — we would expect it to have full *co*variance structure.[5]

In the noiseless regime there are only a few ways in which the predictive distribution could not reduce to SD. Firstly, we may not want the assumption $\mathbf{\Sigma}_M \equiv \mathbf{0}$ to hold. However this would be slightly strange — given we have an observation, an inducing input in the same place, and zero noise, then it seems reasonable to predict with total confidence there. Secondly we may move the inducing inputs away from the training data. Now SD does not make sense. This is exactly what happens when training the SPGP with gradients. Thirdly, we may come up with alternative sparse approximations that do not have equation (2.31) and equation (2.36) as the functional forms for their predictive mean and variance.

2.4 Classification

The GP approximations that we have discussed in this chapter can also be applied to speed up GP classification. The extra complication is that they have to be embedded inside an expectation propagation (EP) approximation to deal with the non-Gaussian likelihood, as discussed in section 1.7. The IVM [Lawrence et al., 2003] can be considered as the SD approximation inside an EP approximation, together with a greedy forward selection algorithm for choosing the subset (that does depend on all the data). Csató [2002], Csató and Opper [2002]'s sparse online Gaussian processes can be seen as implementing PP and FITC for classification as discussed in

[5] Only the *marginal* predictive variances are constrained in the noisy case: $\Sigma_{mm} = \sigma^2$.

section 2.3.7. Originally these were single sweep online algorithms, but later Csató [2002] did convert to the multiple sweep EP versions. Seeger [2003] also uses the PP approximation with EP for classification.

It would even be possible to extend the SPGP method of optimising pseudo-inputs using gradients of the approximate marginal likelihood from EP. Recently Seeger et al. [2007] have argued much more in favour of the IVM for classification rather than any more sophisticated approximation such as PP or FITC. Their main argument is that the IVM only requires M EP 'site' approximations rather than the full N that are needed by the more accurate PP or FITC. We certainly agree that implementing the SPGP for classification is expensive in this regard — each gradient computation requires running EP until convergence at all N sites. However, the cruder approximation of the IVM with its forward selection algorithm still causes problems for hyperparameter learning. The SPGP would benefit from its smooth joint optimisation of pseudo-inputs and hyperparameters. It will be interesting future work to see if the SPGP for classification is worthwhile or whether the costs of the EP updates are too prohibitive.

2.5 Results

In this section we present some results for the FIC approximation and the SPGP. Since our goal is to achieve the best performance on a test set for a given computation time, we show graphs of test error vs. time. The implementations were done in MATLAB,[6] and experiments made on the same 2.4 GHz machine. We use the SE-ARD covariance function equation (1.21) throughout. Optimisations were made using a standard conjugate gradient optimiser.

2.5.1 Datasets and error measures

We test our methods on three different regression data sets, whose properties are summarised in table 2.1. These are data sets that have been used in the literature to test GP methods before, and so we have preserved the splits of training and test sets to allow direct comparison. `SARCOS` is a data set used by Rasmussen and Williams [2006], and before that by Vijayakumar et al. [2002].[7] The task is to learn the inverse

[6] Dominant computations such as Cholesky decompositions use the appropriate efficient LAPACK routines.

[7] The data is available at `http://www.gaussianprocess.org/gpml/data/`.

Data set	SARCOS	KIN40K	Abalone
Input dimension (D)	21	8	8
Training set size (N)	44,484	10,000	3,133
Test set size (T)	4,449	30,000	1,044

Table 2.1: Properties of the data sets

dynamics of a robot arm — to map from the 21 dimensional joint position, velocity, and acceleration space to the torque at a single joint. This is a highly nonlinear regression problem.

KIN40K is a similar data set, with the object to predict the distance of a robot arm head from a target, based on an 8 dimensional input space consisting of joint positions and twist angles. This data set has been used by Schwaighofer and Tresp [2003] and by Seeger et al. [2003] to assess GP approximations.[8] It is also a highly nonlinear data set.

Abalone [Blake and Merz, 1998] is a smaller data set,[9] that is actually small enough to be handled in reasonable time by a full GP. However it is a different type of data set from the previous two, and illustrates some useful results. The object is to predict the age of abalone from 8 physical measurements.

We use two error measures on the test sets. The first is mean squared error (MSE), which only depends on the mean μ_* of the predictive distribution:

$$\text{MSE} = \frac{1}{T}\sum_{t=1}^{T}(y_t - \mu_t)^2 \,. \tag{2.38}$$

For KIN40K we scale this MSE by $\frac{1}{2}$ to allow comparison to Seeger et al. [2003]. The second error measure we use is negative log predictive density (NLPD) which also takes into account the predictive variance σ_*^2:

$$\text{NLPD} = \frac{1}{T}\sum_{t=1}^{T}\left[\frac{(y_t - \mu_t)^2}{2\sigma_t^2} + \frac{1}{2}\log(2\pi\sigma_t^2)\right] \,. \tag{2.39}$$

Just like for MSE, smaller is better for NLPD.

[8] The data is available at http://ida.first.fraunhofer.de/~anton/data.html.
[9] The data is available at http://www.ics.uci.edu/~mlearn/MLSummary.html.

2.5.2 Prediction times

In section 2.1 we discussed how we may require different strategies depending on whether prediction cost or training cost is the more important factor to the user. In this section we test one of these extremes by looking at performance as a function purely of prediction time. The aim is to assess firstly whether the FIC approximation of section 2.3.4 performs better than the baseline SD method of section 2.1.1, and secondly whether the SPGP method of finding pseudo-inputs (section 2.2.1) performs better than random subset selection.

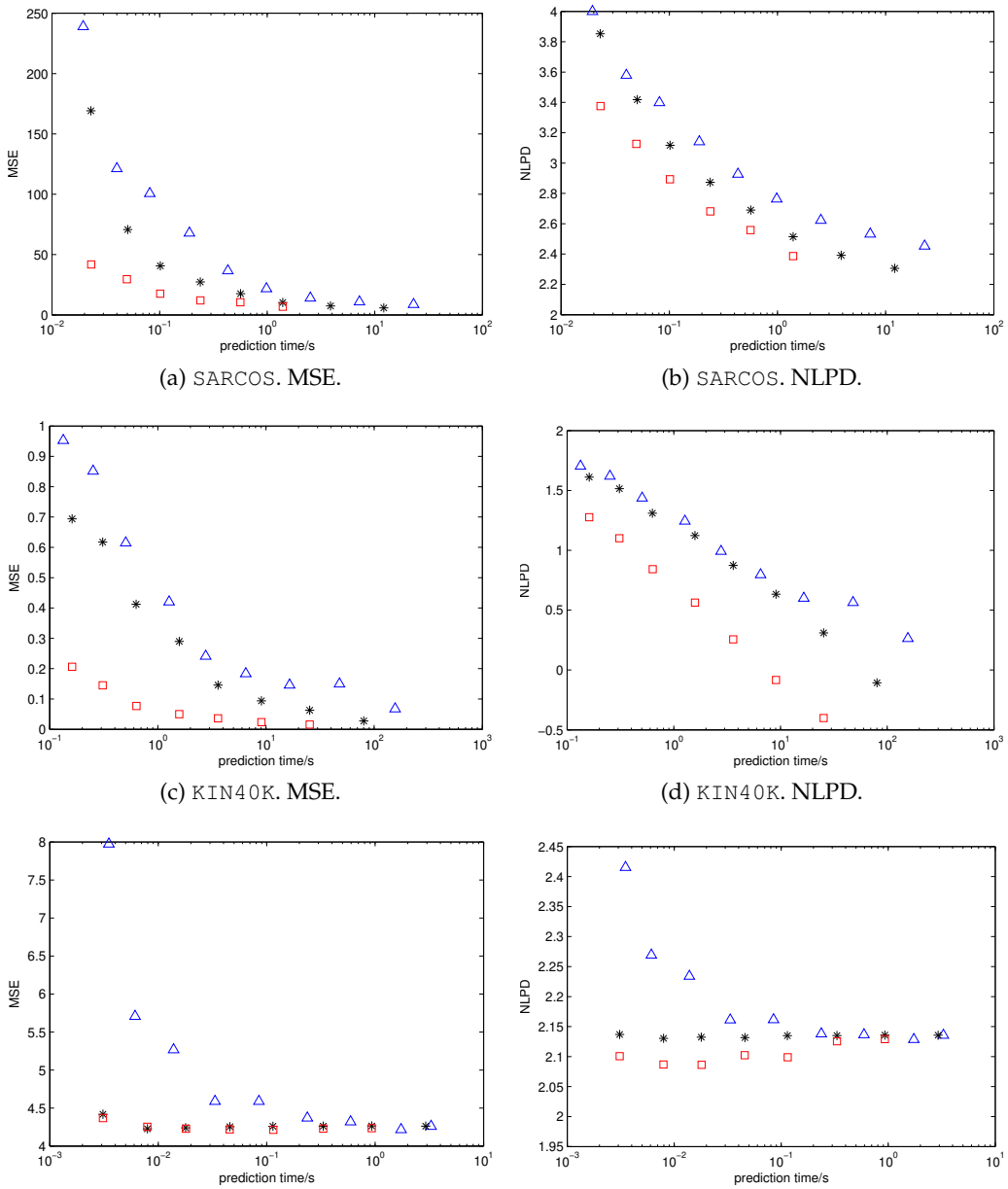
To make this comparison we use a 'ground truth' set of hyperparameters that are obtained by maximising the SD marginal likelihood on a large subset of size 2,048, for all three data sets. Since we are assessing pure prediction times here, we do not worry about the training cost of the hyperparameter learning. Section 2.5.3 looks at this cost.

Figure 2.7 shows plots of test MSE or NLPD vs. total test prediction time (after any precomputations) on the three data sets of section 2.5.1, for three different methods. The first method is SD, with points plotted as blue traingles. We vary the size of the subset M in powers of two from $M = 16$ to $M = 4,096$,[10] and the subset is chosen randomly from the training data. The FIC approximation is plotted as black stars, and uses exactly the same random subset as SD as its inducing inputs. Finally we show the pseudo-input optimisation of the SPGP as the red squares, where the initialisation is on the random subset.[11] This pseudo-input optimisation does not include the joint hyperparameter optimisation as discussed in section 2.2.1. Here we want to separate out these two effects and just show how the pseudo-input optimisation improves FIC with random subset selection. In this section all three methods rely on the same 'ground truth' hyperparameters.

Figure 2.7 helps us assess which method one should choose if one wants to obtain the best performance for a given test time. Curves lying towards the bottom left of the plots are therefore better. This type of requirement would occur if you had an unlimited offline training time, but you need to make a rapid series of online predictions. Since prediction time for all these methods only depends on the subset/inducing set size, and is $\mathcal{O}(M^2)$ per test case, these plots also essentially show how performance varies with subset size.

[10] For `Abalone` the final point is the entire training set of size 3,133.

[11] For FIC and SPGP we stop at a smaller maximum subset size than SD, depending on the data set, due to the training or precomputation costs becoming unreasonably large.



(a) SARCOS. MSE. (b) SARCOS. NLPD.

(c) KIN40K. MSE. (d) KIN40K. NLPD.

(e) Abalone. MSE. (f) Abalone. NLPD.

Figure 2.7: Test error vs. prediction time for the three data sets SARCOS, KIN40K, and Abalone. Blue triangles: SD + random subset. Black stars: FIC + random subset. Red squares: SPGP (with fixed hyperparameters). The different plotted points were obtained by varying M, the size of the subset or number of inducing points.

Figures 2.7 (a)–(d) show qualitatively similar trends for the highly nonlinear data sets of SARCOS and KIN40K. Firstly the FIC approximation is significantly better than SD for the same random subset. Using information from all the training data clearly helps as compared to throwing it away. These plots then show that optimising inducing inputs with the method of the SPGP makes a further significant improvement in accuracy for a given cost. The SPGP is able to achieve very high accuracy for only a small inducing set. The complexity of these data sets means that there is not much of a saturation effect — as the subset size is increased or more time is spent then performance keeps increasing. Using the SPGP however helps drive towards saturation much earlier.

Figures (e) and (f) show slightly different effects for Abalone. The main difference is that there is a definite saturation effect. This data set is very likely to be much simpler than SARCOS and KIN40K, and so a maximum performance level is reached. However for SD we still require a relatively large subset to reach this saturation. This is in contrast to FIC and the SPGP, which reach this saturation with only a tiny inducing set of $M = 32$. Here the full GP performance is matched, and so there is no point increasing the inducing set size further. Prediction times are consequently extremely fast. In this case optimising inducing points with the SPGP does not improve much beyond random, apart from a slight bettering of NLPD.

In summary, if you want to get the most out of a set of inducing inputs, then you should use the FIC approximation and optimise them with the SPGP. The caveat of course is that the SPGP optimisation is a much more expensive operation than random subset selection, and we also have hyperparameter training to take into account. These issues are addressed in the next section.

2.5.3 Training times

In this section we investigate the opposite regime to section 2.5.2. We look at performance as a function of training time exclusively. We want to compare FIC with random subset selection to the full SPGP optimisation. At first sight it seems that FIC will be a clear winner because random selection is so much cheaper than pseudo-input optimisation. However, the advantage of the SPGP is that it can obtain hyperparameters at the same time as pseudo-inputs, using the SPGP marginal likelihood. In section 2.5.2 we used 'ground truth' hyperparameters obtained by training SD on a very large subset. This 'ground truth' hyperparameter training is very expensive. It would not be particularly fair to FIC to simply add on this train-

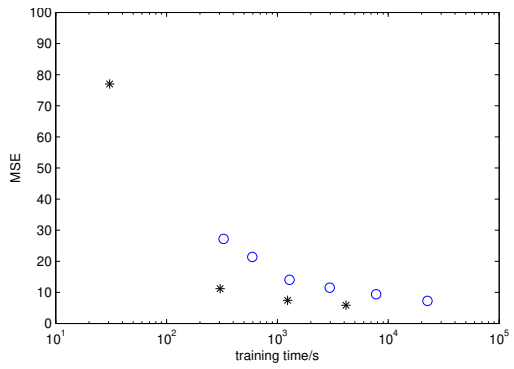
ing time, because perhaps we can get away with finding hyperparameters from a much smaller subset.

The various strategies available for hyperparameter learning make comparison difficult. We decided to test one reasonable strategy for FIC: for each value of M in the FIC approximation we obtain hyperparameters by training SD on the same size subset. However, a problem with using SD to find hyperparameters is that if M is too small there is simply not enough data to determine reasonable hyperparameters, and we get very poor results. We set a threshold of $M = 256$, below which this strategy for FIC is not viable. We therefore tested this strategy with $M = 256$, 512, 1024, and 2048. For the SPGP there is no such complication: we simply optimise the SPGP marginal likelihood jointly to find both pseudo-inputs and hyperparameters.

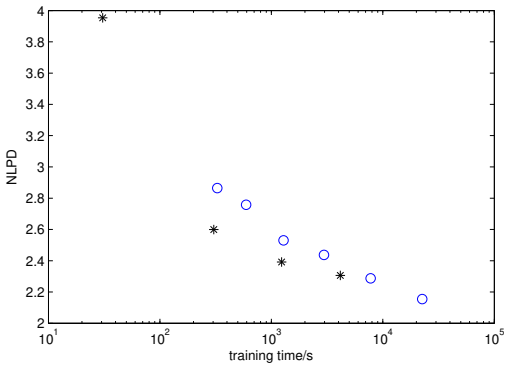
Figure 2.8 shows that even when looking purely at training times, the SPGP optimisation is not as expensive as might be first thought when hyperparameter learning is factored into the comparison. However it is fair to say, at least for `SARCOS` and `KIN40K`, that FIC with random selection is probably the better strategy. This means that if you want to get the best accuracy for a given training time, and you do not care about prediction time, then you should use SD to first find hyperparameters, and then use the FIC approximation with the same random inducing set. By following this strategy though you sacrifice speed in the predictions. For `Abalone` we see the saturation effect again, but also a curious rising SPGP NLPD line which we discuss in the next section.

We could have investigated total training and test time, rather than each individually. However the total training and test time is not really meaningful when the overall data set has been split arbitrarily into training and test sets for the purposes of testing machine learning methods. It very much depends on the application as to whether figure 2.7 or figure 2.8 is the more relevant figure.

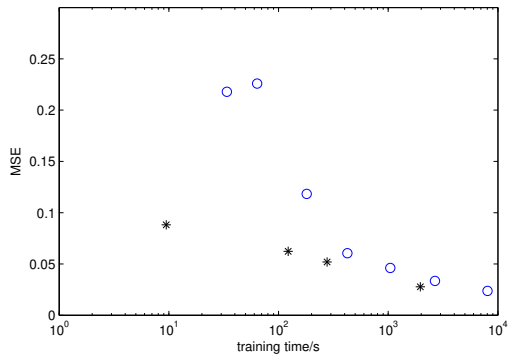
In some low dimensional problems it may be possible to avoid the marginal likelihood based hyperparameter learning by using cross validation to choose them instead. However for any reasonably sized input space, when we use the SE-ARD covariance with a lengthscale per dimension, the number of hyperparameters is simply too big to use CV.



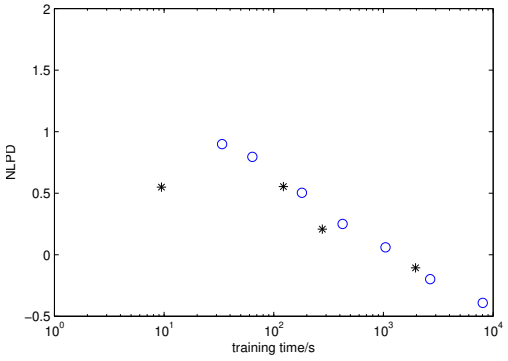
(a) SARCOS. MSE.



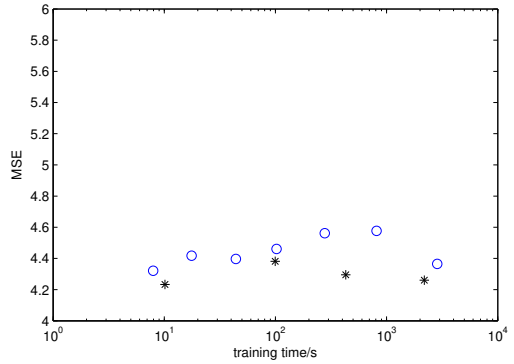
(b) SARCOS. NLPD.



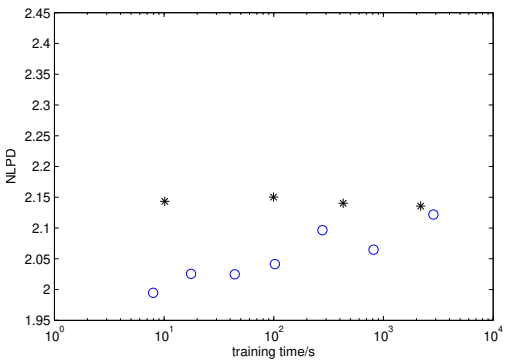
(c) KIN40K. MSE.



(d) KIN40K. NLPD.



(e) Abalone. MSE.



(f) Abalone. NLPD.

Figure 2.8: Test error vs. training time for the three data sets SARCOS, KIN40K, and Abalone. Black stars: FIC + random subset. Blue circles: SPGP, including hyperparameter learning. The different plotted points were obtained by varying M, the size of the subset or number of inducing points.

2.5.4 SPGP hyperparameters

A further point which figure 2.7 and figure 2.8 somewhat obscure, is whether the SPGP marginal likelihood gives better hyperparameters than the 'ground truth'. In figure 2.9 we replot the SPGP test errors as a function of the number of pseudo-inputs M. We replot the red squares from figure 2.7 showing the SPGP with optimised pseudo-inputs but fixed 'ground truth' hyperparameters. We replot the blue circles from figure 2.8 showing the SPGP with both pseudo-inputs and hyperparameters optimised.

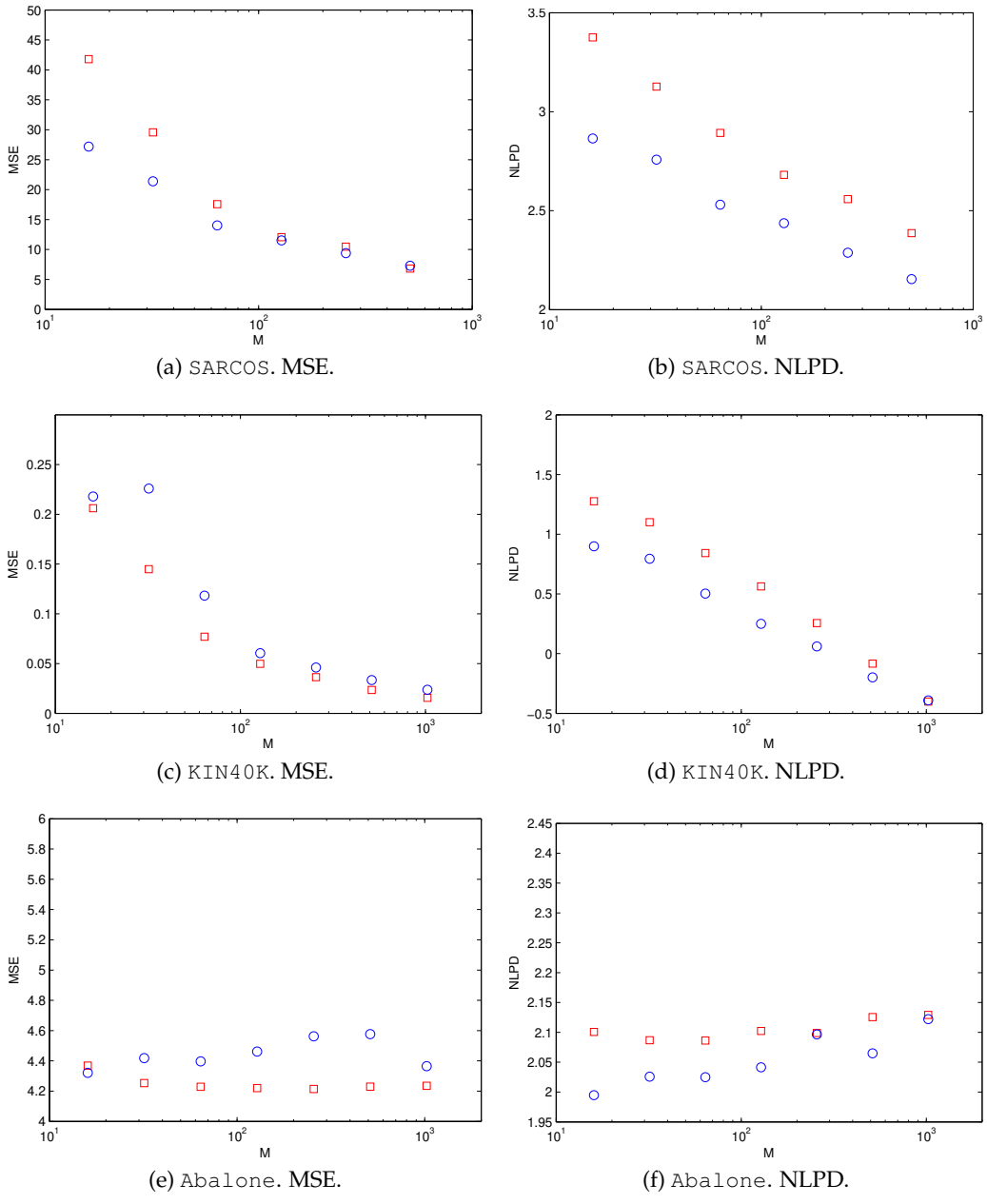
The results for NLPD are clear. Using the SPGP marginal likelihood to learn hyperparameters jointly with pseudo-inputs offers a significant performance increase for the same inducing set size, compared with obtaining hyperparameters using a large SD. Though the 'ground truth' hyperparameters were obtained from an SD of size 2,048, the SPGP marginal likelihood uses *all* training data even when M is small, and therefore extra benefit is obtained. Interestingly this seems to come at expense of a slightly worse MSE for both `KIN40K` and `Abalone`. If you care about the quality of your predictive variances then SPGP hyperparameter optimisation is clearly worthwhile.

The final interesting observation is the *rising* line of blue circles on figure 2.9f for `Abalone` NLPD, as the number of pseudo-inputs is increased. What seems to be happening is that the SPGP is able to find a higher likelihood solution with a smaller set of pseudo-inputs, and that this is leading to better performance than even a full GP. When the number of pseudo-inputs is increased, local minima prevent the SPGP from finding such a good solution, and the error rises back towards that of the full GP. This is an example of the ability of the SPGP to model some data better than a full GP due to the inherent extra flexibility in the SPGP covariance. We discuss this at length in chapter 4.

2.5.5 Other subset selection methods

To give an idea of how the SPGP pseudo-input optimisation performs compared to other subset selection methods in the literature, we reproduce some plots from Seeger et al. [2003] in figure 2.10.[12] This shows test MSE for `KIN40K` using a set of 'ground truth' hyperparameters, just as in figure 2.7c except with a different range of M. Figure 2.10a shows Seeger et al. [2003]'s very cheap info-gain selection

[12] We would like to thank the authors Seeger et al. [2003] for allowing us to use their figure.

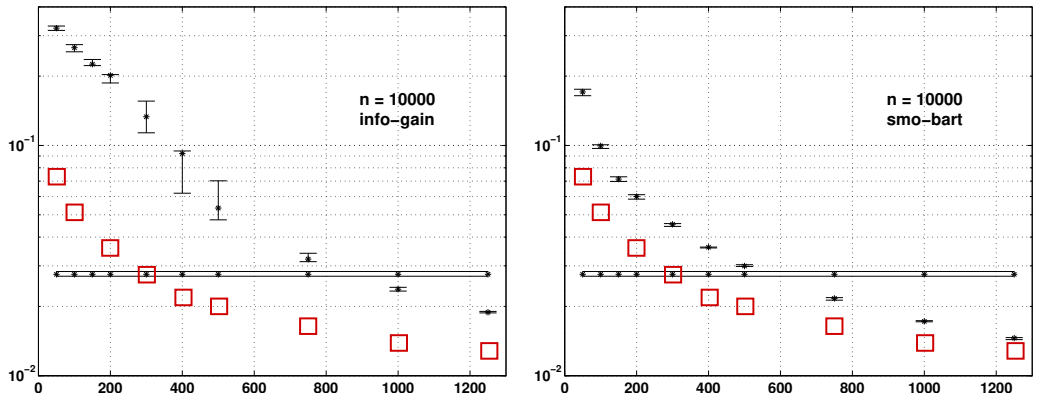


(a) SARCOS. MSE. (b) SARCOS. NLPD.

(c) KIN40K. MSE. (d) KIN40K. NLPD.

(e) Abalone. MSE. (f) Abalone. NLPD.

Figure 2.9: Test error vs. number of pseudo-inputs for the three data sets SARCOS, KIN40K, and Abalone. Red squares: SPGP with fixed 'ground truth' hyperparameters. Blue circles: SPGP with both pseudo-inputs and hyperparameters optimised.



(a) Info-gain (b) Smola-Bartlett

Figure 2.10: Comparison of SPGP to subset selection methods. Plotted is test MSE for the KIN40K data set as a function of M. (a) shows Seeger et al. [2003]'s info-gain selection method, and (b) shows Smola and Bartlett [2001]'s selection method, using the PP approximation. Overlaid as the red squares is the SPGP. The horizontal lines are the performance of a subset of data (SD) of size 2000. Hyperparameters are fixed as 'ground truth' for all three methods.

method, and figure 2.10b shows Smola and Bartlett [2001]'s more expensive selection method, in conjunction with the PP approximation of section 2.3.3. We overlay the SPGP results onto these plots as red squares.

We see that the SPGP gradient optimisation provides a significant increase in performance over the subset selection methods, for a given size of inducing set M. The SPGP also has the advantages of hyperparameter optimisation as discussed in section 2.5.4, as compared to the problematic interleaving of subset selection and hyperparameter learning as discussed in [Seeger et al., 2003]. We also note that Keerthi and Chu [2006] use a similar selection method to Smola and Bartlett [2001] that gives comparable performance but is cheaper to implement.

2.6 Remarks

In this chapter we have introduced a new technique for speeding up Gaussian process regression. The SPGP consists of both a new batch regression approximation FIC, and a gradient method for selecting the inducing inputs. We have justified theoretically why FIC makes a closer approximation to the full GP than other approximations such as SR and PP, and we have highlighted several problems with these previous approaches. We have shown how the pseudo-inputs of the SPGP

can be seen as extra parameters in a more complex flexible covariance function, which can therefore be optimised using the SPGP marginal likelihood.

Experimental results show that this pseudo-input optimisation achieves very high accuracy for a given inducing set size. Therefore the SPGP is particularly useful if one cares about prediction costs. Experiments also show that jointly learning hyperparameters and pseudo-inputs is a reliable way to get high accuracy predictions, especially in terms of predictive variances. This SPGP hyperparameter learning also avoids the complications and unreliabilities inherent in other hyperparameter training strategies. These strategies include the interleaving of subset selection and hyperparameter learning in subset selection methods, and also the pre-learning of hyperparameters using a GP on a large subset of data.

Chapter 3

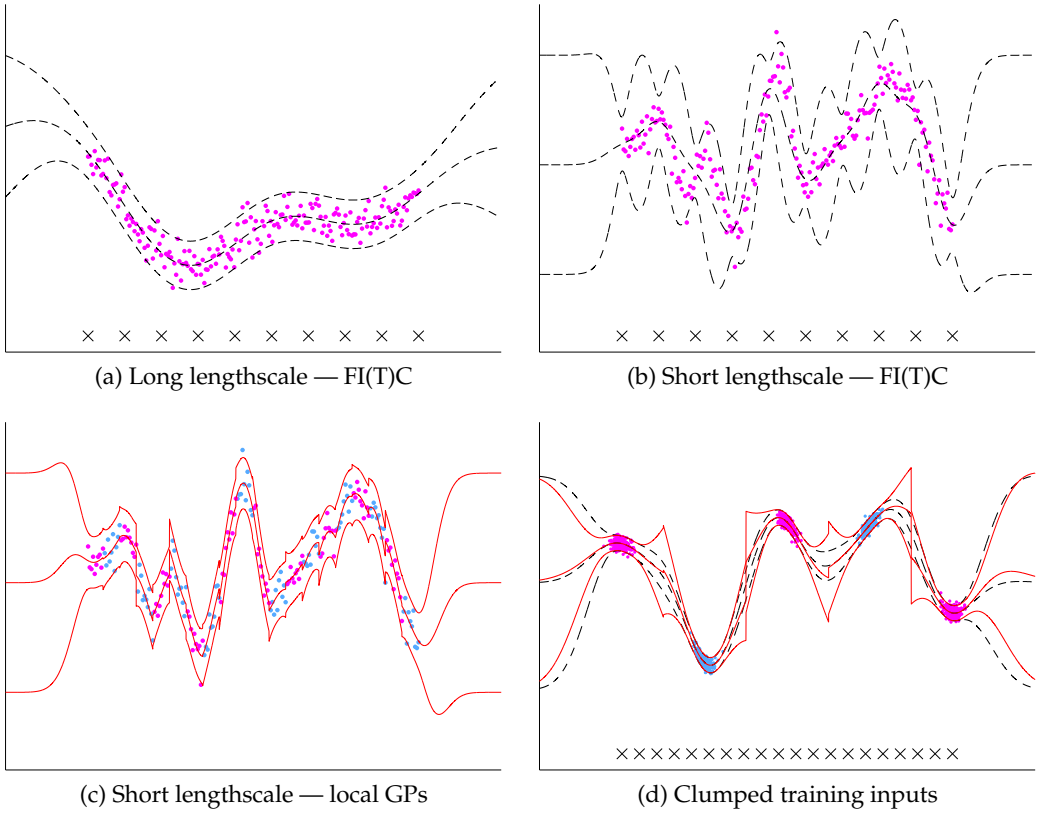
Local and global Gaussian process approximations

In chapter 2 we examined a class of approximations based on a set of inducing inputs, and we looked at a method for choosing inducing inputs based on continuous optimisation. We could refer to this class of approximations as *global*, because the M support points are essentially summarising *all* N data points.

There is a rather different type of approach which has been somewhat forgotten as a way of speeding up GP regression. This is a *local* type of approximation, where only training data points nearby to the test point in question are used to make a prediction. In this chapter we examine the regimes in which a local or global approximation is most suitable. For example, a very complex 'wiggly' data set may not be well summarised by a small number of inducing points, and a local regression scheme may be faster and more accurate.

A natural question to ask is whether there is an approximation that combines the best of both worlds: a combination of a local and global approximation that will be suitable for all regimes. In this chapter we develop such an approximation, and show how it can be derived from a natural extension of the approximation framework of Quiñonero Candela and Rasmussen [2005] outlined in section 2.3.1. This chapter is based on Snelson and Ghahramani [2007].

Throughout this chapter, we assume we have already obtained suitable hyperparameters for the covariance function, and we just concern ourselves with examining the nature of the approximations themselves.



(a) Long lengthscale — FI(T)C

(b) Short lengthscale — FI(T)C

(c) Short lengthscale — local GPs

(d) Clumped training inputs

Figure 3.1: 1D comparison of local and global GP approximations. Mean predictions and two standard deviation error lines are plotted, as black dashed lines for FI(T)C and red solid lines for local GPs. For FI(T)C the x positions of the inducing inputs are marked by black crosses. In (c) and (d) the local training blocks are demarcated by alternating the colors of the data points.

3.1 Local or global approximations

To understand the regimes in which a global approximation such as FI(T)C works well and not so well, it is simplest to look at an example. Figure 3.1a shows some sample data drawn from a GP with a fairly long lengthscale relative to the input point sampling. The FI(T)C prediction is plotted, using just 10 evenly spaced inducing inputs. The approximation is clearly extremely good — a full GP prediction looks essentially identical. Figure 3.1b shows the same number of data points drawn from a GP with a much shorter lengthscale. The FI(T)C prediction is plotted again using only 10 inducing inputs, and is clearly much worse, particularly in the gaps between inducing inputs. The training and prediction costs for the examples in figure 3.1a and figure 3.1b are exactly the same. In this simple example, we could

just increase the number of inducing inputs in figure 3.1b to take into account the extra complexity of the function. However, in a more realistic problem we may not be able to afford the extra cost to do this. For a very complex function we may find ourselves needing almost as many inducing inputs as data points to model the function well, and that takes us back towards $\mathcal{O}(N^3)$ complexity. Although each inducing input only affects predictions in a local region around itself, we refer to this type of approximation as global because *all* N data points contribute to each prediction made, via the inducing points.

An alternative type of approach to the data in figure 3.1b is to use a series of local GPs. This approach is shown in figure 3.1c. The training points are grouped into blocks of 10 points each, and independent GP predictors formed from each of the blocks. The nearest block's GP is used to predict at a given test point. This is a particular unsmoothed example of *local nonlinear regression*, similar in flavor to e.g. LOESS [Cleveland and Devlin, 1988, Grosse, 1989]. It is also a trivial unsmoothed example of a mixture of GP experts [Rasmussen and Ghahramani, 2002]. Viewed as an approximation to a GP, it is taking advantage of the fact that typical covariance functions like the squared exponential of equation (1.6) are local in nature.

The independence between the blocks leads to the discontinuous nature of the prediction in figure 3.1c, but if we ignore the ugly aesthetics, the prediction is actually a much better fit than that of figure 3.1b. If as in this illustration we choose equal block sizes of size B, then the training cost is $\mathcal{O}(N/B \times B^3) = \mathcal{O}(NB^2)$, and prediction cost per test case is $\mathcal{O}(B^2)$.[1] For figures (b) and (c) we chose $B = M = 10$, so the costs are essentially equivalent. In this regime therefore the local type of approximation is the more efficient option.

Apart from the ugly discontinuities, the local GP approach would actually work pretty well for the longer lengthscale example of figure 3.1a. However there are certainly situations where the local approach can be poor. Figure 3.1d shows some data where the training inputs have been sampled in a non-uniform manner. Such a situation often happens in real world examples due to artifacts in the data collection process, and is more pronounced in high dimensions. In this situation, if we take the clusters as separate blocks and use the local GP approach the extrapolation between clusters is very poor, because the blocks are all independent from each other. The FI(T)C predictions are much better because they take into account the correlations between the clusters and extrapolate well.

[1] This ignores the clustering cost (see section 3.2.3).

3.2 A combined local and global approximation

With the discussion of the previous section in mind, it would be nice to have an approximation that combined the ideas of both the global and local approaches, so that it would be suitable to use in all regimes. In this section we develop such an approximation and show how it is naturally derived as an extension of the theoretical framework of section 2.3.1. To lead into this we review one further GP approximation, which we omitted from our discussion in chapter 2, that has some local aspects.

3.2.1 The partially independent training conditional (PITC) approximation

In their unifying paper, Quiñonero Candela and Rasmussen [2005] suggest a further improved approximation to FI(T)C, which they call the *partially independent training conditional* (PITC) approximation. It also turns out to be a more general inductive form of the BCM [Tresp, 2000, Schwaighofer and Tresp, 2003], which we discussed briefly in section 2.1.3. The BCM can be seen as a special case of PITC with the inducing inputs chosen to be the test inputs.

Rather than assume complete training conditional independence as in FITC equation (2.26a), PITC only assumes partial independence. The training points are grouped into 'blocks' or 'clusters' $\{\mathbf{X}_{B_s}, \mathbf{f}_{B_s}\}_{s=1}^{S}$, and conditional independence is only assumed between blocks:

$$q(\mathbf{f}|\bar{\mathbf{f}}) = \prod_s p(\mathbf{f}_{B_s}|\bar{\mathbf{f}}) \tag{3.1a}$$

$$q(\mathbf{f}_T|\bar{\mathbf{f}}) = p(\mathbf{f}_T|\bar{\mathbf{f}}) \,. \tag{3.1b}$$

As in FITC, the test conditional (3.1b) remains exact. Assuming these approximate conditionals leads to the PITC training and test covariance:

$$\tilde{\mathbf{K}}_{N+T}^{\text{PITC}} = \begin{bmatrix} \mathbf{Q}_N + \text{bkdiag}[\mathbf{K}_N - \mathbf{Q}_N] & \mathbf{Q}_{NT} \\ \mathbf{Q}_{TN} & \mathbf{K}_T \end{bmatrix} . \tag{3.2}$$

This covariance is shown in a more pictorial manner in figure 3.2b, and can be compared to FITC in figure 3.2a. We see that it is not just the diagonal that is exact, but rather blocks of the diagonal are exact.

$$\begin{bmatrix} K_{11} & & \mathbf{Q}_N & & \vdots \\ & K_{22} & & & \mathbf{Q}_{NT} \\ \mathbf{Q}_N & & \ddots & & \vdots \\ & & & K_{NN} & \vdots \\ \dots & \mathbf{Q}_{TN} & \dots & \dots & \begin{bmatrix} \mathbf{K}_T \end{bmatrix} \end{bmatrix}$$

(a) FITC

$$\begin{bmatrix} \begin{bmatrix} \mathbf{K}_{B_1} \end{bmatrix} & & \mathbf{Q}_N & & \vdots \\ & \begin{bmatrix} \mathbf{K}_{B_2} \end{bmatrix} & & & \mathbf{Q}_{NT} \\ \mathbf{Q}_N & & \ddots & & \vdots \\ & & & \begin{bmatrix} \mathbf{K}_{B_S} \end{bmatrix} & \vdots \\ \dots & \mathbf{Q}_{TN} & \dots & \dots & \begin{bmatrix} \mathbf{K}_T \end{bmatrix} \end{bmatrix}$$

(b) PITC

$$\begin{bmatrix} \begin{bmatrix} \mathbf{K}_{B_1} \end{bmatrix} & & \mathbf{Q}_N & & \vdots \\ & \begin{bmatrix} \mathbf{K}_{B_2} \end{bmatrix} & & & \mathbf{Q}_{(N \setminus B_S)*} \\ \mathbf{Q}_N & & \ddots & & \vdots \\ & & & \begin{bmatrix} \mathbf{K}_{B_S} & \mathbf{K}_{B_S *} \\ \mathbf{K}_{* B_S} & K_* \end{bmatrix} & \\ \dots & \mathbf{Q}_{*(N \setminus B_S)} & \dots & & \end{bmatrix}$$

(c) PIC

Figure 3.2: FITC, PITC, and PIC approximate prior covariances

The predictive distribution for a single test point becomes:

$$p(y_*|\mathbf{y}) = \mathcal{N}(\mu_*^{\text{PITC}}, (\sigma_*^2)^{\text{PITC}}) \,, \tag{3.3a}$$

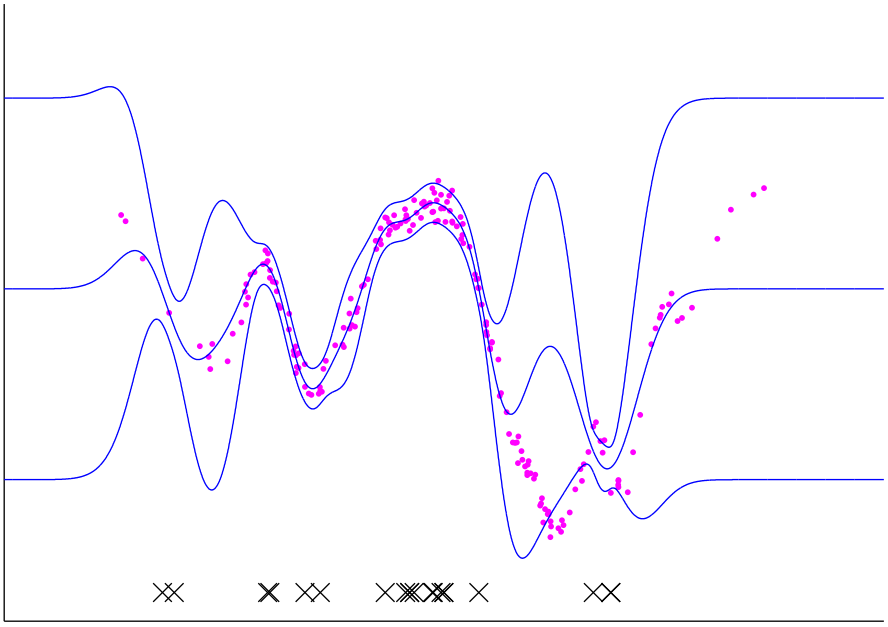
$$\begin{aligned} \mu_*^{\text{PITC}} &= \mathbf{Q}_{*N}[\tilde{\mathbf{K}}_N^{\text{PITC}} + \sigma^2\mathbf{I}]^{-1}\mathbf{y} \\ (\sigma_*^2)^{\text{PITC}} &= K_* - \mathbf{Q}_{*N}[\tilde{\mathbf{K}}_N^{\text{PITC}} + \sigma^2\mathbf{I}]^{-1}\mathbf{Q}_{N*} + \sigma^2 \,, \end{aligned} \tag{3.3b}$$

where $\tilde{\mathbf{K}}_N^{\text{PITC}} = \mathbf{Q}_N + \text{bkdiag}[\mathbf{K}_N - \mathbf{Q}_N]$. Just as for FI(T)C the PITC mean predictor is simply a weighted sum of M basis functions. The cost per test case is therefore exactly the same: $\mathcal{O}(M)$ for the mean and $\mathcal{O}(M^2)$ for the variance. How about the precomputations? The cost to invert $\tilde{\mathbf{K}}_N^{\text{PITC}} + \sigma^2\mathbf{I}$ depends on the sizes of the blocks $\{B_s\}$. There is no requirement for the blocks to be of equal size, but for simplicity suppose they all have size B. Then for the same reasons as in section 3.1, the extra precomputations cost $\mathcal{O}(NB^2)$ (we must invert $\mathbf{K}_{B_s} - \mathbf{Q}_{B_s} + \sigma^2\mathbf{I}$ for each block).

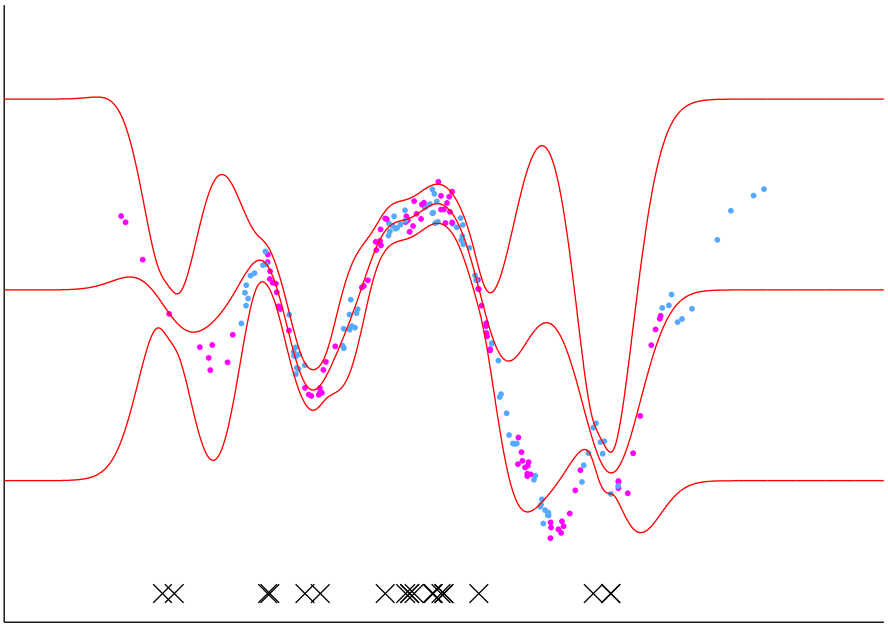
At training time the blocks need to be chosen in some way. Typical covariance functions used for regression, such as the squared exponential, are local in nature. The covariance between two points decays to zero as the points are separated farther apart in input space. This means that it makes sense to cluster points into local blocks in input space, so as to gain most advantage from the blocked approximation. We will discuss particular clustering schemes in section 3.2.3.

The PITC approximation certainly has the flavour of trying to combine a local approximation with a global one. Figure 3.3 shows a 1D example of the predictive distributions of FI(T)C and PITC, with the inducing inputs randomly placed. Figure 3.3a shows that the FI(T)C approximation is poor away from the locations of the inducing points — the predictive variances grow large and the mean tends back towards zero. Based on our earlier arguments that the blocked PITC approximation is closer to the full GP than FI(T)C, we might expected PITC to do much better in figure 3.3b. In fact this is not the case — the predictive distribution is almost identical to FI(T)C, even though we have blocked the training points perfectly into 20 blocks of 10 points based on their ordering along the x axis. Why is this the case? The answer is easy to see by referring back to the PITC predictive distribution of equation (3.3). Looking at the mean prediction: it is still just a weighted sum of basis functions centered on the same inducing points as in FI(T)C. The blocking has only altered the weights slightly. Fundamentally, when the basis functions are local such as the squared exponential, PITC is as incapable of modelling well away from inducing inputs as FI(T)C.

We see that PITC does not have the local characteristics we are looking for in its



(a) FI(T)C



(b) PITC

Figure 3.3: One dimensional comparison of FI(T)C and PITC predictions. The inducing inputs are chosen randomly from the training input points, and are the same for both (a) and (b). For PITC the blocks are denoted by alternating the colours of the data points, blue and magenta. Here there are 20 blocks, each with 10 data points.

predictions. The fact that it gives very similar predictions to FI(T)C is the reason why we did not include the approximation in chapter 2 — it does not seem worth the extra overhead of the blocking for negligible gain in predictive accuracy.

3.2.2 The partially independent conditional (PIC) approximation

In this section we develop a new approximation that successfully combines the ideas of the global and local approximations. Another way to understand why the PITC predictions are not much different to FI(T)C is to look again at the PITC prior covariance of figure 3.2b. The structure of this covariance is such that the training inputs have been blocked separately from the test inputs — the test inputs have effectively been placed in a block of their own. This means that the PITC approximation cannot be considered a GP model with a particular covariance function, as the decision of which block in which to place an input depends on whether that input is a training input or test input. The consequence of this separation of training and test inputs into different blocks is that they only interact with each other via the M inducing inputs. This in turn leads to the predictive distribution being very similar to FI(T)C, and largely governed by the positioning of the inducing inputs.

The separation of the test points into their own block came about because of the first assumption about sparse GP approximations made by Quiñonero Candela and Rasmussen [2005]: the conditional independence of training and test points, denoted $\mathbf{f} \perp \mathbf{f}_T|\bar{\mathbf{f}}$, in equation (2.16). To derive a new approximation we relax this assumption and consider what happens if we block the *joint* training and test conditional. We treat the training and test inputs equivalently, and group them into blocks according only to their $\mathbf{x}$ positions. For ease of notation, and because we will only use the marginal predictive variance, we consider a single test input $\mathbf{x}_*$. Suppose that on the basis of its position this test input was grouped with training block B_S. Then the approximate conditional is:

$$p(\mathbf{f}, f_*|\bar{\mathbf{f}}) \approx q(\mathbf{f}, f_*|\bar{\mathbf{f}}) = p(\mathbf{f}_{B_S}, f_*|\bar{\mathbf{f}}) \prod_{s=1}^{S-1} p(\mathbf{f}_{B_s}|\bar{\mathbf{f}}) \,. \tag{3.4}$$

It seems logical to follow the naming convention introduced by Quiñonero Candela and Rasmussen [2005], and call this approximation the *partially independent conditional* (PIC) approximation. The PIC training and test covariance is:

$$\tilde{\mathbf{K}}_{N+T}^{\text{PIC}} = \mathbf{Q}_{N+T} + \text{bkdiag}[\mathbf{K}_{N+T} - \mathbf{Q}_{N+T}] \,. \tag{3.5}$$

This can be seen pictorially, and with a single test point for clarity, in figure 3.2c.

Notice that unlike PITC, PIC *can* correspond to a standard GP with a particular covariance function. For example, suppose we divided the input space up into disjoint regions *before seeing any data*. Then if two points (training or test) fall into the same region they are placed in the same block. This corresponds to the following covariance function:

$$\tilde{K}^{\text{PIC}}(\mathbf{x}, \mathbf{x}') = Q(\mathbf{x}, \mathbf{x}') + \psi(\mathbf{x}, \mathbf{x}')[K(\mathbf{x}, \mathbf{x}') - Q(\mathbf{x}, \mathbf{x}')] \,, \tag{3.6a}$$

$$\text{where} \qquad \psi(\mathbf{x}, \mathbf{x}') = \begin{cases} 1 & \text{if } \mathbf{x}, \mathbf{x}' \text{ are in the same region} \\ 0 & \text{otherwise} \end{cases} . \tag{3.6b}$$

In practice typical clustering schemes we will use will rely on all the training data to define regions in input space, and so will not technically correspond to the covariance function of equation (3.6). At a high level though, equation (3.6) is a good description of the PIC covariance.

For ease of notation when discussing the predictive distribution, we refer to the training block that the test point $\mathbf{x}_*$ belongs to as B. As shorthand for all the *training* points excluding B, we use $\not{B}$. The PIC single test point predictive distribution is:

$$p(y_*|\mathbf{y}) = \mathcal{N}(\mu_*^{\text{PIC}}, (\sigma_*^2)^{\text{PIC}}) \,, \tag{3.7a}$$

$$\begin{aligned} \mu_*^{\text{PIC}} &= \tilde{\mathbf{K}}_{*N}^{\text{PIC}} [\tilde{\mathbf{K}}_N^{\text{PITC}} + \sigma^2 \mathbf{I}]^{-1} \mathbf{y} \\ (\sigma_*^2)^{\text{PIC}} &= K_* - \tilde{\mathbf{K}}_{*N}^{\text{PIC}} [\tilde{\mathbf{K}}_N^{\text{PITC}} + \sigma^2 \mathbf{I}]^{-1} \tilde{\mathbf{K}}_{N*}^{\text{PIC}} + \sigma^2 \,, \end{aligned} \tag{3.7b}$$

where $\tilde{\mathbf{K}}_{*N}^{\text{PIC}} = [\mathbf{Q}_{*\not{B}} \,, \mathbf{K}_{*B}]$ (see figure 3.2c).

Let us look first at the mean predictor μ_*^{PIC}. Part of the mean predictor, which we define $\mathbf{p} = [\tilde{\mathbf{K}}_N^{\text{PITC}} + \sigma^2 \mathbf{I}]^{-1} \mathbf{y}$, is exactly the same as for PITC (3.3). We can expand μ_*^{PIC} further:

$$\begin{aligned} \mu_*^{\text{PIC}} &= \mathbf{Q}_{*\not{B}} \mathbf{p}_{\not{B}} + \mathbf{K}_{*B} \mathbf{p}_B \\ &= \mathbf{K}_{*M} \boldsymbol{\beta} + \mathbf{K}_{*B} \mathbf{p}_B \,, \end{aligned} \tag{3.8}$$

where the weights $\boldsymbol{\beta} = \mathbf{K}_M^{-1} \mathbf{K}_{M\not{B}} \mathbf{p}_{\not{B}}$. We therefore see that the mean is a weighted sum of basis functions centered at the M inducing inputs *and* at the training inputs in block B. This has the desired feature of being a combination of a local and global predictor. The local information comes from the block B that the test point is assigned to, and the global information comes via the inducing points. A similar

interpretation can be applied to the variance.

Referring to equation (3.5) or figure 3.2c, we see that there are now two limiting processes that will return us to the full GP. If there are N inducing points placed exactly on the training points, then $\mathbf{Q} = \mathbf{K}$, and therefore $\tilde{\mathbf{K}}^{\text{PIC}}_{N+T} = \mathbf{K}_{N+T}$. Similarly if we decrease the number of blocks until we have only one block, then $\tilde{\mathbf{K}}^{\text{PIC}}_{N+T} = \mathbf{K}_{N+T}$. In a practical situation we can push towards both these limits as far as our computational budget will allow.

Taking these limits in the opposite direction gives us some further insight. If we take all the block sizes to one (or equivalently if the points within a block are far separated from each other), then we recover FIC. If we take the number of inducing points to zero (or equivalently move them all far away from the data), then $\mathbf{Q} \to \mathbf{0}$. Referring to figure 3.2c, we see that all the blocks become completely decoupled from each other. We are left with the purely local GP predictor of section 3.1, where only the points in the test point's block are used to make the prediction. We can also see this from equation (3.8), since $\mathbf{p}_B \to [\mathbf{K}_B + \sigma^2\mathbf{I}]^{-1}\mathbf{y}_B$.

FI(T)C and PITC both had $\mathcal{O}(M)$ and $\mathcal{O}(M^2)$ costs per test point for predicting the mean and variance respectively, after precomputations. How about PIC? Looking at equation (3.8), at first it seems that prediction will be too expensive because of the product $\mathbf{Q}_{*\not{B}}\mathbf{p}_{\not{B}}$. In general $\not{B}$ will be close in size to N, and so $\mathcal{O}(\not{B})$ will be too expensive. However, we can rewrite equation (3.8) again:

$$\begin{aligned}\mu_*^{\text{PIC}} &= \mathbf{K}_{*M}\mathbf{K}_M^{-1}\mathbf{K}_{M\not{B}}\mathbf{p}_{\not{B}} + \mathbf{K}_{*B}\mathbf{p}_B \\ &= \mathbf{K}_{*M}\big(\underbrace{\mathbf{K}_M^{-1}\mathbf{K}_{MN}\mathbf{p}}_{\mathbf{w}_M} - \underbrace{\mathbf{K}_M^{-1}\mathbf{K}_{MB}\mathbf{p}_B}_{\mathbf{w}_M^B}\big) + \mathbf{K}_{*B}\mathbf{p}_B \,, \end{aligned} \tag{3.9}$$

where $\mathbf{w}_M = \sum_{s=1}^{S}\mathbf{w}_M^{B_s}$. Hence we can precompute $\mathbf{p}$, then precompute $\mathbf{w}_M^{B_s}$ for each block, and finally precompute $\mathbf{w}_M$. Having done this the cost per test case at test time will be $\mathcal{O}(M + B)$ for the mean. We can play a similar trick for the variance, which then costs $\mathcal{O}\big((M + B)^2\big)$.

KL divergences and graphical covariances

It should be fairly clear that we can extend the optimal KL derivation of FIC as presented in section 2.3.6 to PIC. We follow exactly the same argument, but alter the factorisation constraint to be slightly less restrictive. Rather than the factorisation constraint being over single data points, it is over blocks. We minimise

Figure 3.4: Graphical representation of PIC approximate covariance, as constructed from the low rank and block-diagonal parts.

$\mathrm{KL}[p(\mathbf{f},\bar{\mathbf{f}})\|q(\mathbf{f},\bar{\mathbf{f}})]$ subject to the independence constraint $q(\mathbf{f}|\bar{\mathbf{f}}) = \prod_s q_s(\mathbf{f}_{B_s}|\bar{\mathbf{f}})$, where the minimisation is over all q_s. The solution is simply $q_s(\mathbf{f}_{B_s}|\bar{\mathbf{f}}) = p(\mathbf{f}_{B_s}|\bar{\mathbf{f}})$. Here $\mathbf{f}$ refers to any type of function variable, both training and test — they are treated equally.

In figure 2.4d we showed a graphical representation of the construction of the FIC approximate covariance, for a particular choice of inducing inputs in two clumps. We can also do the same for PIC, and this is shown in figure 3.4. The only difference is that the diagonal is replaced by a block diagonal, and hence it becomes closer to the GP prior covariance. We can also see how part of the band of covariance is filled in by blocks and part is filled in by inducing points.

3.2.3 Clustering schemes

We need a scheme for clustering possibly high dimensional training inputs into blocks for the PIC approximation. We then need to be able to quickly assign a new test point to a block at test time. We do not want too costly a method, especially because even if the clustering is poor we still 'fall back' on FIC. We suggest two simple schemes. The more complicated one is *farthest point clustering* [Gonzales, 1985]. The number of clusters S is chosen in advance. A random input point is picked as the first cluster center. The farthest point from this is chosen as the next center. The farthest point from both of these is chosen as the next, and so on, until we have S centers. Then each point in the training set is assigned to its nearest cluster center. At test time, a test point is simply assigned to the nearest cluster center. We also consider an even simpler algorithm which we call *random clustering*. It is exactly as above except that the cluster centers are picked randomly (without replacement) from the training input points.

The naive costs for these algorithms are $\mathcal{O}(NS)$ training time and $\mathcal{O}(S)$ test time per test case. However with suitable data structures (e.g. KD-trees [Preparata and Shamos, 1985]) and implementation these costs can be reduced to $\mathcal{O}(N \log S)$ and $\mathcal{O}(\log S)$ [Feder and Greene, 1988].

The rough difference between the two is that farthest point clustering leads to regions of fairly equal dimension in input space, but not necessarily equal block sizes B_s if the input distribution is non-uniform. Random clustering leads to more uniform block sizes, but these may not correspond to similar dimensions in input space.

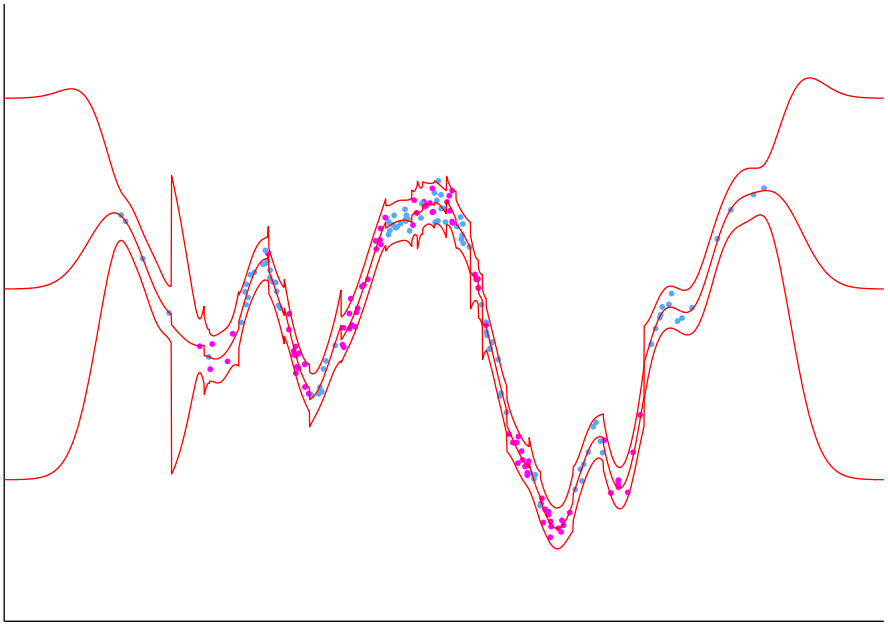
3.3 Results

The first thing to note is that PIC subsumes both the local GP approach and FI(T)C. By varying the number of inducing inputs and the size of the blocks we can obtain an approximation that is close to one or the other.

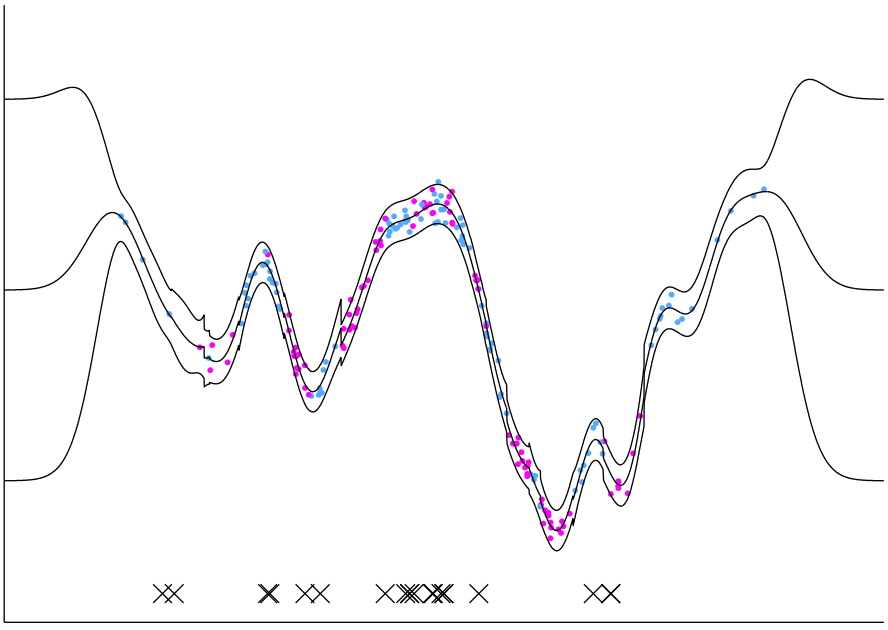
3.3.1 One dimensional examples

In a one dimensional example it is easy to form a perfect clustering into evenly sized clusters. This is what we did for PITC in figure 3.3b. However to more realistically show what might happen in higher dimensions, we apply the technique of random clustering as mentioned in section 3.2.3 to cluster points in the same 1D example of figure 3.5. In figure 3.5a we show what happens when you use the purely local GP, i.e. there are no inducing points. The general trend is pretty good, but there are some very big artifacts at the boundaries of the blocks, especially for blocks that just contain a few points. Figure 3.5b shows what happens when we introduce a few random inducing points, and use the PIC approximation. Where the inducing points are present the local boundary artifacts are smoothed over, and where inducing points are not present, the local prediction takes over. Essentially the global and local approximations complement each other to produce a predictive distribution very close to that of a full GP. Referring back to figure 3.3a, we see how much better PIC is doing compared to FI(T)C with the same inducing points.

The type of regime in which the combined PIC approach has a significant advantage over either one or the other can be seen by examining the failure modes as we did in section 3.1. Referring back to figure 3.1: FI(T)C fails for complex functions



(a) local GPs



(b) PIC

Figure 3.5: One dimensional example comparing local GPs and the PIC approximation. The inducing inputs for PIC are chosen randomly from the training input points. The blocks are denoted by alternating the colours of the data points, blue and magenta. Here the blocks were determined using random clustering.

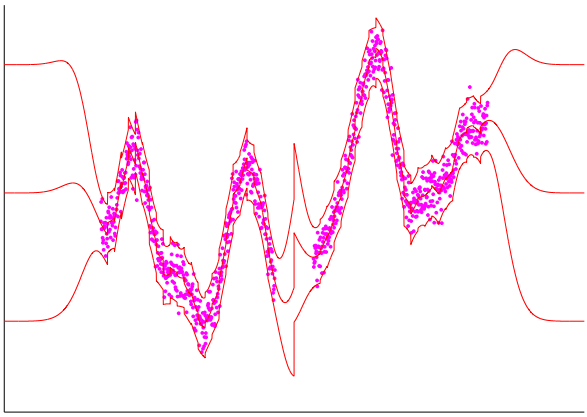
where we cannot afford to tile the space with inducing inputs; local GPs fail when we need to do extrapolation well. Figure 3.6 shows another illustrative 1D example where these problems are solved by the combined approach. In figure 3.6a the local GP approach works well apart from in the extrapolation between the two groups of data points. This can be completely fixed by the PIC approximation in figure 3.6b with the addition of a few well placed inducing points. The FI(T)C prediction based on these inducing points alone is shown in figure 3.6c. We see that the PIC predictor is a combination of the best parts of the local GP and FI(T)C predictors of (a) and (c). In order to do well using FI(T)C alone we would need to tile the space densely with inducing points.

In a real world example, to obtain the maximum advantage from PIC, it would be useful to have schemes to place the inducing points in optimal locations. We could attempt to maximise marginal likelihood as in the SPGP, or we could use simpler heuristics to place the inducing inputs well. We will have to leave a full evaluation of such procedures to future work.

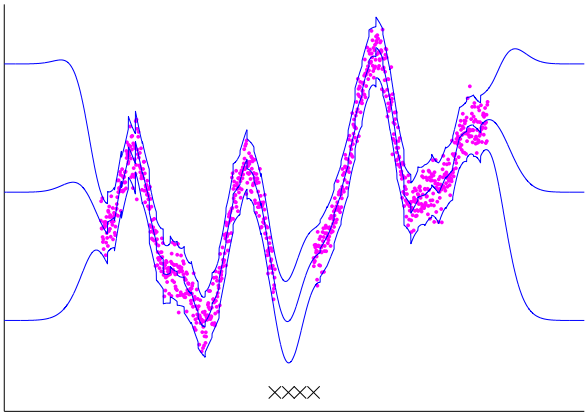
3.3.2 Real world examples

As real world examples we use the same three data sets we tested in section 2.5. We measure test set error as a function of computation time for the three methods: FI(T)C, local GPs, and PIC. Since it is not the goal of this chapter to investigate hyperparameter learning and inducing point selection, we simply use 'ground truth' hyperparameters obtained by training a GP on a large subset of the training data, and we use inducing points optimised as in the SPGP. The computation time reported is the combined precomputation and prediction time on the test sets. For local GPs and PIC, the time includes the extra clustering time, which was simply done by the random clustering method discussed in section 3.2.3. Points on the error/time plots of figure 3.7 were then obtained by varying the number of inducing points for FI(T)C, the number of blocks for local GPs, and both for PIC.

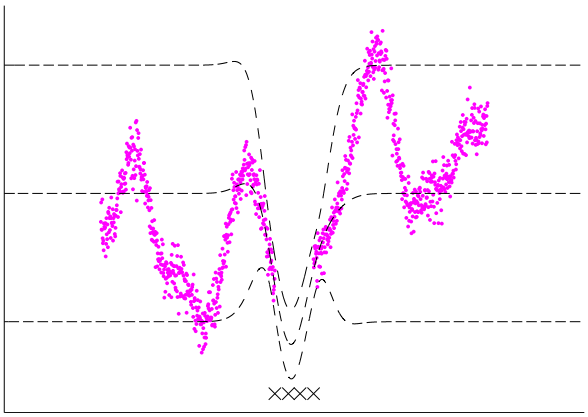
The three different data sets show three different behaviours. Firstly figures (a) and (b) show clearly that the local GP method is superior for `SARCOS`. Using only small block sizes, we can achieve an extremely low MSE and NLPD, comparable to FI(T)C with a large number of inducing inputs. The intuition for this is that `SARCOS` is a very complex nonlinear dataset similar in regime to figures 3.1b and 3.1c, and for the same reasons as there, the local GPs outperform FI(T)C. The combined PIC approximation also benefits from the blocking and performs well, but the inducing



(a) local GPs

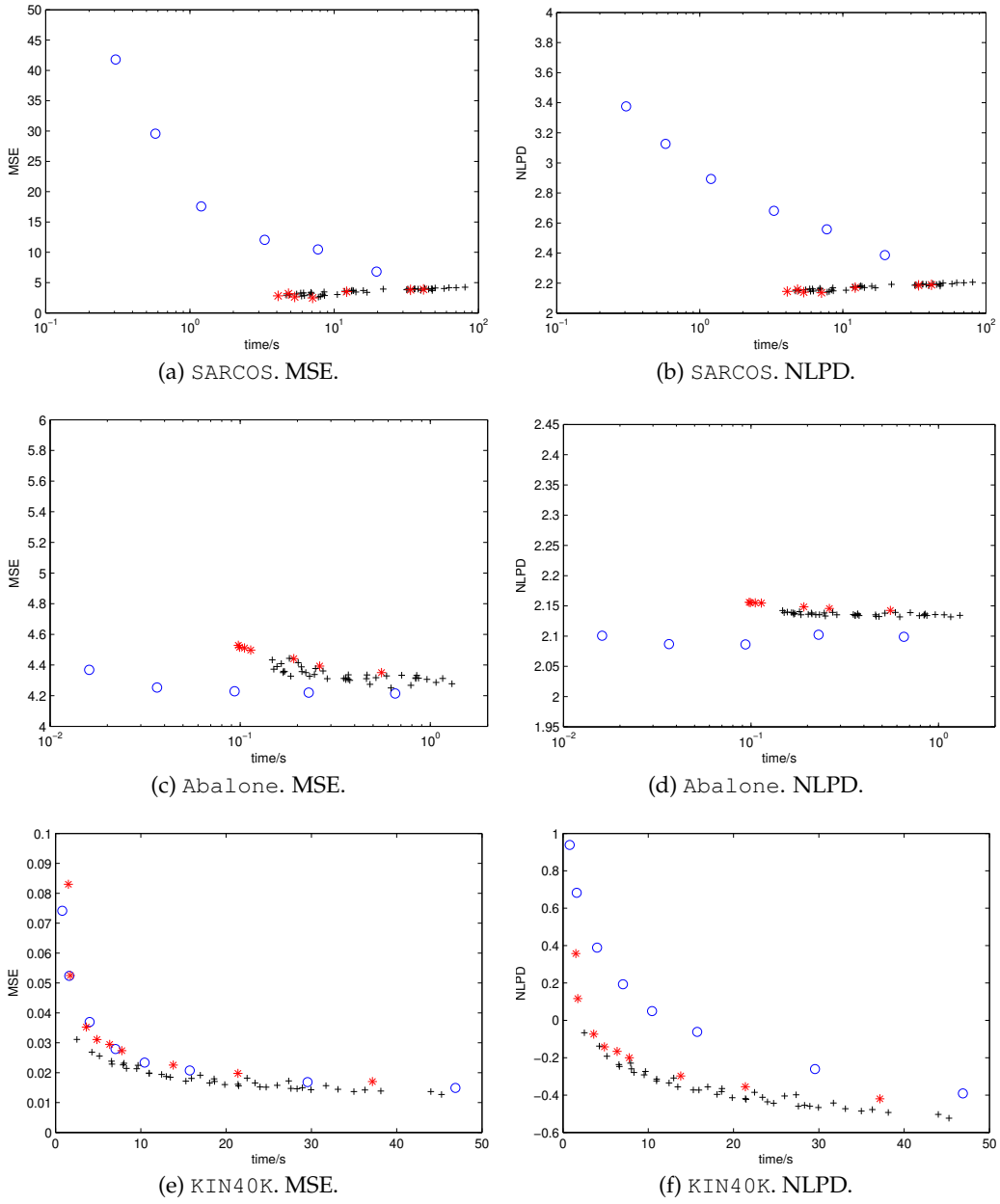


(b) PIC



(c) FI(T)C

Figure 3.6: 1D comparison of local, combined, and global GP approximations. Mean predictions and two standard deviation error lines are plotted, as black dashed lines for FI(T)C, red solid lines for local GPs, and blue solid lines for PIC. In (a) and (b) the blocks are not marked for clarity, because they are very small.



(a) SARCOS. MSE. (b) SARCOS. NLPD.

(c) Abalone. MSE. (d) Abalone. NLPD.

(e) KIN40K. MSE. (f) KIN40K. NLPD.

Figure 3.7: Test set error vs. computation time. Blue circles: FI(T)C, red stars: local GPs, black crosses: PIC. Points are obtained by varying the number of inducing points, or number of blocks, or both (for PIC).

points do not add any accuracy in this case.

Figures 3.7c and 3.7d show almost the opposite effect for `Abalone`. The FI(T)C approximation is able to achieve high accuracy using only a few inducing points, and so it is far faster than local GPs. The `Abalone` data set is much simpler than `SARCOS`, and so it is very well modelled by FI(T)C with only a few inducing points, in a similar way to figure 3.1a.

We show the results for `KIN40K` in figures 3.7e and 3.7f, on a slightly more densely sampled scale to highlight the differences in the methods. Figure 3.7e shows FI(T)C and local GPs performing very similarly in terms of MSE, with the combined PIC approach giving a small but significant gain. Figure 3.7f shows PIC and local GPs performing well in terms of NLPD error, with FI(T)C performing worse. This data set is clearly somewhere in between `SARCOS` and `Abalone` in complexity, in a regime where the combined PIC approximation is useful.

A further regime in which we might expect the local approach to perform less well is for higher dimensional input spaces, where the nearby points alone cannot provide enough information. We confirmed this hypothesis for the 106 dimensional `Temp` data set, which we use in chapter 4, where FI(T)C easily outperformed a local GP approach.

As we have seen, which approximation to use very much depends on the type of data. The advantage of PIC is that you are guarded against both failure modes of the individual local or global style approximations. We might expect further advantages for PIC when we select inducing points in conjunction with the blocks, but this is beyond the scope of this thesis.

3.4 Remarks

In this chapter we have developed a computationally efficient approximation that combines the advantages of the local regression/experts approach with the global inducing input based approach. From a theoretical point of view, PIC in some sense completes the sequence of approximations as set out in the framework of Quiñonero Candela and Rasmussen [2005].

From a practical point of view, we have explored the different types of regimes in which either the local or the global based approximations are more efficient, and we have demonstrated situations where the combined PIC method improves

upon both of these. In practice the PIC approximation allows a user to vary the number of clusters and the number of inducing inputs to find the best performance. There are several interesting future directions to pursue, for example to try to learn inducing inputs in relation to the clustering, perhaps by the maximization of the PIC marginal likelihood.

The original local GP expert approach of Rasmussen and Ghahramani [2002] was not designed with computational efficiency in mind, but rather to develop non-stationarity by allowing different regions to have different hyperparameters, e.g. lengthscales. It might be possible to develop a PIC-like model which allows different hyperparameters for the individual blocks, whilst still maintaining the naturally computationally efficient structure.

One important lesson from this chapter is not to assume that the global style of inducing point based approximation, which dominates the GP approximation literature, will necessarily be the best approach to every data set.

Chapter 4

Variable noise and dimensionality reduction

In this chapter we return to the SPGP we developed in section 2.2, and look at a few simple extensions that can be made. This chapter is based on Snelson and Ghahramani [2006b].

One limitation of the SPGP is that learning the pseudo-inputs becomes impractical for the case of a high dimensional input space. For M pseudo-inputs and a D dimensional input space we have a continuous $M \times D$ dimensional optimisation task. In this chapter we overcome this limitation by learning a projection of the input space into a lower dimensional space. The pseudo-inputs live in this low dimensional space and hence the optimisation problem is much smaller. This can be seen as performing supervised dimensionality reduction.

We also examine the ability of the SPGP to model data with input-dependent noise (heteroscedasticity). In section 2.2.1 we already hinted that the SPGP covariance function could be viewed as inherently more flexible than the original GP, with further advantages other than a purely computational device. Heteroscedastic regression is something that is very difficult to achieve with a standard GP without resorting to expensive sampling [Goldberg et al., 1998]. In this chapter we explore the capabilities of the SPGP for heteroscedastic regression tasks, and we develop a further extension of the model that allows an even greater degree of flexibility in this regard. We do this by learning individual uncertainty parameters for the pseudo-inputs.

4.1 Dimensionality reduction

The SPGP improves the accuracy of its approximation by adjusting the positions of the pseudo-inputs to fit the data well. However a limitation of this procedure is that whereas the standard GP only had a small number $|\boldsymbol{\theta}|$ of parameters to learn, the SPGP has a much larger number: $MD + |\boldsymbol{\theta}|$. Whilst we can adjust the number of pseudo-inputs M depending on our time available for computation, if we have a high dimensional (D) input space the optimisation is impractically large. In this section we address this problem by learning a low dimensional projection of the input space.

In order to achieve this dimensionality reduction we adapt an idea of Vivarelli and Williams [1999] to the SPGP. They replaced the ARD lengthscale hyperparameters $\boldsymbol{\lambda}$ in the SE-ARD covariance function equation (1.21) with a general positive definite matrix $\mathbf{W}$, in order to provide a richer covariance structure between dimensions:

$$K(\mathbf{x}, \mathbf{x}') = a^2 \exp\left[-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^\top \mathbf{W}(\mathbf{x} - \mathbf{x}')\right] . \tag{4.1}$$

$\mathbf{W}$ need not be totally general — it can be restricted to be low rank by decomposing it as $\mathbf{W} = \mathbf{P}^\top\mathbf{P}$, where $\mathbf{P}$ is a $G \times D$ matrix and $G < D$. This is clearly exactly equivalent to making a linear low dimensional projection of each data point $\mathbf{x}^{\text{new}} = \mathbf{P}\mathbf{x}$, and has the covariance function:

$$K(\mathbf{x}, \mathbf{x}') = a^2 \exp\left[-\frac{1}{2}\big(\mathbf{P}(\mathbf{x} - \mathbf{x}')\big)^\top \mathbf{P}(\mathbf{x} - \mathbf{x}')\right] . \tag{4.2}$$

We use exactly this covariance structure for dimensionality reduction in the SPGP. However the SPGP covariance function equation (2.13) is constructed from covariances between data-points and pseudo-inputs $K(\mathbf{x}, \bar{\mathbf{x}})$, and from the covariances of the pseudo-inputs themselves $K(\bar{\mathbf{x}}, \bar{\mathbf{x}}')$. The projection means that we only need to consider the pseudo-inputs living in the reduced dimensional (G) space. Finally we therefore use the following covariances:

$$K(\mathbf{x}, \bar{\mathbf{x}}) = a^2 \exp\left[-\frac{1}{2}(\mathbf{P}\mathbf{x} - \bar{\mathbf{x}})^\top(\mathbf{P}\mathbf{x} - \bar{\mathbf{x}})\right] \tag{4.3}$$

$$K(\bar{\mathbf{x}}, \bar{\mathbf{x}}') = a^2 \exp\left[-\frac{1}{2}(\bar{\mathbf{x}} - \bar{\mathbf{x}}')^\top(\bar{\mathbf{x}} - \bar{\mathbf{x}}')\right] , \tag{4.4}$$

where $\bar{\mathbf{x}}$ is a G dimensional vector. Note that it is not necessary to introduce ex-

tra lengthscale hyperparameters for the pseudo-inputs themselves because they would be redundant. The pseudo-inputs are free to move, and the projection matrix $\mathbf{P}$ can scale the real data points arbitrarily to 'bring the data to the pseudo-inputs'.

Setting aside computational issues for the moment it is worth noting that even with $G < D$ the covariance of equation (4.2) may be more suitable for a particular data set than the standard ARD covariance equation (1.21), because it is capable of mixing dimensions together. However this is not our principal motivation. The SPGP with ARD covariance has $MD + D + 2$ parameters to learn, while with dimensionality reduction it has $(M + D)G + 2$. Clearly whether this is a smaller optimisation space depends on the exact choices for M and G, but we will show on real problems in section 4.3 that G can often be chosen to be very small.

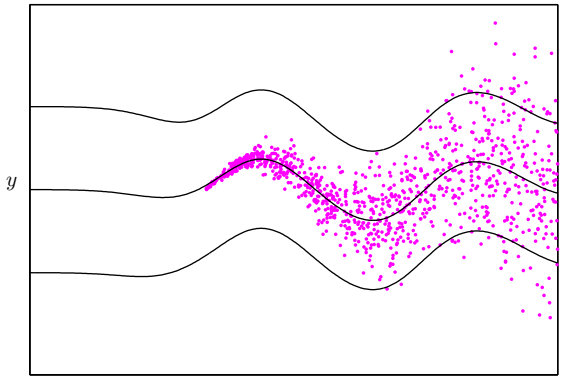
To clarify: the training procedure for the dimensionality reduced SPGP (SPGP+DR) is to maximise the marginal likelihood of equation (2.9) using gradients with respect to the pseudo-inputs $\bar{\mathbf{X}}$, the projection matrix $\mathbf{P}$, the amplitude a, and the noise σ^2.[1] The procedure can be considered to perform supervised dimensionality reduction — an ideal linear projection is learnt for explaining the target data. This is in contrast to the many unsupervised dimensionality reduction methods available (e.g. PCA), which act on the inputs alone.

4.2 Variable noise

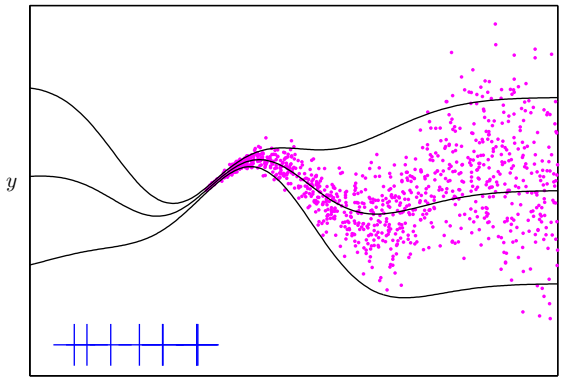
Although GPs are very flexible regression models, they are still limited by the form of the covariance function. To model more complex data than simple stationary processes we require more complex covariance functions. There has been some work done to develop particular classes of nonstationary covariance functions [Higdon et al., 1999, Paciorek and Schervish, 2004], especially with regard to variable lengthscales. Goldberg et al. [1998] modelled input dependent (variable) noise using an additional GP to represent the noise process, but this requires expensive sampling.

Although the SPGP was not originally designed for modelling nonstationarity or variable noise, the SPGP covariance function equation (2.13) is a particular type of nonstationary covariance that arises naturally from the low rank construction. It is therefore interesting to evaluate whether the SPGP can actually handle some type

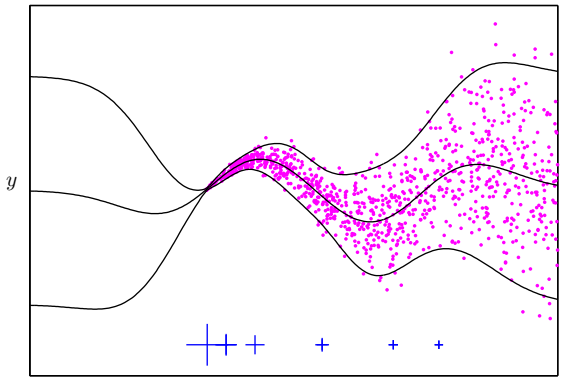
[1] The gradient derivations are a minor modification of the SPGP gradients presented in appendix C.



(a) GP



(b) SPGP



(c) SPGP+HS. The size of a blue cross is a function of the inverse uncertainty associated with that pseudo-input.

Figure 4.1: The predictive distributions after training on a synthetic heteroscedastic data set are shown for the standard GP, SPGP, and SPGP+HS. $\mathbf{x}$ locations of pseudo-inputs are shown as blue crosses (the y positions are not meaningful).

of data better than the original GP. In other words we can take the view of the SPGP not as an approximation, but as a more flexible GP in its own right.

Figure 4.1 shows a simple 1D data set with a variable noise level. Figure 4.1a shows the standard GP solution with SE covariance, with the hyperparameters optimised. The SE covariance is stationary with a global noise level, so the fit to the data is poor. Figure 4.1b shows the SPGP solution to the same data set, where the pseudo-inputs have also been optimised.[2] Although the SPGP also has a single global noise level σ^2, the predictive variances will only drop to this level in regions close to pseudo-inputs. Away from pseudo-inputs the predictive variance rises to $a^2 + \sigma^2$ because correlations cannot be modelled in these regions. During training, the SPGP can adjust its pseudo-inputs to take advantage of this by-product of the non-stationarity of the sparse covariance function. By shifting all the pseudo-inputs to the left in figure 4.1b, the SPGP models the variable noise vastly better than the standard GP does in figure 4.1a.

However the SPGP solution in figure 4.1b is still not entirely satisfactory. By moving all the pseudo-inputs to the left to model the variable noise, the correlations that are still present in the data towards the right cannot be modelled. There are no pseudo-inputs present there to handle the correlations. We propose a further extension to the SPGP model to get around these problems and improve the modelling capabilities of the SPGP.

We introduce extra uncertainties associated with each pseudo-point. This means altering the covariance of the pseudo-points in the following way:

$$\mathbf{K}_M \rightarrow \mathbf{K}_M + \mathrm{diag}(\mathbf{h}) \,, \tag{4.5}$$

where $\mathbf{h}$ is a positive vector of uncertainties to be learnt. These uncertainties allow the pseudo-inputs to be gradually 'switched off' as the uncertainties are increased. If $h_m = 0$ then that particular pseudo-input behaves exactly as in the SPGP. As h_m grows, that pseudo-input has less influence on the predictive distribution. This means that the pseudo-inputs' role is not 'all or nothing' as it was in the SPGP. A pseudo-input can be partly turned off to allow a larger noise variance in the prediction whilst still modeling correlations in that region. As $h_m \rightarrow \infty$, the pseudo-input is totally ignored. We refer to this heteroscedastic extension as the SPGP+HS.

[2] It should be said that there are local optima in this problem, and other solutions looked closer to the standard GP. We ran the method 5 times with random initialisations. All runs had higher likelihood than the GP; the one with the highest likelihood is plotted.

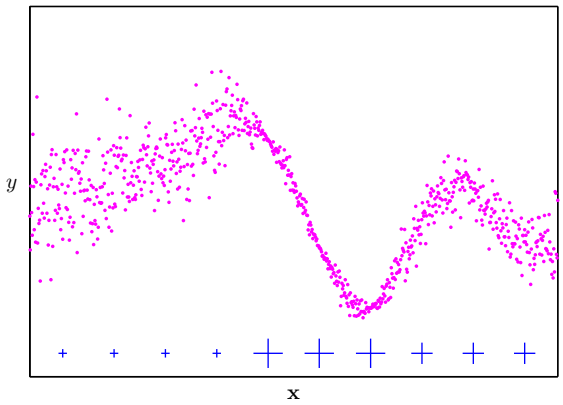


Figure 4.2: Sample data drawn from the SPGP+HS marginal likelihood for a particular choice of pseudo-input locations (blue crosses), hyperparameters, and pseudo uncertainties. The size of a blue cross is related to the inverse of the uncertainty associated to that pseudo-input.

Figure 4.2 shows sample data drawn from the marginal likelihood of the SPGP+HS model, where the components of $\mathbf{h}$ have been set to three different values. These values are indicated by the sizes of the blue crosses representing the pseudo-inputs – a larger cross means a lower uncertainty. Notice the different noise regimes in the generated data.

To train the model, we follow the same procedure as earlier – we include $\mathbf{h}$ as extra parameters to be learned by gradient based maximum likelihood. We tested this on the synthetic data of figure 4.1, and the predictive distribution is shown in figure 4.1c. Now the pseudo-inputs do not all have a tendency to move to the left, but rather the right-most ones can partly turn themselves off, enabling the correlations present towards the right of the data set to be modelled very well.

Our visual intuition is borne out when we look at negative log predictive density (NLPD, with smallest being best) and mean squared error (MSE) scores on a withheld test set for this data set. These are shown below. On NLPD the GP does badly, the SPGP better, but the new method SPGP+HS does best of all, because it models the noise process well. The SPGP is not so good on MSE, because it is forced to sacrifice modeling the correlations on the right side of the data set.

Method	NLPD	MSE
GP	3.09	14.16
SPGP	2.74	16.98
SPGP+HS	2.57	14.37

Data set	Temp	SO2	Synthetic
Dimension (D)	106	27	1
Training set size	7117	15304	256
Validation set size	3558	7652	128
Test set size	3560	7652	1024

Table 4.1: Properties of the competition data sets

4.3 Results

We decided that an ideal test bed for the SPGP and its extensions considered in this chapter would be the data sets of the WCCI-2006 Predictive Uncertainty in Environmental Modeling Competition, run by Gavin Cawley.[3] Some of the data sets have a fairly large number of dimensions (>100), and Gavin Cawley suggested that heteroscedastic modelling techniques are likely to be necessary to perform well on this environmental data. The competition required probabilistic predictions and was to be scored by NLPD on a withheld test set. Unfortunately by the time the competition closed we had only made a submission on one data set (Temp), on which we scored first place. However since then, we have experimented further with our methods on the data sets, and Gavin Cawley kindly agreed to evaluate several more submissions on the test sets, which we report here.

Some properties of the data sets we considered are shown in table 4.1.[4] Most of the results shown in the following sections are obtained by training on the training set *only* and evaluating on the validation set. This is because the test set targets are not publicly available. These results serve as useful comparisons between our different methods. However some results were obtained by training on the training *and* validation sets, before being sent to Gavin Cawley for evaluation on the withheld test set. With these results we can see how our methods fare against a host of competing algorithms whose performance is shown on the competition web site[3].

[3] http://theoval.sys.uea.ac.uk/competition/

[4] The competition also had a further data set Precip, which we have not considered. This is because a histogram of the targets showed a very large spike at exactly zero, which we felt would be best modeled by a hierarchy of a classifier and regressor. The SO2 data set was somewhat similar but not nearly so extreme, so here we could get away with a $\log(y + a)$ preprocessing transform.

4.3.1 `Temp` data set

The targets of the `Temp` data set are maximum daily temperature measurements, and are to be predicted from 106 input variables representing large-scale circulation information. We conducted a series of experiments to see how dimensionality reduction performed, and these are presented in table 4.2a. To compare, we ran the standard SPGP with no dimensionality reduction (which took a long time to train). Although the dimensionality reduction did not produce better performance than the standard SPGP, we see that we are able to reduce the dimensions from 106 to just 5 with only a slight loss in accuracy. The main thing to notice is the training and test times, where reducing the dimension to 5 has sped up training and testing by an order of magnitude over the standard SPGP.[5] Clearly some care is needed in selecting the reduced dimension G. If it is chosen too small then the representation is not sufficient to explain the targets well, and if it is too large then there are probably too many parameters in the projection $\mathbf{P}$ to be fit from the data. Cross-validation is a robust way of selecting G.

Of course a much simpler way of achieving a linear projection of the input space is to do PCA before using the standard SPGP on the smaller dimensional space. In this case the projection is made completely ignoring the target values. The idea behind the SPGP+DR is that the target values should help in choosing the projection in a supervised manner, and that better performance should result. To test this we used PCA to reduce the dimension to 5, before using the SPGP. The results are shown in table 4.2a as well. We see that the SPGP+PCA performs significantly worse than the SPGP+DR both on NLPD and MSE scores. The equivalent reduction to 5 dimensions using the SPGP+DR does not cost too much more than the PCA method either, in terms of training or test time.

Our entry to the competition was made by using the SPGP+DR with dimensionality reduction to $G = 5$, and $M = 10$ pseudo-inputs. We trained on the training set and validation sets, and obtained test set NLPD of 0.0349 and MSE of 0.066, which placed us first place on the `Temp` data set on both scores (see the competition web site[3]). This provides justification that the SPGP+DR is a very competitive algorithm, managing to beat other entries from MLPs to Support Vector Regression, and requiring little training and test time.

We then decided to investigate the heteroscedastic capabilities of the SPGP, and the

[5] The actual training and test times are affected not just by the number of parameters to be optimised, but also by details of the gradient calculations, where memory/speed trade-offs have to be made. We have tried to implement both versions efficiently.

SPGP+HS extension proposed in section 4.2. Table 4.2a reports the performance of the SPGP+HS when combined with a dimensionality reduction to $G = 5$. In this case the extension did not perform better than the standard SPGP. However, it could be that either the `Temp` data set is not particularly heteroscedastic, or that the SPGP itself is already doing a good job of modelling the variable noise. To investigate this we trained a standard GP on a small subset of the training data of 1000 points. We compared the performance on the validation set to the SPGP ($M = 10$) trained on the same 1000 points. Since the SPGP is an approximation to the GP, naïvely one would expect it to perform worse. However the SPGP (NLPD 0.16, MSE 0.08) significantly outperformed the GP (NLPD 0.56, MSE 0.11). The SPGP does a good job of modelling heteroscedasticity in this data set — something the GP cannot do. The SPGP+HS proposed in section 4.2 could do no better in this case.

Of course the gradient optimisation of the likelihood is a difficult non-convex problem, with many local minima. However the performance seems fairly stable to repeated trials, with relatively low variability. For initialising $\mathbf{P}$ we used a random set of orthogonal projections. We then initialised $\bar{\mathbf{X}}$ by projecting a random subset of the input data using $\mathbf{P}$.

4.3.2 SO2 data set

For the `SO2` data set the task is to forecast the concentration of SO_2 in an urban environment twenty-four hours in advance, based on current SO_2 levels and meteorological conditions. The results presented in table 4.2b show a similar story to those on the `Temp` data set. In this case there are a very large number of data points, but a smaller number of dimensions $D = 27$. Although in this case it is perfectly feasible to train the SPGP in a reasonable time without dimensionality reduction, we decided to investigate its effects. Again we find that we can achieve a significant speed up in training and testing for little loss in accuracy. When we compare reducing the dimension to 5 using PCA to using the SPGP+DR, we again find that PCA does not perform well. There is certainly information in the targets which is useful for finding a low dimensional projection.

We also tested the SPGP+HS (with no dimensionality reduction), and we see similar, perhaps slightly better, performance than the standard SPGP. We therefore decided to compile a competition submission using the SPGP+HS, training on the training and validation sets, to give to Gavin Cawley for evaluation on the test set.

	Validation		Time /s	
Method	NLPD	MSE	Train	Test
SPGP	0.063	0.0714	4420	0.567
+DR 2	0.106(2)	0.0754(5)	180(10)	0.043(1)
+DR 5	0.071(8)	0.0711(7)	340(10)	0.061(1)
+DR 10	0.112(10)	0.0739(12)	610(20)	0.091(1)
+DR 20	0.181(5)	0.0805(7)	1190(50)	0.148(1)
+DR 30	0.191(6)	0.0818(7)	1740(50)	0.206(3)
+HS,DR 5	0.077(5)	0.0728(3)	360(10)	0.062(3)
+PCA 5	0.283(1)	0.1093(1)	200(10)	0.047(2)

(a) `Temp`. $M = 10$ pseudo-inputs used.

	Validation		Time /s	
Method	NLPD	MSE	Train	Test
SPGP	4.309(2)	0.812(1)	890(40)	0.723(6)
+DR 2	4.349(1)	0.814(2)	80(5)	0.165(2)
+DR 5	4.325(1)	0.815(4)	160(5)	0.233(1)
+DR 10	4.323(3)	0.809(5)	290(15)	0.342(2)
+DR 15	4.341(3)	0.803(6)	400(10)	0.458(5)
+DR 20	4.350(3)	0.807(2)	530(15)	0.562(4)
+HS	4.306(1)	0.809(2)	860(30)	0.714(4)
+PCA 5	4.395(1)	0.855(2)	170(10)	0.255(3)

(b) `SO2`. $M = 20$ pseudo-inputs used.

Table 4.2: Results showing NLPD and MSE score (smaller is better) on the validation sets of two competition data sets, `Temp` and `SO2`. Times to train on the training set and test on the validation set are also shown. SPGP indicates the standard SPGP, +DR G indicates dimensionality reduction to dimension G, +HS indicates the heteroscedastic extension to the SPGP has been used, +PCA G means PCA to dimension G before standard SPGP. Where possible trials were repeated 5 times and standard errors in the means have been reported – numbers in parentheses refer to errors on final digit(s).

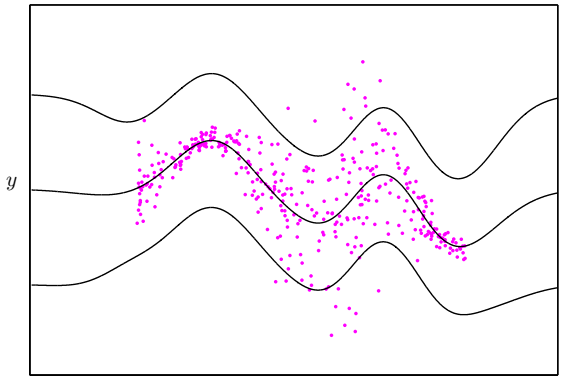
We scored an NLPD of 4.28, and MSE of 0.82. Had we managed to submit this entry to the competition before the deadline, we would have been placed second on this data set, again showing the competitiveness of our methods. This time when a GP is compared to the SPGP on a subset of training data of size 1000, the performance is very similar, leading us to suspect that there is not too much to be gained from heteroscedastic methods on this data.

4.3.3 `Synthetic` data set

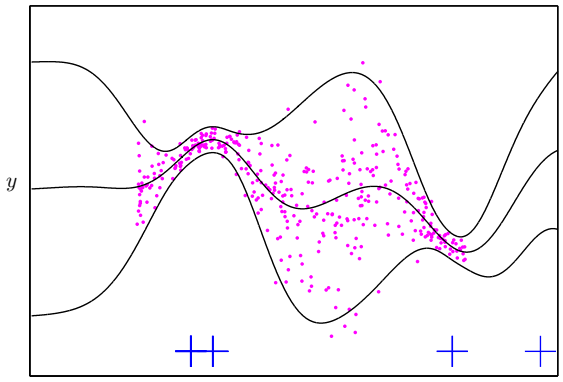
The final competition data set is a small 1D data set particularly generated to test heteroscedastic methods. Figure 4.3 shows plots of the data, and the predictive distributions obtained using a GP, a standard SPGP, and the SPGP+HS. These plots show again that the SPGP itself is very capable of modelling certain types of heteroscedasticity. The SPGP+HS creates a very similar predictive distribution, but is able to refine it slightly by using more pseudo-inputs to model the correlations. Both of these look much better than the GP. We sent submissions of all three methods to Gavin Cawley for him to evaluate on the test set. Either the SPGP (NLPD 0.380, MSE 0.571), or the SPGP+HS (NLPD 0.383, MSE 0.562), would have been placed first under NLPD score. In contrast the GP (NLPD 0.860, MSE 0.573) performed poorly on NLPD score as expected. So again we have further evidence that the SPGP can be a very good model for heteroscedastic noise alone. The SPGP+HS extension may improve matters in certain circumstances — here it actually seems to slightly improve MSE over the SPGP, just as we saw for the synthetic data set of section 4.2.

4.3.4 Motorcycle data set

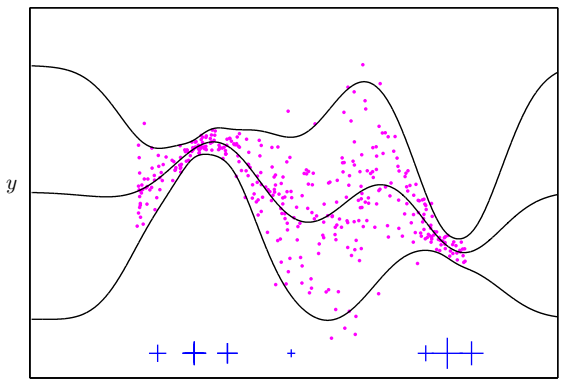
We finally tested our methods on a data set from Silverman [1985] — data from a simulated motorcycle accident. This is a very small (133 points) 1D data set, which is known for its non-stationarity. We removed 10 random points for testing, trained on the remainder, repeated the procedures 100 times, and the results are shown below. Here we have to report a failure of our methods. The SPGP does not do much better than a standard GP because it cannot deal with this degree of non-stationarity. The SPGP+HS fails completely because it overfits the data badly. The reason for the overfitting is a bad interaction between all the hyperparameters, where the lengthscale is driven too small, and the pseudo-noise parameters allow



(a) GP



(b) SPGP



(c) SPGP+HS

Figure 4.3: The predictive distributions on the competition `Synthetic` data set are shown for the standard GP, SPGP, and SPGP+HS.

the predictive distribution to pinch in on some individual training data points. Essentially, for such a small data set, we have allowed too much flexibility in our covariance function for all the hyperparameters to be fitted using maximum likelihood.

Method	NLPD	MSE
GP	4.6	2.6×10^2
SPGP	4.5	2.6×10^2
SPGP+HS	11.2	2.8×10^2

4.4 Remarks

In this chapter we have demonstrated the capabilities of the SPGP and its extensions for modelling data sets with a wide range of properties. The original SPGP could handle data sets with a large number of data points. However it was impractical for data sets with high dimensional input spaces. By learning a linear projection we achieve supervised dimensionality reduction, and greatly speed up the original SPGP for little loss in accuracy. We also have shown the advantage of this supervised dimensionality reduction over the obvious unsupervised linear projection, PCA.

We have also investigated the use of the SPGP for modelling heteroscedastic noise. We find that the original SPGP is a surprisingly good model for heteroscedastic noise, at least in the predictive uncertainty competition data sets. We have also developed an extension of the SPGP more specifically designed for heteroscedastic noise, which although not improving performance on the competition data sets, should provide advantages for some types of problem. However the increase in flexibility to the covariance function can cause overfitting problems for certain data sets, and it is future work to improve the robustness of the method. We could certainly try various forms of regularization and even full Bayesian inference.

Since these extensions are based on the efficient SPGP covariance function, computational tractability is retained first and foremost.

Chapter 5

Warped Gaussian processes

Chapters 2 and 3 were concerned with purely computational improvements to GPs. Chapter 4 was based on computationally efficient methods, but also concerned the development of more flexible GP models. In this chapter we continue this theme by developing another method to increase the flexibility of GPs. This chapter is based on Snelson et al. [2004].

In their simplest form GPs are limited by their assumption that the observation data is distributed as a multivariate Gaussian, with Gaussian noise. Often it is unreasonable to assume that, in the form the data is obtained, the noise will be Gaussian, and the data well modelled as a GP. For example, the observations may be positive quantities varying over many orders of magnitude, where it makes little sense to model these quantities directly assuming homoscedastic Gaussian noise. In these situations it is standard practice in the statistics literature to take the $\log$ of the data. Then modelling proceeds assuming that this transformed data has Gaussian noise and will be better modelled by the GP. The $\log$ is just one particular transformation that could be done; there is a continuum of transformations that could be applied to the observation space to bring the data into a form well modelled by a GP. Making such a transformation should really be a full part of the probabilistic modelling; it seems strange to first make an ad-hoc transformation, and then use a principled Bayesian probabilistic model.

In this chapter we show how such a transformation or 'warping' of the observation space can be made entirely automatically, fully encompassed into the probabilistic framework of the GP. The *warped GP* makes a transformation from a latent space to the observation, such that the data is best modelled by a GP in the latent space.

It can also be viewed as a generalisation of the GP, since in observation space it is a non-Gaussian process, with non-Gaussian and asymmetric noise in general.

5.1 Warping the observation space

In this section we present a method of warping the observation space through a nonlinear monotonic function to a latent space, whilst retaining the full probabilistic framework to enable learning and prediction to take place consistently. Let us consider a vector of latent observations $\mathbf{z}$ and suppose that this vector *is* modelled by a GP with Gaussian noise of variance σ^2:

$$p(\mathbf{z}|\boldsymbol{\theta}) = \mathcal{N}(\mathbf{0}, \mathbf{C}) \,, \tag{5.1}$$

where $\mathbf{C}(\boldsymbol{\theta}) = \mathbf{K} + \sigma^2\mathbf{I}$, and $\boldsymbol{\theta}$ is the vector of hyperparameters. Alternatively we may consider the negative log marginal likelihood $\mathcal{L}_\mathbf{z}$ equation (1.20):

$$\begin{aligned} \mathcal{L}_\mathbf{z} &= -\log p(\mathbf{z}|\boldsymbol{\theta}) \\ &= \frac{1}{2}\log\det\mathbf{C} + \frac{1}{2}\mathbf{z}^\top\mathbf{C}^{-1}\mathbf{z} + \frac{N}{2}\log(2\pi) \,. \end{aligned} \tag{5.2}$$

Now we make a transformation from the true observation space to the latent space by mapping each observation through the same monotonic function f,

$$z = f(y; \boldsymbol{\Psi}) \,, \tag{5.3}$$

where $\boldsymbol{\Psi}$ parameterises the transformation.[1] We require f to be monotonic and mapping on to the whole of the real line; otherwise probability measure will not be conserved in the transformation, and we will not induce a valid distribution over the observations $\mathbf{y}$. Including the Jacobian term that takes the transformation into account, the negative log likelihood $\mathcal{L}$ now becomes:

$$\begin{aligned} \mathcal{L} &= -\log p(\mathbf{y}|\boldsymbol{\theta}, \boldsymbol{\Psi}) \\ &= \frac{1}{2}\log\det\mathbf{C} + \frac{1}{2}\mathbf{f}(\mathbf{y})^\top\mathbf{C}^{-1}\mathbf{f}(\mathbf{y}) - \sum_{n=1}^{N}\log\frac{\partial f(y)}{\partial y}\bigg|_{y_n} + \frac{N}{2}\log(2\pi) \,, \end{aligned} \tag{5.4}$$

[1] We could have defined the transformation the other way around. The choice comes down to whether you want to evaluate the inverse function in training or for prediction. With this choice, the inverse function is needed for prediction (see section 5.1.2). We will not in general have an analytic form for the inverse, so the decision slightly affects the training vs. test computation times.

where $\mathbf{f}(\mathbf{y})$ is the vector $[f(y_1), f(y_2), \ldots, f(y_N)]$.

5.1.1 Training the warped GP

Learning in this extended model is achieved by simply taking derivatives of the negative log likelihood function equation (5.4) with respect to both $\boldsymbol{\theta}$ and $\boldsymbol{\Psi}$ parameter vectors, and using a gradient optimisation method to compute maximum likelihood parameter values. In this way the form of both the covariance matrix and the nonlinear transformation are learnt simultaneously under the same probabilistic framework. Since the computational limiter to a GP is inverting the covariance matrix, adding a few extra parameters into the likelihood is not really costing us anything. All we require is that the derivatives of f are easy to compute (both with respect to y and $\boldsymbol{\Psi}$), and that we don't introduce so many extra parameters that we have problems with over-fitting. Of course a prior over both $\boldsymbol{\theta}$ and $\boldsymbol{\Psi}$ may be included to compute a MAP estimate, or the parameters could be integrated out using a sampling technique with associated extra cost.

5.1.2 Predictions with the warped GP

For a particular setting of the covariance function hyperparameters $\boldsymbol{\theta}$ (for example $\boldsymbol{\theta}_{\mathrm{ML}}$), in *latent* variable space the predictive distribution at a new point is just as for a regular GP (equation (1.19)), except the observations have been passed through the nonlinearity f:

$$p(z_*|\mathbf{y}, \boldsymbol{\theta}, \Psi) = \mathcal{N}\left(\mu_*^z, (\sigma_*^z)^2\right) , \tag{5.5a}$$

$$\begin{aligned} \mu_*^z &= \mathbf{K}_{*N}[\mathbf{K}_N + \sigma^2\mathbf{I}]^{-1}\mathbf{z} \\ &= \mathbf{K}_{*N}[\mathbf{K}_N + \sigma^2\mathbf{I}]^{-1}\mathbf{f}(\mathbf{y}) , \end{aligned} \tag{5.5b}$$

$$(\sigma_*^z)^2 = K_* - \mathbf{K}_{*N}[\mathbf{K}_N + \sigma^2\mathbf{I}]^{-1}\mathbf{K}_{N*} + \sigma^2 .$$

To find the predictive distribution in the observation space we pass that Gaussian back through the nonlinear warping function, giving

$$p(y_*|\mathbf{y}, \boldsymbol{\theta}, \boldsymbol{\Psi}) = \frac{f'(y_*)}{\sqrt{2\pi(\sigma_*^z)^2}} \exp\left[-\frac{1}{2}\left(\frac{f(y_*) - \mu_*^z}{\sigma_*^z}\right)^2\right] . \tag{5.6}$$

The shape of this distribution depends on the form of the warping function f, but in general it may be asymmetric and multimodal.

If we require a point prediction to be made, rather than the whole distribution over y_*, then the value we will predict depends on our loss function. If our loss function is absolute error, then the median of the distribution should be predicted, whereas if our loss function is squared error, then it is the mean of the distribution. For a standard GP where the predictive distribution is Gaussian, the median and mean lie at the same point. For the warped GP in general they are at different points. The median is particularly easy to calculate:

$$y_*^{\text{med}} = f^{-1}(\mu_*^z) \,. \tag{5.7}$$

Notice we need to compute the inverse warping function. In general we are unlikely to have an analytical form for f^{-1}, because we have parameterised the function in the opposite direction. However since we have access to derivatives of f, a few iterations of Newton's method with a good enough starting point is enough.

It is often useful to give an indication of the shape and range of the distribution by giving the positions of various 'percentiles'. For example we may want to know the positions of '2σ' either side of the median so that we can say that approximately 95% of the density lies between these bounds. These points in observation space are calculated in exactly the same way as the median - simply pass the values through the inverse function:

$$y_*^{\text{med}\pm 2\sigma} = f^{-1}(\mu_*^z \pm 2\sigma_*^z) \,. \tag{5.8}$$

To calculate the predictive mean, we need to integrate y_* over the density of equation (5.6). Rewriting this integral back in latent space we get:

$$\begin{aligned} \mathcal{E}[y_*] &= \int \mathrm{d}z f^{-1}(z) \mathcal{N}_z\big(\mu_*^z, (\sigma_*^z)^2\big) \\ &= \mathcal{E}[f^{-1}] \,. \end{aligned} \tag{5.9}$$

This is a simple one dimensional integral under a Gaussian density, so Gauss-Hermite quadrature [e.g. Press et al., 1992] may be used to accurately compute it with a weighted sum of a small number of evaluations of the inverse function f^{-1} at appropriate places.

5.1.3 Choosing a monotonic warping function

We wish to design a warping function that will allow for complex transformations, but we must constrain the function to be monotonic. There are various ways to do

this, an obvious one being a neural-net style sum of $\tanh$ functions,

$$f(y;\boldsymbol{\Psi}) = \sum_{i=1}^{I} a_i \tanh\left(b_i(y+c_i)\right) \qquad a_i, b_i \geq 0 \quad \forall i \;, \tag{5.10}$$

where $\boldsymbol{\Psi} = \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$. This produces a series of smooth steps, with $\mathbf{a}$ controlling the size of the steps, $\mathbf{b}$ controlling their steepness, and $\mathbf{c}$ their position. Of course the number of steps I needs to be set, and that will depend on how complex a function one wants. The derivatives of this function with respect to either y, or the warping parameters $\boldsymbol{\Psi}$, are easy to compute. In the same spirit, sums of error functions, or sums of logistic functions, would produce a similar series of steps, and so these could be used instead.

The problem with using equation (5.10) as it stands is that it is bounded; the inverse function $f^{-1}(z)$ does not exist for values of z outside the range of these bounds. As explained earlier, this will not lead to a proper density in y space, because the density in z space is Gaussian, which covers the whole of the real line. We can fix this up by using instead:

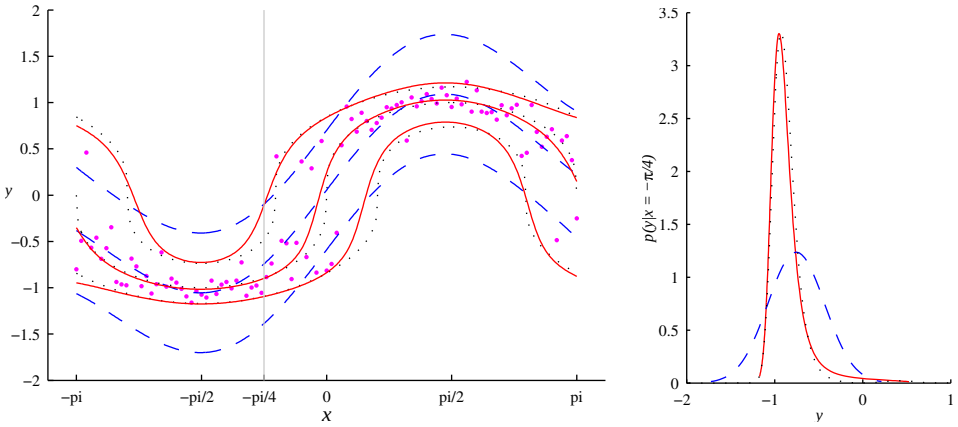
$$f(y;\boldsymbol{\Psi}) = y + \sum_{i=1}^{I} a_i \tanh\left(b_i(y+c_i)\right) \qquad a_i, b_i \geq 0 \quad \forall i \;. \tag{5.11}$$

which has linear trends away from the $\tanh$ steps. In doing so, we have restricted ourselves to only making warping functions with $f' \geq 1$, but because the amplitude of the covariance function is free to vary, the *effective* gradient can be made arbitrarily small by simply making the range of the data in the latent space arbitrarily big.

A more flexible system of linear trends may be made by including, in addition to the neural-net style function equation (5.10), some functions of the form:

$$g(y) = \frac{1}{\beta}\log\left[e^{\beta m_1(y-d)} + e^{\beta m_2(y-d)}\right] \qquad m_1, m_2 \geq 0 \;. \tag{5.12}$$

This function effectively splices two straight lines of gradients m_1 and m_2 smoothly together with a 'curvature' parameter β, and at position d. The sign of β determines whether the join is convex or concave.



(a) The triplets of lines represent the median, and 2σ percentiles in each case

(b) Probability densities at $x = -\pi/4$; i.e. a cross section through (a) at the solid grey line

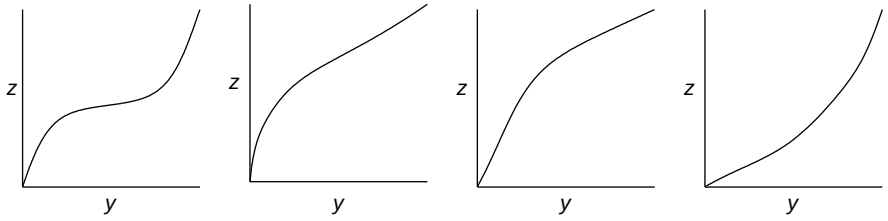
Figure 5.1: A 1D regression task. The black dotted lines show the true generating distribution, the blue dashed lines show a GP's predictions, and the red solid lines show the warped GP's predictions.

5.2 A 1D regression task

A simple 1D regression task was created to show a situation where the warped GP should, and does, perform significantly better than the standard GP. 101 points, regularly spaced from $-\pi$ to π on the x axis, were generated with Gaussian noise about a sine function. These points were then warped through the function $y = z^{1/3}$, to arrive at the dataset $\mathbf{y}$ which is shown as the dots in figure 5.1a.

A GP and a warped GP were trained independently on this dataset using a conjugate gradient minimisation procedure and randomly initialised parameters, to obtain maximum likelihood parameters. For the warped GP, the warping function equation (5.11) was used with just two $\tanh$ functions. For both models the SE covariance function equation (1.6) was used. Hybrid Monte Carlo was also implemented to integrate over all the parameters, or just the warping parameters (much faster since no matrix inversion is required with each step), but with this data set (and the real data sets of section 5.3) no significant differences were found from ML.

Predictions from the GP and warped GP were made, using the ML parameters. The predictions made were the median and 2σ percentiles in each case, and these are plotted as triplets of lines on figure 5.1a. The predictions from the warped GP are



(a) `Sine` (b) `Creep` (c) `Abalone` (d) `Ailerons`

Figure 5.2: Warping functions learnt for the four regression tasks carried out in this paper. Each plot is made over the range of the observation data, from y_{min} to y_{max}.

found to be much closer to the true generating distribution than the standard GP, especially with regard to the 2σ lines. The mean line was also computed, and found to lie close, but slightly skewed, from the median line.

Figure 5.1b emphasises the point that the warped GP finds the shape of the whole predictive *distribution* much better, not just the median or mean. In this plot, one particular point on the x axis is chosen, $x = -\pi/4$, and the predictive densities from the GP and warped GP are plotted alongside the true density (which can be written down analytically). Note that the standard GP must necessarily predict a symmetrical Gaussian density, even when the density from which the points are generated is highly asymmetrical, as in this case.

Figure 5.2a shows the warping function learnt for this regression task. The $\tanh$ functions have adjusted themselves so that they mimic a y^3 nonlinearity over the range of the observation space, thus inverting the $z^{1/3}$ transformation imposed when generating the data.

5.3 Results for some real data sets

It is not surprising that the method works well on the toy dataset of section 5.2 since it was generated from a known nonlinear warping of a smooth function with Gaussian noise. To demonstrate that nonlinear transformations also help on real data sets we have run the warped GP comparing its predictions to an ordinary GP on three regression problems. These data sets are summarised in table 5.1 which shows the range of the targets (y_{min}, y_{max}), the number of input dimensions (D), and the size of the training and test sets (N, T) that we used.

The dataset `Creep` is a materials science set, with the objective to predict creep

Dataset	D	y_{min}	y_{max}	N	T
`Creep`	30	18 MPa	530 MPa	800	1266
`Abalone`	8	1 yr	29 yrs	1000	3177
`Ailerons`	40	-3.0×10^{-3}	-3.5×10^{-4}	1000	6154

Table 5.1: Properties of the data sets

rupture stress (in MPa) for steel given chemical composition and other inputs [Cole et al., 2000].[2] With `Abalone` the aim is to predict the the age of Abalone from various physical inputs [Blake and Merz, 1998]. `Ailerons` is a simulated control problem, with the aim to predict the control action on the Ailerons of an F16 aircraft [Camacho, 2000].[3]

For datasets `Creep` and `Abalone`, which consist of positive observations only, standard practice may be to model the log of the data with a GP. So for these datasets we have compared three models: a GP directly on the data, a GP on the fixed log-transformed data, and the warped GP directly on the data. The predictive points and densities were always compared in the original data space, accounting for the Jacobian of both the log and the warped transforms. The models were run as in the 1D task: ML parameter estimates only, SE-ARD covariance (equation (1.21)), and warping function equation (5.11) with three tanh functions.

The results we obtain for the three datasets are shown in table 5.2. We show three measures of performance over independent test sets: mean absolute error, mean squared error, and the mean negative log predictive density (NLPD) evaluated at the test points. This final measure was included to give some idea of how well the model predicts the entire density, not just point predictions.

On these three sets, the warped GP always performs significantly better than the standard GP. For `Creep` and `Abalone`, the fixed log transform clearly works well too, but particularly in the case of `Creep`, the warped GP learns a better transformation. Figure 5.2 shows the warping functions learnt, and indeed figure 5.2b and figure 5.2c are clearly log-like in character. On the other hand figure 5.2d, for the `Ailerons` set, is exponential-like. This shows the warped GP is able to flexibly handle these different types of data sets. The shapes of the learnt warping functions were also found to be very robust to random initialisation of the parameters. Finally, the warped GP also makes a better job of predicting the distributions, as

[2] Materials Algorithms Project (MAP) Program and Data Library. http://www.msm.cam.ac.uk/map/entry.html.

[3] L. Torgo. http://www.liacc.up.pt/~ltorgo/Regression/.

Dataset	Model	Absolute error	Squared error	NLPD
Creep	GP	16.4	654	4.46
	GP + log	15.6	587	4.24
	warped GP	15.0	554	4.19
Abalone	GP	1.53	4.79	2.19
	GP + log	1.48	4.62	2.01
	warped GP	1.47	4.63	1.96
Ailerons	GP	1.23×10^{-4}	3.05×10^{-8}	-7.31
	warped GP	1.18×10^{-4}	2.72×10^{-8}	-7.45

Table 5.2: Results of testing the GP, warped GP, and GP with log transform, on three real datasets. The units for absolute error and squared error are as for the original data.

shown by the difference in values of the negative log density.

5.4 Remarks

We have shown that the warped GP is a useful extension to the standard GP for regression, capable of finding extra structure in the data through the transformations it learns. From another viewpoint, it allows standard preprocessing transforms, such as log, to be discovered automatically and improved on, rather than be applied in an ad-hoc manner.

Of course some data sets are well modelled by a GP already, and applying the warped GP model simply results in a linear 'warping' function. It has also been found that data sets that have been censored, i.e. many observations at the edge of the range lie on a single point, cause the warped GP problems. The warping function attempts to model the censoring by pushing those points far away from the rest of the data, and it suffers in performance especially for ML learning. To deal with this properly a censorship model is required.

As a further extension, one might consider warping the input space in some non-linear fashion. In the context of geostatistics this has actually been dealt with by Schmidt and O'Hagan [2003], where a transformation is made from an input space which can have non-stationary and non-isotropic covariance structure, to a latent space in which the usual conditions of stationarity and isotropy hold.

Gaussian process classifiers can also be thought of as warping the outputs of a GP,

through a mapping onto the $(0, 1)$ probability interval. However, the observations in classification are discrete, not points in this warped continuous space. Therefore the likelihood is different. Diggle et al. [1998] consider various other fixed nonlinear transformations of GP outputs.

The warped GP is just one way in which to relax the strict Gaussianity assumptions of the standard GP. The warped GP takes both the process *and* the noise, and passes them together through a nonlinear deterministic function. The major advantage of this is that it is very cheap to apply — hardly more expensive than the original GP. This type of all encompassing preprocessing transformation will not be suitable for all data. For example, some data may be much better modelled by a GP with a non-Gaussian noise model. Alternatively one can imagine making a nonlinear transformation to a GP *before* adding the noise. The problem with any of these alternatives is that they inevitably require costly approximations due to non-Gaussian high dimensional integrals. Finally, problems with input-dependent noise are better dealt with by the techniques discussed in chapter 4. Of course the warped GP need not be applied in a standalone way, and can be used in conjunction with many other GP modelling techniques.

Chapter 6

Discussion

In this thesis we have examined and developed a variety of techniques to deal with two general problems in Gaussian process modelling: their computational expense for large data sets, and their lack of flexibility to model different types of data. We now summarise the main contributions of the thesis and discuss some opportunities for future work based on these ideas.

In chapter 2 we developed the sparse pseudo-input Gaussian process (SPGP), based on a pseudo data set parameterising a GP approximation. This consisted of a new batch regression approximation for GPs coupled together with an evidence based gradient optimisation to find the locations of the pseudo-inputs and hyperparameters. We compared and contrasted this method to previous sparse GP approximations, both from a theoretical and from an empirical point of view. We found that the gradient optimisation of the SPGP gives very accurate solutions for a given number of pseudo-inputs, making it very suitable for applications where fast predictions are paramount. We also found another major advantage was the ability to reliably choose hyperparameters as part of the same smooth optimisation, and that these hyperparameters gave better results than preselecting them by training a GP on a large subset of data.

From a theoretical point of view we discussed the unifying framework for GP approximations of Quiñonero Candela and Rasmussen [2005], and how the SPGP approximation fits into the framework as 'FITC'. We then showed how FITC can be derived as a minimal KL (p to q) approximation with a conditional factorisation assumption. We discussed the equivalence of FITC as a batch version of Csató and Opper [2002]'s sparse online learning scheme, and argued how this equivalence

arises theoretically. We then illustrated the 'breaking' of the similar projected process (PP) approximation in the low noise regime, and we argued how this arises from the minimal KL (q to p) derivation as opposed to FITC's (p to q).

In terms of future work, there are still many areas to be addressed. From a theoretical point of view we saw how the PP approximation goes wrong in the low noise regime. However we also saw how the FITC approximation avoids this by collapsing onto the subset of data solution. Whilst it is good that FITC is a safe approximation, we would really like to do better than subset of data in this situation. The essential problem is that FITC is completely constrained in this situation — it is tied down at the inducing inputs. An interesting question is whether a different category of approximations can be developed that do not fall under these constraints.

From an empirical point of view more work needs doing to compare the inducing point based approximations discussed in this thesis, to approximations such as the IFGT [Yang et al., 2005] which seek to make fast matrix vector products embedded within CG. Whilst there are known limitations with these methods it would be very worthwhile to do a full empirical study to compare and assess in which regimes they work well. An interesting line of thought in this direction is to actually to combine a matrix vector product approximation with an inducing point based approximation such as FITC. One advantage of the matrix vector product approximations is that they do have some forms of error guarantees, i.e. a tolerance can be set. However, these guarantees are made on the results of the matrix vector products, and it is difficult to relate them to actual quantities of interest such as the predictive distribution and marginal likelihood. The SPGP, like all inducing point based approaches, requires a choice of the number of pseudo-inputs. The good thing is that this choice is directly related to computation, but unfortunately we do not have accuracy guarantees. It would be interesting to look at more automated ways of choosing the number of pseudo-inputs.

As we discussed previously, it would be interesting to test the SPGP methods for other non-Gaussian likelihoods and noise models. Whilst the FITC approximation has been used for classification by Csató and Opper [2002], the full SPGP method of learning pseudo-inputs and hyperparameters using gradients of the marginal likelihood has not been fully investigated.[1] It remains to be seen if the refinement of the locations of the pseudo-inputs is as useful for classification as it is for regression.

[1] We are aware of some unpublished work by Andrew Naish-Guzman that partly addresses this.

In chapter 3 we compared local type approximations to the global inducing point based ones, and we examined for which types of data they are most appropriate. The local type of approximation has been somewhat forgotten as a specific means of speeding up GP models, but it can be surprisingly effective, especially for low dimensional highly complex data sets. The inducing point based global approximation such as FITC is more suited to smoother higher dimensional data sets where global correlations and extrapolations are very important. We then developed a combined local and global approximation that can be seen as a direct extension of the framework of Quiñonero Candela and Rasmussen [2005], where we have relaxed the assumption that the training and test variables are conditionally independent. We showed how this PIC approximation drastically improves the predictions of the previously proposed PITC approximation, and subsumes both the purely local and purely global approaches.

There are many ideas for future directions associated with this work. Firstly we have not yet investigated choosing the local clusters in conjunction with the inducing points for the combined PIC approximation, perhaps based on the PIC marginal likelihood, or on heuristics. We feel this may improve the performance of PIC further over the purely local and purely global approaches. To complete the picture we could also try to choose hyperparameters from the PIC marginal likelihood. One of the reasons the global approach sometimes suffers in comparison to the local one, is that it unnecessarily computes covariances between inducing points and data points far away from each other, when their covariance is essentially zero. It would be interesting to investigate hierarchical type approximations where not only are the data points clustered, but the inducing points themselves are clustered, such that the only covariances computed are to their local regions. In a sense this idea has the same flavour as combining a fast matrix vector product approximation with an inducing point based one, as discussed above. A final idea, not purely along computational lines, is to try to develop PIC-like covariance functions that are naturally efficient, but also allow different region-specific properties such as varying lengthscales. This would nicely align with the original motivations for local GP experts of Rasmussen and Ghahramani [2002].

In chapter 4 we extended the modelling capabilities of the SPGP by learning a low-dimensional projection to deal with high dimensional data sets, and by learning uncertainties for the pseudo-variables to deal with heteroscedastic data sets. In fact we found that the originally SPGP was surprisingly flexible with regard to variable noise, even without the extension. This can be understood by viewing the SPGP

covariance function as a heavily parameterised non-stationary flexible covariance, which is naturally efficient by construction. For data that is not well modelled by a simple underlying stationary GP with fixed noise level, we gain a lot more than computational efficiency by using the SPGP. For the dimensionality reduction we showed that computation can be dramatically reduced by learning a supervised projection to a much lower dimensional space than the original, often for little loss in accuracy.

It is true that the variable noise with the SPGP is a useful bi-product of the sparse approximation, rather than an explicitly designed property, although this is somewhat addressed by the extended model. It would be interesting to compare with more specific heteroscedastic models such as Goldberg et al. [1998]. The problem is that such models tend to be computationally intractable for anything but very small data sets, whereas the SPGP is of course efficient by construction.

In chapter 5 we developed the warped Gaussian process — a method for automatically learning a nonlinear preprocessing transformation to handle data better that is not well modelled initially by a GP. Examples of such data include output distributions that are naturally skewed, or perhaps specified only on the positive real line. The advantage of the warped GP is that the preprocessing is not ad-hoc such as a simple $\log$ transformation, but is considered as a full part of the probabilistic model. We showed that on a variety of real data sets different types of transformations can be learnt, from exponential-like to $\log$-like, with corresponding improvements in performance. An attractive feature of the warped GP is that it is computationally cheap to apply. As we discussed in section 5.4, further work might include combination with an input warping for non-stationarity, and also combination with some of the sparse GP methods discussed earlier in the thesis.

In summary, this thesis provides a set of useful techniques that both approximate and extend Gaussian processes, allowing them to be applied to a greater range of machine learning problems. In all our methods we have sought to preserve the important aspects of Gaussian processes, namely that they are fully probabilistic nonlinear nonparametric models, allowing flexible modelling whilst handling uncertainty in data.

Appendix A

Notation

A.1 General

$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	multivariate normal distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$
$\mathcal{E}[f]$	expectation of f under $p(f)$
$\mathbf{I}$	identity matrix
$\mathrm{tr}(\mathbf{A})$	trace of a matrix $\mathbf{A}$
$\lvert\mathbf{A}\rvert$	determinant of a matrix $\mathbf{A}$
$\mathrm{KL}[p(x)\|q(x)]$	the Kullback-Leibler divergence between probability distributions p and q
$\mathrm{diag}(\mathbf{A})$	diagonal matrix formed from the leading diagonal of $\mathbf{A}$

A.2 Data points

We denote a D dimensional input point as a vector $\mathbf{x}$, and a scalar valued output as y. In a standard regression problem, we have a training data set $\mathcal{D}$ consisting of N pairs of inputs and outputs, and a corresponding test data set $\mathcal{D}_T$:

$$\mathcal{D} = \{\mathbf{x}_n, y_n\}_{n=1}^{N} , \tag{A.1a}$$

$$\mathcal{D}_T = \{\mathbf{x}_t, y_t\}_{t=1}^{T} . \tag{A.1b}$$

When working with GPs we assume there is an underlying noise-free latent function $f(\mathbf{x})$ that we are modelling. At each observation in the training or test set there

is therefore a latent function variable which we denote f_n or f_t.

The SPGP of section 2.2 is based on a set of M pseudo-points. We highlight these pseudo-points with a bar to distinguish them from real data points: $\{\bar{\mathbf{x}}_m, \bar{f}_m\}_{m=1}^M$. We never need to consider an equivalent noisy pseudo-output $\bar{y}_m$.

It is helpful to be able to refer collectively to the inputs and outputs of these data sets:

$$\mathbf{X} = \{\mathbf{x}_n\}_{n=1}^N \qquad \mathbf{f} = \{f_n\}_{n=1}^N \qquad \mathbf{y} = \{y_n\}_{n=1}^N \tag{A.2a}$$

$$\mathbf{X}_T = \{\mathbf{x}_t\}_{t=1}^T \qquad \mathbf{f}_T = \{f_t\}_{t=1}^T \qquad \mathbf{y}_T = \{y_t\}_{t=1}^T \tag{A.2b}$$

$$\bar{\mathbf{X}} = \{\bar{\mathbf{x}}_m\}_{m=1}^M \qquad \bar{\mathbf{f}} = \{\bar{f}_m\}_{m=1}^M \,. \tag{A.2c}$$

The PI(T)C approximation of section 3.2 requires that training points are grouped into disjoint clusters or 'blocks':

$$\mathbf{X}_{B_s} \subset \mathbf{X} = \bigcup_{s=1}^S \mathbf{X}_{B_s} \qquad \mathbf{f}_{B_s} \subset \mathbf{f} = \bigcup_{s=1}^S \mathbf{f}_{B_s} \qquad \sum_{s=1}^S B_s = N \,. \tag{A.3}$$

In keeping with past GP references we refer to a single general test point as $(\mathbf{x}_*, f_*, y_*)$.

A.3 Covariances

We need to be able to construct covariance matrices and vectors from various combinations of training, test and pseudo- inputs. The starting point is the covariance function $K(\mathbf{x}, \mathbf{x}')$, which maps two arbitrary input points to a real number. Our notation uses $\mathbf{K}$ to represent any covariance matrix that is directly constructed from the covariance function, but with different indices to show which two sets of input points are involved. For example, the $(N \times M)$ rectangular covariance matrix between training points and pseudo-points is denoted $\mathbf{K}_{NM}$. It is constructed from the covariance function:

$$[\mathbf{K}_{NM}]_{nm} = K(\mathbf{x}_n, \bar{\mathbf{x}}_m) \,, \tag{A.4}$$

or, with some abuse of notation, $\mathbf{K}_{NM} = K(\mathbf{X}, \bar{\mathbf{X}})$. The indices N and M are really shorthand for the two sets of input points $\mathbf{X}$ and $\bar{\mathbf{X}}$, but they also allow us to easily read off the size of any given covariance matrix.

Rather than use transpose symbols we simply swap the indices: $\mathbf{K}_{NM}^\top \equiv \mathbf{K}_{MN}$, since

Notation	Size	Covariance between	Transpose notation
$\mathbf{K}_N$	$N \times N$	$\mathbf{X} \leftrightarrow \mathbf{X}$	$\mathbf{K}_N$
$\mathbf{K}_M$	$M \times M$	$\bar{\mathbf{X}} \leftrightarrow \bar{\mathbf{X}}$	$\mathbf{K}_M$
$\mathbf{K}_{NM}$	$N \times M$	$\mathbf{X} \leftrightarrow \bar{\mathbf{X}}$	$\mathbf{K}_{MN}$
$\mathbf{K}_{B_sM}$	$B_s \times M$	$\mathbf{X}_{B_s} \leftrightarrow \bar{\mathbf{X}}$	$\mathbf{K}_{MB_s}$
$\mathbf{K}_{nM}$	$1 \times M$	$\mathbf{x}_n \leftrightarrow \bar{\mathbf{X}}$	$\mathbf{K}_{Mn}$
$\mathbf{K}_{TN}$	$T \times N$	$\mathbf{X}_T \leftrightarrow \mathbf{X}$	$\mathbf{K}_{NT}$
$\mathbf{K}_{*M}$	$1 \times M$	$\mathbf{x}_* \leftrightarrow \bar{\mathbf{X}}$	$\mathbf{K}_{M*}$
$\mathbf{K}_{\mathbf{x}M}$	$1 \times M$	$\mathbf{x} \leftrightarrow \bar{\mathbf{X}}$	$\mathbf{K}_{M\mathbf{x}}$

Table A.1: Covariance matrix notation

the covariance function is a symmetric function. To save some further space, we contract the two indices of square or self covariances to one index, e.g. $\mathbf{K}_{NN} \equiv \mathbf{K}_N$.

Sometimes we only need to specify a covariance vector rather than a full matrix. For example, the $(M \times 1)$ covariance between all pseudo-points and one training point, which we denote $\mathbf{K}_{Mn}$. The lower case n indicates that we are referring to a single training point $\mathbf{x}_n$, rather than the whole set $\mathbf{X}$.

Often we need only refer to a single general test point of arbitrary location. We denote such a test point $(\mathbf{x}_*, f_*, y_*)$, in keeping with past GP references. Covariances that refer to this single test point have starred indices, e.g. $\mathbf{K}_{N*}$.

Finally, we occasionally need to abuse notation further to preserve the *function* part of the covariance function in a concise notation. For example, we use $\mathbf{K}_{\mathbf{x}M}$ as shorthand for the vector *function* $[K(\mathbf{x}, \bar{\mathbf{x}}_1), K(\mathbf{x}, \bar{\mathbf{x}}_2), \ldots, K(\mathbf{x}, \bar{\mathbf{x}}_M)]$. Here $\mathbf{K}_{\mathbf{x}M}$ is strictly regarded as a function of $\mathbf{x}$, not just a simple vector. As an example of this use, we define the 'low-rank' covariance function Q:

$$Q(\mathbf{x}, \mathbf{x}') = \mathbf{K}_{\mathbf{x}M}\mathbf{K}_M^{-1}\mathbf{K}_{M\mathbf{x}'} \ . \tag{A.5}$$

Any covariance matrix $\mathbf{Q}$ constructed from (A.5) will have maximum rank M.

We summarize all this information in table A.1 by giving some common examples of covariance matrices and their properties.

Appendix B

Mathematical background

B.1 Matrix identities

B.1.1 Matrix inversion lemma

The matrix inversion lemma is also known as the Woodbury, Sherman and Morrison formula:

$$(\mathbf{A} + \mathbf{U}\mathbf{B}\mathbf{U}^\top)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{B}^{-1} + \mathbf{U}^\top\mathbf{A}^{-1}\mathbf{U})^{-1}\mathbf{U}^\top\mathbf{A}^{-1} \,, \tag{B.1}$$

where $\mathbf{A}$ and $\mathbf{B}$ are invertible square matrices of possibly different sizes, and $\mathbf{V}$ is a rectangular matrix. The equivalent relation for determinants is:

$$|\mathbf{A} + \mathbf{U}\mathbf{B}\mathbf{U}^\top| = |\mathbf{A}|\,|\mathbf{B}|\,|\mathbf{B}^{-1} + \mathbf{U}^\top\mathbf{A}^{-1}\mathbf{U}| \,. \tag{B.2}$$

B.1.2 Inverse of a partitioned matrix

If a symmetric invertible matrix $\mathbf{A}$ and its inverse are partitioned:

$$\mathbf{A} = \begin{bmatrix} \mathbf{F} & \mathbf{G} \\ \mathbf{G}^\top & \mathbf{H} \end{bmatrix} \,, \qquad \mathbf{A}^{-1} = \begin{bmatrix} \tilde{\mathbf{F}} & \tilde{\mathbf{G}} \\ \tilde{\mathbf{G}}^\top & \tilde{\mathbf{H}} \end{bmatrix} \,, \tag{B.3}$$

where $\mathbf{F}$ and $\tilde{\mathbf{F}}$ are square matrices of the same size etc. Then these submatrices may be expressed:

$$\begin{aligned}\tilde{\mathbf{F}} &= \mathbf{F}^{-1} + \mathbf{F}^{-1}\mathbf{G}\mathbf{M}\mathbf{G}^\top\mathbf{F}^{-1} \\ \tilde{\mathbf{G}} &= -\mathbf{F}^{-1}\mathbf{G}\mathbf{M} \\ \tilde{\mathbf{H}} &= \mathbf{M}\,,\end{aligned} \tag{B.4}$$

where $\mathbf{M} = (\mathbf{H} - \mathbf{G}^\top\mathbf{F}^{-1}\mathbf{G})^{-1}$.

B.1.3 Matrix derivatives

Suppose we have a matrix $\mathbf{A}$ which is a function of a parameter θ. The derivative of the inverse matrix w.r.t. θ is:

$$\frac{\partial}{\partial\theta}\mathbf{A}^{-1} = -\mathbf{A}^{-1}\frac{\partial\mathbf{A}}{\partial\theta}\mathbf{A}^{-1}\,, \tag{B.5}$$

where $\frac{\partial\mathbf{A}}{\partial\theta}$ is the matrix of elementwise derivatives of $\mathbf{A}$. If $\mathbf{A}$ is positive definite symmetric, the derivative of the $\log$ determinant is:

$$\frac{\partial}{\partial\theta}\log|\mathbf{A}| = \mathrm{tr}(\mathbf{A}^{-1}\frac{\partial\mathbf{A}}{\partial\theta})\,. \tag{B.6}$$

B.2 Cholesky decompositions

When implementing Gaussian processes and their approximations we are always faced with inversion of a symmetric positive definite matrix $\mathbf{A}$. However we rarely require the inverse $\mathbf{A}^{-1}$ itself. Common forms we require are $\mathbf{y}^\top\mathbf{A}^{-1}\mathbf{y}$, $\mathbf{A}^{-1}\mathbf{y}$ and $|\mathbf{A}|$, for some vector $\mathbf{y}$. The most computationally stable and efficient way to obtain these forms is via Cholesky decomposition:

$$\mathbf{A} = \mathbf{L}\mathbf{L}^\top\,, \tag{B.7}$$

where $\mathbf{L}$ is a triangular matrix called the Cholesky factor. This operation is also known as the matrix square root. It has the same order cost as matrix inversion $\mathcal{O}(N^3)$, but is actually cheaper in terms of constant factors.

The symmetric form $\mathbf{y}^\top \mathbf{A}^{-1}\mathbf{y}$ can be rewritten:

$$\begin{aligned}\mathbf{y}^\top \mathbf{A}^{-1}\mathbf{y} &= \mathbf{y}^\top (\mathbf{L}\mathbf{L}^\top)^{-1}\mathbf{y} \\ &= \mathbf{y}^\top \mathbf{L}^{-\top}\mathbf{L}^{-1}\mathbf{y} \\ &= \|\mathbf{L}^{-1}\mathbf{y}\|^2 \ .\end{aligned} \tag{B.8}$$

The vector $\mathbf{L}^{-1}\mathbf{y}$ can be easily found in $\mathcal{O}(N^2)$ time by forward substitution because $\mathbf{L}$ is triangular. If we require $\mathbf{A}^{-1}\mathbf{y}$ then we make a further back substitution: $\mathbf{L}^{-\top}(\mathbf{L}^{-1}\mathbf{y})$. The determinant $|\mathbf{A}|$ can be easily found because $|\mathbf{A}| = \prod_n \mathbf{L}_{nn}^2$.

If the matrix $\mathbf{A}$ is a *noiseless* covariance matrix, it is sometimes necessary to add a small 'jitter' to its diagonal, to make the Cholesky decomposition well conditioned: $\mathbf{A} + \epsilon\mathbf{I}$. In practice this jitter can be made small enough so that any differences in results are negligible.

A further use for the Cholesky decomposition is to generate a sample from the multivariate Gaussian distribution: $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. First factorise $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^\top$, and then a sample $\mathbf{f}$ is made from $\mathbf{f} = \boldsymbol{\mu} + \mathbf{L}\mathbf{u}$, where $\mathbf{u}$ is a sample from $\mathcal{N}(\mathbf{0}, \mathbf{I})$.

B.3 Gaussian identities

Suppose we have a Gaussian distribution on $\mathbf{y}$, with a mean that is a linear function of some other random variables $\mathbf{f}$:

$$p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\mathbf{V}\mathbf{f}, \mathbf{A}) \ . \tag{B.9}$$

Then suppose we have a zero mean Gaussian distribution on $\mathbf{f}$:

$$p(\mathbf{f}) = \mathcal{N}(\mathbf{0}, \mathbf{B}) \ . \tag{B.10}$$

The following identity is useful:

$$\begin{aligned}p(\mathbf{y}) &= \int \mathrm{d}\mathbf{f}\, p(\mathbf{y}|\mathbf{f})p(\mathbf{f}) \\ &= \mathcal{N}(\mathbf{0}, \mathbf{A} + \mathbf{V}\mathbf{B}\mathbf{V}^\top) \ .\end{aligned} \tag{B.11}$$

B.4 KL divergence

The Kullback-Leibler (KL) divergence between two probability densities $p(\mathbf{x})$ and $q(\mathbf{x})$ is:

$$\mathrm{KL}[p(\mathbf{x})\|q(\mathbf{x})] = \int \mathrm{d}\mathbf{x}\, p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} \,. \tag{B.12}$$

It is asymmetric, and hence we use the shorthand (p to q) to refer to the KL as above.

Appendix C

SPGP derivatives

C.1 SPGP marginal likelihood

Our aim in this section is to maximise the SPGP marginal likelihood equation (2.9) with respect to hyperparameters $\boldsymbol{\theta}$ and pseudo-inputs $\bar{\mathbf{X}}$. We first define the diagonal matrix $\boldsymbol{\Gamma}$:

$$\sigma^2\boldsymbol{\Gamma} = \mathrm{diag}(\mathbf{K}_N - \mathbf{Q}_N) + \sigma^2\mathbf{I} \,. \tag{C.1}$$

This scaling of $\boldsymbol{\Gamma}$ with the noise seems to help stability somewhat. With this definition we restate the SPGP marginal likelihood:

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{0}, \mathbf{Q}_N + \sigma^2\boldsymbol{\Gamma}) \,. \tag{C.2}$$

We want to minimise $\mathcal{L} = -\log p(\mathbf{y})$. Therefore:

$$2\mathcal{L} = \underbrace{\log|\mathbf{Q}_N + \sigma^2\boldsymbol{\Gamma}|}_{2\mathcal{L}_1} + \underbrace{\mathbf{y}^\top(\mathbf{Q}_N + \sigma^2\boldsymbol{\Gamma})^{-1}\mathbf{y}}_{2\mathcal{L}_2} + N\log 2\pi \,. \tag{C.3}$$

Equation (C.3) is not in a computationally efficient form and needs to be rewritten with the matrix inversion lemma equation (B.1) and its equivalent for determinants equation (B.2). We deal first with $\mathcal{L}_1$:

$$\begin{aligned} 2\mathcal{L}_1 &= \log|\mathbf{K}_M + \sigma^{-2}\mathbf{K}_{MN}\boldsymbol{\Gamma}^{-1}\mathbf{K}_{NM}||\mathbf{K}_M^{-1}||\sigma^2\boldsymbol{\Gamma}| \\ &= \log|\mathbf{A}| - \log|\mathbf{K}_M| + \log|\boldsymbol{\Gamma}| + (N-M)\log\sigma^2 \,, \end{aligned} \tag{C.4}$$

where $\mathbf{A} = \sigma^2\mathbf{K}_M + \mathbf{K}_{MN}\mathbf{\Gamma}^{-1}\mathbf{K}_{NM}$.[1] Now $\mathcal{L}_2$:

$$2\mathcal{L}_2 = \sigma^{-2}\mathbf{y}^\top(\mathbf{\Gamma}^{-1} - \mathbf{\Gamma}^{-1}\mathbf{K}_{NM}\mathbf{A}^{-1}\mathbf{K}_{MN}\mathbf{\Gamma}^{-1})\mathbf{y} \ . \tag{C.5}$$

For the actual implementation we use the Cholesky decompositions (see section B.2) of all the square symmetric matrices involved, e.g. $\mathbf{A} = \mathbf{A}^{\frac{1}{2}}\mathbf{A}^{\frac{\top}{2}}$. The equivalent for $\mathbf{\Gamma}$ is simply the square-root because it is diagonal. The advantage for the derivation is that the symmetric nature of the quadratic form equation (C.5) can be emphasised:

$$2\mathcal{L}_2 = \sigma^{-2}\big(\|\underline{\mathbf{y}}\|^2 - \|\mathbf{A}^{-\frac{1}{2}}\underline{\mathbf{K}}_{MN}\underline{\mathbf{y}}\|^2\big) \ , \tag{C.6}$$

where we have defined $\underline{\mathbf{y}} = \mathbf{\Gamma}^{-\frac{1}{2}}\mathbf{y}$ and $\underline{\mathbf{K}}_{NM} = \mathbf{\Gamma}^{-\frac{1}{2}}\mathbf{K}_{NM}$. Finally we combine equation (C.4) and equation (C.6) to obtain the negative log marginal likelihood:

$$\mathcal{L} = \mathcal{L}_1 + \mathcal{L}_2 + \frac{N}{2}\log 2\pi \ . \tag{C.7}$$

C.2 Derivatives

We now take the derivative of $\mathcal{L}$ with respect to a general arbitrary parameter (a hyperparameter or element of a pseudo-input vector).[2] We start by differentiating equation (C.4) using the identity equation (B.6), ignoring the noise variance σ^2 for now:

$$\begin{aligned} 2\dot{\mathcal{L}}_1 &= \mathrm{tr}\big(\mathbf{A}^{-\frac{1}{2}}\dot{\mathbf{A}}\mathbf{A}^{-\frac{\top}{2}}\big) - \mathrm{tr}\big(\mathbf{K}_M^{-\frac{1}{2}}\dot{\mathbf{K}}_M\mathbf{K}_M^{-\frac{\top}{2}}\big) + \mathrm{tr}\big(\mathbf{\Gamma}^{-\frac{1}{2}}\dot{\mathbf{\Gamma}}\mathbf{\Gamma}^{-\frac{1}{2}}\big) \\ &= \mathrm{tr}\big(\mathbf{A}^{-\frac{1}{2}}\dot{\mathbf{A}}\mathbf{A}^{-\frac{\top}{2}}\big) - \mathrm{tr}\big(\mathbf{K}_M^{-\frac{1}{2}}\dot{\mathbf{K}}_M\mathbf{K}_M^{-\frac{\top}{2}}\big) + \mathrm{tr}\,\underline{\dot{\mathbf{\Gamma}}} \ , \end{aligned} \tag{C.8}$$

where we have used the dot notation as a shorthand for derivative, and we have defined $\underline{\dot{\mathbf{\Gamma}}} = \mathbf{\Gamma}^{-\frac{1}{2}}\dot{\mathbf{\Gamma}}\mathbf{\Gamma}^{-\frac{1}{2}}$. We next differentiate equation (C.6):

$$\begin{aligned} \dot{\mathcal{L}}_2 = \sigma^{-2}\Big[-\tfrac{1}{2}\underline{\mathbf{y}}^\top\underline{\dot{\mathbf{\Gamma}}}\underline{\mathbf{y}} + \big(\mathbf{A}^{-\frac{1}{2}}\underline{\mathbf{K}}_{MN}\underline{\mathbf{y}}\big)^\top\Big(\tfrac{1}{2}\mathbf{A}^{-\frac{1}{2}}\dot{\mathbf{A}}\mathbf{A}^{-\frac{\top}{2}}\big(\mathbf{A}^{-\frac{1}{2}}\underline{\mathbf{K}}_{MN}\underline{\mathbf{y}}\big) \\ - \mathbf{A}^{-\frac{1}{2}}\underline{\dot{\mathbf{K}}}_{MN}\underline{\mathbf{y}} + \mathbf{A}^{-\frac{1}{2}}\underline{\mathbf{K}}_{MN}\underline{\dot{\mathbf{\Gamma}}}\underline{\mathbf{y}}\Big] \ , \end{aligned} \tag{C.9}$$

[1] To compare to section 2.2, $\mathbf{A} = \sigma^2\mathbf{B}$ and $\sigma^2\mathbf{\Gamma} = \mathbf{\Lambda} + \sigma^2\mathbf{I}$.

[2] For an alternative derivation of the gradients see [Lawrence, 2007], where the FITC approximation is used within the GPLVM for dimensionality reduction and visualisation on large data sets. There the matrix chain rule is used to break the derivation down into simpler stages.

where $\underline{\dot{\mathbf{K}}}_{NM} = \boldsymbol{\Gamma}^{-\frac{1}{2}}\dot{\mathbf{K}}_{NM}$. To complete the derivative of $\mathcal{L}$ we need $\dot{\mathbf{A}}$ and $\dot{\boldsymbol{\Gamma}}$:

$$\begin{aligned}\dot{\mathbf{A}} &= \sigma^2\dot{\mathbf{K}}_M + 2\,\mathrm{sym}\big(\dot{\mathbf{K}}_{MN}\boldsymbol{\Gamma}^{-1}\mathbf{K}_{NM}\big) - \mathbf{K}_{MN}\boldsymbol{\Gamma}^{-1}\dot{\boldsymbol{\Gamma}}\boldsymbol{\Gamma}^{-1}\mathbf{K}_{NM} \\ &= \sigma^2\dot{\mathbf{K}}_M + 2\,\mathrm{sym}\big(\underline{\dot{\mathbf{K}}}_{MN}\underline{\mathbf{K}}_{NM}\big) - \underline{\mathbf{K}}_{MN}\underline{\dot{\boldsymbol{\Gamma}}}\underline{\mathbf{K}}_{NM}\ .\end{aligned} \tag{C.10}$$

$$\dot{\boldsymbol{\Gamma}} = \sigma^{-2}\,\mathrm{diag}\big(\dot{\mathbf{K}}_N - 2\dot{\mathbf{K}}_{NM}\mathbf{K}_M^{-1}\mathbf{K}_{MN} + \mathbf{K}_{NM}\mathbf{K}_M^{-1}\dot{\mathbf{K}}_M\mathbf{K}_M^{-1}\mathbf{K}_{MN}\big) \tag{C.11}$$

$$\Rightarrow \quad \underline{\dot{\boldsymbol{\Gamma}}} = \sigma^{-2}\,\mathrm{diag}\big(\underline{\dot{\mathbf{K}}}_N - 2\underline{\dot{\mathbf{K}}}_{NM}\mathbf{K}_M^{-1}\underline{\mathbf{K}}_{MN} + \underline{\mathbf{K}}_{NM}\mathbf{K}_M^{-1}\dot{\mathbf{K}}_M\mathbf{K}_M^{-1}\underline{\mathbf{K}}_{MN}\big)\ , \tag{C.12}$$

where $\mathrm{sym}(\mathbf{B}) = (\mathbf{B} + \mathbf{B}^\top)/2$. By substituting equation (C.10) and equation (C.12) into equation (C.8) and equation (C.9) we have the derivative for $\mathcal{L}$ in terms of the covariance derivatives $\dot{\mathbf{K}}_M$, $\dot{\mathbf{K}}_{NM}$, and $\mathrm{diag}(\dot{\mathbf{K}}_N)$. $\dot{\mathbf{K}}_M$ and $\dot{\mathbf{K}}_{NM}$ are of course functions of both the hyperparameters and the pseudo-inputs. $\mathrm{diag}(\dot{\mathbf{K}}_N)$ is simply a function of the hyperparameters. Notice that the sym can be dropped from equation (C.10) when it is substituted in either equation (C.8) and equation (C.9).

C.3 Kernel derivatives

The kernel derivatives $\dot{\mathbf{K}}_M$, $\dot{\mathbf{K}}_{NM}$, and $\mathrm{diag}(\dot{\mathbf{K}}_N)$ with respect to the hyperparameters $\boldsymbol{\theta}$ are generally easy to compute. For example let us consider the amplitude hyperparameter a of the SE covariance equation (1.6):

$$\frac{\partial}{\partial a^2}K(\mathbf{x}_n, \bar{\mathbf{x}}_m) = \frac{1}{a^2}K(\mathbf{x}_n, \bar{\mathbf{x}}_m) \tag{C.13}$$

$$\Rightarrow \qquad \dot{\mathbf{K}}_{NM} = \frac{1}{a^2}\mathbf{K}_{NM}\ . \tag{C.14}$$

Similarly $\dot{\mathbf{K}}_M = \frac{1}{a^2}\mathbf{K}_M$ and $\mathrm{diag}(\dot{\mathbf{K}}_N) = \mathbf{I}$. In practice we differentiate with respect to $\log a^2$ so that unconstrained optimisation may be used. Derivatives with respect to other hyperparameters are also straightforward.

Derivatives with respect to the pseudo-inputs are made slightly more complicated by the fact that an element of $\mathbf{K}_{NM}$ and $\mathbf{K}_M$ is only a function of one or two pseudo-inputs. Consider the function that is defined by taking the derivative of the kernel function with respect to a single dimension d of its first argument: $\frac{\partial K}{\partial x^d}(\mathbf{x}, \mathbf{x}')$. Then the derivative of $\mathbf{K}_{NM}$ with respect to a single dimension of a single pseudo-input $\bar{x}^d_{m'}$ is:

$$\frac{\partial K_{nm}}{\partial \bar{x}^d_{m'}} = \delta_{mm'}\frac{\partial K}{\partial x^d}(\bar{\mathbf{x}}_{m'}, \mathbf{x}_n)\ . \tag{C.15}$$

The delta function indicates that only the m' column of $\dot{\mathbf{K}}_{NM}$ is non-zero. Similarly for $\dot{\mathbf{K}}_M$:

$$\frac{\partial K_{mm'}}{\partial \bar{x}^d_{m''}} = \delta_{mm''}\frac{\partial K}{\partial x^d}(\bar{\mathbf{x}}_{m''}, \bar{\mathbf{x}}_{m'}) + \delta_{m''m'}\frac{\partial K}{\partial x^d}(\bar{\mathbf{x}}_{m''}, \bar{\mathbf{x}}_{m}) \; . \tag{C.16}$$

Here only the m'' row and column of $\dot{\mathbf{K}}_M$ are non-zero. When equations (C.15) and (C.16) are substituted in (C.8), (C.9), (C.10), and (C.12), the delta functions cause various contractions to occur.

C.4 Noise derivative

We have left the noise derivative to last because it is actually simpler than the other hyperparameter derivatives, since it is not a kernel parameter. It is best derived directly from equation (C.3), before applying the matrix inversion lemma:

$$\begin{aligned} 2\frac{\partial \mathcal{L}_1}{\partial \sigma^2} &= \operatorname{tr}(\mathbf{Q}_N + \sigma^2\mathbf{\Gamma})^{-1} \\ &= \sigma^{-2}\operatorname{tr}(\mathbf{\Gamma}^{-1} - \mathbf{\Gamma}^{-1}\mathbf{K}_{NM}\mathbf{A}^{-1}\mathbf{K}_{MN}\mathbf{\Gamma}^{-1}) \\ &= \sigma^{-2}\operatorname{tr}(\mathbf{\Gamma}^{-1}) - \sigma^{-2}\operatorname{tr}(\underline{\underline{\mathbf{K}}}_{NM}\mathbf{A}^{-1}\underline{\underline{\mathbf{K}}}_{MN}) \; , \end{aligned} \tag{C.17}$$

where we have used $\frac{\partial}{\partial \sigma^2}(\sigma^2\mathbf{\Gamma}) = \mathbf{I}$. Similarly:

$$\begin{aligned} 2\frac{\partial \mathcal{L}_2}{\partial \sigma^2} &= -\|(\mathbf{Q}_N + \sigma^2\mathbf{\Gamma})^{-1}\mathbf{y}\|^2 \\ &= -\sigma^{-4}\|(\mathbf{\Gamma}^{-1} - \mathbf{\Gamma}^{-1}\mathbf{K}_{NM}\mathbf{A}^{-1}\mathbf{K}_{MN}\mathbf{\Gamma}^{-1})\mathbf{y}\|^2 \\ &= -\sigma^{-4}\big(\|\underline{\mathbf{y}}\|^2 + \|\underline{\underline{\mathbf{K}}}_{NM}\mathbf{A}^{-1}\underline{\underline{\mathbf{K}}}_{MN}\underline{\mathbf{y}}\|^2 - 2\underline{\mathbf{y}}^\top\underline{\underline{\mathbf{K}}}_{NM}\mathbf{A}^{-1}\underline{\underline{\mathbf{K}}}_{MN}\underline{\mathbf{y}}\big) \; . \end{aligned} \tag{C.18}$$

C.5 Derivative costs

There is a precomputation cost of $\mathcal{O}(NM^2)$ for any of the derivatives. After this the cost per hyperparameter is $\mathcal{O}(NM)$ in general. Since the pseudo-inputs are treated as extra hyperparameters, and since they constitute MD parameters in total, one might think that the cost for all pseudo-input derivatives would therefore be $\mathcal{O}(NM^2D)$. However, as described in section C.3, the pseudo-inputs are special in that they only affect certain rows and columns of $\mathbf{K}_M$ and $\mathbf{K}_{NM}$. *All* pseudo-input derivatives can actually be obtained in $\mathcal{O}(NMD)$, after precomputation of $\mathcal{O}(NM^2)$.

Bibliography

C. M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995. (page 16)

C. L. Blake and C. J. Merz. UCI repository of machine learning databases, 1998. http://www.ics.uci.edu/~mlearn/MLRepository.html. (pages 64 and 112)

R. Camacho. *Inducing models of human control skills*. PhD thesis, University of Porto, 2000. (page 112)

W. S. Cleveland and S. J. Devlin. Locally weighted regression: an approach to regression analysis by local fitting. *Journal of the American Statistical Association*, 83:596–610, 1988. (page 76)

D. Cole, C. Martin-Moran, A. G. Sheard, H. K. D. H. Bhadeshia, and D. J. C. MacKay. Modelling creep rupture strength of ferritic steel welds. *Science and Technology of Welding and Joining*, 5:81–90, 2000. (page 112)

L. Csató. *Gaussian Processes — Iterative Sparse Approximations*. PhD thesis, Aston University, UK, 2002. (pages 36, 55, 56, 62, and 63)

L. Csató and M. Opper. Sparse online Gaussian processes. *Neural Computation*, 14: 641–668, 2002. (pages 36, 55, 56, 57, 62, 115, and 116)

N. de Freitas, Y. Wang, M. Mahdaviani, and D. Lang. Fast Krylov methods for N-body learning. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*. MIT Press, 2006. (page 36)

P. J. Diggle, J. A. Tawn, and R. A. Moyeed. Model-based geostatistics. *Applied Statistics*, 47(3):299–350, 1998. (page 114)

Y. Engel, S. Mannor, and R. Meir. Bayes meets Bellman: The Gaussian process approach to temporal difference learning. In T. Fawcett and N. Mishra, editors,

Proceedings of the Twentieth International Conference on Machine Learning, 2003. (page 16)

T. Feder and D. Greene. Optimal algorithms for approximate clustering. In *Proceedings of the 20th ACM Symposium on Theory of Computing*, pages 434–444. ACM Press, New York, USA, 1988. ISBN 0-89791-264-0. doi: http://doi.acm.org/10.1145/62212.62255. (page 85)

M. N. Gibbs. *Bayesian Gaussian processes for Regression and Classification*. PhD thesis, University of Cambridge, 1997. (page 36)

M. N. Gibbs and D. J. C. MacKay. Variational Gaussian process classifiers. *IEEE Transactions on Neural Networks*, 11(6):1458–1464, 2000. (page 30)

M. Girolami and S. Rogers. Variational Bayesian multinomial probit regression with Gaussian process priors. *Neural Computation*, 18(8):1790–1817, 2006. (page 30)

P. W. Goldberg, C. K. I. Williams, and C. M. Bishop. Regression with input-dependent noise: A Gaussian process treatment. In *Advances in Neural Information Processing Systems 10*. MIT Press, 1998. (pages 92, 94, and 118)

T. Gonzales. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38(2–3):293–306, 1985. (page 84)

A. Gray. Fast kernel matrix-vector multiplication with application to Gaussian process learning. Technical Report CMU-CS-04-110, School of Computer Science, Carnegie Mellon University, 2004. (page 36)

E. Grosse. LOESS: Multivariate smoothing by moving least squares. In *Approximation Theory VI*, volume 1, pages 299–302. Academic Press, 1989. (page 76)

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001. (page 16)

D. Higdon, J. Swall, and J. Kern. Non-stationary spatial modeling. In J. M. Bernardo, J. O. Berger, A. P. Dawid, and A. F. M. Smith, editors, *Bayesian Statistics 6*, volume 6. OUP, 1999. (page 94)

S. Keerthi and W. Chu. A matching pursuit approach to sparse Gaussian process regression. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 643–650. MIT Press, Cambridge, MA, 2006. (pages 36 and 72)

A. N. Kolmogorov. Interpolation und Extrapolation von stationären zufäligen Folgen. *Izv. Akad. Nauk SSSR*, 5:3–14, 1941. (page 15)

S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951. (page 54)

M. Kuss and C. E. Rasmussen. Assessing approximations for Gaussian process classification. *Journal of Machine Learning Research*, 6:1679–1704, 2005. (pages 16 and 30)

N. D. Lawrence. Probabilistic non-linear principal component analysis with Gaussian process latent variable models. *Journal of Machine Learning Research*, 6:1783–1816, 2005. (page 16)

N. D. Lawrence. Learning for larger datasets with the Gaussian process latent variable model. In M. Meila and X. Shen, editors, *Proceedings of the Eleventh International Workshop on Artificial Intelligence and Statistics*. Omnipress, 2007. (page 127)

N. D. Lawrence, M. Seeger, and R. Herbrich. Fast sparse Gaussian process methods: The informative vector machine. In S. T. S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 609–616. MIT Press, Cambridge, MA, 2003. (pages 33 and 62)

D. J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003. (page 16)

G. Matheron. The intrinsic random functions and their applications. *Advances in Applied Probability*, 5:439–468, 1973. (page 15)

T. Minka. *A family of algorithms for approximate Bayesian inference*. PhD thesis, Massachusetts Institute of Technology, 2001. (page 30)

J. Moody. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(281–294), 1989. (page 44)

R. M. Neal. Bayesian learning for neural networks. In *Lecture Notes in Statistics 118*. Springer, 1996. (pages 16 and 27)

A. O'Hagan. Curve fitting and optimal design for prediction. *Journal of the Royal Statistical Society B*, 40:1–42, 1978. (page 15)

C. J. Paciorek and M. J. Schervish. Nonstationary covariance functions for Gaussian process regression. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in*

Neural Information Processing Systems 16. MIT Press, Cambridge, MA, 2004. (page 94)

A. Papoulis. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, New York, third edition, 1991. (pages 15 and 17)

T. Poggio and F. Girosi. Networks for approximation and learning. *Proceedings of IEEE*, 78:1481–1497, 1990. (page 35)

F. P. Preparata and M. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985. (page 85)

W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, second edition, 1992. (page 108)

J. Quiñonero Candela. *Learning with Uncertainty - Gaussian Processes and Relevance Vector Machines*. PhD thesis, Technical University of Denmark, 2004. (page 35)

J. Quiñonero Candela and C. E. Rasmussen. A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6:1939–1959, Dec 2005. (pages 13, 37, 45, 47, 48, 49, 50, 51, 56, 74, 77, 81, 90, 115, and 117)

C. E. Rasmussen and Z. Ghahramani. Infinite mixtures of Gaussian process experts. In *Advances in Neural Information Processing Systems 14*. MIT Press, 2002. (pages 76, 91, and 117)

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006. (pages 16, 22, and 63)

A. M. Schmidt and A. O'Hagan. Bayesian inference for nonstationary spatial covariance structure via spatial deformations. *Journal of the Royal Statistical Society, Series B*, 65:745–758, 2003. (page 113)

B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, 2002. (pages 28 and 29)

A. Schwaighofer and V. Tresp. Transductive and inductive methods for approximate Gaussian process regression. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 953–960. MIT Press, Cambridge, MA, 2003. (pages 64 and 77)

M. Seeger. *Bayesian Gaussian Process Models: PAC-Bayesian Generalisation Error Bounds and Sparse Approximations*. PhD thesis, University of Edinburgh, 2003. (page 63)

M. Seeger, C. K. I. Williams, and N. D. Lawrence. Fast forward selection to speed up sparse Gaussian process regression. In C. M. Bishop and B. J. Frey, editors, *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, 2003. (pages 36, 42, 54, 64, 70, and 72)

M. Seeger, N. Lawrence, and R. Herbrich. Efficient nonparametric Bayesian modelling with sparse Gaussian process approximations. Submitted for publication. Available at http://www.kyb.tuebingen.mpg.de/bs/people/seeger/, 2007. (page 63)

Y. Shen, A. Ng, and M. Seeger. Fast Gaussian process regression using KD-trees. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*. MIT Press, 2006. (page 36)

B. W. Silverman. Some aspects of the spline smoothing approach to non-parametric curve fitting. *Journal of the Royal Statistical Society*, 1985. (page 102)

A. J. Smola and P. Bartlett. Sparse greedy Gaussian process regression. In T. K. Leen, T. G. Diettrich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 619–625. MIT Press, Cambridge, MA, 2001. (pages 31, 35, and 72)

A. J. Smola and B. Schölkopf. Sparse greedy matrix approximation for machine learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000. (page 35)

E. Snelson and Z. Ghahramani. Sparse Gaussian processes using pseudo-inputs. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 1257–1264. MIT Press, Cambridge, MA, 2006a. (pages 12, 37, and 56)

E. Snelson and Z. Ghahramani. Variable noise and dimensionality reduction for sparse Gaussian processes. In R. Dechter and T. S. Richardson, editors, *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 2006b. (pages 13 and 92)

E. Snelson and Z. Ghahramani. Local and global sparse Gaussian process approximations. In M. Meila and X. Shen, editors, *Proceedings of the Eleventh International Workshop on Artificial Intelligence and Statistics*. Omnipress, 2007. (pages 13 and 74)

E. Snelson, C. E. Rasmussen, and Z. Ghahramani. Warped Gaussian processes. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004. (pages 13 and 105)

V. Tresp. A Bayesian committee machine. *Neural Computation*, 12:2719–2741, 2000. (pages 36 and 77)

G. E. Uhlenbeck and L. S. Ornstein. On the theory of Brownian motion. *Physical Review*, 36:823–841, 1930. (page 21)

S. Vijayakumar, A. D'Souza, T. Shibata, J. Conradt, and S. Schaal. Statistical learning for humanoid robots. *Autonomous Robot*, 12(1):55–69, 2002. (page 63)

F. Vivarelli and C. K. I. Williams. Discovering hidden features with Gaussian processes regression. In *Advances in Neural Information Processing Systems 11*. MIT Press, 1999. (page 93)

G. Wahba. *Spline Models for Observational Data*. CBMS-NSF Regional Conference series in applied mathematics. Society for Industrial and Applied Mathematics, 1990. (page 35)

N. Wiener. *Extrapolation, Interpolation and Smoothing of Stationary Time Series*. MIT Press, 1949. (page 15)

C. K. I. Williams. Computation with infinite neural networks. *Neural Computation*, 10:1203–1216, 1998. (page 22)

C. K. I. Williams and D. Barber. Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1342–1351, 1998. (pages 16 and 30)

C. K. I. Williams and C. E. Rasmussen. Gaussian processes for regression. In *Advances in Neural Information Processing Systems 8*. MIT Press, 1996. (page 16)

C. K. I. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In T. K. Leen, T. G. Diettrich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 682–688. MIT Press, Cambridge, MA, 2001. (page 35)

C. Yang, R. Duraiswami, and L. Davis. Efficient kernel machines using the improved fast Gauss transform. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1561–1568. MIT Press, Cambridge, MA, 2005. (pages 36 and 116)