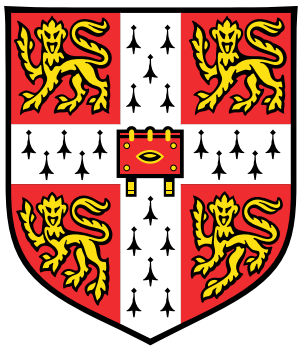
# Automatic Model Construction with Gaussian Processes



**David Kristjanson Duvenaud**

University of Cambridge

This dissertation is submitted for the degree of

*Doctor of Philosophy*

Pembroke College

June 2014

# Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text. This dissertation contains less than 60,000 words and has less than 150 figures.

David Kristjanson Duvenaud
June 2014

# Acknowledgements

First, I would like to thank my supervisor, Carl Rasmussen, for so much advice and encouragement. It was wonderful working with someone who has spent many years thinking deeply about the business of modeling. Carl was patient while I spent the first few months of my PhD chasing half-baked ideas, and then gently suggested a series of ideas which actually worked.

I'd also like to thank my advisor, Zoubin Ghahramani, for providing much encouragement, support, feedback and advice, and for being an example. I'm also grateful for Zoubin's efforts to constantly populate the lab with interesting visitors.

Thanks to Michael Osborne, whose thesis was an inspiration early on, and to Roman Garnett for being a sane third party. Sinead Williamson, Peter Orbanz and John Cunningham gave me valuable advice when I was still bewildered. Ferenc Huszár and Dave Knowles made it clear that constantly asking questions was the right way to go.

Tom Dean and Greg Corrado made me feel at home at Google. Philipp Hennig was a mentor to me during my time in Tübingen, and demonstrated an inspiring level of effectiveness. Who else finishes their conference papers with entire days to spare? I'd also like to thank Ryan Adams for hosting me during my visit to Harvard, for our enjoyable collaborations, and for building such an awesome group. Josh Tenenbaum's broad perspective and enthusiasm made the projects we worked on very rewarding. The time I got to spend with Roger Grosse was mind-expanding – he constantly suprised me by pointing out basic unsolved questions about decades-old methods, and had extremely high standards for his own work.

I'd like to thank Andrew McHutchon, Konstantina Palla, Alex Davies, Neil Houlsby, Koa Heaukulani, Miguel Hernández-Lobato, Yue Wu, Ryan Turner, Roger Frigola, Sae Franklin, David Lopez-Paz, Mark van der Wilk and Rich Turner for making the lab feel like a family. I'd like to thank James Lloyd for many endless and enjoyable discussions, and for keeping a level head even during the depths of deadline death-marches. Christian Steinruecken showed me that amazing things are hidden all around.

Thanks to Mel for reminding me that I needed to write a thesis, for supporting me

during the final push, and for her love.

My graduate study was supported by the National Sciences and Engineering Research Council of Canada, the Cambridge Commonwealth Trust, Pembroke College, a grant from the Engineering and Physical Sciences Research Council, and a grant from Google.

# Abstract

This thesis develops a method for automatically constructing, visualizing and describing a large class of models, useful for forecasting and finding structure in domains such as time series, geological formations, and physical dynamics. These models, based on Gaussian processes, can capture many types of statistical structure, such as periodicity, changepoints, additivity, and symmetries. Such structure can be encoded through *kernels*, which have historically been chosen by hand by experts. We show how to automate this task, creating a system which explores an open-ended space of models and reports the structures discovered.

To automatically construct Gaussian process models, we define a grammar over kernels and simply optimize approximate marginal likelihood using a breadth-first search. We also develop automatic model decomposition, visualization and description procedures. Combining these, we present a procedure which takes in a dataset and outputs an automatically-constructed model, along with a detailed report with graphs and automatically generated text illustrating the qualitatively different, and sometimes novel, types of structure discovered in that dataset.

The introductory chapters contain a tutorial showing how to express many types of structure through kernels, and how adding and multiplying different kernels combines their properties. We also show how composite kernels can produce priors over topological manifolds such as cylinders, toruses, and Möbius strips, as well as their higher-dimensional analogues.

This thesis also explores several extensions to Gaussian process models. First, building on earlier work relating Gaussian processes and neural nets, we analyze natural extensions of these models to *deep kernels* and *deep Gaussian processes*. Second, we model sums of functions of all subsets of input variables, connecting this model class to the recently-developed regularization method of *dropout*. Third, we combine Gaussian processes with the Dirichlet process to produce the *warped mixture model*: a Bayesian clustering model with nonparametric cluster shapes, and a corresponding latent space in which each cluster has an interpretable parametric form.

# Contents

# List of Figures

# List of Tables

# Notation

Unbolded $x$ represents a single number, while boldface $\mathbf{x}$ represents a vector, and capital boldface $\mathbf{X}$ represents a matrix. An individual element of a vector is denoted with a subscript and without boldface. For example, the $i$th element of a vector $\mathbf{x}$ is $x_i$. A bold lower-case letter with an index such as $\mathbf{x}_j$ represents a particular row of matrix $\mathbf{X}$.

| Symbol | Description |
|---|---|
| SE | The squared-exponential kernel, also known as the radial-basis function (RBF) kernel, or the Gaussian kernel. |
| RQ | The rational-quadratic kernel. |
| Per | The periodic kernel. |
| Lin | The linear kernel. |
| WN | The white-noise kernel. |
| C | The constant kernel. |
| $\boldsymbol{\sigma}$ | The changepoint kernel, $\boldsymbol{\sigma}(x, x') = \sigma(x)\sigma(x')$, where $\sigma(x)$ is a sigmoidal function such as the logistic function. |
| $k_a + k_b$ | Addition of kernels, shorthand for $k_a(\mathbf{x}, \mathbf{x}') + k_b(\mathbf{x}, \mathbf{x}')$ |
| $k_a \times k_b$ | Multiplication of kernels, shorthand for $k_a(\mathbf{x}, \mathbf{x}') \times k_b(\mathbf{x}, \mathbf{x}')$ |
| $k(\mathbf{X}, \mathbf{X})$ | The Gram matrix, whose $i, j$th element is $k(\mathbf{x}_i, \mathbf{x}_j)$. |
| $\mathbf{K}$ | Shorthand for the Gram matrix $k(\mathbf{X}, \mathbf{X})$ |
| $\boldsymbol{f}(\mathbf{X})$ | A vector of function values, whose $i$th element is given by $f(\mathbf{x}_i)$. |
| $\mathrm{mod}(i, j)$ | The modulo operator, giving the remainder after dividing $i$ by $j$. |
| $\mathcal{O}(\cdot)$ | The big-O asymptotic complexity of an algorithm. |

# Chapter 1

# Introduction

> "All models are wrong, but yours are stupid too."
>
> @ML_Hipster (2013)

Prediction, extrapolation, and induction are all examples of learning a function from data. There are many ways to learn functions, but one particularly elegant way is by *inference*. Inference procedures set up a group of hypotheses (a *model*), then weight those hypotheses based on how well their predictions match the data. Keeping around all the hypotheses that match the data helps guard against over-fitting. Comparing different models is a way to find which sorts of structure are present in a dataset.

To be able to learn a wide variety of structures, we would like to have an expressive language of models of functions. We would like to be able to represent simple kinds of functions, such as linear functions or polynomials. We would also like to have models of arbitrarily complex functions, specified in terms of high-level properties such as how smooth they are, whether they repeat over time, or which symmetries they have.

The beginning of thesis will show how to build such a language using Gaussian processes (GPs), a tractable class of models of functions. This chapter will introduce the basic properties of GPs. The next chapter will describe how to model different types of structure using GPs.

## 1.1 Gaussian Process models

Gaussian processes are a simple and general class of models of functions. To be precise, a GP is any distribution over functions such that any finite subset of function values $f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots f(\mathbf{x}_N)$ have a joint Gaussian distribution (Rasmussen and Williams,

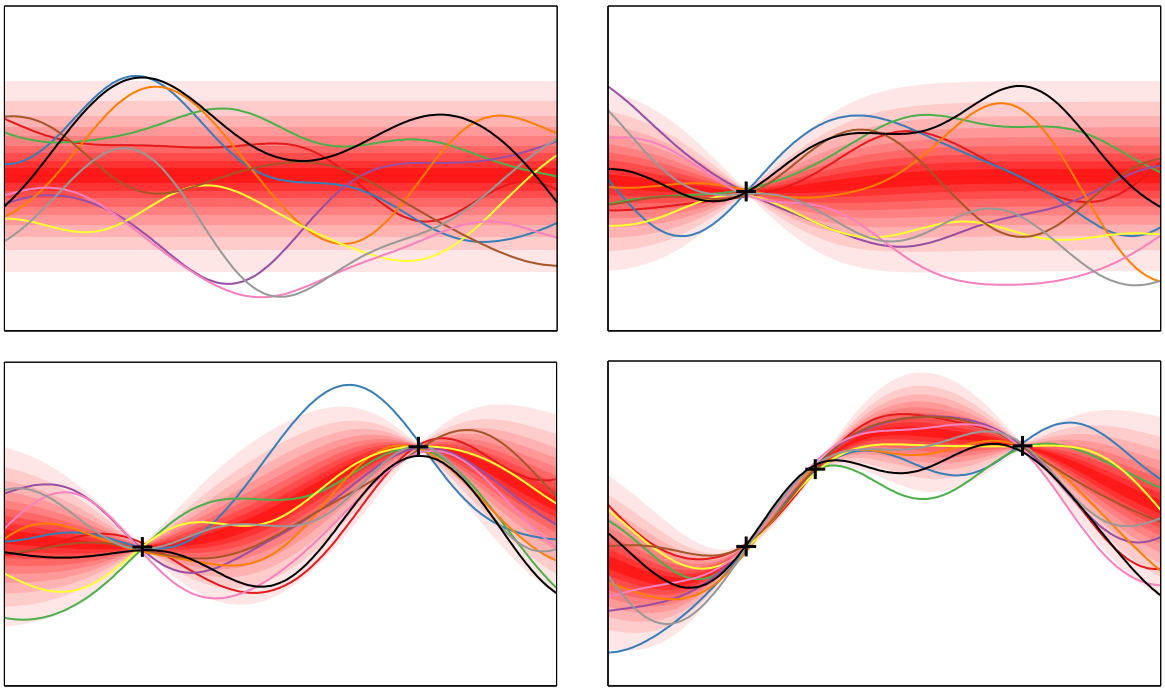


Figure 1.1 A visual representation of a Gaussian process modeling a one-dimensional function. Different shades of red correspond to deciles of the predictive density at each input location. Coloured lines show samples from the process – examples of some of the hypotheses included in the model. *Top left:* A GP not conditioned on any datapoints. *Remaining plots:* The posterior after conditioning on different amounts of data. All plots have the same axes.

2006). A GP model, before conditioning on data, is completely specified by its mean function,

$$\mathbb{E}\left[f(\mathbf{x})\right] = \mu(\mathbf{x}) \tag{1.1}$$

and its covariance function, also called the *kernel*:

$$\mathrm{Cov}\left[f(\mathbf{x}), f(\mathbf{x}')\right] = k(\mathbf{x}, \mathbf{x}') \tag{1.2}$$

It is common practice to assume that the mean function is simply zero everywhere, since uncertainty about the mean function can be taken into account by adding an extra term to the kernel.

After accounting for the mean, the kind of structure which can be captured by a GP model is entirely determined by its kernel. The kernel determines how the model

generalizes, or extrapolates to new data.

There are many possible choices of covariance function, and we can specify a wide range of models just by specifying the kernel of a GP. For example, linear regression, splines, and Kalman filters are all examples of GPs with particular kernels. However, these are just a few familiar examples out of a wide range of possibilities. One of the main difficulties in using GPs is constructing a kernel which represents the particular structure present in the data being modelled.

### 1.1.1 Model selection

The crucial property of GPs that allows us to automatically construct models is that we can compute the *marginal likelihood* of a dataset given a particular model, also known as the *evidence* (MacKay, 1992). The marginal likelihood allows us to compare models and automatically discover the appropriate amount of detail to use, due to Bayesian Occam's razor (MacKay, 2003; Rasmussen and Ghahramani, 2001).

Choosing a kernel, or kernel parameters, by maximizing the marginal likelihood will typically result in selecting the *least* flexible model which still captures all the structure in the data. For example, if a kernel has a parameter which controls the smoothness of the functions it models, the maximum-likelihood estimate of that parameter will usually correspond to the smoothest family of functions consistent with the observed function values.

Here is the marginal likelihood under a GP prior of observing a set of function values $[f(\mathbf{x}_1), f(\mathbf{x}_2), \dots f(\mathbf{x}_N)] := \boldsymbol{f}(\mathbf{X})$ at locations $[\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N]^\mathsf{T} := \mathbf{X}$:

$$\begin{aligned} p(\boldsymbol{f}(\mathbf{X})|\mathbf{X}, \mu(\cdot), k(\cdot,\cdot)) &= \mathcal{N}(\boldsymbol{f}(\mathbf{X})|\mu(\mathbf{X}), k(\mathbf{X},\mathbf{X})) \\ &= (2\pi)^{-\frac{N}{2}} \times \underbrace{|k(\mathbf{X},\mathbf{X})|^{-\frac{1}{2}}}_{\text{discourages flexibility}} \\ &\quad \times \underbrace{\exp\left\{-\frac{1}{2}\left(\boldsymbol{f}(\mathbf{X}) - \boldsymbol{\mu}(\mathbf{X})\right)^\mathsf{T} k(\mathbf{X},\mathbf{X})^{-1}\left(\boldsymbol{f}(\mathbf{X}) - \boldsymbol{\mu}(\mathbf{X})\right)\right\}}_{\text{encourages fit with data}} \end{aligned} \tag{1.3}$$

This multivariate Gaussian density is referred to as the *marginal* likelihood because it implicitly integrates (marginalizes) over all possible functions values $\boldsymbol{f}(\bar{\mathbf{X}})$, where $\bar{\mathbf{X}}$ is the set of all locations where we have not observed the function.

### 1.1.2 Prediction

We can ask the model which function values are likely to occur at any location, given the observations seen so far. By the formula for Gaussian conditionals (given in appendix A), the predictive distribution of a function value $f(\mathbf{x}^\star)$ at a test point $\mathbf{x}^\star$ has the form:

$$p(f(\mathbf{x}^\star)|\boldsymbol{f}(\mathbf{X}), \mathbf{X}, \mu(\cdot), k(\cdot,\cdot)) = \mathcal{N}\Big(f(\mathbf{x}^\star) \,|\, \underbrace{\mu(\mathbf{x}^\star) + k(\mathbf{x}^\star, \mathbf{X})k(\mathbf{X}, \mathbf{X})^{-1}\left(\boldsymbol{f}(\mathbf{X}) - \mu(\mathbf{X})\right)}_{\text{predictive mean follows observations}}, \underbrace{k(\mathbf{x}^\star, \mathbf{x}^\star) - k(\mathbf{x}^\star, \mathbf{X})k(\mathbf{X}, \mathbf{X})^{-1}k(\mathbf{X}, \mathbf{x}^\star)}_{\text{predictive variance shrinks given more data}}\Big) \tag{1.4}$$

These expressions may look complex, but only require a few matrix operations to evaluate.

Sampling a function from a GP is also straightforward: a sample from a GP at a finite set of locations is just a single sample from a single multivariate Gaussian distribution, given by equation (1.4). Figure 1.1 shows prior and posterior samples from a GP, as well as contours of the predictive density.

Our use of probabilities does not mean that we are assuming the function being learned is stochastic or random in any way; it is simply a consistent method of keeping track of our uncertainty.

### 1.1.3 Useful properties of Gaussian processes

There are several reasons why GPs in particular are well-suited for building a language of regression models:

- **Analytic inference.** Given a kernel function and some observations, the predictive posterior distribution can be computed exactly in closed form. This is a rare property for nonparametric models to have.

- **Expressivity.** Through the choice of covariance function, we can express a very wide range of modeling assumptions. Some examples will be shown in chapter 2.

- **Integration over hypotheses.** The fact that a GP posterior, given a fixed kernel, lets us integrate exactly over a wide range of hypotheses means that overfitting is less of an issue than in comparable model classes. For example, compared to

neural networks, relatively few parameters need to be estimated, which lessens the need for the complex optimization or regularization schemes.

- **Model Selection.** A side benefit of being able to integrate over all hypotheses is that we can compute the marginal likelihood of the data given a model. This gives us a principled way of comparing different models.

- **Closed-form predictive distribution.** The predictive distribution of a GP at a set of test points is simply a multivariate Gaussian distribution. This means that GPs can easily be composed with other models or decision procedures.

- **Easy to analyze.** It may seem unsatisfying to restrict ourselves to a limited model class, as opposed to trying to do inference in the set of all computable functions. However, simple models can be used as well-understood building blocks for constructing more interesting models.

  For example, consider linear models. Although they form an extremely limited model class, they are simple, easy to analyze, and easy to incorporate into other models or procedures. Gaussian processes can be seen as an extension of linear models which retain these attractive properties (Rasmussen and Williams, 2006).

### 1.1.4 Limitations of Gaussian processes

There are several issues which sometimes make GPs difficult to use:

- **Slow inference.** Computing the matrix inverse in equations (1.3) and (1.4) takes $\mathcal{O}(N^3)$ time, making exact inference prohibitively slow for more than a few thousand datapoints. However, this problem can be addressed by approximate inference schemes (Hensman et al., 2013; Quiñonero-Candela and Rasmussen, 2005; Snelson and Ghahramani, 2006). Most GP software packages implement several of these methods (Hensman et al., 2014b; Rasmussen and Nickisch, 2010; Vanhatalo et al., 2013).

- **Light tails of the predictive distribution.** The predictive distribution of a standard GP model is Gaussian. We may sometimes with to use non-Gaussian predictive likelihoods, for example to be robust to outliers, or to perform classification. Using non-Gaussian likelihoods requires approximate inference. Fortunately, mature software packages exist which can automatically perform approximate inference for a wide variety of non-Gaussian likelihoods.

- **The need to choose a kernel.** The flexibility of GP models raises the question of which kernel to use for a given problem. Choosing a useful kernel is equivalent to learning a useful representation of the input. Kernel parameters can be set automatically by maximizing the marginal likelihood, but until recently, human experts were required to choose the parametric form of the kernel. Chapter 3 will show a way in which kernels can be automatically constructed for a given dataset.

## 1.2 Outline and contributions of thesis

The main contribution of this thesis is to show how to automate the discovery and description of structure in functions, by searching through an open-ended language of regression models. This thesis also includes a set of related results showing how Gaussian processes can be extended or composed with other models.

Chapter 2 is a tutorial showing how to build a wide variety of structured models of functions by constructing appropriate covariance functions. We will also show how GPs can produce nonparametric models of manifolds with diverse topological structures, such as cylinders, toruses and Möbius strips.

Chapter 3 shows how to search over a general, open-ended language of models, built by combining different kernels. Since we can evaluate each model by the marginal likelihood, we can automatically construct custom models for each dataset by a breadth-first search. The nature of GPs allow the resulting models to be visualized by decomposing them into diverse, interpretable components, each capturing a different type of structure. Capturing this high-level structure sometimes allows us to extrapolate beyond the range of the data.

One benefit of using a compositional model class is that the resulting models are interpretable. Chapter 4 demonstrates a system which automatically describes the structure implied by a given kernel on a given dataset, generating reports with graphs and English-language text describing the resulting model. Combined with the automatic model search developed in chapter 3, this system represents the beginnings of what could be called an "automatic statistician".

Chapter 5 analyzes deep neural network models by characterizing the prior over functions obtained by composing GP priors to form *deep Gaussian processes*. We show that, as the number of layers increase, the amount of information retained about the original input diminishes to a single degree of freedom. A simple change to the network architecture fixes this pathology. We relate these models to neural networks, and as a

side effect derive several forms of *infinitely deep kernels*.

Chapter 6 examines a more limited, but much faster way of discovering structure using GPs. Specifying a kernel with many different types of structure, we use kernel parameters to discard whichever types of structure are *not* found in the current dataset. The model class we examine is called *additive Gaussian processes*, a model summing over exponentially-many GPs, each depending on a different subset of the input variables. We give a polynomial-time inference algorithm for this model, and relate it to other model classes. For example, additive GPs are shown to have the same covariance as a GP that uses *dropout*, a recently discovered regularization technique for neural networks.

Chapter 7 develops a Bayesian clustering model in which the clusters have nonparametric shapes, called the infinite warped mixture model. The density manifolds learned by this model follow the contours of the data density, and have interpretable, parametric forms in the latent space. The marginal likelihood lets us infer the effective dimension and shape of each cluster separately, as well as the number of clusters.

# Chapter 2

# Expressing Structure with Kernels

This chapter shows how to use kernels to build models of functions with many different kinds of structure: additivity, symmetry, periodicity, interactions between variables, and changepoints. We also show several ways to encode group invariants into kernels. Combining a few simple kernels through addition and multiplication will give us a rich, open-ended language of models.

The properties of kernels discussed in this chapter are mostly known in the literature. The original contribution of this chapter is to gather them into a coherent whole and to offer a tutorial showing the implications of different kernel choices, and some of the structures which can be obtained by combining them.

## 2.1 Definition

A kernel (also called a covariance function, kernel function, or covariance kernel), is a positive-definite function of two inputs $\mathbf{x}, \mathbf{x}'$. In this chapter, $\mathbf{x}$ and $\mathbf{x}'$ are usually vectors in a Euclidean space, but kernels can also be defined on graphs, images, discrete or categorical inputs, or even text.

Gaussian process models use a kernel to define the prior covariance between any two function values:

$$\mathrm{Cov}\left[f(\mathbf{x}), f(\mathbf{x}')\right] = k(\mathbf{x}, \mathbf{x}') \tag{2.1}$$

Colloquially, kernels are often said to specify the similarity between two objects. This is slightly misleading in this context, since what is actually being specified is the similarity between two values of a *function* evaluated on each object. The kernel specifies which

functions are likely under the GP prior, which in turn determines the generalization properties of the model.

## 2.2 A few basic kernels

To begin understanding the types of structures expressible by GPs, we will start by briefly examining the priors on functions encoded by some commonly used kernels: the squared-exponential (SE), periodic (Per), and linear (Lin) kernels. These kernels are defined in figure 2.1.

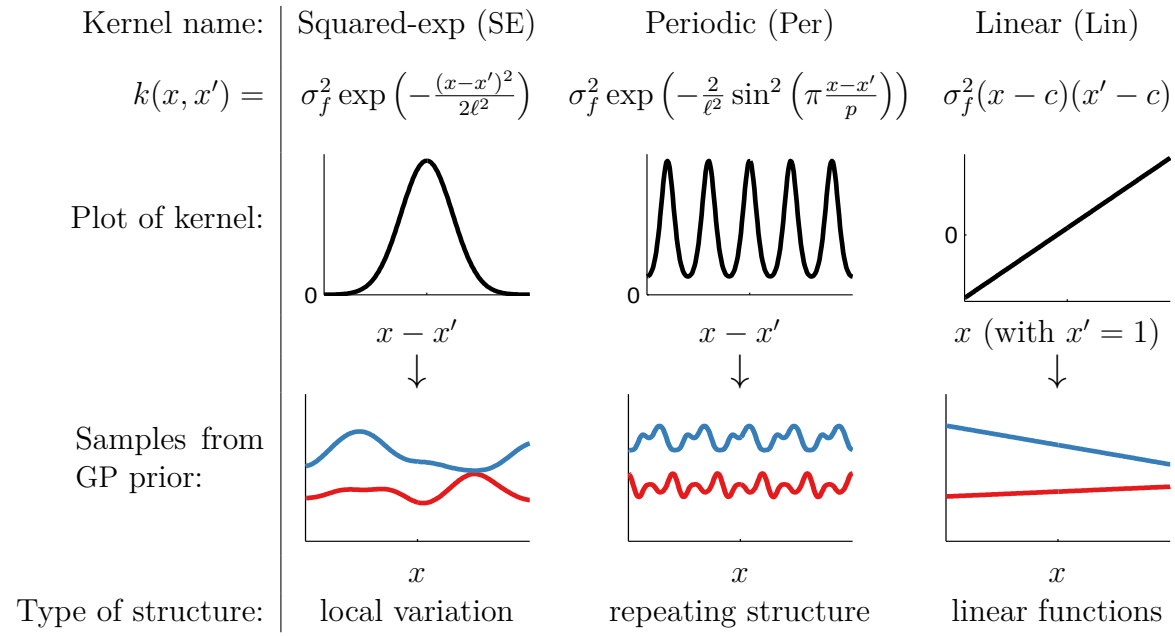



Figure 2.1 Examples of structures expressible by some basic kernels.

Each covariance function corresponds to a different set of assumptions made about the function we wish to model. For example, using a squared-exp (SE) kernel implies that the function we are modeling has infinitely many derivatives. There exist many variants of "local" kernels similar to the SE kernel, each encoding slightly different assumptions about the smoothness of the function being modeled.

**Kernel parameters** Each kernel has a number of parameters which specify the precise shape of the covariance function. These are sometimes referred to as *hyper-parameters*, since they can be viewed as specifying a distribution over function parameters, instead of being parameters which specify a function directly. An example would be the lengthscale

parameter $\ell$ of the SE kernel, which specifies the width of the kernel and thereby the smoothness of the functions in the model.

**Stationary and Non-stationary** The SE and Per kernels are *stationary*, meaning that their value only depends on the difference $x - x'$. This implies that the probability of observing a particular dataset remains the same even if we move all the $\mathbf{x}$ values by the same amount. In contrast, the linear kernel (Lin) is non-stationary, meaning that the corresponding GP model will produce different predictions if the data were moved while the kernel parameters were kept fixed.

## 2.3 Combining kernels

What if the kind of structure we need is not expressed by any known kernel? For many types of structure, it is possible to build a "made to order" kernel with the desired properties. The next few sections of this chapter will explore ways in which kernels can be combined to create new ones with different properties. This will allow us to include as much high-level structure as necessary into our models.

### 2.3.1 Notation

Below, we will focus on two ways of combining kernels: addition and multiplication. We will often write these operations in shorthand, without arguments:

$$k_a + k_b = k_a(\mathbf{x}, \mathbf{x}') + k_b(\mathbf{x}, \mathbf{x}') \tag{2.2}$$

$$k_a \times k_b = k_a(\mathbf{x}, \mathbf{x}') \times k_b(\mathbf{x}, \mathbf{x}') \tag{2.3}$$

All of the basic kernels we considered in section 2.2 are one-dimensional, but kernels over multi-dimensional inputs can be constructed by adding and multiplying between kernels on different dimensions. The dimension on which a kernel operates is denoted by a subscripted integer. For example, $\mathrm{SE}_2$ represents an SE kernel over the second dimension of vector $\mathbf{x}$. To remove clutter, we will usually refer to kernels without specifying their parameters.

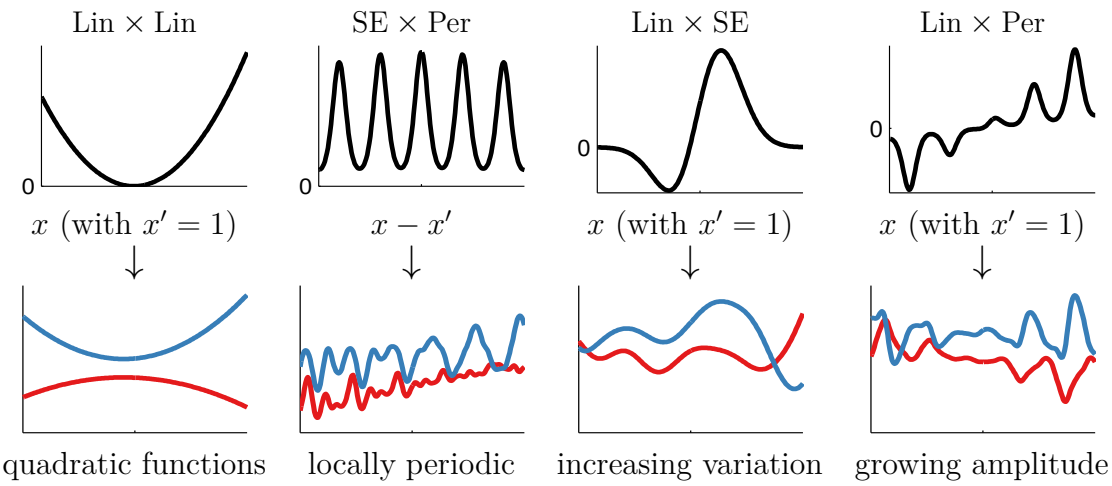



Figure 2.2 Examples of one-dimensional structures expressible by multiplying kernels. Plots have same meaning as in figure 2.1.

### 2.3.2 Combining properties through multiplication

Multiplying two positive-definite kernels together always results in another positive-definite kernel. But what properties do these new kernels have? Figure 2.2 shows some kernels obtained by multiplying two basic kernels together.

Working with kernels, rather than the parametric form of the function itself, allows us to express high-level properties of functions that do not necessarily have a simple parametric form. Here, we discuss a few examples:

- **Polynomial Regression.** By multiplying together $T$ linear kernels, we obtain a prior on polynomials of degree $T$. The first column of figure 2.2 shows a quadratic kernel.

- **Locally Periodic Functions.** In univariate data, multiplying a kernel by SE gives a way of converting global structure to local structure. For example, Per corresponds to exactly periodic structure, whereas Per$\times$SE corresponds to locally periodic structure, as shown in the second column of figure 2.2.

- **Functions with Growing Amplitude.** Multiplying by a linear kernel means that the marginal standard deviation of the function being modeled grows linearly away from the location given by kernel parameter $c$. The third and fourth columns of figure 2.2 show two examples.

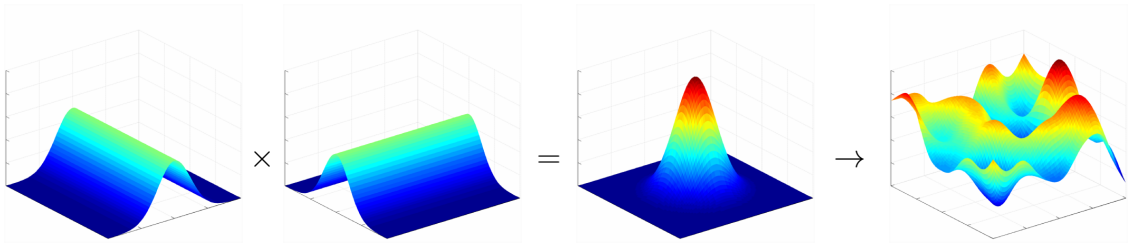
We can multiply any number of kernels together in this way to produce kernels combining several high-level properties. For example, the kernel SE×Lin×Per is a prior on functions which are locally periodic with linearly growing amplitude. We will a real datasets with this kind of structure in chapter 3.

### 2.3.3 Building multi-dimensional models

A flexible way to model functions having more than one input is to multiply together kernels defined on each individual input. For example, a product of SE kernels over different dimensions, each having a different lengthscale parameter, is called the SE-ARD kernel:

$$\text{SE-ARD}(\mathbf{x}, \mathbf{x}') = \prod_{d=1}^{D} \sigma_d^2 \exp\left(-\frac{1}{2}\frac{(x_d - x'_d)^2}{\ell_d^2}\right) = \sigma_f^2 \exp\left(-\frac{1}{2}\sum_{d=1}^{D}\frac{(x_d - x'_d)^2}{\ell_d^2}\right) \tag{2.4}$$

Figure 2.3 illustrates the SE-ARD kernel in two dimensions.




$\text{SE}_1(x_1, x'_1)$ $\text{SE}_2(x_2, x'_2)$ $\text{SE}_1 \times \text{SE}_2$ $f(x_1, x_2)$ drawn from $\mathcal{GP}(0, \text{SE}_1 \times \text{SE}_2)$

Figure 2.3 A product of two one-dimensional kernels gives rise to a prior on functions which depend on both dimensions.

ARD stands for Automatic Relevance Determination, so named because estimating the lengthscale parameters $\ell_1, \ell_2, \ldots, \ell_D$, implicitly determines the "relevance" of each dimension. Input dimensions with relatively large lengthscales imply relatively little variation along those dimensions in the function being modeled.

SE-ARD kernels are the default kernel in most applications of GPs. This may be partly because they have relatively few parameters to estimate, and those parameters are relatively interpretable. In addition, there is a theoretical reason to use them: they are *universal* kernels (Micchelli et al., 2006), capable of learning any continuous function given enough data, under some conditions.

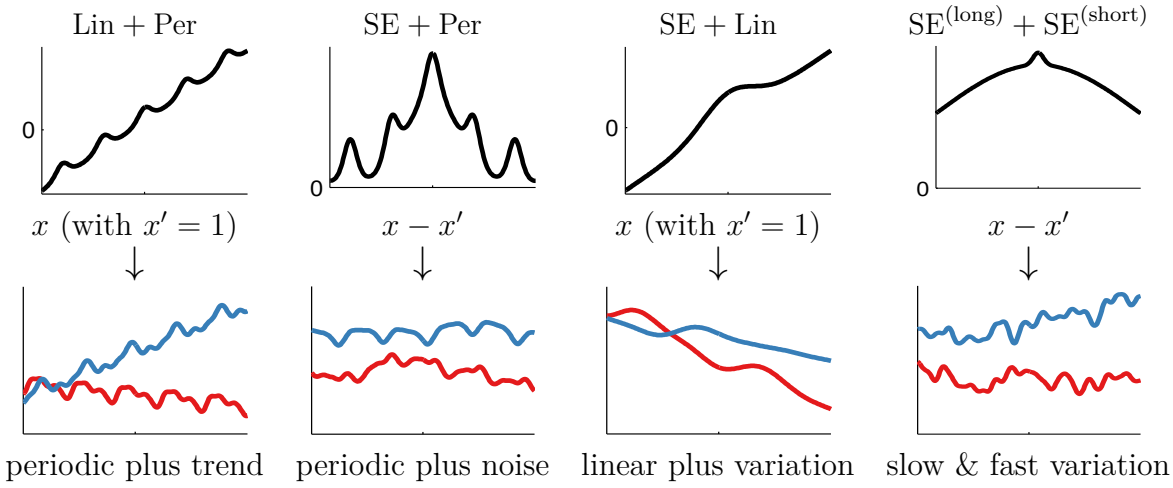



Figure 2.4 Examples of one-dimensional structures expressible by adding kernels. Rows have the same meaning as in figure 2.1. $\mathrm{SE}^{(\mathrm{long})}$ denotes a SE kernel whose lengthscale is long relative to that of $\mathrm{SE}^{(\mathrm{short})}$

However, this flexibility means that they can sometimes be relatively slow to learn, due to the *curse of dimensionality* (Bellman, 1956). In general, the more structure we account for, the less data we need - the *blessing of abstraction* (Goodman et al., 2011) counters the curse of dimensionality. Below, we will investigate ways to encode more structure into kernels.

## 2.4 Modeling sums of functions

An additive function is one which can be expressed as $f(\mathbf{x}) = f_a(\mathbf{x}) + f_b(\mathbf{x})$. Additivity is a useful modeling assumption in a wide variety of contexts, especially if it allows us to make strong assumptions about the individual components which make up the sum. Restricting the flexibility of component functions often aids in building interpretable models, and sometimes enables extrapolation in high dimensions.

It is easy to encode additivity into GP models. Suppose functions $f_a, f_b$ are drawn independently from GP priors:

$$f_a \sim \mathcal{GP}(\mu_a, k_a) \tag{2.5}$$

$$f_b \sim \mathcal{GP}(\mu_b, k_b) \tag{2.6}$$

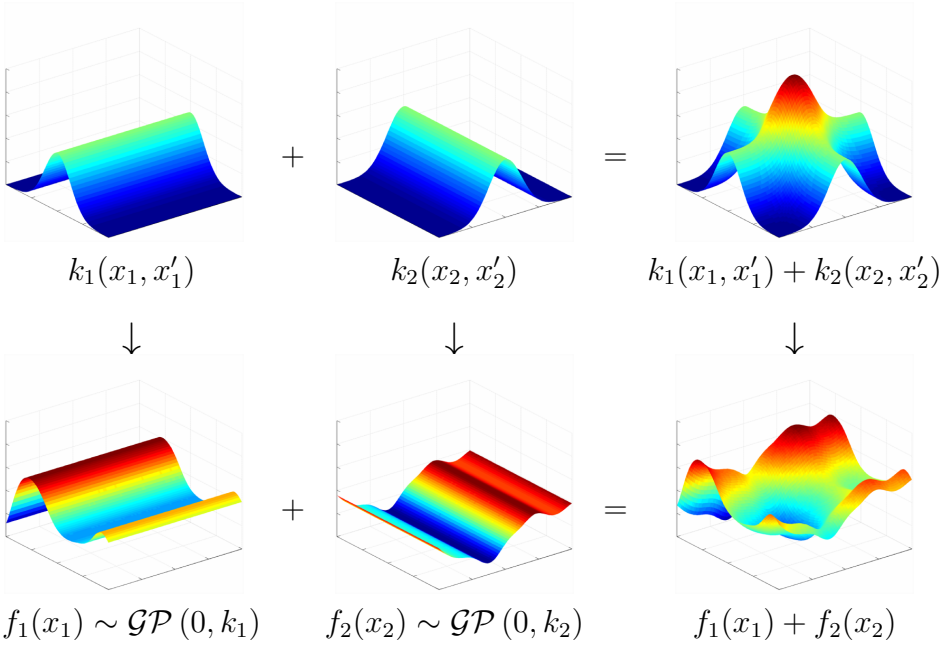



Figure 2.5 A sum of two orthogonal one-dimensional kernels. *Top row:* An additive kernel is any sum of kernels. *Bottom row:* A draw from an additive kernel corresponds to a sum of draws from independent GP priors, each having the corresponding kernel.

Then the sum of those functions is simply another GP:

$$f_a + f_b \sim \mathcal{GP}(\mu_a + \mu_b, k_a + k_b). \tag{2.7}$$

Kernels $k_a$ and $k_b$ can be of different types, allowing us to model the data as a sum of independent functions, each possibly representing a different type of structure. We can sum any number of components this way.

### 2.4.1 Additivity across multiple dimensions

When modeling functions of multiple dimensions, summing kernels can give rise to additive structure across different dimensions. To be more precise, if the kernels being added together are functions of only a subset of input dimensions, then the implied prior over functions decomposes in the same way. For example,

$$f(x_1, x_2) \sim \mathcal{GP}(0, k_1(x_1, x_1') + k_2(x_2, x_2')) \tag{2.8}$$

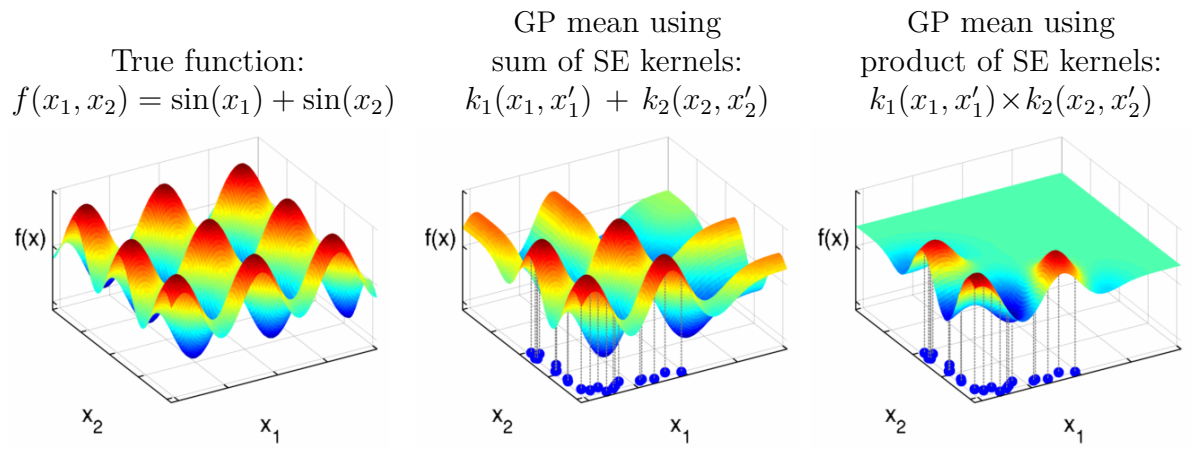



Figure 2.6 *Left:* A function with additive structure. *Center:* A GP with an additive kernel can extrapolate away from the training data. *Right:* A GP with a product kernel allows a different function value for every combination of inputs, and so is uncertain about function values away from the training data. This causes the predictions to revert to the mean.

is equivalent to the model

$$f_1(x_1) \sim \mathcal{GP}(0, k_1(x_1, x_1')) \tag{2.9}$$

$$f_2(x_2) \sim \mathcal{GP}(0, k_2(x_2, x_2')) \tag{2.10}$$

$$f(x_1, x_2) = f_1(x_1) + f_2(x_2)\,. \tag{2.11}$$

Figure 2.5 illustrates a decomposition of this form. Note that the product of two kernels does not have an analogous interpretation as the product of two functions.

### 2.4.2 Extrapolation through additivity

Additive structure sometimes allows us to make predictions far from the training data. Figure 2.6 compares the extrapolations made by additive versus product-kernel GP models, conditioned on data from a sum of two axis-aligned sine functions. The training points were evaluated in a small, L-shaped area. In this example, the additive model is able to correctly predict the height of the function at an unseen combinations of inputs. The product-kernel model is more flexible, and so remains uncertain about the function away from the data.

These types of additive models have been well-explored in the statistics literature. For example, generalized additive models (Hastie and Tibshirani, 1990) have seen wide

adoption. In high dimensions, we can also consider sums of functions of multiple input dimensions. Chapter 6 considers this model class in more detail.

### 2.4.3 Example: An additive model of concrete strength

To illustrate how additive kernels give rise to interpretable models, we built an additive model of the strength of concrete as a function of the amount of seven different ingredients, plus the age of the concrete (Yeh, 1998).

Our simple additive model looks like

$$\begin{aligned} f(\mathbf{x}) = f_1(\text{cement}) + f_2(\text{slag}) + f_3(\text{fly ash}) + f_4(\text{water}) \\ + f_5(\text{plasticizer}) + f_6(\text{coarse}) + f_7(\text{fine}) + f_8(\text{age}) + \text{noise} \end{aligned} \tag{2.12}$$

where noise $\overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma_n^2)$. The corresponding kernel has 9 additive components. After learning the kernel parameters by maximizing the marginal likelihood of the data, we can visualize the predictive distribution of each component of the model.

Figure 2.7 shows the marginal posterior distribution of each of the eight components in equation (2.12). The parameters controlling the variance of two of the components, Coarse and Fine, were set to zero, meaning that the marginal likelihood preferred a parsimonious model which did not depend on these dimensions. This is an example of the automatic sparsity that arises by maximizing marginal likelihood in GP models, and another example of automatic relevance determination (ARD) (Neal, 1995).

The ability to learn kernel parameters in this way is much more difficult when using non-probabilistic methods such as Support Vector Machines (Cortes and Vapnik, 1995), for which cross-validation is often the best method to select kernel parameters.

### 2.4.4 Posterior variance of additive components

Here we derive the posterior variance and covariance of all of the additive components of a GP. These formulas allow one to make plots such as figure 2.7.

First, we will write down the joint prior over the sum of two functions drawn from GP priors. We distinguish between $\boldsymbol{f}(\mathbf{X})$ (the function values at the training locations) and $\boldsymbol{f}(\mathbf{X}^\star)$ (the function values at some set of query locations).

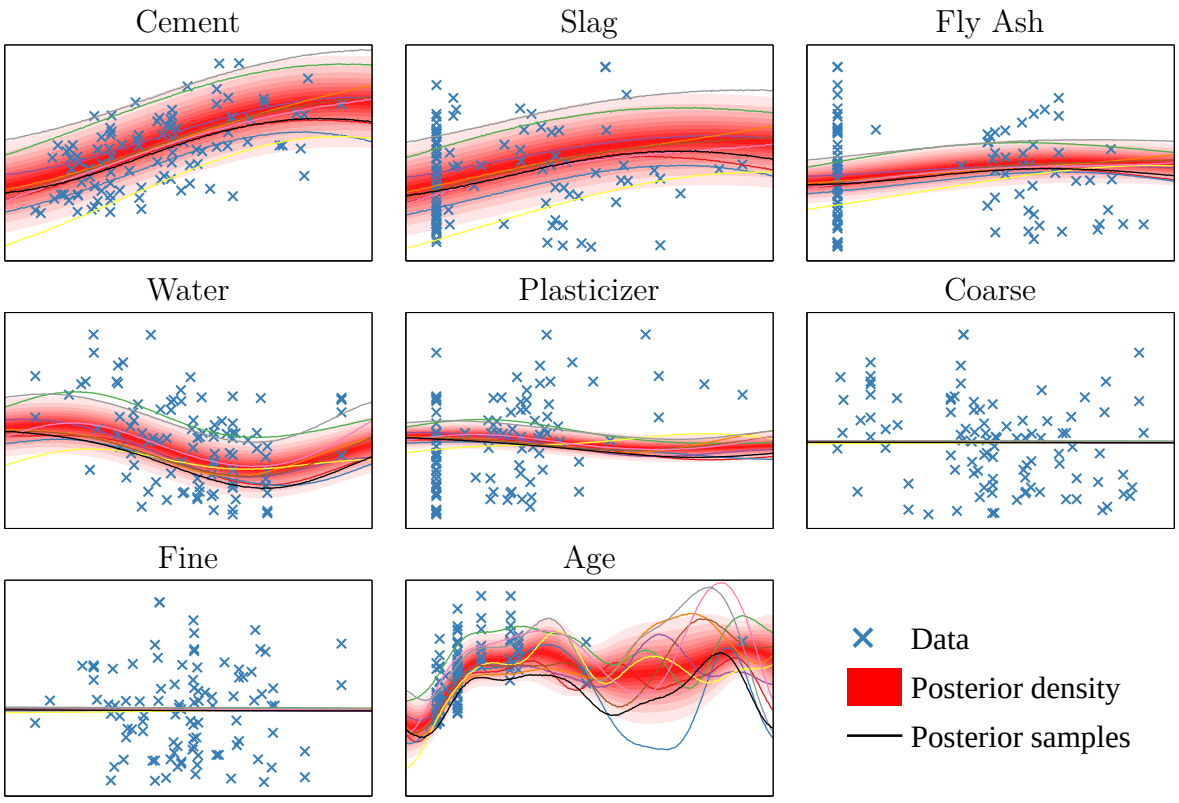



Figure 2.7 The predictive distribution of each component of a multi-dimensional additive model. Blue crosses indicate the original data projected on to each dimension, red indicates the marginal posterior density, and colored lines are samples from the marginal posterior of each component. The vertical axis is the same for all plots.

If $f_1$ and $f_2$ are *a priori* independent, and $f_1 \sim \mathrm{GP}(\mu_1, k_1)$ and $f_2 \sim \mathrm{GP}(\mu_2, k_2)$, then

$$\begin{bmatrix} \boldsymbol{f}_1(\mathbf{X}) \\ \boldsymbol{f}_1(\mathbf{X}^\star) \\ \boldsymbol{f}_2(\mathbf{X}) \\ \boldsymbol{f}_2(\mathbf{X}^\star) \\ \boldsymbol{f}_1(\mathbf{X}) + \boldsymbol{f}_2(\mathbf{X}) \\ \boldsymbol{f}_1(\mathbf{X}^\star) + \boldsymbol{f}_2(\mathbf{X}^\star) \end{bmatrix} \sim \mathcal{N}\left( \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_1^\star \\ \boldsymbol{\mu}_2 \\ \boldsymbol{\mu}_2^\star \\ \boldsymbol{\mu}_1 + \boldsymbol{\mu}_2 \\ \boldsymbol{\mu}_1^\star + \boldsymbol{\mu}_2^\star \end{bmatrix}, \begin{bmatrix} \mathbf{K}_1 & \mathbf{K}_1^\star & 0 & 0 & \mathbf{K}_1 & \mathbf{K}_1^\star \\ \mathbf{K}_1^\star & \mathbf{K}_1^{\star\star} & 0 & 0 & \mathbf{K}_1^\star & \mathbf{K}_1^{\star\star} \\ 0 & 0 & \mathbf{K}_2 & \mathbf{K}_2^\star & \mathbf{K}_2 & \mathbf{K}_2^\star \\ 0 & 0 & \mathbf{K}_2^\star & \mathbf{K}_2^{\star\star} & \mathbf{K}_2^\star & \mathbf{K}_2^{\star\star} \\ \mathbf{K}_1 & \mathbf{K}_1^\star & \mathbf{K}_2 & \mathbf{K}_2^\star & \mathbf{K}_1 + \mathbf{K}_2 & \mathbf{K}_1^\star + \mathbf{K}_2^\star \\ \mathbf{K}_1^\star & \mathbf{K}_1^{\star\star} & \mathbf{K}_2^\star & \mathbf{K}_2^{\star\star} & \mathbf{K}_1^\star + \mathbf{K}_2^\star & \mathbf{K}_1^{\star\star} + \mathbf{K}_2^{\star\star} \end{bmatrix} \right) \tag{2.13}$$

where we represent the Gram matrices, whose $i, j$th entry is given by $k(\mathbf{x}_i, \mathbf{x}_j)$ by

$$\mathbf{K}_1 = k_1(\mathbf{X}, \mathbf{X}) \tag{2.14}$$
$$\mathbf{K}_1^\star = k_1(\mathbf{X}^\star, \mathbf{X}) \tag{2.15}$$
$$\mathbf{K}_1^{\star\star} = k_1(\mathbf{X}^\star, \mathbf{X}^\star) \tag{2.16}$$

The formula for Gaussian conditionals, equation (A.2), can be used to give the conditional distribution of a GP-distributed function conditioned on its sum with another GP-distributed function:

$$\boldsymbol{f}_1(\mathbf{X}^\star)|\boldsymbol{f}_1(\mathbf{X}) + \boldsymbol{f}_2(\mathbf{X}) \sim \mathcal{N}\Big(\boldsymbol{\mu}_1^\star + \mathbf{K}_1^{\star\mathsf{T}}(\mathbf{K}_1 + \mathbf{K}_2)^{-1}\left[\boldsymbol{f}_1(\mathbf{X}) + \boldsymbol{f}_2(\mathbf{X}) - \boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\right], \\ \mathbf{K}_1^{\star\star} - \mathbf{K}_1^{\star\mathsf{T}}(\mathbf{K}_1 + \mathbf{K}_2)^{-1}\mathbf{K}_1^\star\Big) \tag{2.17}$$

These formulas express the model's posterior uncertainty about the different components of the signal, integrating over the possible configurations of the other components. To extend these formulas to a sum of more than two functions, the term $\mathbf{K}_1 + \mathbf{K}_2$ can simply be replaced by $\sum_i \mathbf{K}_i$ everywhere.

#### Posterior covariance of additive components

We can also compute the posterior covariance between any two components, conditioned on their sum:

$$\mathrm{Cov}\left[\boldsymbol{f}_1(\mathbf{X}^\star), \boldsymbol{f}_2(\mathbf{X}^\star)|\boldsymbol{f}(\mathbf{X})\right] = -\mathbf{K}_1^{\star\mathsf{T}}(\mathbf{K}_1 + \mathbf{K}_2)^{-1}\mathbf{K}_2^\star \tag{2.18}$$

If this quantity is negative, it means that there is ambiguity about which of the two components explains the data at that location. Figure 2.8 shows the posterior correlation between all non-zero components of the concrete model. Most of the correlation occurs within components, but there is also negative correlation between the "Cement" and "Slag" variables. This reflects an ambiguity in the model about which one of these functions is high and the other low.

## 2.5 Changepoints

An example of how combining kernels can give rise to more structured priors is given by changepoint kernels. Changepoints kernels can be defined through addition and

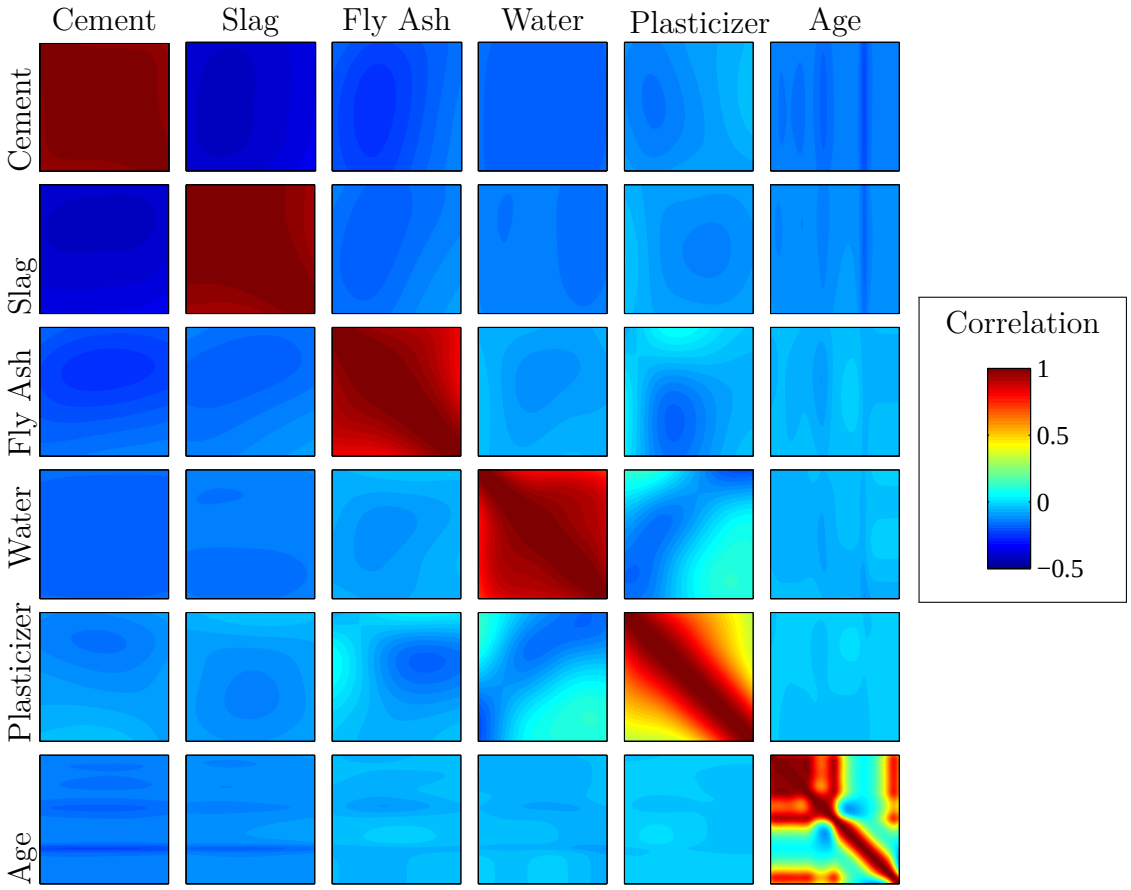



Figure 2.8 Posterior correlations between, and within, the additive components explaining the concrete dataset. Each plot shows the posterior correlations between two components, plotted over the domain of the data ±5%. Red indicates high correlation, teal indicates no correlation, and blue indicates negative correlation. Plots on the diagonal show posterior correlations within each component. Dimensions 'Coarse' and 'Fine' are not shown, because their variance is zero.

multiplication with sigmoidal functions such as $\sigma(x) = 1/1+\exp(-x)$:

$$\mathrm{CP}(k_1, k_2)(x, x') = \sigma(x)k_1(x, x')\sigma(x') + (1 - \sigma(x))k_2(x, x')(1 - \sigma(x')) \tag{2.19}$$

which can be written in shorthand as

$$\mathrm{CP}(k_1, k_2) = k_1 \times \boldsymbol{\sigma} \; + \; k_2 \times \bar{\boldsymbol{\sigma}} \tag{2.20}$$

where $\boldsymbol{\sigma} = \sigma(x)\sigma(x')$ and $\bar{\boldsymbol{\sigma}} = (1 - \sigma(x))(1 - \sigma(x'))$.

This compound kernel expresses a change from one kernel to another. The parameters of the sigmoid determine where, and how rapidly, this change occurs. Figure 2.9 shows some examples.

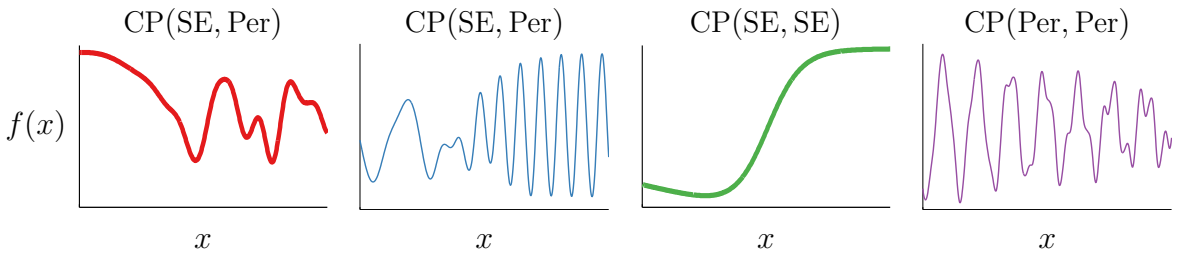



Figure 2.9 Draws from different priors on using changepoint kernels, constructed by adding and multiplying together base kernels with sigmoidal functions.

We can also build a model of functions whose structure changes only within some interval – a *change-window* – by replacing $\sigma(x)$ with a product of two sigmoids, one increasing and one decreasing.

### 2.5.1 Multiplication by a known function

More generally, we can model an unknown function that's been multiplied by any fixed, known function $a(x)$, by multiplying the kernel by $a(\mathbf{x})a(\mathbf{x}')$. Formally,

$$f(\mathbf{x}) = a(\mathbf{x})g(\mathbf{x}), \quad g \sim \mathcal{GP}(\,0, k(\mathbf{x}, \mathbf{x}')) \quad \Longleftrightarrow \quad f \sim \mathcal{GP}(\,0, a(\mathbf{x})k(\mathbf{x}, \mathbf{x}')a(\mathbf{x}'))\,. \tag{2.21}$$

## 2.6 Feature representation of kernels

By Mercer's theorem (Mercer, 1909), any positive-definite kernel can be represented as the inner product between a fixed set of features, evaluated at $\mathbf{x}$ and at $\mathbf{x}'$:

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{h}(\mathbf{x})^\mathsf{T}\mathbf{h}(\mathbf{x}') \tag{2.22}$$

For example, the squared-exponential kernel (SE) on the real line has a representation in terms of infinitely many radial-basis functions of the form $h_i(x) \propto \exp(-\frac{1}{2}\frac{(x-c_i)^2}{2\ell^2})$. More generally, any stationary kernel can be represented by a set of sines and cosines - a Fourier representation (Bochner, 1959). In general, any particular feature representation of a kernel is not unique (Minh et al., 2006).

In some cases, $\mathcal{X}$ can even be the infinite-dimensional feature mapping of another kernel. Composing feature maps in this way leads to *deep kernels*, explored in section 5.5.

### 2.6.1 Relation to linear regression

Surprisingly, GP regression is equivalent to Bayesian linear regression on the implicit features $\mathbf{h}(\mathbf{x})$ which give rise to the kernel:

$$f(\mathbf{x}) = \mathbf{w}^\mathsf{T}\mathbf{h}(\mathbf{x}), \quad \mathbf{w} \sim \mathcal{N}(0, \mathbf{I}) \quad \iff \quad f \sim \mathcal{GP}\Big(0, \mathbf{h}(\mathbf{x})^\mathsf{T}\mathbf{h}(\mathbf{x})\Big) \tag{2.23}$$

The link between Gaussian processes, linear regression, and neural networks is explored further in section 5.1.

### 2.6.2 Feature-space view of combining kernels

We can also view kernel addition and multiplication as a combination of the features of the original kernels. For example, given two kernels

$$k_a(\mathbf{x}, \mathbf{x}') = \mathbf{a}(\mathbf{x})^\mathsf{T}\mathbf{a}(\mathbf{x}') \tag{2.24}$$

$$k_b(\mathbf{x}, \mathbf{x}') = \mathbf{b}(\mathbf{x})^\mathsf{T}\mathbf{b}(\mathbf{x}') \tag{2.25}$$

their addition has the form:

$$k_a(\mathbf{x}, \mathbf{x}') + k_b(\mathbf{x}, \mathbf{x}') = \mathbf{a}(\mathbf{x})^\mathsf{T}\mathbf{b}(\mathbf{x}') + \mathbf{a}(\mathbf{x})^\mathsf{T}\mathbf{b}(\mathbf{x}') = \begin{bmatrix} \mathbf{a}(\mathbf{x}) \\ \mathbf{b}(\mathbf{x}) \end{bmatrix}^\mathsf{T} \begin{bmatrix} \mathbf{a}(\mathbf{x}') \\ \mathbf{b}(\mathbf{x}') \end{bmatrix} \tag{2.26}$$

meaning that the features of $k_a + k_b$ are the concatenation of the features of each kernel.

We can examine kernel multiplication in a similar way:

$$k_a(\mathbf{x}, \mathbf{x}') \times k_b(\mathbf{x}, \mathbf{x}') = \left[\mathbf{a}(\mathbf{x})^\mathsf{T}\mathbf{a}(\mathbf{x}')\right] \times \left[\mathbf{b}(\mathbf{x})^\mathsf{T}\mathbf{b}(\mathbf{x}')\right] \tag{2.27}$$

$$= \sum_i a_i(\mathbf{x})a_i(\mathbf{x}') \times \sum_j b_j(\mathbf{x})b_j(\mathbf{x}') \tag{2.28}$$

$$= \sum_{i,j} \left[a_i(\mathbf{x})b_j(\mathbf{x})\right]\left[a_i(\mathbf{x}')b_j(\mathbf{x}')\right] \tag{2.29}$$

In words, the features of $k_a \times k_b$ are made of up all pairs of the original two sets of features. For example, the features of the product of two one-dimensional SE kernels cover the plane with two-dimensional radial-basis functions of the form

$$h_{ij}(x_1, x_2) \propto \exp\left(-\frac{1}{2}\frac{(x_1 - c_i)^2}{2\ell_1^2}\right) \exp\left(-\frac{1}{2}\frac{(x_2 - c_j)^2}{2\ell_2^2}\right) \tag{2.30}$$

## 2.7 Expressing symmetries and invariances

When modeling functions, encoding known symmetries can improve predictive accuracy. This section looks at different ways to encode symmetries into a prior on functions. Many types of symmetry can be enforced through operations on the kernel.

We will demonstrate the properties of the resulting models by sampling functions from their priors. By using these functions to define warpings from $\mathbb{R}^2 \to \mathbb{R}^3$, we will show how to build a nonparametric prior on an open-ended family of topological manifolds, such as cylinders, toruses, and Möbius strips.

### 2.7.1 Three recipes for invariant priors

Consider the scenario where we have a finite set of transformations of the input space $\{g_1, g_2, \ldots\}$ to which we wish our function to remain invariant:

$$f(\mathbf{x}) = f(g(\mathbf{x})) \quad \forall \mathbf{x} \in \mathcal{X}, \quad \forall g \in G \tag{2.31}$$

As an example, imagine we want to build a model of functions invariant to swapping their inputs: $f(x_1, x_2) = f(x_2, x_1)$. Being invariant to a set of operations is equivalent to being invariant to all compositions of those operations, the set of which form a group. (Armstrong et al., 1988, chapter 21). In our example, the elements of the group $G_{\text{swap}}$

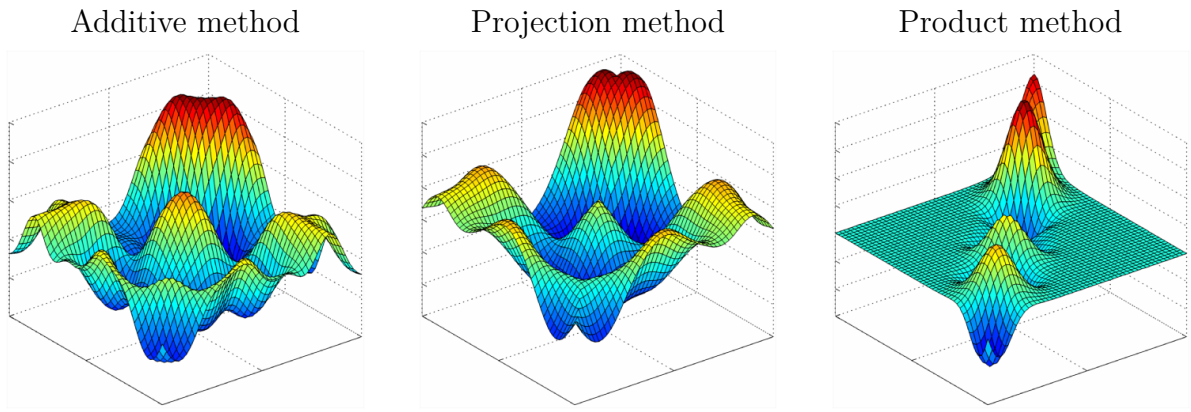



Figure 2.10 Three methods of introducing symmetry, illustrated through draws from the corresponding priors. All three methods introduce a different type of nonstationarity.

containing the operations the functions are invariant to has two elements:

$$g_1([x_1, x_2]) = [x_2, x_1] \qquad \text{(swap)} \tag{2.32}$$

$$g_2([x_1, x_2]) = [x_1, x_2] \qquad \text{(identity)} \tag{2.33}$$

How can we construct a prior on functions which respect these symmetries?

Ginsbourger et al. (2012) and Ginsbourger et al. (2013) showed that the only way to construct a GP prior on functions which respect a set of invariances is to construct a kernel which respects the same invariances with respect to each of its two inputs:

$$k(\mathbf{x}, \mathbf{x}') = k(g(\mathbf{x}), g(\mathbf{x}')), \quad \forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}, \quad \forall g, g' \in G \tag{2.34}$$

Formally, given a finite group $G$ whose elements are operations to which we wish our function to remain invariant, and $f \sim \mathcal{GP}(0, k(\mathbf{x}, \mathbf{x}'))$, then every $f$ is invariant under $G$ (up to a modification) if and only if $k(\cdot, \cdot)$ is argument-wise invariant under $G$.

It might not always be clear how to construct a kernel respecting such argument-wise invariances. Fortunately, there are a few simple ways to do this for any finite group:

1. **Sum over the orbit.** Ginsbourger et al. (2012) and Kondor (2008) suggest enforcing invariances through a double sum over the orbits of $\mathbf{x}$ and $\mathbf{x}'$ with respect

to G:

$$k_{\text{sum}}(\mathbf{x}, \mathbf{x}') = \sum_{g,\in G} \sum_{g' \in G} k(g(\mathbf{x}), g'(\mathbf{x}')) \tag{2.35}$$

For the group $G_{\text{swap}}$, this operation results in the kernel:

$$\begin{aligned} k_{\text{switch}}(\mathbf{x}, \mathbf{x}') &= \sum_{g \in G_{\text{swap}}} \sum_{g' \in G_{\text{swap}}} k(g(\mathbf{x}), g'(\mathbf{x}')) && (2.36) \\ &= k(x_1, x_2, x_1', x_2') + k(x_1, x_2, x_2', x_1') \\ &\quad + k(x_2, x_1, x_1', x_2') + k(x_2, x_1, x_2', x_1') && (2.37) \end{aligned}$$

For stationary kernels, some pairs of elements in this sum will be identical, and can be ignored. Figure 2.10(left) shows a draw from a GP prior with a product of SE kernels symmetrized in this way. This construction has the property that the marginal variance is doubled near $x_1 = x_2$, which may or may not be desirable.

2. **Project onto a fundamental domain.** Ginsbourger et al. (2013) also explore the possibility of projecting each datapoint into a fundamental domain of the group, using a mapping $A_G$:

$$k_{\text{proj}}(\mathbf{x}, \mathbf{x}') = k(A_G(\mathbf{x}), A_G(\mathbf{x}')) \tag{2.38}$$

For the group $G_{\text{swap}}$, a fundamental domain is $\{x_1, x_2 : x_1 < x_2\}$, which can be mapped to using $A_{G_{\text{swap}}}(x_1, x_2) = \Big[\min(x_1, x_2), \max(x_1, x_2)\Big]$. Constructing a kernel using this method introduces a non-differentiable "seam" along $x_1 = x_2$, as shown in figure 2.10(center).

3. **Multiply over the orbit.** Ryan P. Adams (personal communication) suggested a construction enforcing invariances through products over the orbits:

$$k_{\text{sum}}(\mathbf{x}, \mathbf{x}') = \prod_{g \in G} \prod_{g' \in G} k(g(\mathbf{x}), g'(\mathbf{x}')) \tag{2.39}$$

This method can sometimes produce GP priors with zero variance in some regions of the space, as in figure 2.10(right).

There are many possible ways to achieve a given symmetry, but we must be careful to do so without compromising other qualities of the model we are constructing. For example,

simply setting $k(\mathbf{x}, \mathbf{x}') = 0$ gives rise to a GP prior which obeys *all possible* symmetries, but this is presumably not a model we wish to use.

### 2.7.2 Example: Periodicity

Periodicity in a one-dimensional function corresponds to the invariance

$$f(x) = f(x + \tau) \tag{2.40}$$

where $\tau$ is the period.

The most popular method for building a periodic kernel is due to MacKay (1998), who used the projection method in combination with an SE kernel. A fundamental domain of the symmetry group is a circle, so the kernel

$$\mathrm{Per}(x, x') = \mathrm{SE}\left(\sin(x), \sin(x')\right) \times \mathrm{SE}\left(\cos(x), \cos(x')\right) \tag{2.41}$$

achieves the invariance in equation (2.40). Simple algebra reduces this kernel to the form shown in figure 2.1.

### 2.7.3 Example: Symmetry about zero

Another simple example of an easily-enforceable symmetry is symmetry about zero:

$$f(x) = f(-x) \tag{2.42}$$

using the sum over orbits method, by the transform

$$k_{\mathrm{reflect}}(x, x') = k(x, x') + k(x, -x') + k(-x, x') + k(-x, -x') \tag{2.43}$$

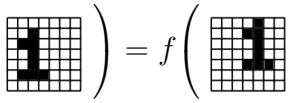
### 2.7.4 Example: Translation invariance in images

Many models of images are invariant to spatial translations (LeCun and Bengio, 1995). Similarly, many models of sounds are also invariant to translation through time.

This sort of translation invariance is completely distinct from the stationarity of kernels such as SE or Per. A stationary kernel implies that the prior is invariant to translations of the entire training and test set. In contrast, here we use translation invariance to refer to situations where the signal has been discretized, and each pixel (or the audio equivalent) corresponds to a different input dimension. We are interested

in creating priors on functions that are invariant to swapping pixels in a manner that corresponds to shifting the signal in some direction:

$$f\left(\quad\right) = f\left(\quad\right) \tag{2.44}$$



For example, in a one-dimensional image or audio signal, translation of an input vector by $i$ pixels can be defined as

$$\text{shift}(\mathbf{x}, i) = \left[x_{\text{mod}(i+1,D)}, x_{\text{mod}(i+2,D)}, \ldots, x_{\text{mod}(i+D,D)}\right]^{\mathsf{T}} \tag{2.45}$$

As above, translation invariance in one dimension can be achieved by a double sum over the orbit

$$k_{\text{invariant}}\left(\mathbf{x}, \mathbf{x}'\right) = \sum_{i=1}^{D}\sum_{j=1}^{D} k(\text{shift}(\mathbf{x}, i), \text{shift}(\mathbf{x}, j))\,, \tag{2.46}$$

defining the covariance between two signals to be the sum of all covariances between all translations of those two signals.

The extension to two dimensions, $\text{shift}(\mathbf{x}, i, j)$, is straightforward, but notationally cumbersome. Kondor (2008) built a more elaborate kernel between images, approximately invariant to both translation and rotation, using the projection method.

## 2.8 Generating topological manifolds

In this section we give a geometric illustration of the symmetries encoded by different compositions of kernels. The work presented in this section is based on a collaboration with David Reshef, Roger Grosse, Joshua B. Tenenbaum, and Zoubin Ghahramani. The derivation of the Möbius kernel was an original contribution by myself.

Priors on functions obeying invariants can be used to create a prior on topological manifolds by using such functions to warp a simply-connected surface into a higher-dimensional space. For example, to build a prior on 2-dimensional manifolds embedded in 3-dimensional space, we simply need a prior on mappings from $\mathbb{R}^2$ to $\mathbb{R}^3$, which we can construct using three independent functions $[f_1(\mathbf{x}), f_2(\mathbf{x}), f_3(\mathbf{x})]$, each mapping from $\mathbb{R}^2$ to $\mathbb{R}$. GP priors on the mapping functions implictly give rise to a prior on warped surfaces. Symmetries in $[f_1, f_2, f_3]$ will connect different parts of the manifolds, giving rise to non-trivial topologies on the sampled surfaces. Figure 2.11 shows 2D meshes

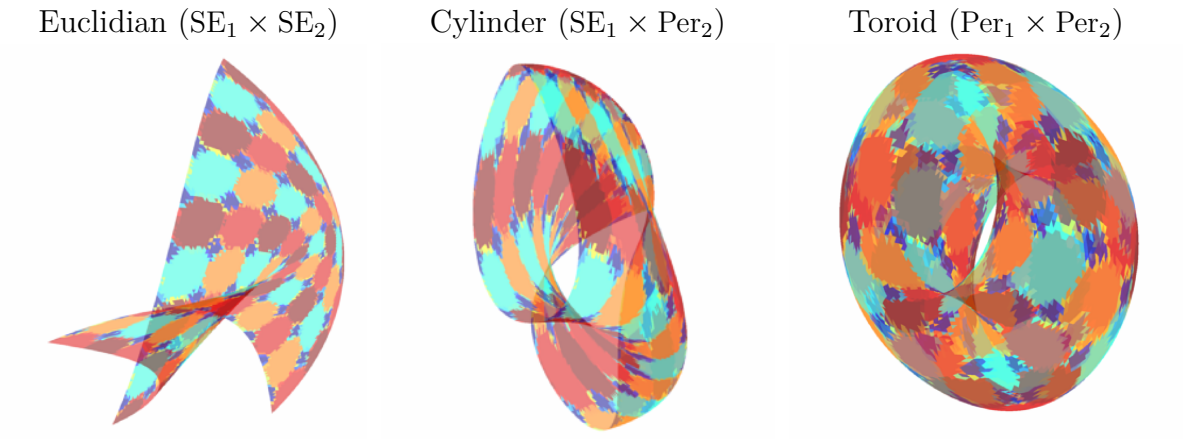



Figure 2.11 Generating 2D manifolds with different topologies. By enforcing that the functions mapping from $\mathbb{R}^2$ to $\mathbb{R}^3$ obey certain symmetries, the surfaces created have corresponding topologies, ignoring self-intersections.

warped into 3D by functions drawn from GP priors with different kernels, giving rise to a variety of different topologies. Higher-dimensional analogues of these shapes can be constructed by increasing the latent dimension and including corresponding terms in the kernel. For example, an $N$-dimensional space with kernel $\mathrm{Per}_1 \times \mathrm{Per}_2 \times \ldots \times \mathrm{Per}_N$ will give rise to a prior on $N$-dimensional toruses.

This construction is similar in spirit to the GP latent variable model (GP-LVM) of Lawrence (2005), which learns a latent embedding of the data into a low-dimensional space, using a GP prior on the mapping from the latent space to the observed space.

### 2.8.1 Möbius strips

A space having the topology of a Möbius strip can be constructed by enforcing the following invariance to the following operations (Reid and Szendrői, 2005, chapter 7):

$$g_{p_1}([x_1, x_2]) = [x_1 + \tau, x_2] \qquad \text{(periodic in } x_1\text{)} \tag{2.47}$$

$$g_{p_2}([x_1, x_2]) = [x_1, x_2 + \tau] \qquad \text{(periodic in } x_2\text{)} \tag{2.48}$$

$$g_s([x_1, x_2]) = [x_2, x_1] \qquad \text{(symmetric about } x_1 = x_2\text{)} \tag{2.49}$$

Section 2.7 already showed how to build GP priors invariant to each of these types of transformations. We'll call a kernel which enforces all of these symmetries a *Möbius*

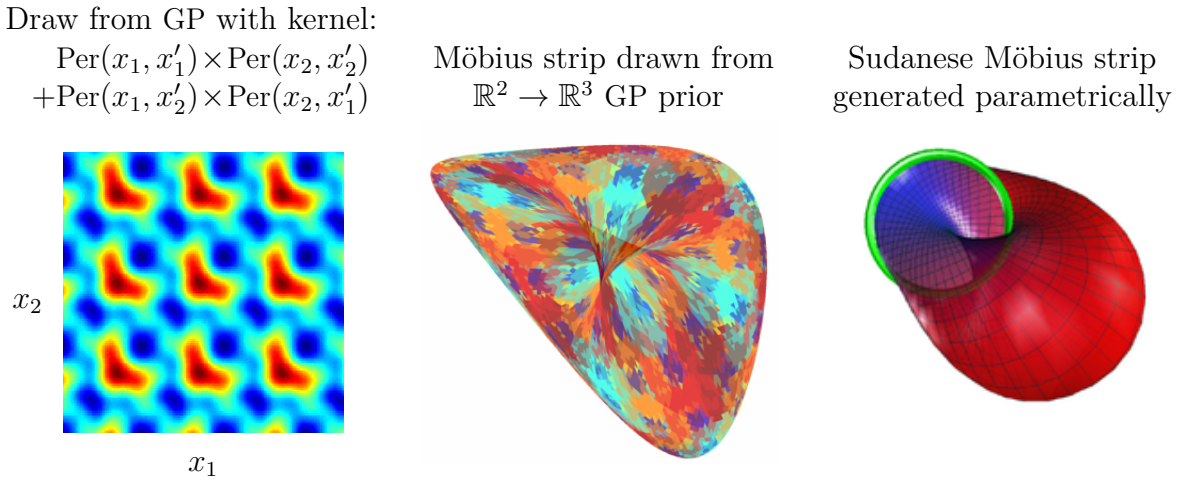



Figure 2.12 Generating Möbius strips. *Left:* A function drawn from a GP prior obeying the symmetries given by equations (2.47) to (2.49). *Center:* Simply-connected surfaces mapped from $\mathbb{R}^2$ to $\mathbb{R}^3$ by functions obeying those symmetries have a topology corresponding to a Möbius strip. Surfaces generated this way do not have the familiar shape of a flat surface connected to itself with a half-twist. Instead, they tend to look like *Sudanese* Möbius strips (Lerner and Asimov, 1984), whose edge has a circular shape. *Right:* A Sudanese projection of a Möbius strip. Image adapted from Wikimedia Commons (2005).

*kernel*. An example of such a kernel is:

$$k(x_1, x_2, x_1', x_2') = \mathrm{Per}(x_1, x_1') \times \mathrm{Per}(x_2, x_2') \; + \; \mathrm{Per}(x_1, x_2') \times \mathrm{Per}(x_2, x_1') \tag{2.50}$$

Moving along the diagonal $x_1 = x_2$ of a function drawn from the corresponding GP prior is equivalent to moving along the edge of a notional Möbius strip which has had that function mapped on to its surface. Figure 2.12(a) shows an example of a function drawn from such a prior. Figure 2.12(b) shows an example of a 2D mesh mapped to 3D by functions drawn from such a prior. This surface doesn't resemble the typical representation of a Möbius strip, but instead resembles an embedding known as the Sudanese Möbius strip (Lerner and Asimov, 1984), shown in figure 2.12(c).

## 2.9 Kernels on categorical variables

Categorical variables are variables which can take values only from a discrete, unordered set, such as $\{\texttt{blue}, \texttt{green}, \texttt{red}\}$. A flexible way to construct a kernel over categorical variables is to represent that variable by a set of binary variables, using a one-of-k

encoding. For example, if $\mathbf{x}$ can take one of four values, $x \in \{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}\}$, then a one-of-k encoding of $x$ will correspond to four binary inputs, and one-of-k($\texttt{C}$) $= [0, 0, 1, 0]$. Given a one-of-k encoding, we can place any multi-dimensional kernel on that space, such as the SE-ARD:

$$k_{\text{categorical}}(x, x') = \text{SE-ARD}(\text{one-of-k}(x), \text{one-of-k}(x')) \tag{2.51}$$

Short lengthscales on any particular dimension of the SE-ARD kernel indicate that the function value corresponding to that category is uncorrelated with the others.

A more flexible parameterization suggested by Kevin Swersky (personal communication) allows complete flexibility about which pairs of categories are similar to one another, replacing the SE-ARD kernel with a fully-parameterized kernel, SE-full:

$$\text{SE-full}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left(-\frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{x}'\right) \tag{2.52}$$

where $\mathbf{L}$ is a symmetric matrix, individually parameterizing the covariance between each pair of function values of categories.

## 2.10 Building a kernel in practice

This chapter outlined ways to choose the parametric form of a kernel in order to express different sorts of structure. Once the parametric form has been chosen, one still needs to choose, or integrating over, the kernel parameters. If the kernel we construct has relatively few parameters, these parameters can be estimated by maximum marginal likelihood, using gradient-based optimizers. The kernel parameters estimated in the examples above were optimized using the GPML toolbox, available at http://www.gaussianprocess.org/gpml/code.

A systematic search over kernel parameters is necessary when appropriate parameters aren't known. Likewise, sometimes appropriate kernel structures are hard to guess. The next chapter will show how to perform an automatic search not just over the kernel parameters, but also over the open-ended space of kernel expressions.

### Source code

Source code to produce all figures and examples in this chapter is available at http://www.github.com/duvenaud/phd-thesis.

# Chapter 3

# Automatic Model Construction

> "It would be very nice to have a formal apparatus that gives us some 'optimal' way of recognizing unusual phenomena and inventing new classes of hypotheses that are most likely to contain the true one; but this remains an art for the creative human mind."
>
> – E. T. Jaynes (1985)

In chapter 2, we saw that the choice of kernel determines the type of structure that can be learned by a GP model, and that a wide variety of models could be constructed by adding and multiplying a few base kernels together. However, we did not answer the difficult question of which kernel to use for a given problem. Even for experts, choosing the kernel in GP regression remains something of a black art.

The contribution of this chapter is to show a way to automate the process of building kernels for GP models. We define an open-ended space of kernels by adding and multiplying together kernels from a fixed set. We the define a procedure to search over this space to find a kernel which matches the structure in the data.

Searching over such a large, structured model class has two main benefits. First, this procedure has good predictive accuracy, since it tries out a large number of different regression models. Second, this procedure can sometimes discover interpretable structure in datasets. Because GP posteriors can be decomposed (as in section 2.4.3), we can examine the resulting structures visually. In chapter 4, we also show how to automatically generate English-language descriptions of the resulting models.

This chapter is based on work done in collaboration with James Robert Lloyd, Roger Grosse, Joshua B. Tenenbaum, and Zoubin Ghahramani. It was published in Duvenaud et al. (2013) and Lloyd et al. (2014). Myself, James Lloyd and Roger Grosse jointly developed the idea of searching through a grammar-based language of GP models, inspired

by Grosse et al. (2012), and wrote the first versions of the code together. James Lloyd ran most of the experiments, while I constructed most of the figures.

## 3.1 Ingredients of an automatic statistician

Gelman (2013) asks: "How can an artificial intelligence do statistics? ... It needs not just an inference engine, but also a way to construct new models and a way to check models. Currently, those steps are performed by humans, but the AI would have to do it itself". This section will discuss the different parts we think are required to build an artificial intelligence that can do statistics.

1. **An open-ended language of models.** Many learning algorithms consider all models in a class of fixed size. For example, graphical model learning algorithms (Eaton and Murphy, 2007; Friedman and Koller, 2003) search over different connectivity graphs for a given set of nodes. Such methods can be powerful, but human statisticians are sometimes capable of deriving *novel* model classes when required. An automatic search through an open-ended class of models can achieve some of this flexibility, possibly combining existing structures in novel ways.

2. **A search through model space.** An open-ended space of models cannot be searched exhaustively. Just as human researchers iteratively refine their models, search procedures can propose new search directions based on the results of previous model fits. Because any search in an open-ended space must start with relatively simple models before moving on to more complex ones, any search strategy is likely to resemble an iterative model-building procedure.

3. **A model comparison procedure.** A search strategy requires an objective to optimize. In this work, we use approximate marginal likelihood to compare models, penalizing complexity using the Bayesian Information Criterion as a heuristic. More generally, an automatic statistician needs to somehow check the models it has constructed. Gelman and Shalizi (2012) review the literature on model checking.

4. **A model description procedure.** Part of the value of statistical models comes from helping humans to understand a dataset or a phenomenon. Furthermore, a clear description of the statistical structure found in a dataset helps a user to notice when the dataset has errors, the wrong question was asked, the model-

building procedure failed to capture known structure, a relevant piece of data or constraint is missing, or when a novel statistical structure has been found.

In this chapter, we introduce a system containing all the above ingredients. We call this system the automatic Bayesian covariance discovery (ABCD) system. The next four sections of this chapter describe the mechanisms we use to incorporate these four ingredients into a limited example of an artificial intelligence which does statistics.

## 3.2 A language of regression models

As shown in chapter 2, we can construct a wide variety of kernel structures by adding and multiplying a small number of base kernels. We can therefore define a language of GP regression models simply by specifying a language of kernels.

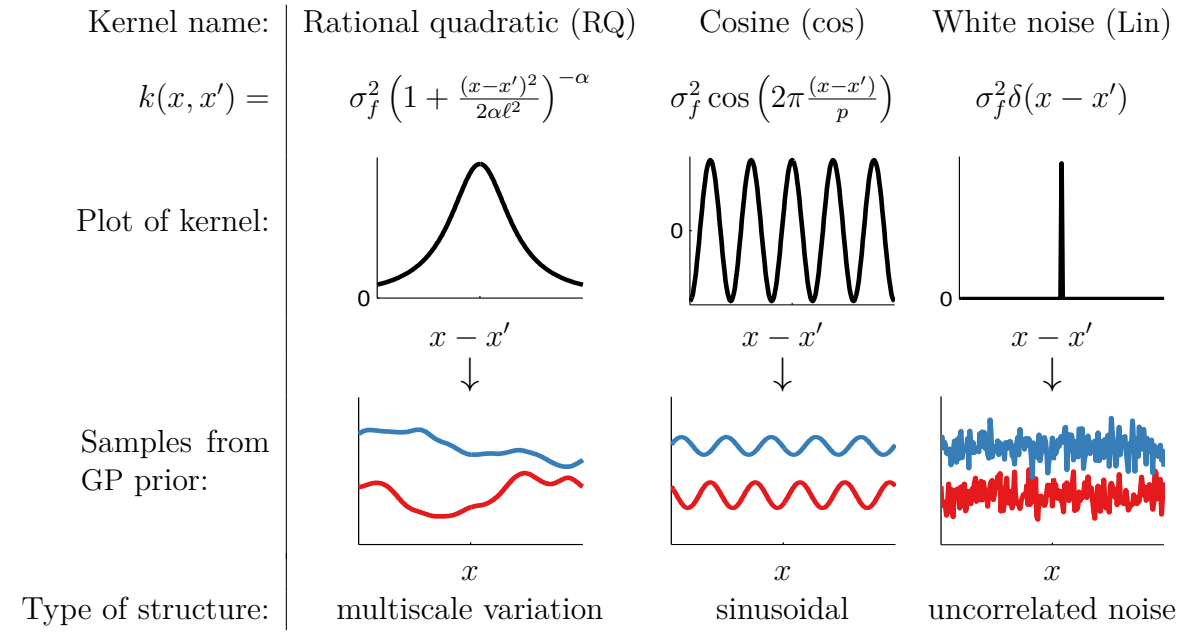



Figure 3.1 New base kernels introduced in this chapter, and the types of structure they encode. Other types of kernels can be constructed by adding and multiplying base kernels together.

Our language of models is specified by a set of base kernels which capture different properties of functions, and a set of rules which combine kernels to yield other valid kernels. In this chapter, we will use such base kernels as white noise (WN), constant (C), linear (Lin), squared-exponential (SE), rational-quadratic (RQ), sigmoidal ($\boldsymbol{\sigma}$) and

periodic (Per). We use a form of Per due to James Lloyd (personal communication) which has its constant component removed, and $\cos(x - x')$ as a special case. Figure 3.1 shows the new kernels introduced in this chapter. For precise definitions of all kernels, see appendix B.

To specify an open-ended language of structured kernels, we consider the set of all kernels that can be built by adding and multiplying these base kernels together, which we write in shorthand by:

$$k_1 + k_2 = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \tag{3.1}$$

$$k_1 \times k_2 = k_1(\mathbf{x}, \mathbf{x}') \times k_2(\mathbf{x}, \mathbf{x}') \tag{3.2}$$

The space of kernels produced by adding and multiplying the above set of kernels contains many existing regression models. Table 3.1 lists some of these, which are discussed in more detail in section 3.7.

| Regression model | Kernel structure | Example of related work |
|---|---|---|
| Linear regression | $\mathrm{C} + \mathrm{Lin} + \mathrm{WN}$ | |
| Polynomial regression | $\mathrm{C} + \prod \mathrm{Lin} + \mathrm{WN}$ | |
| Semi-parametric | $\mathrm{Lin} + \mathrm{SE} + \mathrm{WN}$ | Ruppert et al. (2003) |
| Multiple kernel learning | $\sum \mathrm{SE} + \mathrm{WN}$ | Gönen and Alpaydın (2011) |
| Trend, cyclical, irregular | $\sum \mathrm{SE} + \sum \mathrm{Per} + \mathrm{WN}$ | Lind et al. (2006) |
| Fourier decomposition | $\mathrm{C} + \sum \cos + \mathrm{WN}$ | |
| Sparse spectrum GPs | $\sum \cos + \mathrm{WN}$ | Lázaro-Gredilla et al. (2010) |
| Spectral mixture | $\sum \mathrm{SE} \times \cos + \mathrm{WN}$ | Wilson and Adams (2013) |
| Changepoints | e.g. $\mathrm{CP}(\mathrm{SE}, \mathrm{SE}) + \mathrm{WN}$ | Garnett et al. (2010) |
| Time-changing variance | e.g. $\mathrm{SE} + \mathrm{Lin} \times \mathrm{WN}$ | |
| Interpretable + flexible | $\sum_d \mathrm{SE}_d + \prod_d \mathrm{SE}_d$ | Plate (1999) |
| Additive GPs | *e.g.* $\prod_d (1 + \mathrm{SE}_d)$ | Chapter 6 |

Table 3.1 Common regression models expressible by sums and products of base kernels. $\cos(\cdot, \cdot)$ is a special case of our reparametrized $\mathrm{Per}(\cdot, \cdot)$.

## 3.3 A model search procedure

We explore this open-ended space of regression models using a simple greedy search. At each stage, we choose the highest scoring kernel, and propose modifying it by applying an operation to one of its parts, combining or replacing that part with another base kernel. The basic operations we can perform on any part $k$ of a kernel are:

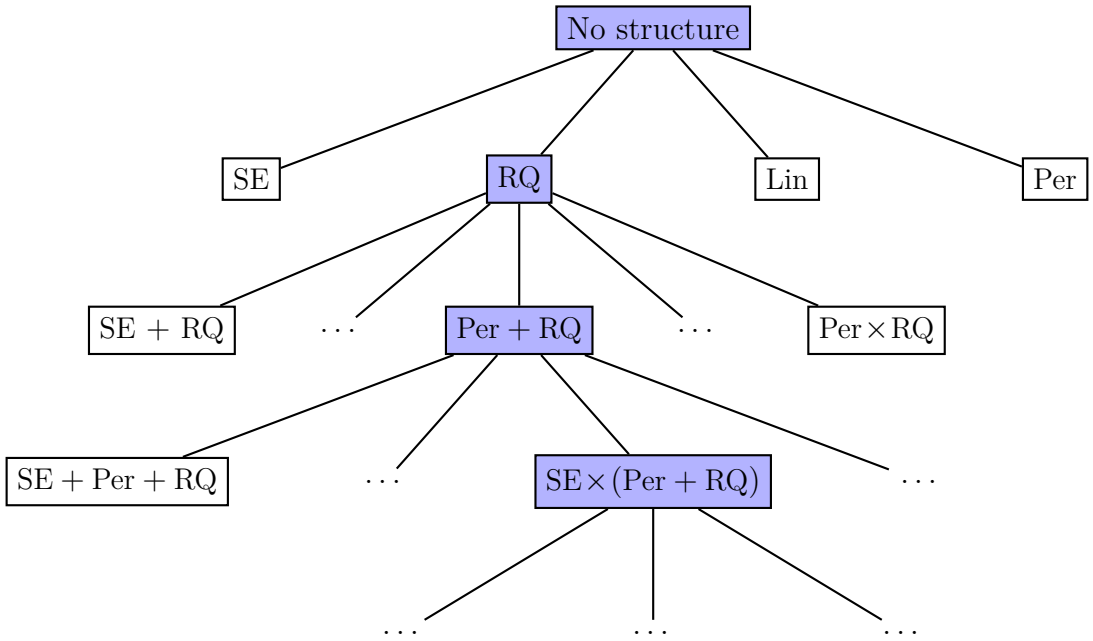



Figure 3.2 An example of a search tree over kernel expressions. Figure 3.3 shows the corresponding model increasing in sophistication as the kernel expression grows.

$$\begin{aligned} \text{Replacement:} \quad & k \quad \rightarrow \quad k' \\ \text{Addition:} \quad & k \quad \rightarrow \quad k + k' \\ \text{Multiplication:} \quad & k \quad \rightarrow \quad k \times k' \end{aligned}$$

where $k'$ is a new base kernel. These operators can generate all possible algebraic expressions involving addition and multiplication of base kernels. To see this, observe that if we restricted the addition and multiplication rules to only apply to base kernels, we would obtain a context-free grammar which generates the set of algebraic expressions.

Figure 3.2 shows an example search tree followed by our algorithm. Figure 3.3 shows how the resulting model changes as the search is followed. In practice, we also include extra operators which propose commonly-occuring structures, such as changepoints. A complete list is contained in appendix C.

Our search operators have rough parallels with strategies used by human researchers to construct regression models. In particular,

- One can look for structure in the residuals of a model, such as periodicity, and then extend the model to capture that structure. This corresponds to adding a new kernel to the existing structure.
- One can start with structure which is assumed to hold globally, such as linear-

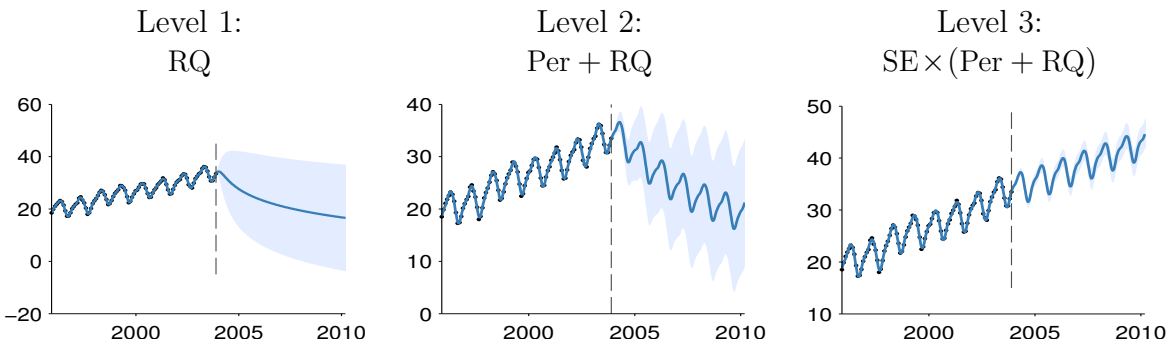



Figure 3.3 Posterior mean and variance for different depths of kernel search on the Mauna Loa dataset. The dashed line marks the end of the dataset. *Left:* First, the function is only modeled as a locally smooth function, and the extrapolation is poor. *Middle:* A periodic component is added, and the extrapolation improves. *Right:* At depth 3, the kernel can capture most of the relevant structure, and is able to extrapolate reasonably.

ity, but find that it only holds locally. This corresponds to multiplying a kernel structure by a local kernel such as SE.

- One can incorporate features incrementally, analogous to algorithms like boosting, back-fitting, or forward selection. This corresponds to adding or multiplying with kernels on dimensions not yet included in the model.

#### Hyperparameter initialization

Unfortunately, optimizing the marginal likelihood over parameters is not a convex optimization problem, and the space can have many local optima. For example, in data with periodic structure, integer multiples of the true period (harmonics) are often local optima. We take advantage of our search procedure to provide reasonable initializations: all parameters which were part of the previous kernel are initialized to their previous values. Newly introduced paramters are initialized randomly. In the newly proposed kernel, all parameters are then optimized using conjugate gradients. This procedure is not guaranteed to find the global optimum, but it implements the commonly used heuristic of iteratively modeling residuals.

## 3.4 A model comparison procedure

Choosing a kernel requires a method for comparing models. We choose marginal likelihood as our criterion, since it balances the fit and complexity of a model (Rasmussen and

Ghahramani, 2001). Conditioned on kernel parameters, the marginal likelihood of a GP can be computed analytically by equation (1.3). In addition, if we compare GP models by the maximum likelihood over their kernel parameters, then all else being equal, the model having more free parameters will be chosen. This introduces a bias in favor of more complex models.

We could avoid overfitting by integrating the marginal likelihood over all free parameters, but this integral is difficult to do in general. Instead, we approximate this integral using the Bayesian information criterion (BIC) (Schwarz, 1978):

$$\mathrm{BIC}(M) = \log p(D \mid M) - \frac{1}{2}|M| \log N \tag{3.3}$$

where $p(D|M)$ is the marginal likelihood of the data evaluated at the maximum-likelihood kernel parameters, $|M|$ is the number of kernel parameters, and $N$ is the number of data points. BIC simply penalizes the marginal likelihood in proportion to how many parameters the model has. Because BIC is a function of the number of parameters in a model, we did not count kernel parameters known to not affect the model. For example, when two kernels are multiplied, one of their output variance parameters becomes redundant, and can be ignored.

Other more sophisticated approximations are possible, such as Laplace's approximation. We chose to try BIC first because of its simplicity, and it performed reasonably well in our experiments.

## 3.5 A model description procedure

As discussed in chapter 2, a GP whose kernel is a sum of kernels can be viewed as a sum of functions drawn from different GPs. We can always express any kernel structure as a sum of products of kernels by distributing all products of sums. For example,

$$\mathrm{SE} \times (\mathrm{RQ} + \mathrm{Lin}) = \mathrm{SE} \times \mathrm{RQ} + \mathrm{SE} \times \mathrm{Lin} \tag{3.4}$$

When all kernels in a product apply to the same dimension, we can use the formulas in section 2.4.4 to visualize the marginal posterior distribution of that component. This decomposition into additive components provides a method of visualizing GP models which disentangles the different types of structure in the model.

The following section shows examples of such decomposition plots. In chapter 4, we extend this model visualization method to include automatically generated English text

explaining types of structure discovered.

## 3.6 Structure discovery in time series

To investigate our method's ability to discover structure, we ran the kernel search on several time-series. In the following examples, the search was run to depth 10, using SE, RQ, Lin, Per and WN as base kernels.

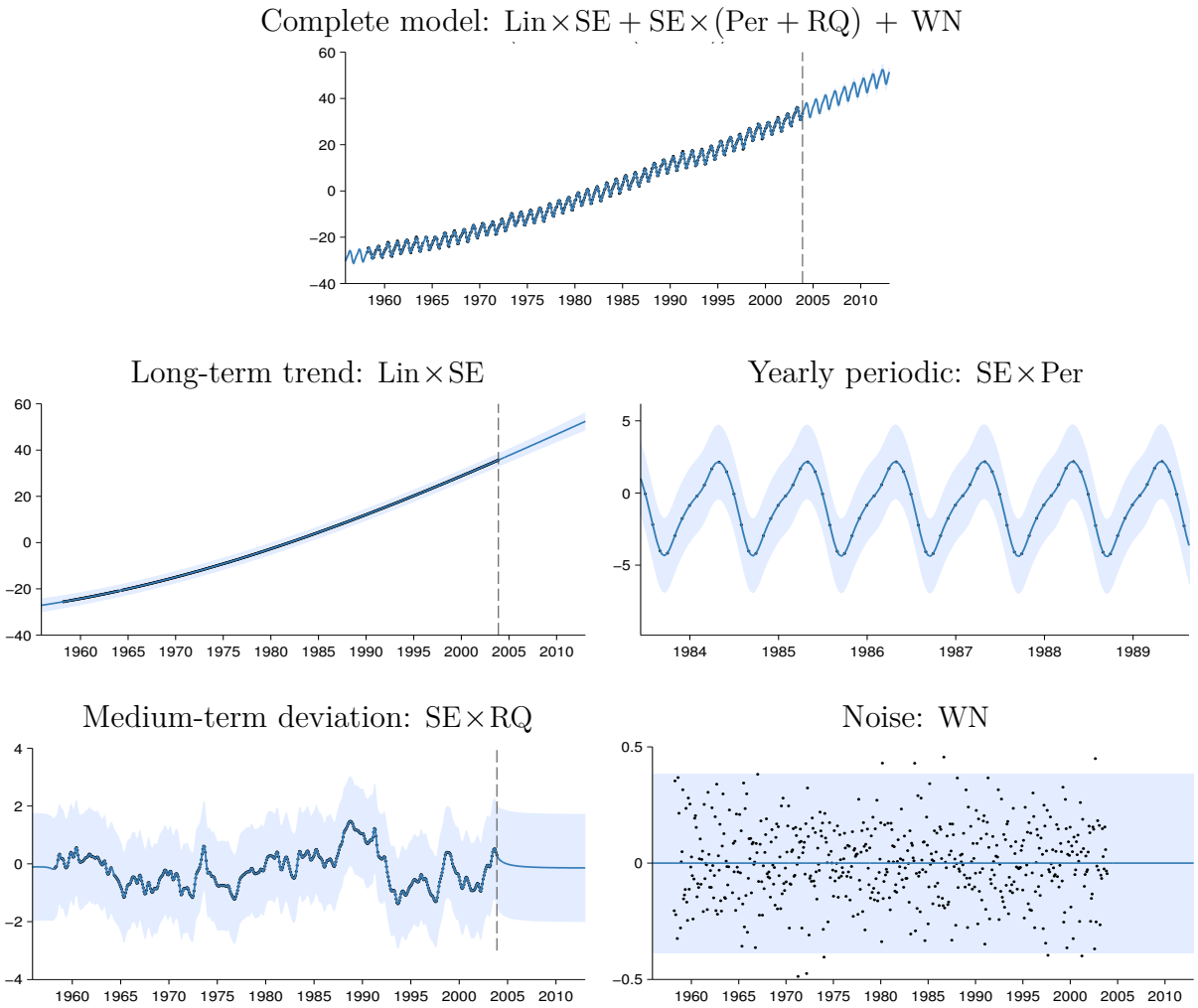



Figure 3.4 *First row:* The full posterior on the Mauna Loa dataset, after a search of depth 10. *Subsequent rows:* The automatic decomposition of the time series. The model is a sum of long-term, yearly periodic, medium-term components, and residual noise, respectively. The yearly periodic component has been rescaled for visibility.

### 3.6.1 Mauna Loa atmospheric $CO_2$

Our analyzed records of carbon dioxide levels recorded at the Mauna Loa observatory. Since this dataset was analyzed in detail by Rasmussen and Williams (2006), we can compare the kernel chosen by our method to a kernel constructed by human experts.

Figure 3.3 shows the posterior mean and variance on this dataset as the search depth increases. While the data can be smoothly interpolated by a model with only a single base kernel, the extrapolations improve dramatically as the increased search depth allows more structure to be included.

Figure 3.4 shows the final model chosen by our method together with its decomposition into additive components. The final model exhibits plausible extrapolation and interpretable components: a long-term trend, annual periodicity, and medium-term deviations. These are the same components chosen in the kernel hand-constructed by Rasmussen and Williams (2006, chapter 5).

We also plot the residuals modeled by a white noise (WN) component, showing that there is little obvious structure left in the data. More generally, some components capture slowly-changing structure while others capture quickly-varying struture, but there is no hard distinction between "signal" components and "noise" components.

### 3.6.2 Airline passenger counts

Figure 3.5 shows the decomposition produced by applying our method to monthly totals of international airline passengers (Box et al., 1970). We observe similar components to those in the Mauna Loa dataset: a long term trend, annual periodicity, and medium-term deviations. In addition, the composite kernel captures the near-linearity of the long-term trend, and the linearly growing amplitude of the annual oscillations.

The model search can be run without modification on multi-dimensional datasets, but the resulting structures are more difficult to visualize.

## 3.7 Related work

#### Building kernel functions by hand

Rasmussen and Williams (2006, chapter 5) devoted 4 pages to manually constructing a composite kernel to model the Mauna Loa dataset. Other examples of papers whose main contribution is to manually construct and fit a composite GP kernel are Preotiuc-Pietro and Cohn (2013), Lloyd (2013), and Klenske et al. (2013). These papers show that

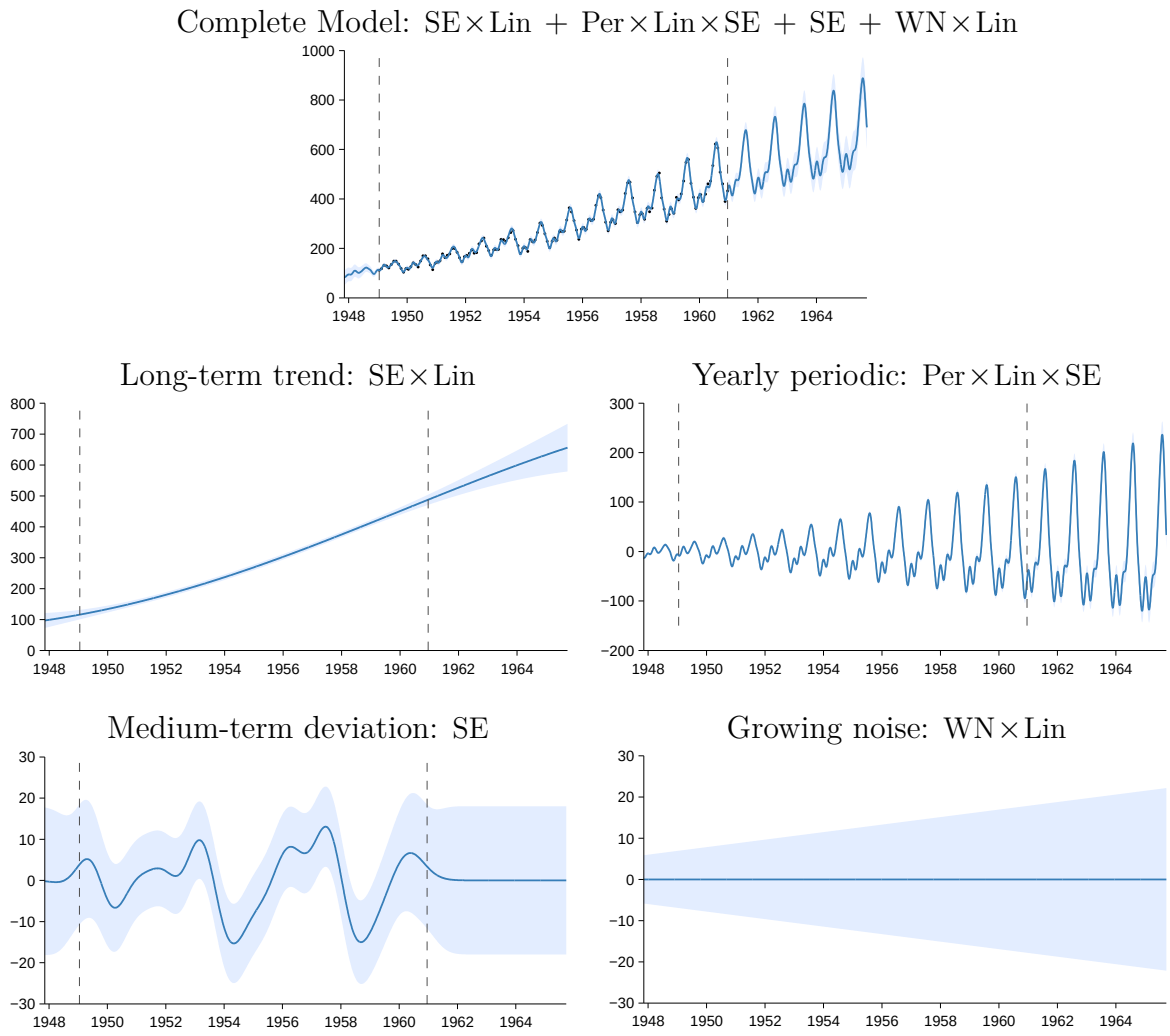



Figure 3.5 *First row:* The airline dataset and posterior after a search of depth 10. *Subsequent rows:* Additive decomposition of posterior into long-term smooth trend, yearly variation, and short-term deviations. Due to the linear kernel, the marginal variance grows over time, making this a heteroskedastic model.

experts are capable of constructing kernels, in one dimension, of similar complexity to the ones shown in this chapter. However, a more systematic search can consider possibilities that might otherwise be missed. For example, the kernel structure SE × Per × Lin, while appropriate for the airline dataset, had never been considered by the authors before it was chosen by the automatic search.

### Nonparametric regression in high dimensions

Nonparametric regression methods such as splines, locally-weighted regression, and GP regression are popular because they are capable of learning arbitrary smooth functions of the data. Unfortunately, they suffer from the curse of dimensionality: it is very difficult for these models to generalize well in more than a few dimensions.

Applying nonparametric methods in high-dimensional spaces can require imposing additional structure on the model. One such structure is additivity. Generalized additive models (Hastie and Tibshirani, 1990) assume the regression function is a transformed sum of functions defined on the individual dimensions: $\mathbb{E}[f(\mathbf{x})] = g^{-1}(\sum_{d=1}^{D} f_d(x_d))$. These models have a limited compositional form, but one which is interpretable and often generalizes well. In our grammar, we can capture such structure through sums of base kernels along different dimensions, although we have not yet tried incorporating a warping function $g(\cdot)$.

It is possible to add more flexibility to additive models by considering higher-order interactions between different dimensions. Chapter 6 considers GP models whose kernel implicitly sums over all possible interactions of input variables. Plate (1999) constructs a special case of this model class, summing an SE kernel along each dimension (for interpretability) plus a single SE-ARD kernel over all dimensions (for flexibility). Both of these models can be expressed in our grammar.

A closely related procedure is smoothing-splines ANOVA (Gu, 2002; Wahba, 1990). This model is a linear combinations of splines along each dimension, all pairs of dimensions, and possibly higher-order combinations. Because the number of terms to consider grows exponentially in the order, in practice, only terms of first and second order are usually considered.

Semi-parametric regression (e.g. Ruppert et al., 2003) attempts to combine interpretability with flexibility by building a composite model out of an interpretable, parametric part (such as linear regression) and a 'catch-all' nonparametric part (such as a GP with an SE kernel). This model class can be represented through the kernel SE + Lin.

### Kernel learning

There is a large body of work attempting to construct rich kernels through a weighted sum of base kernels, called multiple kernel learning (MKL) (e.g. Bach et al., 2004; Gönen and Alpaydın, 2011). These approaches usually have a convex objective function. However the component kernels, as well as their parameters, must be specified in advance.

We compare to a Bayesian variant of MKL in section 3.8, expressed as a restriction of our language of kernels.

Salakhutdinov and Hinton (2008) use a deep neural network with unsupervised pre-training to learn an embedding $g(\mathbf{x})$ onto which a GP with an SE kernel is placed: $\mathrm{Cov}\left[f(\mathbf{x}), f(\mathbf{x}')\right] = k(g(\mathbf{x}), g(\mathbf{x}'))$. This is a flexible approach to kernel learning, but relies mainly on finding structure in the input density $p(\mathbf{x})$. Instead, we focus on domains where most of the interesting structure is in $f(\mathbf{x})$.

Sparse spectrum GPs (Lázaro-Gredilla et al., 2010) approximate the spectral density of a stationary kernel function using delta functions which corresponds to kernels of the form $\sum \cos$. Similarly, Wilson and Adams (2013) introduced spectral mixture kernels, which approximate the spectral density using a mixture of Gaussians, corresponding to kernels of the form $\sum \mathrm{SE} \times \cos$. Both groups demonstrated, using Bochner's theorem (Bochner, 1959), that these kernels can approximate any stationary covariance function. Our language of kernels includes both of these kernel classes (see table 3.1).

### Changepoints

There is a wide body of work on changepoint modeling. Adams and MacKay (2007) developed a Bayesian online changepoint detection method which segments time-series into independent parts. This approach was extended by Saatçi et al. (2010) to Gaussian process models. Garnett et al. (2010) developed a family of kernels which modeled changepoints occurring abruptly at a single point. The changepoint kernel (CP) used in this work is a straightforward extension to smooth changepoints.

### Equation learning

Todorovski and Džeroski (1997), Washio et al. (1999) and Schmidt and Lipson (2009) learned parametric forms of functions specifying time series or relations between quantities. In contrast, ABCD learns a parametric form for the covariance, allowing it to model functions which do not have a simple parametric form but still have high-level structure. An examination of the structure discovered by the automatic equation-learning software Eureqa (Schmidt and Lipson, Accessed February 2013) on the airline and Mauna Loa datasets can be found in Lloyd et al. (2014).

**Structure discovery through grammars**

Kemp and Tenenbaum (2008) learned the structural form of a graph used to model human similarity judgements. Their grammar on graph structures includes planes, trees, and cylinders. Some of their discrete graph structures have continuous analogues in our own space. For example, $\mathrm{SE}_1 \times \mathrm{SE}_2$ and $\mathrm{SE}_1 \times \mathrm{Per}_2$ can be seen as mapping the data to a Euclidian surface and a cylinder, respectively. Section 2.8 examined these structures in more detail.

Diosan et al. (2007) and Bing et al. (2010) learned composite kernels for support vector machines and relevance vector machines, respectively, using genetic search algorithms to optimize cross-validation error. Similarly, Kronberger and Kommenda (2013) searched over composite kernels for GPs using genetic programming, optimizing the unpenalized marginal likelihood. These methods explore similar languages of kernels to the one explored in this chapter. It is not clear whether the complex genetic searches used by these methods offer advantages over the straightforward but naïve greedy search used in this chapter. Our search criterion has the advantages of being both differentiable with respect to kernel parameters, and of trading off model fit and complexity automatically. This prior work also did not explore the automatic model decomposition, summarization and description made possible by the use of GP models.

Grosse et al. (2012) performed a greedy search over a compositional model class for unsupervised learning, using a grammar of matrix decomposition models, and a greedy search procedure based on held-out likelihood. This model class contains many existing unsupervised models as special cases, and was able to discover diverse forms of structure, such as co-clustering or sparse latent feature models, automatically from data. Our framework takes a similar approach, but in a supervised setting.

Similarly, Steinruecken (2014) showed to automatically perform inference in arbitrary compositions of discrete sequence models. More generally, Dechter et al. (2013) and Liang et al. (2010) constructed grammars over programs, and automatically searched the resulting spaces.

## 3.8 Experiments

### 3.8.1 Interpretability versus accuracy

BIC trades off model fit and complexity by penalizing the number of parameters in a kernel expression. This can result in ABCD favoring kernel expressions with nested products

of sums, producing descriptions involving many additive components after expanding these structures. While these models typically have good predictive performance, their large number of components can make them less interpretable. We experimented with not allowing parentheses during the search, discouraging nested expressions. This was done by distributing all products immediately after each search operator was applied. We call this procedure ABCD-interpretability, in contrast to the unrestricted version of the search, ABCD-accuracy.

### 3.8.2 Predictive accuracy on time series

We evaluate the performance of the algorithms listed below on 13 real time-series from various domains from the time series data library (Hyndman, accessed July 2013). The pre-processed datasets used in our experiments are available at
`http://github.com/jamesrobertlloyd/gpss-research/tree/master/data/tsdlr`

#### Algorithms

We compare ABCD to equation learning using Eureqa (Schmidt and Lipson, Accessed February 2013), as well as six other regression algorithms: linear regression, GP regression with a single SE kernel (squared exponential), a Bayesian variant of multiple kernel learning (MKL) (e.g. Bach et al., 2004; Gönen and Alpaydın, 2011), changepoint modeling (e.g. Fox and Dunson, 2013; Garnett et al., 2010; Saatçi et al., 2010), spectral mixture kernels (Wilson and Adams, 2013) (spectral kernels), and trend-cyclical-irregular models (e.g. Lind et al., 2006).

We set Eureqa's search objective to the default mean-absolute-error. All algorithms besides Eureqa can be expressed as restrictions of our modeling language (see table 3.1), so we perform inference using the same search and objective function, with appropriate restrictions to the language.

We restricted to regression algorithms for comparability; we did not include models which regress on previous values of times series, such as auto-regressive or moving-average models (e.g. Box et al., 1970). Constructing a language of autoregressive time-series models would be an interesting area for future research.

#### Extrapolation experiments

To test extrapolation, we trained all algorithms on the first 90% of the data, predicted the remaining 10% and then computed the root mean squared error (RMSE). The

RMSEs were then standardised by dividing by the smallest RMSE for each data set, so the best performance on each data set will have a value of 1.

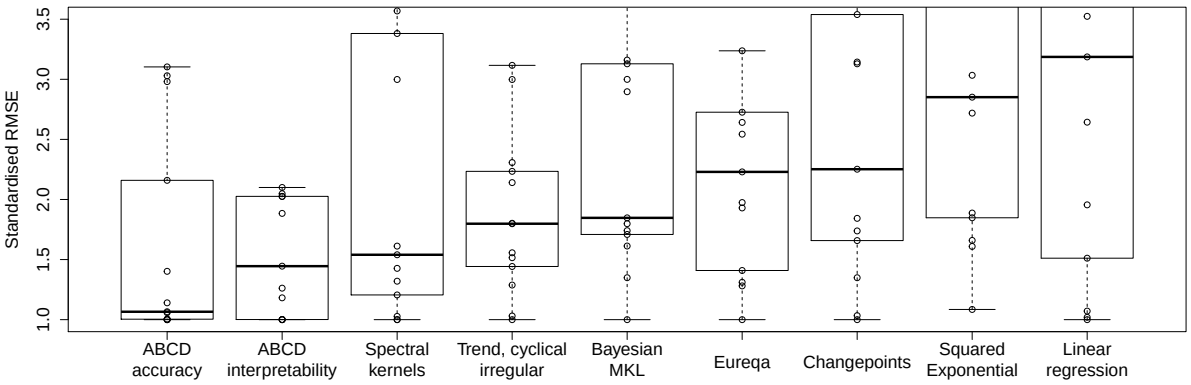



Figure 3.6 Box plot (showing median and quartiles) of standardised extrapolation RMSE (best performance = 1) on 13 time-series. The methods are ordered by median.

Figure 3.6 shows the standardised RMSEs across algorithms. ABCD-accuracy usually outperforms ABCD-interpretability. Both algorithms had lower quartiles than all other methods.

Overall, the model construction methods having richer languages of models perform better: ABCD outperforms trend-cyclical-irregular, which outperforms Bayesian MKL, which outperforms squared-exponential. Despite searching over a rich model class, Eureqa performs relatively poorly. This may be because few datasets are parsimoniously explained by a parametric equation, or because of the limited regularization ability of this procedure.

Not shown on the plot are large outliers for spectral kernels, Eureqa, squared exponential and linear regression with normalized RMSEs of 11, 493, 22 and 29 respectively.

### 3.8.3 Multi-dimensional prediction

ABCD can be applied to multidimensional regression problems without modification. An experimental comparison with other methods can be found in section 6.6, where it has the best performance on every dataset.

### 3.8.4 Structure recovery on synthetic data

The structure found in the examples above may seem reasonable, but we may wonder to what extent ABCD is consistent – that is, does it recover all the structure in any given dataset? It is difficult to tell from predictive accuracy alone if the search procedure is finding the correct structure, especially in multiple dimensions. To address this question, we tested our method's ability to recover known structure on a set of synthetic datasets.

For several composite kernel expressions, we constructed synthetic data by first sampling 300 points uniformly at random, then sampling function values at those points from a GP prior. We then added i.i.d. Gaussian noise to the functions at various signal-to-noise ratios (SNR).

Table 3.2 Kernels chosen by ABCD on synthetic data generated using known kernel structures. $D$ denotes the dimension of the function being modeled. SNR indicates the signal-to-noise ratio. Dashes (–) indicate no structure was found. Each kernel implicitly has a WN kernel added to it.

| True kernel | $D$ | SNR = 10 | SNR = 1 | SNR = 0.1 |
|---|---|---|---|---|
| $\mathrm{SE} + \mathrm{RQ}$ | 1 | SE | $\mathrm{SE} \times \mathrm{Per}$ | SE |
| $\mathrm{Lin} \times \mathrm{Per}$ | 1 | $\mathrm{Lin} \times \mathrm{Per}$ | $\mathrm{Lin} \times \mathrm{Per}$ | SE |
| $\mathrm{SE}_1 + \mathrm{RQ}_2$ | 2 | $\mathrm{SE}_1 + \mathrm{SE}_2$ | $\mathrm{Lin}_1 + \mathrm{SE}_2$ | $\mathrm{Lin}_1$ |
| $\mathrm{SE}_1 + \mathrm{SE}_2 \times \mathrm{Per}_1 + \mathrm{SE}_3$ | 3 | $\mathrm{SE}_1 + \mathrm{SE}_2 \times \mathrm{Per}_1 + \mathrm{SE}_3$ | $\mathrm{SE}_2 \times \mathrm{Per}_1 + \mathrm{SE}_3$ | – |
| $\mathrm{SE}_1 \times \mathrm{SE}_2$ | 4 | $\mathrm{SE}_1 \times \mathrm{SE}_2$ | $\mathrm{Lin}_1 \times \mathrm{SE}_2$ | $\mathrm{Lin}_2$ |
| $\mathrm{SE}_1 \times \mathrm{SE}_2 + \mathrm{SE}_2 \times \mathrm{SE}_3$ | 4 | $\mathrm{SE}_1 \times \mathrm{SE}_2 + \mathrm{SE}_2 \times \mathrm{SE}_3$ | $\mathrm{SE}_1 + \mathrm{SE}_2 \times \mathrm{SE}_3$ | $\mathrm{SE}_1$ |
| $(\mathrm{SE}_1 + \mathrm{SE}_2) \times (\mathrm{SE}_3 + \mathrm{SE}_4)$ | 4 | $(\mathrm{SE}_1 + \mathrm{SE}_2) \times \ldots$ $(\mathrm{SE}_3 \times \mathrm{Lin}_3 \times \mathrm{Lin}_1 + \mathrm{SE}_4)$ | $(\mathrm{SE}_1 + \mathrm{SE}_2) \times \ldots$ $\mathrm{SE}_3 \times \mathrm{SE}_4$ | – |

Table 3.2 shows the results. For the highest SNR, the method finds all relevant structure except in one case. The reported additional linear structure in the last row is can be explained the fact that functions sampled from SE kernels with long length-scales occasionally have near-linear trends. As the noise increases, our method generally backs off to simpler structures rather than over-fitting.

#### Source code

Source code to perform all experiments is available at
http://www.github.com/jamesrobertlloyd/gp-structure-search.

All GP parameter optimisation was performed by automated calls to the GPML toolbox, available at http://www.gaussianprocess.org/gpml/code.

## 3.9 Discussion

This chapter presented a system which constructs an appropriate model from an open-ended language and automatically generates plots decomposing the different types of structure present in the model.

This was done by introducing a space of kernels defined by sums and products of a small number of base kernels. The set of models in this space includes many standard regression models. We proposed a search procedure for this space of kernels, and argued that this search process parallels the process of model-building by statisticians.

We found that the learned structures are often capable of accurate extrapolation in time-series datasets, and are competitive with widely used kernel classes on a variety of prediction tasks. The learned kernels can yield decompositions of a signal into diverse and interpretable components, enabling model-checking by humans. We hope that this procedure has the potential to make powerful statistical model-building techniques accessible to non-experts.

# Chapter 4

# Automatic Model Description

> "Not a wasted word. This has been a main point to my literary thinking all my life."
>
> – *Hunter S. Thompson*

The previous chapter showed how to automatically build structured models by searching through a language of kernels. It also showed how to decompose the resulting models into the different types of structure present, and how to visually illustrate the type of structure captured by each component. This chapter shows how automatically describe the resulting model structures with English text.

The main idea is to treat every component of a product kernel as an adjective, or as a short phrase that modifies the description of a kernel. To see how this could work, recall that the model decomposition plots of chapter 3 showed that most of the structure in each component was determined by that component's kernel. Even across different datasets, the meaning of individual parts of kernels is consistent in some ways. For example, Per indicates repeating structure, and SE indicates smooth change over time.

This chapter also presents a system that generates reports combining automatically generated text and plots which highlight interpretable features discovered in a data sets. A complete example of an automatically-generated report can be found in appendix D.

The work appearing in this chapter was written in collaboration with James Robert Lloyd, Roger Grosse, Joshua B. Tenenbaum, and Zoubin Ghahramani, and was published in Lloyd et al. (2014). The procedure translating kernels into adjectives developed out of discussions between James and myself. James Lloyd wrote the code to automatically generate reports, and ran all of the experiments. The paper (upon which this chapter is based) was written mainly by James Lloyd and myself.

## 4.1 Generating descriptions of composite kernels

There are two main features of our language of GP models that allow description to be performed automatically. Firstly, any kernel expression in the language can be simplified into a sum of products. As discussed in section 2.4, a sum of kernels corresponds to a sum of functions, so each product can be described separately as part of a sum. Second, each kernel in a product modifies the resulting model in a consistent way. Therefore, one can describe a product of kernels by concatenating descriptions of the effect of each part of the product. One part needs to be described using a noun.

For example, one can describe the product of kernels $\mathrm{Per} \times \mathrm{SE}$ by representing Per by a noun ("a periodic function") modified by a phrase representing the effect of the SE kernel ("whose shape varies smoothly over time"). To simplify the system, we restricted base kernels to the set {C, Lin, WN, SE, Per, and $\boldsymbol{\sigma}$}. Recall that the sigmoidal kernel $\boldsymbol{\sigma}(x, x') = \sigma(x)\sigma(x')$ allows changepoints and change-windows.

### 4.1.1 Simplification rules

In order to be able to use the same phrase to describe the effect of each base kernel in different circumstances, our system converts each kernel expression into a standard, simplified form.

First, our system distributes all products of sums into sums of products. Then, it applies several simplification rules to the kernel expression:

- Products of two or more SE kernels can be equivalently replaced by a single SE with different parameters.
- Multiplying the white-noise kernel (WN) by any stationary kernel (C, WN, SE, or Per) gives another WN kernel.
- Multiplying any kernel by the constant kernel (C) only changes the parameters of the original kernel, and so can be factored out of any product in which it appears.

After applying these rules, any composite kernel expressible by the grammar can be written as a sum of terms of the form:

$$K \prod_m \mathrm{Lin}^{(m)} \prod_n \boldsymbol{\sigma}^{(n)}, \tag{4.1}$$

where $K$ is one of $\{\mathrm{WN}, \mathrm{C}, \mathrm{SE}, \prod_k \mathrm{Per}^{(k)}\}$ or $\{\mathrm{SE} \prod_k \mathrm{Per}^{(k)}\}$, and $\prod_i k^{(i)}$ denotes a product of kernels, each with different parameters. Superscripts denote different instances of the same kernel appearing in a product: $\mathrm{SE}^{(1)}$ can have different kernel parameters than $\mathrm{SE}^{(2)}$.

### 4.1.2 Describing each part of a product of kernels

Each kernel in a product modifies the resulting GP model in a consistent way. This allows one to describe the contribution of each kernel in a product as an adjective, or more generally as a modifier of a noun.

We now describe how each of the kernels in our grammar modifies a GP model:

- **Multiplication by SE** removes long range correlations from a model, since $\mathrm{SE}(x, x')$ decreases monotonically to 0 as $|x - x'|$ increases. This converts any global correlation structure into local correlation only.

- **Multiplication by Lin** is equivalent to multiplying the function being modeled by a linear function. If $f(x) \sim \mathrm{GP}(0, k)$, then $x \times f(x) \sim \mathrm{GP}(0, \mathrm{Lin} \times k)$. This causes the standard deviation of the model to vary linearly, without affecting the correlation between function values.

- **Multiplication by $\sigma$** is equivalent to multiplying the function being modeled by a sigmoid, which means that the function goes to zero before or after some point.

- **Multiplication by Per** removes correlation between all pairs of function values not close to one period apart, allowing variation within each period, but maintaining correlation between periods.

- **Multiplication by any kernel** modifies the covariance in the same way as multiplying by a function drawn from a corresponding GP prior. This follows from the fact that if $f_1(x) \sim \mathrm{GP}(0, k_1)$ and $f_2(x) \sim \mathrm{GP}(0, k_2)$ then

$$\mathrm{Cov}\left[f_1(x) f_2(x), f_1(x') f_2(x')\right] = k_1(x, x') k_2(x, x'). \tag{4.2}$$

  Put more plainly, a GP whose covariance is a product of kernels has the same covariance as a product of two functions, each drawn from the corresponding GP prior. However, the distribution of $f_1 \times f_2$ is not GP distributed – it has higher central moments as well. This identity can be used to generate a cumbersome

"worst-case" description in cases where a more concise description of the effect of a kernel is not known. For example, it is used in our system to describe the product of more than one periodic kernel.

Table 4.1 gives the corresponding description of the effect of each type of kernel in a product, written as a post-modifier.

| Kernel | Postmodifier phrase |
|---|---|
| SE | whose shape changes smoothly |
| Per | modulated by a periodic function |
| Lin | with linearly varying amplitude |
| $\prod_k \mathrm{Lin}^{(k)}$ | with polynomially varying amplitude |
| $\prod_k \boldsymbol{\sigma}^{(k)}$ | which applies until / from [changepoint] |

Table 4.1 Descriptions of the effect of each kernel, written as a post-modifier.

Table 4.2 gives the corresponding description of each kernel before it has been multiplied by any other, written as a noun phrase.

| Kernel | Noun phrase |
|---|---|
| WN | uncorrelated noise |
| C | constant |
| SE | smooth function |
| Per | periodic function |
| Lin | linear function |
| $\prod_k \mathrm{Lin}^{(k)}$ | {quadratic, cubic, quartic, ...} function |

Table 4.2 Noun phrase descriptions of each type of kernel.

### 4.1.3 Combining descriptions into noun phrases

In order to build a noun phrase describing a product of kernels, our system chooses one kernel to act as the head noun, which is then modified by appending descriptions of the other kernels in the product.

As an example, a kernel of the form $\mathrm{Per} \times \mathrm{Lin} \times \boldsymbol{\sigma}$ could be described as a

$$\underbrace{\mathrm{Per}}_{\text{periodic function}} \times \underbrace{\mathrm{Lin}}_{\text{with linearly varying amplitude}} \times \underbrace{\boldsymbol{\sigma}}_{\text{which applies until 1700.}}$$

where Per was chosen to be the head noun.

In our system, the head noun is chosen according to the following ordering:

$$\text{Per}, \text{WN}, \text{SE}, \text{C}, \prod_m \text{Lin}^{(m)}, \prod_n \boldsymbol{\sigma}^{(n)} \tag{4.3}$$

Combining tables 4.1 and 4.2 with ordering 4.3 provides a general method to produce descriptions of sums and products of these base kernels.

### Extensions

In practice, the system also incorporate a number of other rules which help to make the descriptions shorter, easier to parse, or clearer:

- The system add extra adjectives depending on kernel parameters. For example, an SE with a relatively short lengthscale might be described as "a rapidly-varying smooth function" as opposed to just "a smooth function".

- Descriptions can include kernel parameters. For example, the system might write that a function is "repeating with a period of 7 days".

- Descriptions can include extra information about the model not contained in the kernel. For example, based on the slope of the posterior mean, the system might write "a linearly increasing function" as opposed to "a linear function".

- Some kernels can be described through pre-modifiers. For example, the system might write "an approximately periodic function" as opposed to "a periodic function whose shape changes smoothly".

### Ordering additive components

The reports generated by our system attempt to present the most interesting or important features of a data set first. As a heuristic, the system orders components by always adding next the component which most reduces the 10-fold cross-validated mean absolute error.

### 4.1.4 Worked example

This section shows an example of our procedure describing a compound kernel containing every type of base kernel in our set:

$$\mathrm{SE} \times (\mathrm{WN} \times \mathrm{Lin} \; + \; \mathrm{CP}(\mathrm{C}, \mathrm{Per})). \tag{4.4}$$

The kernel is first converted into a sum of products, and the changepoint is converted into sigmoidal kernels (recall the definition of changepoint kernels in section 2.5):

$$\mathrm{SE} \times \mathrm{WN} \times \mathrm{Lin} \; + \; \mathrm{SE} \times \mathrm{C} \times \boldsymbol{\sigma} \; + \; \mathrm{SE} \times \mathrm{Per} \times \bar{\boldsymbol{\sigma}} \tag{4.5}$$

which is then simplified using the rules in section 4.1.1 to

$$\mathrm{WN} \times \mathrm{Lin} \; + \; \mathrm{SE} \times \boldsymbol{\sigma} \; + \; \mathrm{SE} \times \mathrm{Per} \times \bar{\boldsymbol{\sigma}}. \tag{4.6}$$

To describe the first component, $(\mathrm{WN} \times \mathrm{Lin})$, the head noun description for WN, "uncorrelated noise", is concatenated with a modifier for Lin, "with linearly increasing standard deviation".

The second component, $(\mathrm{SE} \times \boldsymbol{\sigma})$, is described as "A smooth function with a length-scale of [lengthscale] [units]", corresponding to the SE, "which applies until [changepoint]".

Finally, the third component, $(\mathrm{SE} \times \mathrm{Per} \times \bar{\boldsymbol{\sigma}})$, is described as "An approximately periodic function with a period of [period] [units] which applies from [changepoint]".

## 4.2 Example descriptions

In this section, we demonstrate the ability of our procedure, ABCD, to write intelligible desciptions of the structure present in two time series. The examples presented here describe models produced by the automatic search method presented in chapter 3.

### 4.2.1 Summarizing 400 years of solar activity

First, we show excerpts from the report automatically generated on annual solar irradiation data from 1610 to 2011. This dataset is shown in figure 4.1.

This time series has two pertinent features: First, a roughly 11-year cycle of solar activity. Second, a period lasting from 1645 to 1715 having almost no variance. This flat

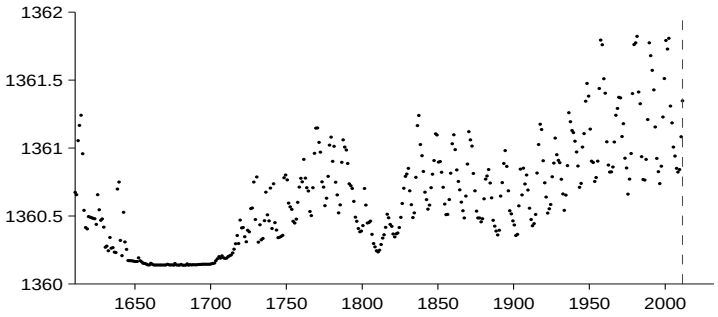



Figure 4.1 Solar irradiance data (Lean et al., 1995).

region corresponds to the Maunder minimum, a period in which sunspots were extremely rare (Lean et al., 1995). The Maunder minimum is an example of the type of structure which can be captured by change-windows.

- A constant.
- A constant. This function applies from 1643 until 1716.
- A smooth function. This function applies until 1643 and from 1716 onwards.
- An approximately periodic function with a period of 10.8 years. This function applies until 1643 and from 1716 onwards.

Figure 4.2 Automatically generated descriptions of the first four components discovered by ABCD on the solar irradiance data set. The dataset has been decomposed into diverse structures with simple descriptions.

The first section of each report generated by ABCD is a summary of the structure found in the dataset. Figure 4.2 shows natural-language summaries of the top four components discovered by ABCD on the solar dataset. From these summaries, we can see that the system has identified the Maunder minimum (second component) and the 11-year solar cycle (fourth component). These components are visualized and described in figures 4.3 and 4.5, respectively. The third component, visualized in figure 4.4, captures the smooth variation over time of the overall level of solar activity.

The complete report generated on this dataset can be found in appendix D. This report also contains samples from the model posterior.

### 4.2.2 Describing changing noise levels

Next, we present excerpts of the description generated by our procedure on a model of international airline passenger counts over time. The model described is the same as

This component is constant. This component applies from 1643 until 1716.

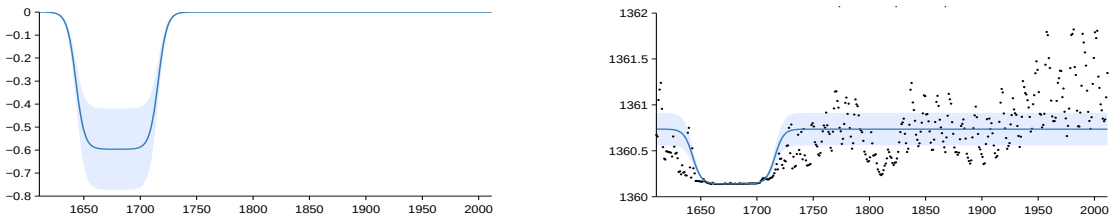



Figure 4: Pointwise posterior of component 2 (left) and the posterior of the cumulative sum of components with data (right)

Figure 4.3 Extract from an automatically-generated report describing the model component corresponding to the Maunder minimum.

This component is a smooth function with a typical lengthscale of 23.1 years. This component applies until 1643 and from 1716 onwards.

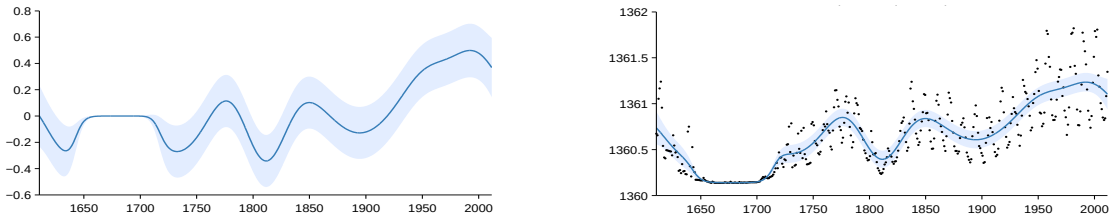



Figure 6: Pointwise posterior of component 3 (left) and the posterior of the cumulative sum of components with data (right)

Figure 4.4 Characterizing the medium-term smoothness of solar activity levels. By allowing other components to explain the periodicity, noise, and the Maunder minimum, ABCD can isolate the part of the signal best explained by a slowly-varying trend.

that shown in figure 3.5. High-level descriptions of the four components discovered are shown in figure 4.6.

The second component, shown in figure 4.7, is accurately described as approximately (SE) periodic (Per) with linearly growing amplitude (Lin).

The description of the fourth component, shown in figure 4.8, expresses the fact that the scale of the unstructured noise in the model grows linearly with time.

The complete report generated on this dataset can be found in the supplementary material of Lloyd et al. (2014). Other example reports describing a wide variety of time-series can be found at `http://mlg.eng.cam.ac.uk/lloyd/abcdoutput/`

This component is approximately periodic with a period of 10.8 years. Across periods the shape of this function varies smoothly with a typical lengthscale of 36.9 years. The shape of this function within each period is very smooth and resembles a sinusoid. This component applies until 1643 and from 1716 onwards.

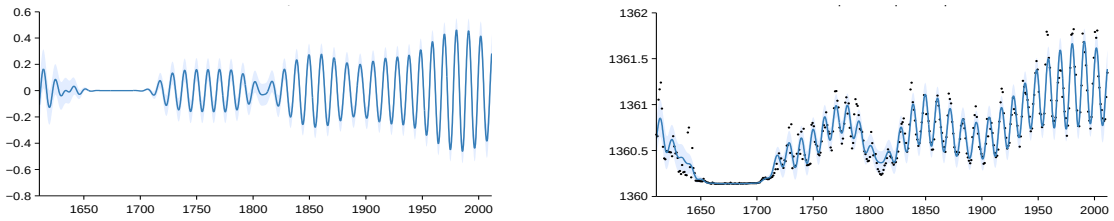



Figure 8: Pointwise posterior of component 4 (left) and the posterior of the cumulative sum of components with data (right)

Figure 4.5 This part of the report isolates and describes the approximately 11-year sunspot cycle, also noting its dissapearing during the Maunder minimum.

## 4.3 Related work

To the best of our knowledge, our procedure is the first example of automatic description of nonparametric statistical models. However, systems with natural language output have been developed for automatic video description (Barbu et al., 2012) and automated theorem proving (Ganesalingam and Gowers, 2013).

Although not a description procedure, Durrande et al. (2013) developed an analytic method for decomposing a GP posterior into entirely periodic and entirely non-periodic parts, even when using non-periodic kernels.

## 4.4 Limitations of this approach

During development, we noted several difficulties with this overall approach:

- **Some kernels are hard to describe.** For instance, we did not include the RQ kernel in the text-generation procedure. This was done for several reasons. First, the RQ kernel can be equivalently expressed as a scale mixture of SE kernels. Second, it was difficult to think of a clear and concise description for the hyperparameter which controls the heaviness of the tails of the RQ kernel. Third, a product of two RQ kernels does not give another RQ kernel, which raises the question of how to concisely describe products of RQ kernels.

- A linearly increasing function.
- An approximately periodic function with a period of 1.0 years and with linearly increasing amplitude.
- A smooth function.
- Uncorrelated noise with linearly increasing standard deviation.

Figure 4.6 Short descriptions of the four components of the airline model.

This component is approximately periodic with a period of 1.0 years and varying amplitude. Across periods the shape of this function varies very smoothly. The amplitude of the function increases linearly. The shape of this function within each period has a typical lengthscale of 6.0 weeks.

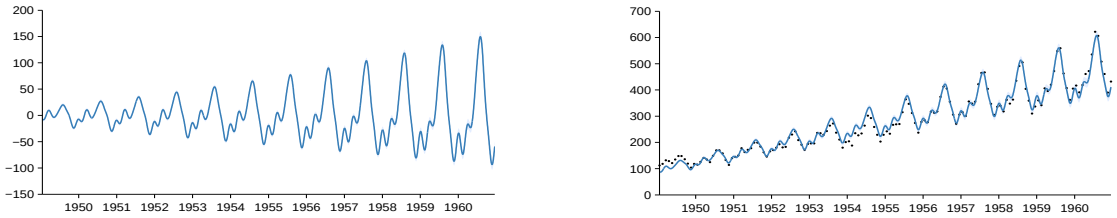



Figure 4: Pointwise posterior of component 2 (left) and the posterior of the cumulative sum of components with data (right)

Figure 4.7 Describing non-stationary periodicity in the airline data.

This component models uncorrelated noise. The standard deviation of the noise increases linearly.

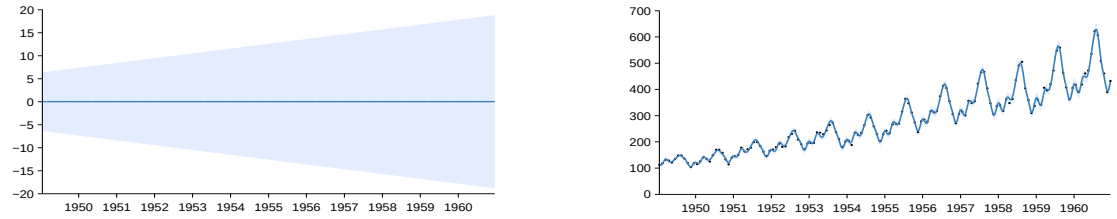



Figure 8: Pointwise posterior of component 4 (left) and the posterior of the cumulative sum of components with data (right)

Figure 4.8 Describing time-changing variance in the airline dataset.

- **Reliance on additivity.** Much of the modularity of the description procedure is due to the additive decomposition. However, additivity is lost under any nonlinear transformation of the output. Such warpings can be learned (Snelson et al., 2004), but descriptions of transformations of the data may not be as clear to the end user.

- **Difficulty of expressing uncertainty.** A natural extension to the model search procedure would be to report a posterior distribution on structures and kernel parameters, rather than point estimates. Describing uncertainty about the hyper-parameters of a particular structure may be feasible, but describing even a few most-probable structures might result in excessively long reports.

### Source code

Source code to perform all experiments is available at
http://www.github.com/jamesrobertlloyd/gpss-research.

## 4.5 Conclusions

This chapter presented a system which automatically generates detailed reports describing patterns captured by a GP model. The properties of GPs and the kernels being used allows a modular description, avoiding an exponential blowup in the number of special cases that needed to be considered.

Combined with the model search of chapter 3, this gives a procedure combining all the elements of an automatic statistician listed in section 3.1: an open-ended language of models, a search through model space, a model comparison procedure, and a model description procedure. Each particular element used in the procedure presented here is merely a proof-of-concept. However, even this simple prototype demonstrated the ability to discover and describe a variety of patterns in time series.

# Chapter 5

# Deep Gaussian Processes

> "I asked myself: On any given day, would I rather be wrestling with a sampler, or proving theorems?"
>
> – *Peter Orbanz*, personal communication

Choosing appropriate architectures and regularization strategies of deep networks is important for good predictive performance. In this chapter, we propose to study this problem by viewing deep nets as priors on functions. By viewing neural networks this way, we can analyze their properties without reference to any particular dataset, loss function, or training method. Instead, we can ask what sorts of information-processing structures these priors give rise to, and check whether those structures are the same sort which we expect to find in useful models.

As a starting point, we will relate neural networks to Gaussian processes, and examine a class of infinitely-wide, deep neural networks called *deep Gaussian processes* – compositions of functions drawn from GP priors. Deep GPs are an attractive model class to study for several reasons. Firstly, Damianou and Lawrence (2013) showed that the probabilistic nature of deep GPs gaurds against overfitting. Second, Hensman et al. (2014a) showed that stochastic variational inference is possible in deep GPs, allowing mini-batch training. Third, the availabilty of an approximate marginal likelihood allows us to automatically tune the model architecture without the need for cross-validation. Finally, Deep GPs are attractive from a model analysis point of view because they abstract away some of the details of finite neural networks.

Our analysis will show that in standard architectures, the representational capacity of standard deep networks tends to decrease as the number of layers increases, retaining only a single degree of freedom in the limit. We propose an alternate network architecture

that connects the input to each layer that does not suffer from this pathology. We also examine *deep kernels*, obtained by composing arbitrarily many fixed feature transforms.

The ideas contained in this chapter were developed through discussions with Oren Rippel, Ryan Adams and Zoubin Ghahramani, and appear in Duvenaud et al. (2014).

## 5.1 Relating deep neural networks to deep GPs

This section gives a precise definition of deep GPS, reviews the precise relationship between neural networks and Gaussian processes, and gives two equivalent ways of constructing neural networks which give rise to deep GPs.

### 5.1.1 Definition of deep GPs

We define a deep GP as a prior on functions constructed by composing draws from GP priors. An example of a deep GP is a composition of vector-valued functions, with each function in each layer drawn independently from GP priors:

$$\begin{aligned} \boldsymbol{f}^{(1:L)}(\mathbf{x}) &= \boldsymbol{f}^{(L)}(\boldsymbol{f}^{(L-1)}(\dots \boldsymbol{f}^{(2)}(\boldsymbol{f}^{(1)}(\mathbf{x}))\dots)) \\ \text{with each} \quad f_d^{(\ell)} &\overset{\text{ind}}{\sim} \text{GP}\left(0, k_d^{(\ell)}(\mathbf{x}, \mathbf{x}')\right) \end{aligned} \tag{5.1}$$

Multilayer neural networks also implement compositions of vector-valued functions, one per layer. Therefore, understanding properties of more general function compositions might helps us gain insight into deep neural networks.

### 5.1.2 Single-hidden-layer models

First, we relate neural networks to standard "shallow" Gaussian processes, using the standard neural network architecture known as the multi-layer perceptrion (MLP) (Rosenblatt, 1962). In the typical definition of an MLP with one hidden layer, the hidden unit activations are defined as:

$$\mathbf{h}(\mathbf{x}) = \sigma\left(\mathbf{b} + \mathbf{V}\mathbf{x}\right) \tag{5.2}$$

where $\mathbf{h}$ are the hidden unit activations, $\mathbf{b}$ is a bias vector, $\mathbf{V}$ is a weight matrix and $\sigma$ is a one-dimensional nonlinear function, usually sigmoidal, applied element-wise. The

output vector $\boldsymbol{f}(\mathbf{x})$ is simply a weighted sum of these hidden unit activations:

$$\boldsymbol{f}(\mathbf{x}) = \mathbf{W}\sigma\left(\mathbf{b} + \mathbf{V}\mathbf{x}\right) = \mathbf{W}\mathbf{h}(\mathbf{x}) \tag{5.3}$$

where $\mathbf{W}$ is another weight matrix.

Neal (1995, chapter 2) showed that neural networks with infinitely many hidden units, one hidden layer, and unknown weights correspond to Gaussian processes. More precisely, for any model of the form

$$f(\mathbf{x}) = \frac{1}{K}\mathbf{w}^{\mathsf{T}}\mathbf{h}(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K} w_i h_i(\mathbf{x}), \tag{5.4}$$

with fixed[1] features $[h_1(\mathbf{x}), \ldots, h_K(\mathbf{x})]^{\mathsf{T}} = \mathbf{h}(\mathbf{x})$ and i.i.d. $w$'s with zero mean and finite variance $\sigma^2$, the central limit theorem implies that as the number of features $K$ grows, any two function values $f(\mathbf{x})$ and $f(\mathbf{x}')$ have a joint distribution approaching a Gaussian:

$$\lim_{K\to\infty} p\left(\begin{bmatrix} f(\mathbf{x}) \\ f(\mathbf{x}') \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \frac{\sigma^2}{K}\begin{bmatrix} \sum_{i=1}^{K} h_i(\mathbf{x})h_i(\mathbf{x}) & \sum_{i=1}^{K} h_i(\mathbf{x})h_i(\mathbf{x}') \\ \sum_{i=1}^{K} h_i(\mathbf{x}')h_i(\mathbf{x}) & \sum_{i=1}^{K} h_i(\mathbf{x}')h_i(\mathbf{x}') \end{bmatrix}\right) \tag{5.5}$$

A joint Gaussian distribution between any set of function values is the definition of a Gaussian process.

The result is surprisingly general: it puts no constraints on the features (other than having uniformly bounded activation), nor does it require that the feature weights $\mathbf{w}$ be Gaussian distributed.

We can also work backwards to derive a one-layer MLP from any GP. Mercer's theorem implies that any positive-definite kernel function corresponds to an inner product of features: $k(\mathbf{x}, \mathbf{x}') = \mathbf{h}(\mathbf{x})^{\mathsf{T}}\mathbf{h}(\mathbf{x}')$. Thus in the one-hidden-layer case, the correspondence between MLPs and GPs is simple: the implicit features $\mathbf{h}(\mathbf{x})$ of the kernel correspond to hidden units of an MLP.

An MLP with a finite number of fixed hidden nodes gives rise to a GP if and only if the weights $\mathbf{w}$ are jointly Gaussian distributed.

[1] The above derivation gives the same result if the parameters of the hidden units are random, since their distribution on outputs always the same with probability one. However, to avoid confusion, we refer to layers with infinitely-many nodes as"fixed".

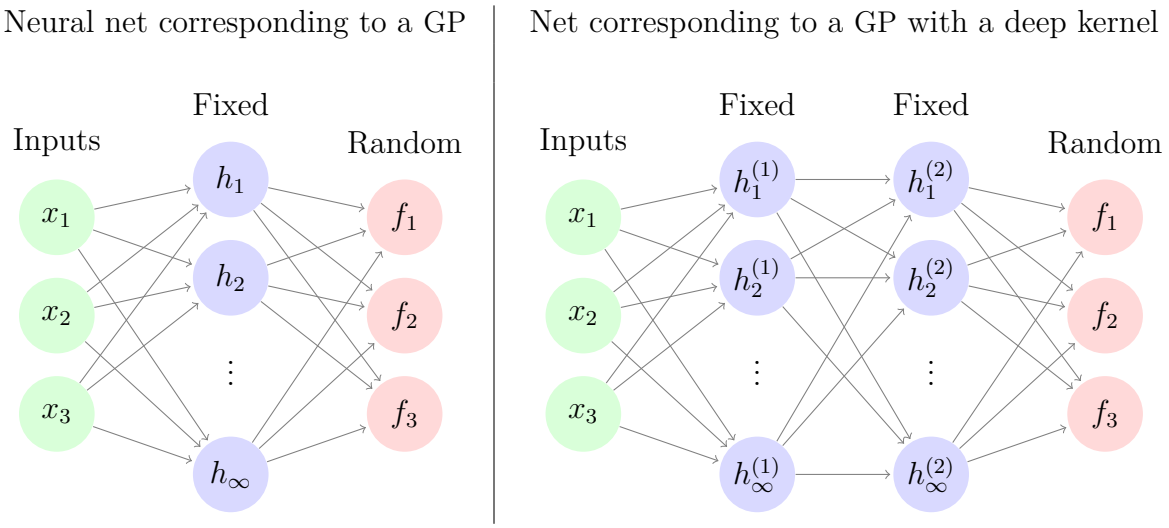



Figure 5.1 *Left:* GPs can be derived as a one-hidden-layer MLP with infinitely many fixed hidden units having unknown weights. *Right:* Multiple layers of fixed hidden units gives rise to a GP with a deep kernel, but not a deep GP.

### 5.1.3 Multiple hidden layers

Next, we examine infinitely-wide MLPs with multiple hidden layers. There are several ways to construct such networks, giving rise to different priors on functions.

In an MLP with multiple hidden layers, activation of the $\ell$th layer units are given by

$$\mathbf{h}^{(\ell)}(\mathbf{x}) = \sigma\left(\mathbf{b}^{(\ell)} + \mathbf{V}^{(\ell)}\mathbf{h}^{(\ell-1)}(\mathbf{x})\right) . \tag{5.6}$$

This architecture is shown on the right of figure 5.1. For example, if we extend the model given by equation (5.3) to have two layers of feature mappings, the resulting model becomes

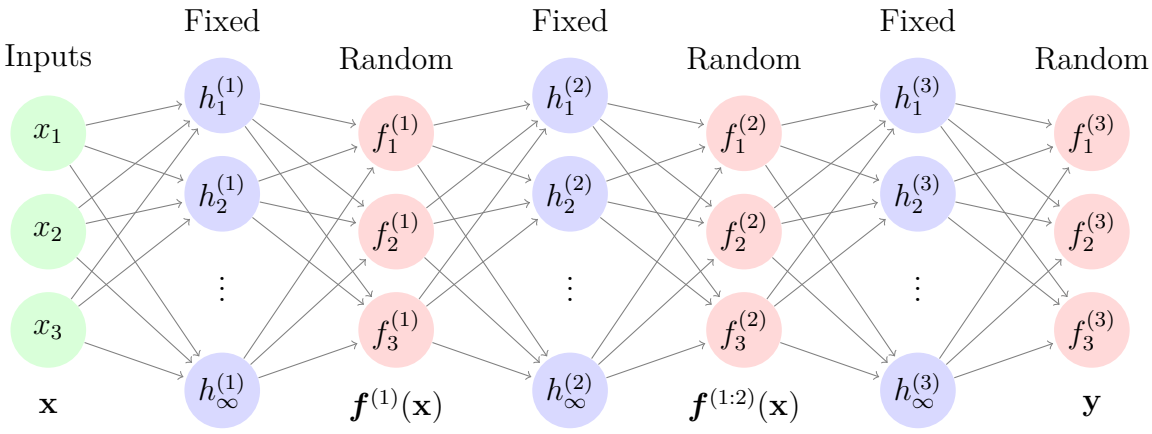
$$f(\mathbf{x}) = \frac{1}{K}\mathbf{w}^\mathsf{T}\mathbf{h}^{(2)}\left(\mathbf{h}^{(1)}(\mathbf{x})\right) . \tag{5.7}$$

If the features $\mathbf{h}^{(1)}(\mathbf{x})$ and $\mathbf{h}^{(2)}(\mathbf{x})$ are fixed with only the last-layer weights $\mathbf{w}$ unknown, this model corresponds to a shallow GP with a *deep kernel*, given by

$$k(\mathbf{x}, \mathbf{x}') = \left(\mathbf{h}^{(2)}(\mathbf{h}^{(1)}(\mathbf{x}))\right)^\mathsf{T} \mathbf{h}^{(2)}(\mathbf{h}^{(1)}(\mathbf{x}')) . \tag{5.8}$$

Deep kernels, explored in section 5.5, imply a fixed representation as opposed to a prior over representations. Thus, unless we richly parameterize these kernels, their

A neural net with fixed activation functions corresponding to a 3-layer deep GP




A net with nonpararametric activation functions corresponding to a 3-layer deep GP

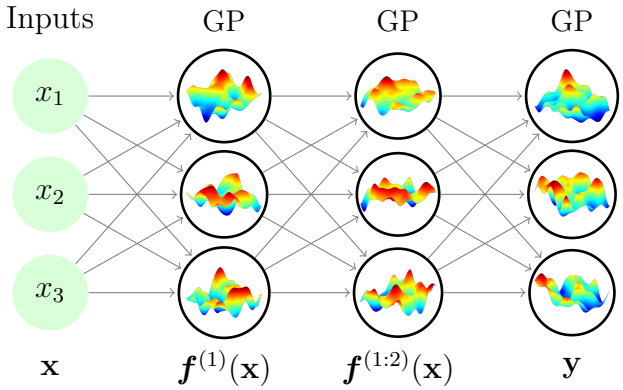



Figure 5.2 Two equivalent views of deep GPs as neural networks. *Top:* A neural network whose every second layer is a weighted sum of an infinite number of fixed hidden units, where the weights are initially unknown. *Bottom:* A neural network with a finite number of hidden units, each with a different unknown non-parametric activation function. The activation functions are visualized by draws from 2-dimensional GPs, although their input dimension will actually be the same as the output dimension of the previous layer.

capacity to learn an appropriate representation is limited in comparison to more flexible models such as deep neural networks or deep GPs.

### 5.1.4 Two network architectures equivalent to deep GPs

There are two equivalent neural network architectures that correspond to deep GPs: one with fixed nonlinearities, and another with GP-distributed nonlinearties.

To construct a neural network corresponding to a deep GP using only fixed nonlinearities, one can start with the infinitely-wide deep GP shown in figure 5.1(right), and introduce a finite set of nodes in between each infinitely-wide set of fixed basis functions. This architecture is shown in the top of figure 5.2. The $D^{(\ell)}$ outputs $\boldsymbol{f}^{(\ell)}(\mathbf{x})$ in between each fixed layer are weighted sums (with random weights) of the fixed hidden units of the layer below, and the next layer's hidden units depend only on these $D^{(\ell)}$ outputs.

This alternating-layer architecture has an interpretation as a series of linear information bottlenecks. To see this, we can simply substitute equation (5.3) into equation (5.6) to get

$$\mathbf{h}^{(\ell)}(\mathbf{x}) = \sigma\left(\mathbf{b}^{(\ell)} + \left[\mathbf{V}^{(\ell)}\mathbf{W}^{(\ell-1)}\right]\mathbf{h}^{(\ell-1)}(\mathbf{x})\right) \tag{5.9}$$

where $\mathbf{W}^{(\ell-1)}$ is the weight matrix connecting $\mathbf{h}^{(\ell-1)}$ to $\boldsymbol{f}^{(\ell-1)}$. Thus, ignoring the intermediate outputs $\boldsymbol{f}^{(\ell)}(\mathbf{x})$, a deep GP is an infinitely-wide, deep MLP with each pair of layers connected by random, rank-$D_\ell$ matrices $\mathbf{V}^{(\ell)}\mathbf{W}^{(\ell-1)}$.

The second, more direct way to construct a network architecture corresponding to a deep GP is to integrate out all $\mathbf{W}^{(\ell)}$, and view deep GPs as a neural network with a finite number of nonparametric, GP-distributed basis functions at each layer, in which $\boldsymbol{f}^{(1:\ell)}(\mathbf{x})$ represent the output of the hidden nodes at the $\ell^{th}$ layer. This second view lets us compare deep GP models to standard neural net architectures more directly. Figure 5.1(bottom) shows this architecture.

## 5.2 Characterizing deep Gaussian process priors

This section develops several theoretical results characterizing the behavior of deep GPs as a function of their depth. Specifically, we show that the size of the derivative of a one-dimensional deep GP becomes log-normal distributed as the network becomes deeper. We will also show that the Jacobian of a multivariate deep GP is a product of independent Gaussian matrices with independent entries. These results will allow us to identify a pathology that emerges in very deep networks in section 5.3.

### 5.2.1 One-dimensional asymptotics

In this section, we derive the limiting distribution of the derivative of an arbitrarily deep, one-dimensional GP with a squared-exp kernel:

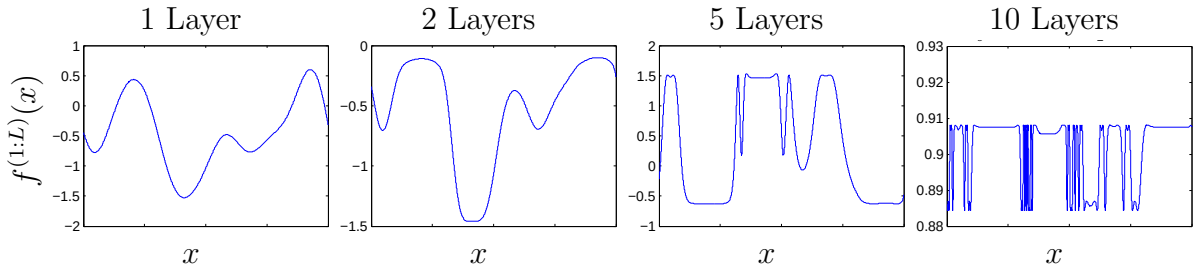



Figure 5.3 A one-dimensional draw from a deep GP prior shown at different depths. The $x$-axis is the same for all plots. After a few layers, the functions begin to be either nearly flat, or highly varying, everywhere. This is a consequence of the distribution on derivatives becoming heavy-tailed. As well, the function values at each layer tend to cluster around the same few values as the depth increases. This happens because once the function values in a particular region are mapped to the same value in an intermediate layer, there is no way for them to be mapped to different values in later layers.

$$\mathrm{SE}(x, x') = \sigma^2 \exp\left(\frac{-(x-x')^2}{2w^2}\right) . \tag{5.10}$$

The parameter $\sigma^2$ controls the variance of functions drawn from the prior, and the lengthscale parameter $w$ controls the smoothness. The derivative of a GP with a squared-exp kernel is point-wise distributed as $\mathcal{N}(0, {}^{\sigma^2}/_{w^2})$. Intuitively, a draw from a GP is likely to have large derivatives if the kernel has high variance and small lengthscales.

By the chain rule, the derivative of a one-dimensional deep GP is simply a product of the derivatives of each layer, which are drawn independently by construction. The distribution of the absolute value of this derivative is a product of half-normals, each with mean $\sqrt{{}^{2\sigma^2}/_{\pi w^2}}$. If we choose kernel parameters so that ${}^{\sigma^2}/_{w^2} = {}^{\pi}/_{2}$, then the expected magnitude of the derivative remains constant regardless of the depth.

The distribution of the log of the magnitude of the derivatives has finite moments:

$$\begin{aligned} m_{\log} &= \mathbb{E}\left[\log\left|\frac{\partial f(x)}{\partial x}\right|\right] = 2\log\left(\frac{\sigma}{w}\right) - \log 2 - \gamma \\ v_{\log} &= \mathbb{V}\left[\log\left|\frac{\partial f(x)}{\partial x}\right|\right] = \frac{\pi^2}{4} + \frac{\log^2 2}{2} - \gamma^2 - \gamma\log 4 + 2\log\left(\frac{\sigma}{w}\right)\left[\gamma + \log 2 - \log\left(\frac{\sigma}{w}\right)\right] \end{aligned} \tag{5.11}$$

where $\gamma \approxeq 0.5772$ is Euler's constant. Since the second moment is finite, by the central limit theorem, the limiting distribution of the size of the gradient approaches log-normal

as L grows:

$$\log\left|\frac{\partial f^{(1:L)}(x)}{\partial x}\right| = \log\prod_{\ell=1}^{L}\left|\frac{\partial f^{(\ell)}(x)}{\partial x}\right| = \sum_{\ell=1}^{L}\log\left|\frac{\partial f^{(\ell)}(x)}{\partial x}\right| \overset{L\to\infty}{\sim} \mathcal{N}\Big(Lm_{\log}, L^2 v_{\log}\Big) \tag{5.12}$$

Even if the expected magnitude of the derivative remains constant, the variance of the log-normal distribution grows without bound as the depth increases.

Because the log-normal distribution is heavy-tailed and its domain is bounded below by zero, the derivative will become very small almost everywhere, with rare but very large jumps. Figure 5.3 shows this behavior in a draw from a 1D deep GP prior. This figure also shows that once the derivative in one region of the input space becomes very large or very small, it is likely to remain that way in subsequent layers.

### 5.2.2 Distribution of the Jacobian

fNext, we characterize the distribution on Jacobians of multivariate functions drawn from deep GP priors, finding them to be products of independent Gaussian matrices with independent entries.

**Lemma 5.2.1.** *The partial derivatives of a function mapping $\mathbb{R}^D \to \mathbb{R}$ drawn from a GP prior with a product kernel are independently Gaussian distributed.*

*Proof.* Because differentiation is a linear operator, the derivatives of a function drawn from a GP prior are also jointly Gaussian distributed. The covariance between partial derivatives with respect to input dimensions $d_1$ and $d_2$ of vector $\mathbf{x}$ are given by Solak et al. (2003):

$$\operatorname{cov}\left(\frac{\partial f(\mathbf{x})}{\partial x_{d_1}}, \frac{\partial f(\mathbf{x})}{\partial x_{d_2}}\right) = \left.\frac{\partial^2 k(\mathbf{x},\mathbf{x}')}{\partial x_{d_1}\partial x'_{d_2}}\right|_{\mathbf{x}=\mathbf{x}'} \tag{5.13}$$

If our kernel is a product over individual dimensions $k(\mathbf{x},\mathbf{x}') = \prod_d^D k_d(x_d, x'_d)$, then the off-diagonal entries are zero, implying that all elements are independent. □

For example, in the case of the multivariate squared-exp kernel, the covariance be-

tween derivatives has the form:

$$f(\mathbf{x}) \sim \mathrm{GP}\left(0, \sigma^2 \prod_{d=1}^{D} \exp\left(-\frac{1}{2}\frac{(x_d - x'_d)^2}{w_d^2}\right)\right)$$
$$\implies \mathrm{cov}\left(\frac{\partial f(\mathbf{x})}{\partial x_{d_1}}, \frac{\partial f(\mathbf{x})}{\partial x_{d_2}}\right) = \begin{cases} \frac{\sigma^2}{w_{d_1}^2} & \text{if } d_1 = d_2 \\ 0 & \text{if } d_1 \neq d_2 \end{cases} \tag{5.14}$$

**Lemma 5.2.2.** *The Jacobian of a set of $D$ functions $\mathbb{R}^D \to \mathbb{R}$ drawn from independent GP priors, each having product kernel is a $D \times D$ matrix of independent Gaussian R.V.'s*

*Proof.* The Jacobian of the vector-valued function $\boldsymbol{f}(\mathbf{x})$ is a matrix $J$ with elements $J_{ij}(\mathbf{x}) = \frac{\partial f_i(\mathbf{x})}{\partial x_j}$. Because the GPs on each output dimension $f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_D(\mathbf{x})$ are independent by construction, it follows that each row of $J$ is independent. Lemma 5.2.1 shows that the elements of each row are independent Gaussian. Thus all entries in the Jacobian of a GP-distributed transform are independent Gaussian R.V.'s. □

**Theorem 5.2.3.** *The Jacobian of a deep GP with a product kernel is a product of independent Gaussian matrices, with each entry in each matrix being drawn independently.*

*Proof.* When composing $L$ different functions, we denote the *immediate* Jacobian of the function mapping from layer $\ell - 1$ to layer $\ell$ as $J^{(\ell)}(\mathbf{x})$, and the Jacobian of the entire composition of $L$ functions by $J^{(1:L)}(\mathbf{x})$. By the multivariate chain rule, the Jacobian of a composition of functions is simply the product of the immediate Jacobian matrices of each function. Thus the Jacobian of the composed (deep) function $\boldsymbol{f}^{(L)}(\boldsymbol{f}^{(L-1)}(\ldots \boldsymbol{f}^{(3)}(\boldsymbol{f}^{(2)}(\boldsymbol{f}^{(1)}(\mathbf{x})))\ldots))$ is

$$J^{(1:L)}(\mathbf{x}) = J^{(L)} J^{(L-1)} \ldots J^{(3)} J^{(2)} J^{(1)}. \tag{5.15}$$

By lemma 5.2.2, each $J_{i,j}^{(\ell)} \overset{\text{ind}}{\sim} \mathcal{N}$, so the complete Jacobian is a product of independent Gaussian matrices, with each entry of each matrix drawn independently. □

This result allows us to analyze the representational properties of a deep Gaussian process by examining the properties of products of independent Gaussian matrices.

## 5.3 Formalizing a pathology

A common use of deep neural networks is building useful representations of data manifolds. What properties make a representation useful? Rifai et al. (2011a) argue that

good representations of data manifolds are invariant in directions orthogonal to the data manifold. They also argue that, conversely, a good representation must also change in directions tangent to the data manifold, in order to preserve relevant information. Figure 5.4 visualizes such a representation.

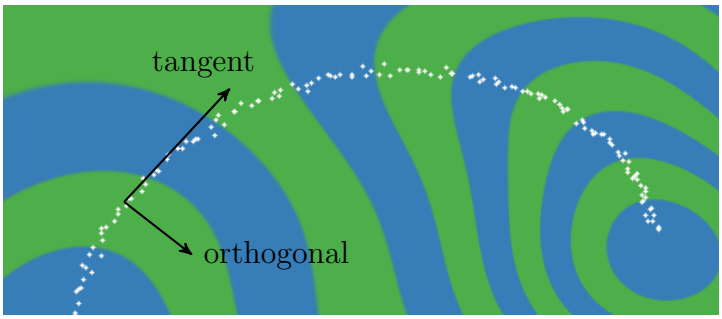



Figure 5.4 Representing a 1-D data manifold. Colors are a function of the computed representation of the input space. The representation (blue & green) varies in directions tangent to the data manifold (white), preserving information for later layers. The representation changes little in directions orthogonal to the manifold, making it robust to noise in those directions.

As in Rifai et al. (2011b), we characterize the representational properties of a function by the singular value spectrum of the Jacobian. The number of relatively large singular values of the Jacobian roughly correspond to the number of directions in data-space in which the representation varies. Figure 5.5 shows the singular value spectrum for 5-dimensional deep GPs of different depths [2]. As the net gets deeper, the largest singular value dominates, implying there is usually only one effective degree of freedom in the representations being computed.

Figure 5.6 demonstrates a related pathology that arises when composing functions to produce a deep density model. The density in the observed space eventually becomes locally concentrated onto one-dimensional manifolds, or *filaments*. This again suggests that, when the width of the network is relatively small, deep compositions of independent functions are unsuitable for modeling manifolds whose underlying dimensionality is greater than one.

To visualize this pathology in another way, figure 5.7 illustrates a color-coding of the representation computed by a deep GP, evaluated at each point in the input space. After 10 layers, we can see that locally, there is usually only one direction that one can move in $\mathbf{x}$-space in order to change the value of the computed representation, or to cross

[2] Rifai et al. (2011b) analyzed the Jacobian at location of the training points, but because the priors we are examining are stationary, the distribution of the Jacobian is identical everywhere.

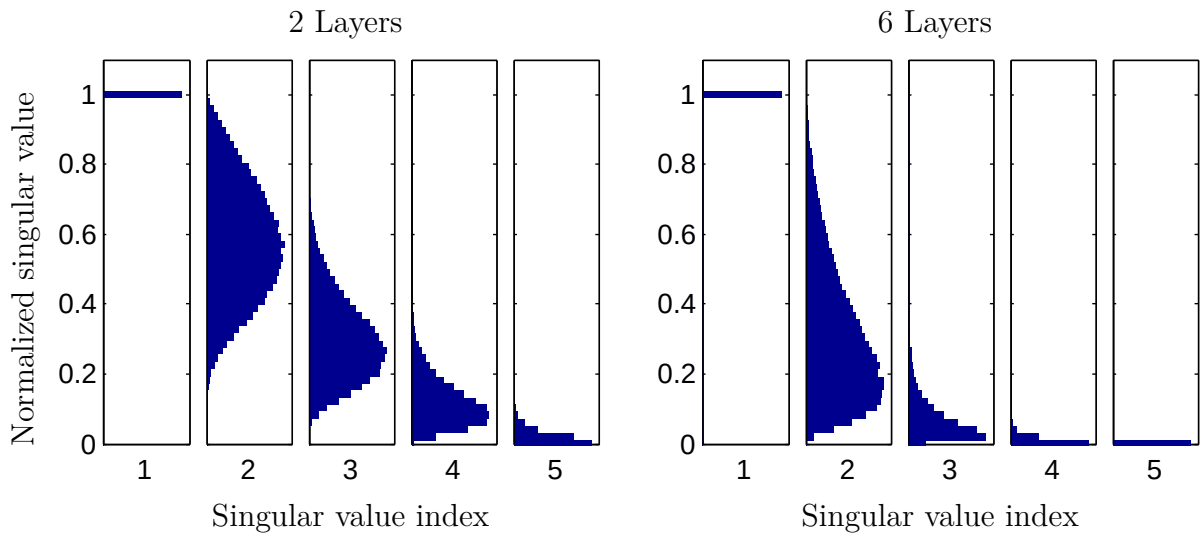



Figure 5.5 The distribution of normalized singular values of the Jacobian of a function drawn from a 5-dimensional deep GP prior 25 layers deep (*Left*) and 50 layers deep (*Right*). As nets get deeper, the largest singular value tends to become much larger than the others. This implies that with high probability, these functions vary little in all directions but one, making them insuitable for computing representations of manifolds of more than one dimension.

a decision boundary. This means that such representations are likely to be unsuitable for decision tasks that depend on more than one property of the input.

To what extent are these pathologies present in the types of neural networks commonly used in practice? In simulations, we found that for deep functions with a fixed hidden dimension $D$, the singular value spectrum remained relatively flat for hundreds of layers as long as $D > 100$. Thus, these pathologies are unlikely to severely effect the relatively shallow, wide networks most commonly used in practice.

## 5.4 Fixing the pathology

As suggested by Neal (1995, chapter 2), we can fix the pathologies exhibited in figures figure 5.6 and 5.7 by simply making each layer depend not only on the output of the previous layer, but also on the original input $\mathbf{x}$. We refer to these models as *input-connected* networks, and denote deep functions having this architecture with the subscript $C$, as in $f_C(\mathbf{x})$. Formally, this functional dependence can be written as

$$\boldsymbol{f}_C^{(1:L)}(\mathbf{x}) = \boldsymbol{f}^{(L)}\left(\boldsymbol{f}_C^{(1:L-1)}(\mathbf{x}), \mathbf{x}\right), \quad \forall L \tag{5.16}$$

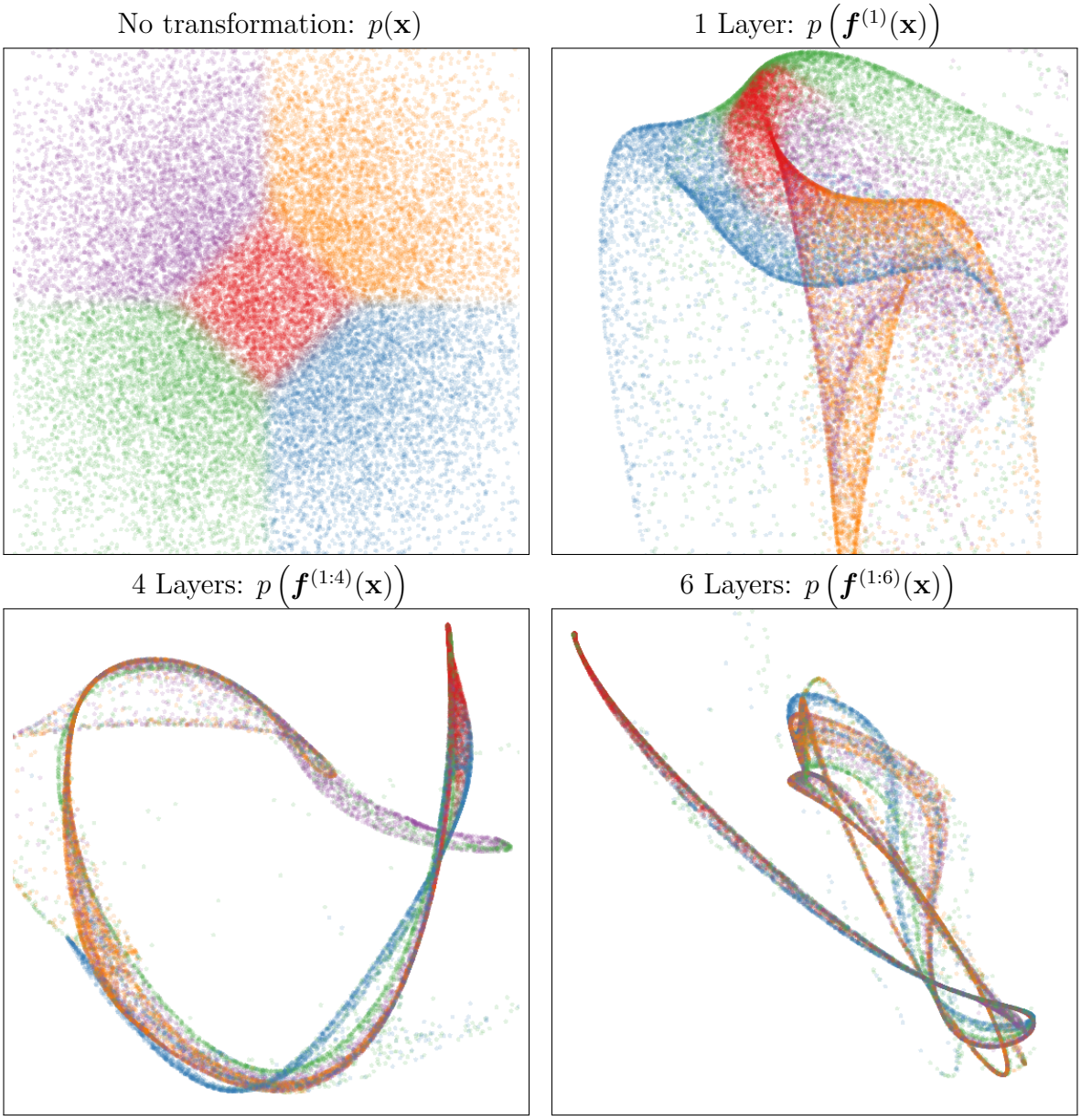



Figure 5.6 Points warped by a function drawn from a deep GP prior. *Top left:* Points drawn from a 2-dimensional Gaussian distribution, color-coded by their location. *Subsequent panels:* Those same points, successively warped by compositions of functions drawn from a GP prior. As the number of layers increases, the density concentrates along one-dimensional filaments. Warpings using random finite neural networks exhibit the same pathology, but also tend to concentrate density into 0-dimensional manifolds (points) due to saturation of the hidden units.

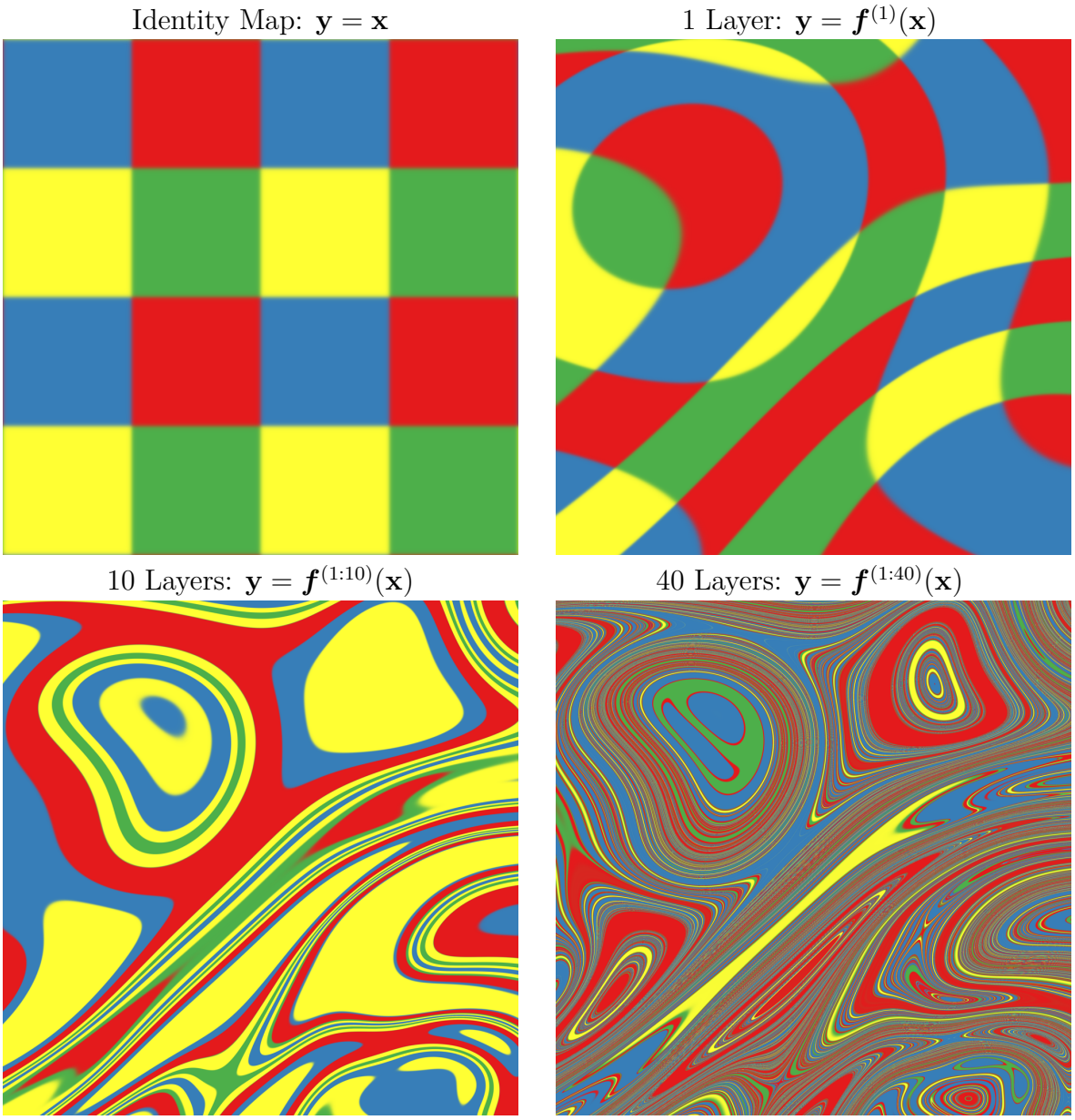



Figure 5.7 A vizualization of the feature map implied by a function $\boldsymbol{f}$ drawn from a deep GP. Colors are a function of the 2D representation $\mathbf{y} = \boldsymbol{f}(\mathbf{x})$ that each point is mapped to. The number of directions in which the color changes rapidly corresponds to the number of large singular values in the Jacobian. Just as the densities in figure 5.6 became locally one-dimensional, there is usually only one direction that one can move $\mathbf{x}$ in locally to change $\mathbf{y}$. This means that $\boldsymbol{f}$ is unlikely to be a suitable representation for decision tasks that depend on more than one aspect of $\mathbf{x}$. Also note that the overall shape of the mapping remains the same as the number of layers increase. For example, the roughly circular shape remains in the top-left corner even after 40 independent warpings.

Figure 5.8 shows a graphical representation of the two connectivity architectures.

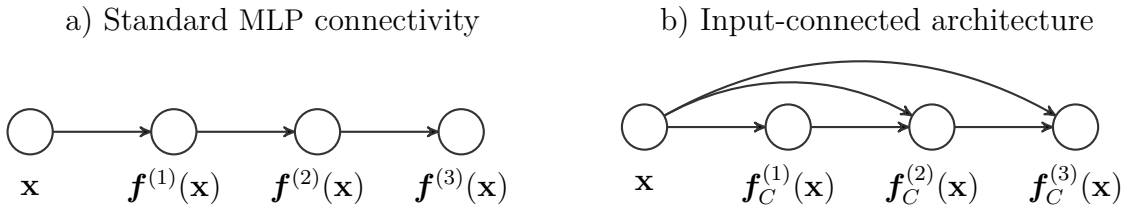



Figure 5.8 Two different architectures for deep neural networks. *Left:* The standard architecture connects each layer's outputs to the next layer's inputs. *Right:* The input-connected architecture also connects the original input $\mathbf{x}$ to each layer.

Similar connections between non-adjacent layers can also be found the primate visual cortex (Maunsell and van Essen, 1983). Visualizations of the resulting prior on functions are shown in figures 5.9, 5.10 and 5.12.

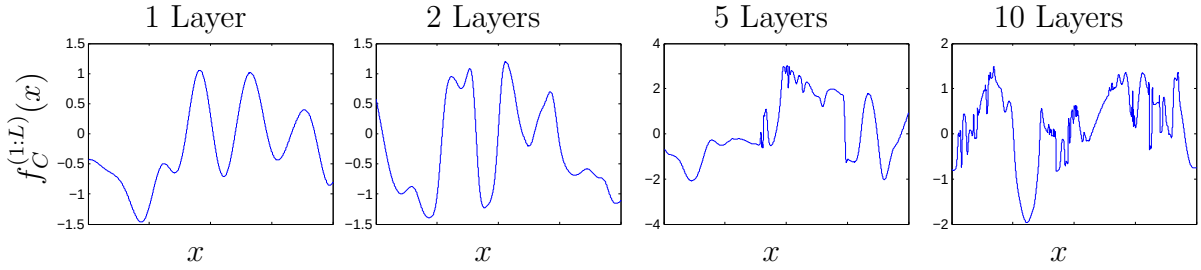



Figure 5.9 A draw from a 1D deep GP prior with each layer connected to the input. The $x$-axis is the same for all plots. Even after many layers, the functions remain relatively smooth in some regions, while varying rapidly in other regions. Compare to standard-connectivity deep GP draws shown in figure 5.3.

The Jacobian of an input-connected deep function is defined by the recurrence

$$J_C^{(1:L)} = J^{(L)} \begin{bmatrix} J_C^{(1:L-1)} \\ I_D \end{bmatrix}. \tag{5.17}$$

where $I_D$ is a $D$-dimensional identity matrix. In the case of a deep GP, the input-connected Jacobian is still a product of independent Gaussian matrices. Figure 5.11 shows that with this architecture, even 50-layer deep GPs have well-behaved singular value spectra.

The pathology examined in this section is an example of the sort of analysis made possible by a well-defined prior on functions. The figures and analysis done in this

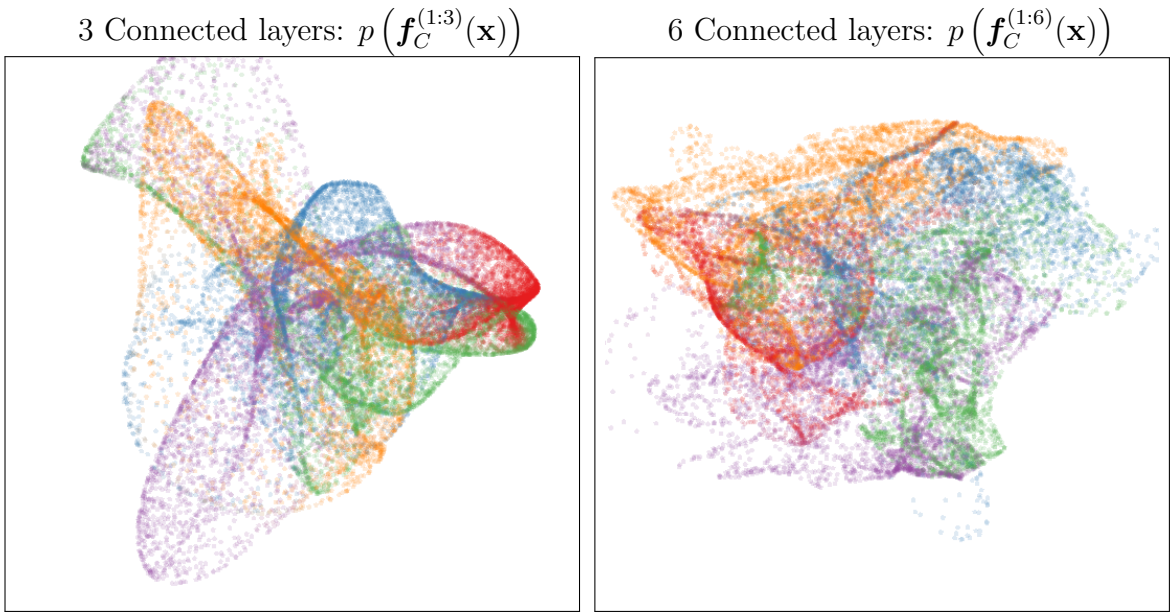



Figure 5.10 Points warped by a draw from a deep GP with each layer connected to the input $\mathbf{x}$. As depth increases, the density becomes more complex without concentrating only along one-dimensional filaments.

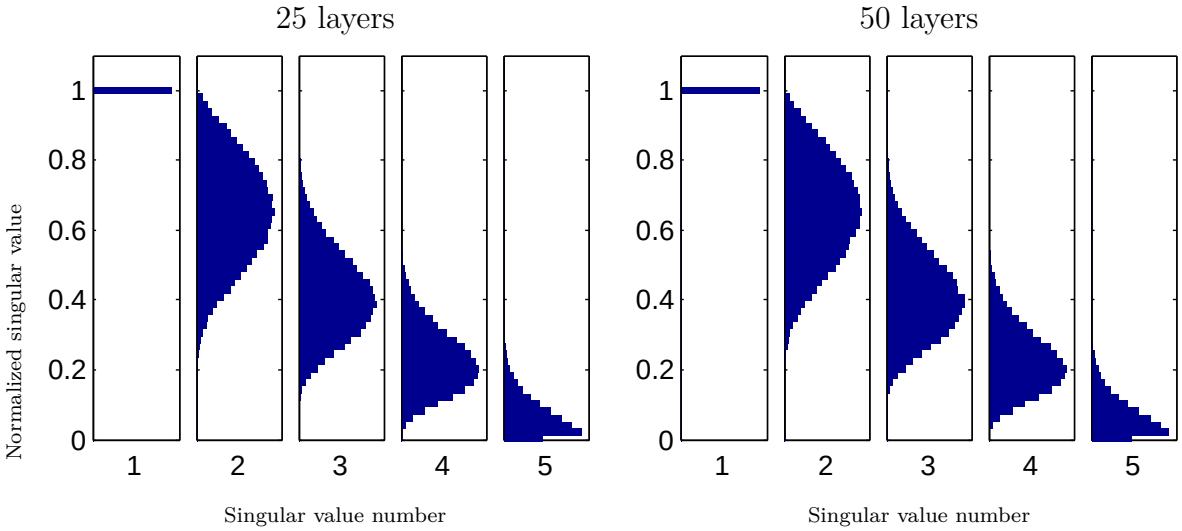



Figure 5.11 The distribution of singular values drawn from 5-dimensional input-connected deep GP priors, 25 layers deep (*Left*) and 50 layers deep (*Right*). Compared to the standard architecure, the singular values are more likely to remain the same size as one another, meaning that the model outputs are more often sensitive to several directions of variation in the input.

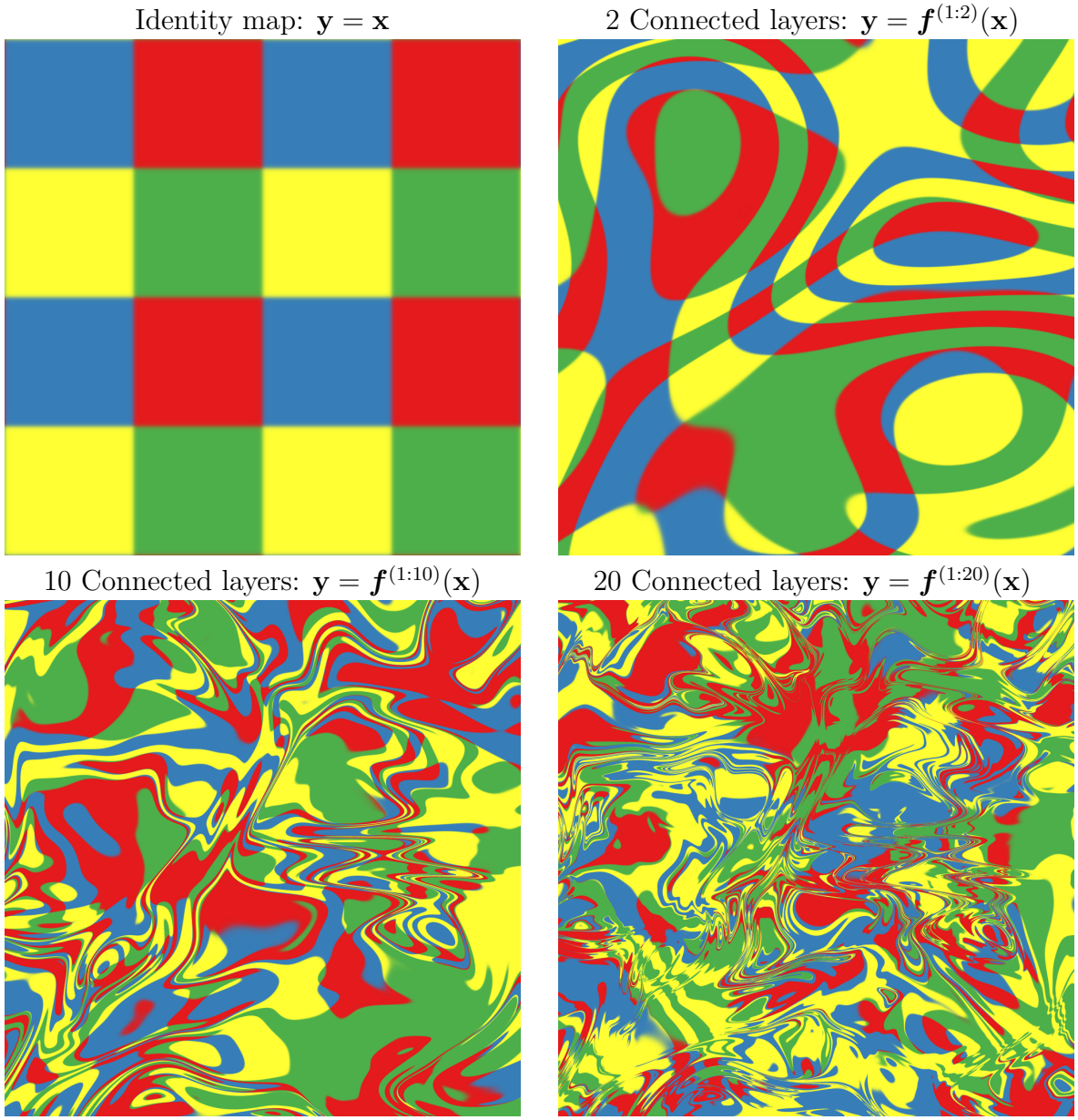



Figure 5.12 The feature map implied by a function $\boldsymbol{f}$ drawn from a deep GP prior with each layer connected to the input $\mathbf{x}$, visualized at various depths. Compare to the map shown in figure 5.7. In the mapping shown here there are sometimes two directions that one can move locally in $\mathbf{x}$ to in order to change the value of $\boldsymbol{f}(\mathbf{x})$. This means that the input-connected prior puts mass on a greater variety of types of representations, some of which depend on all aspects of the input.

section could be done using Bayesian neural networks with finite numbers of nodes, but would be more difficult. In particular, care would need to be taken to ensure that the networks do not produce degenerate mappings due to saturation of the hidden units.

## 5.5 Deep kernels

Bengio et al. (2006) showed that kernel machines have limited generalization ability when they use "local" kernels such as the squared-exp. However, as we saw in chapters 2, 3 and 6, structured, non-local kernels can allow extrapolation. Another way to build non-local kernels is by composing fixed feature maps, creating *deep kernels*. This section builds on the work of Cho and Saul (2009), who derived several kinds of deep kernels by applying multiple layers of feature mappings.

To return to an example given in section 2.7, periodic kernels can be viewed as a 2-layer-deep kernel, in which the first layer maps $x \to [\sin(x), \cos(x)]$, and the second layer maps through basis functions corresponding to the SE kernel.

In principle, we can compose the implicit feature maps of any two kernels $k_a$ and $k_b$ to get a new kernel, which we'll denote by $(k_b \circ k_a)$:

$$k_a(\mathbf{x}, \mathbf{x}') = \mathbf{h}_a(\mathbf{x})^\mathsf{T} \mathbf{h}_a(\mathbf{x}') \tag{5.18}$$

$$k_b(\mathbf{x}, \mathbf{x}') = \mathbf{h}_b(\mathbf{x})^\mathsf{T} \mathbf{h}_b(\mathbf{x}') \tag{5.19}$$

$$(k_b \circ k_a)(\mathbf{x}, \mathbf{x}') = k_b(\mathbf{h}_a(\mathbf{x}), \mathbf{h}_a(\mathbf{x}')) = [\mathbf{h}_b(\mathbf{h}_a(\mathbf{x}))]^\mathsf{T} \mathbf{h}_b(\mathbf{h}_a(\mathbf{x}')) \tag{5.20}$$

However, this composition might not always have a closed form if the number of hidden features of either kernel is infinite.

Fortunately, composing the squared-exp (SE) kernel with the implicit mapping given by any other kernel has a simple closed form. If $k(\mathbf{x}, \mathbf{x}') = \mathbf{h}(\mathbf{x})^\mathsf{T} \mathbf{h}(\mathbf{x}')$, then

$$(\text{SE} \circ k)(\mathbf{x}, \mathbf{x}') = k_{SE}\Big(\mathbf{h}(\mathbf{x}), \mathbf{h}(\mathbf{x}')\Big) \tag{5.21}$$

$$= \exp\left(-\frac{1}{2}||\mathbf{h}(\mathbf{x}) - \mathbf{h}(\mathbf{x}')||_2^2\right) \tag{5.22}$$

$$= \exp\left(-\frac{1}{2}\Big[\mathbf{h}(\mathbf{x})^\mathsf{T}\mathbf{h}(\mathbf{x}) - 2\mathbf{h}(\mathbf{x})^\mathsf{T}\mathbf{h}(\mathbf{x}') + \mathbf{h}(\mathbf{x}')^\mathsf{T}\mathbf{h}(\mathbf{x}')\Big]\right) \tag{5.23}$$

$$= \exp\left(-\frac{1}{2}\Big[k(\mathbf{x}, \mathbf{x}) - 2k(\mathbf{x}, \mathbf{x}') + k(\mathbf{x}', \mathbf{x}')\Big]\right) . \tag{5.24}$$

This formula lets expresses the composed kernel $(\text{SE} \circ k)$ exactly in terms of evaluations of the original kernel $k$.

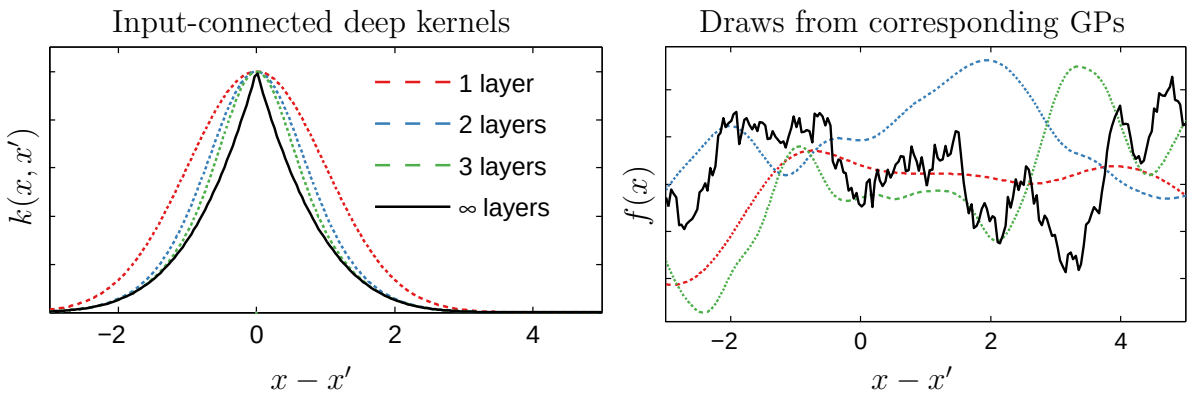



Figure 5.13 *Left:* Input-connected deep kernels of different depths. By connecting the input $\mathbf{x}$ to each layer, the kernel can still depend on its input even after arbitrarily many layers of composition. *Right:* Draws from GPs with deep input-connected kernels.

### 5.5.1 Infinitely deep kernels

What happens when we repeatedly compose feature maps many times, starting with the squared-exp kernel? If the output variance of the SE is normalized to $k(\mathbf{x}, \mathbf{x}) = 1$, then the infinite limit of composition with SE converges to $(\text{SE} \circ \text{SE} \circ \text{SE} \circ \ldots \circ \text{SE})(\mathbf{x}, \mathbf{x}') = 1$ for all pairs of inputs. A constant covariance corresponds to a prior on constant functions $f(\mathbf{x}) = c$.

As above, we can overcome this degeneracy by connecting the input $\mathbf{x}$ to each layer. To do so, we concatenate the composed feature vector at each layer, $\mathbf{h}^{(1:\ell)}(\mathbf{x})$, with the input vector $\mathbf{x}$, to produce an input-connected deep kernel $k_C^{(1:L)}$, defined by:

$$\begin{aligned} k_C^{(1:\ell+1)}(\mathbf{x}, \mathbf{x}') &= \exp\left(-\frac{1}{2}\left|\left|\begin{bmatrix}\mathbf{h}^{(1:\ell)}(\mathbf{x}) \\ \mathbf{x}\end{bmatrix} - \begin{bmatrix}\mathbf{h}^{(1:\ell)}(\mathbf{x}') \\ \mathbf{x}'\end{bmatrix}\right|\right|_2^2\right) \\ &= \exp\left(-\frac{1}{2}\left[k_C^{(1:\ell)}(\mathbf{x}, \mathbf{x}) - 2k_C^{(1:\ell)}(\mathbf{x}, \mathbf{x}') + k_C^{(1:\ell)}(\mathbf{x}', \mathbf{x}') - ||\mathbf{x} - \mathbf{x}'||_2^2\right]\right) \end{aligned} \tag{5.25}$$

Starting with the squared-exp kernel, this repeated mapping satisfies

$$k_C^{(1:\infty)}(\mathbf{x}, \mathbf{x}') - \log\left(k_C^{(1:\infty)}(\mathbf{x}, \mathbf{x}')\right) = 1 + \frac{1}{2}||\mathbf{x} - \mathbf{x}'||_2^2 . \tag{5.26}$$

The solution to this recurrence has no closed form, but for one input dimension, has a similar shape to the Ornstein-Uhlenbeck covariance $\text{OU}(x, x') = \exp(-|x - x'|)$ but with lighter tails. Samples from a GP prior with this kernel are not differentiable, and are locally fractal. Figure 5.13 shows this kernel at different depths, as well as samples from

the resulting GP priors.

We can also consider two related connectivity architectures: one in which each layer is connected to the output layer, and another in which every layer is connected to all subsequent layers. It is easy to show that in the limit of infinite depth of composing SE kernels, both these architectures converge to $k(\mathbf{x}, \mathbf{x}') = \delta(\mathbf{x}, \mathbf{x}')$, the white noise kernel.

### 5.5.2 When are deep kernels useful models?

Kernels correspond to fixed feature maps, and so kernel learning is an example of implicit representation learning. As we saw in chapters 2 and 3, kernels can capture rich structure and can enable many types of generalization. We believe that the relatively uninteresting properties of the deep kernels derived in this section simply reflect the fact that an arbitrary computation, even if it is "deep", is not likely to give rise to a useful representation unless combined with learning. To put it another way, any fixed representation is unlikely to be useful unless it has been chosen specifically for the problem at hand.

## 5.6 Related work

#### Deep Gaussian processs

Neal (1995, chapter 2) explored properties of arbitrarily deep Bayesian neural networks, including those that would give rise to deep GPs. He noted that infinitely deep random neural networks without extra connections to the input would be equivalent to a Markov chain, and therefore would lead to degenerate priors on functions. He also suggested connecting the input to each layer in order to fix this problem. Much of the analysis in this chapter can be seen as a more detailed investigation, and vindication, of these claims.

The first instance of deep GPs being used in practice was (Lawrence and Moore, 2007), who presented a model called "hierarchical GP-LVMs", in which time was mapped through a composition of multiple GPs to produce observations.

The term "deep Gaussian processes" was first used by Damianou and Lawrence (2013), who developed a variational inference method, analyzed the effect of automatic relevance determination, and showed that deep GPS could learn with relatively little data. They used the term "deep GP" to refer both to supervised models (compositions of GPs) and to unsupervised models (compositions of GP-LVMs). This conflation may

be reasonable, since the activations of the hidden layers are themselves latent variables, even in supervised settings: Depending on kernel parameters, each latent variable may or may not depend on the layer below.

In general, supervised models can also be latent-variable models. For example, Wang and Neal (2012) investigated single-layer GP regression models that had additional latent inputs.

### Nonparametric neural networks

Adams et al. (2010) proposed a prior on arbitrarily deep Bayesian networks with an unknown and unbounded number of parametric hidden units in each layer. Their architecture has connections only between adjacent layers, and so may be expected to have similar pathologies to those of deep GPs as the number of layers increases.

Wilson et al. (2012) introduced Gaussian process regression networks, which are defined as a matrix product of draws from GPs priors, rather than a composition. These networks have the form:

$$\mathbf{y}(\mathbf{x}) = \mathbf{W}(\mathbf{x})\boldsymbol{f}(\mathbf{x}) \qquad \text{with each } f_d, W_{d,j} \overset{\text{iid}}{\sim} \mathcal{GP}(\mathbf{0}, \text{SE} + \text{WN})\,. \tag{5.27}$$

We can easily define a "deep" Gaussian process regression network:

$$\mathbf{y}(\mathbf{x}) = \mathbf{W}^{(3)}(\mathbf{x})\mathbf{W}^{(2)}(\mathbf{x})\mathbf{W}^{(1)}(\mathbf{x})\boldsymbol{f}(\mathbf{x}) \tag{5.28}$$

which repeatedly adds and multiplies functions drawn from GPs, in contrast to deep GPs, which repeatedly compose functions. This prior on functions has a similar form to the Jacobian of a deep GP (equation (5.15)), and so might be amenable to a similar analysis to that of section 5.2.

### Information-preserving architectures

Deep density networks (Rippel and Adams, 2013) are constructed through a series of parametric warpings of fixed dimension, with penalty terms encouraging the preservation of information about lower layers. This is another promising approach to fixing the pathology discussed in section 5.3.

### Recurrent networks

Bengio et al. (1994) and Pascanu et al. (2012) analyze a related problem with gradient-based learning in recurrent networks, the "exploding-gradients" problem. They note that in recurrent neural networks, the size of the training gradient can grow or shrink exponentially as it is back-propagated, making gradient-based training difficult.

Hochreiter and Schmidhuber (1997) addressed the exploding-gradients problem by including hidden units designed to have stable gradients.

### Deep kernels

The first systematic examination of deep kernels was done by Cho and Saul (2009), who derived closed-form composition rules for SE, polynomial, and arc-cosine kernels, and showed that deep arc-cosine kernels performed competitively in machine-vision applications when used in a SVM.

Hermans and Schrauwen (2012) constructed deep kernels in a time-series setting, constructing kernels corresponding to infinite-width *recurrent* neural networks. They also proposed concatenating the implicit feature vectors from previous time-steps with the current inputs, resulting in an architecture analogous to the input-connected architecture proposed by Neal (1995, chapter 2).

### Analyses of deep learning

Montavon et al. (2010) performed a layer-wise analysis of deep networks, and noted that the performance of MLPs degrades as the number of layers with random weights increases.

The experiments of Saxe et al. (2011) suggested that most of the performance of convolutional neural networks could be attributed to the architecture alone. Later, Saxe et al. (2013) looked at the dynamics of gradient-based training methods in deep *linear* networks as a tractable approximation to standard deep (nonlinear) neural networks.

### Source code

Source code to produce all figures is available at `http://www.github.com/duvenaud/deep-limits`. This code is also capable of producing visualizations of mappings such as figures 5.7 and 5.12 using neural nets instead of GPs at each layer.

## 5.7 Conclusions

This chapter demonstrated that well-defined priors allow explicit examination of the assumptions being made about functions being modeled by different neural network architectures. As an example of the sort of analysis made possible this way, we attempted to gain insight into the properties of deep neural networks by characterizing the sorts of functions likely to be obtained under different choices of priors on compositions of functions.

First, we identified deep Gaussian processes as an easy-to-analyze model corresponding to multi-layer preceptrons with nonparametric activation functions. We then showed that representations based on repeated composition of independent functions exhibit a pathology where the representations becomes invariant to all but one direction of variation. Finally, we showed that this problem could be alleviated by connecting the input to each layer. We also examined properties of deep kernels, corresponding to arbitrarily many compositions of fixed features.

Much recent work on deep networks has focused on weight initialization (Martens, 2010), regularization (Lee et al., 2007) and network architecture (Gens and Domingos, 2013). If we can identify priors that give our models desirable properties, these in turn may suggest regularization, initialization, and architecture choices that also provide such properties.

Existing neural network practice also requires expensive tuning of model hyperparameters such as the number of layers, the size of each layer, and regularization penalties by cross-validation. One advantage of deep GPs is that the approximate marginal likelihood allows a principled method for automatically determining such model choices.

# Chapter 6

# Additive Gaussian Processes

Chapter 3 showed how to learn the structure of a kernel by building it up piece-by-piece. This chapter presents an alternative approach: starting with many different types of structure in a kernel, adjust kernel parameters to discard whatever structure is *not* present in the current dataset. The advantage of this approach is that we do not need to run an expensive discrete-and-continuous search in order to build a structured model, and implementation is simpler.

This model, which we call *additive Gaussian processes*, is a sum of functions of all possible combinations of input variables. This model can be specified by a weighted sum of all possible products of one-dimensional kernels.

There are $2^D$ combinations of $D$ objects, so naïve computation of this kernel is intractable. Furthermore, if each term has different kernel parameters, fitting or integrating over so many parameters poses severe difficulty. To address these problems, we introduce a restricted parameterization of the kernel which allows efficient evaluation of all interaction terms, but a different weighting of each order of interaction. Empirically, this model has good predictive power in regression tasks, and its parameters are relatively interpretable. This model also has an interpretation as an approximation to *dropout*, a recently-introduced regularization method for neural networks.

The work in this chapter was done in collaboration with Hannes Nickisch and Carl Rasmussen, who derived and coded up the initial model. My role in the project was to examine the properties of the resulting model, clarify the connections to existing methods, to create all figures and run all experiments. That work was published in Duvenaud et al. (2011). The connection to dropout regularization in section 6.4 is an independent contribution.

## 6.1 Different types of multivariate additive structure

Chapter 2 showed how additive structure in a GP prior enabled long-range extrapolation in multivariate regression problems. In general, models of the form

$$f(\mathbf{x}) = g\Big(f(x_1) + f(x_2) + \cdots + f(x_D)\Big) \tag{6.1}$$

are widely used in machine learning and statistics, partly for this reason, and partly because they are relatively easy to fit and interpret. Examples include logistic regression, linear regression, generalized linear models (Nelder and Wedderburn, 1972) and generalized additive models (Hastie and Tibshirani, 1990).

At the other end of the spectrum are models which allow the response to depend on all input variables simultaneously, without any additive decomposition:

$$f(\mathbf{x}) = f(x_1, x_2, \ldots, x_D) \tag{6.2}$$

An example would be a GP with an SE-ARD kernel. Such models are much more flexible than those having the form (6.1), but this flexibility can make it difficult to generalize to new combinations of input variables.

In between these extremes are function classes depending on pairs or triplets of inputs, such as

$$f(x_1, x_2, x_3) = f_{12}(x_1, x_2) + f_{23}(x_2, x_3) + f_{13}(x_1, x_3) \tag{6.3}$$

We call the number of input variables appearing in each term the *order* of a model class. Models of intermediate order such as (6.3) allow more flexibility than models of form (6.1) ($D$-th order), but have more structure than those of form (6.2) (first-order).

Capturing the low-order additive structure present in a function can be expected to improve predictive accuracy. However, if the function being learned depends in some way on an interaction between all input variables, a $D$th-order term is required in order for the model to be consistent.

## 6.2 Defining additive kernels

To define the additive kernels introduced in this chapter, we first assign each dimension $i \in \{1 \ldots D\}$ a one-dimensional *base kernel* $k_i(x_i, x_i')$. Then, the first order, second order

and $n$th order additive kernel are defined as:

$$k_{add_1}(\mathbf{x}, \mathbf{x}') = \sigma_1^2 \sum_{i=1}^{D} k_i(x_i, x_i') \tag{6.4}$$

$$k_{add_2}(\mathbf{x}, \mathbf{x}') = \sigma_2^2 \sum_{i=1}^{D} \sum_{j=i+1}^{D} k_i(x_i, x_i') k_j(x_j, x_j') \tag{6.5}$$

$$k_{add_n}(\mathbf{x}, \mathbf{x}') = \sigma_n^2 \sum_{1 \le i_1 < i_2 < \ldots < i_n \le D} \left[ \prod_{d=1}^{n} k_{i_d}(x_{i_d}, x_{i_d}') \right] \tag{6.6}$$

$$k_{add_D}(\mathbf{x}, \mathbf{x}') = \sigma_D^2 \sum_{1 \le i_1 < i_2 < \ldots < i_D \le D} \left[ \prod_{d=1}^{D} k_{i_d}(x_{i_d}, x_{i_d}') \right] = \sigma_D^2 \prod_{d=1}^{D} k_d(x_d, x_d') \tag{6.7}$$

where $D$ is the dimension of the input space, and $\sigma_n^2$ is the variance assigned to all $n$th order interactions. The $n$th-order kernel is a sum of $\binom{D}{n}$ terms. In particular, the $D$th-order additive kernel has $\binom{D}{D} = 1$ term, a product of each dimension's kernel. In the case where each base kernel is a one-dimensional squared-exponential kernel, the $D$th-order term corresponds to the multivariate squared-exponential kernel, also known as SE-ARD:

$$\prod_{d=1}^{D} \mathrm{SE}(x_d, x_d') = \sigma_D^2 \prod_{d=1}^{D} \exp\left( -\frac{(x_d - x_d')^2}{2\ell_d^2} \right) = \sigma_D^2 \exp\left( -\sum_{d=1}^{D} \frac{(x_d - x_d')^2}{2\ell_d^2} \right) \tag{6.8}$$

The full additive kernel is a sum of the additive kernels of all orders.

The only design choice necessary to specify an additive kernel is the selection of a one-dimensional base kernel for each input dimension. Parameters of the base kernels (such as length-scales $\ell_1, \ell_2, \ldots, \ell_D$) can be learned as usual by maximizing the marginal likelihood of the training data.

### 6.2.1 Weighting different orders of interaction

In addition to the parameters of each dimension's kernel, additive kernels are equipped with a set of $D$ parameters $\sigma_1^2 \ldots \sigma_D^2$. These "order variance" parameters have a useful interpretation: the $d$th order variance parameter specifies how much of the target function's variance comes from interactions of the $d$th order. Table 6.1 shows examples of the variance contributed by different orders of interaction, estimated on real datasets.

On different datasets, the dominant order of interaction estimated by the additive model varies widely. Low-order structure sometimes allows extrapolation, as shown in section 2.4.2. If low-dimensional additive structure is not present, the kernel parameters

Table 6.1 Percentage of variance contributed by each order of interaction of the additive model on different datasets. The maximum order of interaction is set to the input dimension or 10, whichever is smaller.

| | Order of interaction | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Dataset | 1st | 2nd | 3rd | 4th | 5th | 6th | 7th | 8th | 9th | 10th |
| pima | 0.1 | 0.1 | 0.1 | 0.3 | 1.5 | **96.4** | 1.4 | 0.0 | | |
| liver | 0.0 | 0.2 | **99.7** | 0.1 | 0.0 | 0.0 | | | | |
| heart | **77.6** | 0.0 | 0.0 | 0.0 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 22.0 |
| concrete | **70.6** | 13.3 | 13.8 | 2.3 | 0.0 | 0.0 | 0.0 | 0.0 | | |
| pumadyn-8nh | 0.0 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | **99.5** | | |
| servo | **58.7** | 27.4 | 0.0 | 13.9 | | | | | | |
| housing | 0.1 | 0.6 | **80.6** | 1.4 | 1.8 | 0.8 | 0.7 | 0.8 | 0.6 | 12.7 |

can specify a suitably flexible model, with interactions between as many variables as necessary.

### 6.2.2 Efficiently evaluating additive kernels

An additive kernel over $D$ inputs with interactions up to order $n$ has $O(2^n)$ terms. Naïvely summing these terms is intractable. Perhaps surprisingly, one can eactly evaluate the sum over all terms in $O(D^2)$, while also weighting each order of interaction seperately.

To efficiently compute the additive kernel, we exploit the fact that the $n$th order additive kernel corresponds to the $n$th *elementary symmetric polynomial* (Macdonald, 1998) of the base kernels, which we denote $e_n$. For example: if $\mathbf{x}$ has 4 input dimensions ($D = 4$), and if we use the shorthand notation $k_d = k_d(x_d, x'_d)$, then

$$k_{\text{add}_0}(\mathbf{x}, \mathbf{x}') = e_0(k_1, k_2, k_3, k_4) = 1 \tag{6.9}$$

$$k_{\text{add}_1}(\mathbf{x}, \mathbf{x}') = e_1(k_1, k_2, k_3, k_4) = k_1 + k_2 + k_3 + k_4 \tag{6.10}$$

$$k_{\text{add}_2}(\mathbf{x}, \mathbf{x}') = e_2(k_1, k_2, k_3, k_4) = k_1k_2 + k_1k_3 + k_1k_4 + k_2k_3 + k_2k_4 + k_3k_4 \tag{6.11}$$

$$k_{\text{add}_3}(\mathbf{x}, \mathbf{x}') = e_3(k_1, k_2, k_3, k_4) = k_1k_2k_3 + k_1k_2k_4 + k_1k_3k_4 + k_2k_3k_4 \tag{6.12}$$

$$k_{\text{add}_4}(\mathbf{x}, \mathbf{x}') = e_4(k_1, k_2, k_3, k_4) = k_1k_2k_3k_4 \tag{6.13}$$

The Newton-Girard formulas give an efficient recursive form for computing these poly-

nomials:

$$k_{\mathrm{add}_n}(\mathbf{x}, \mathbf{x}') = e_n(k_1, k_2, \ldots, k_D) = \frac{1}{n}\sum_{a=1}^{n}(-1)^{(a-1)} e_{n-a}(k_1, k_2, \ldots, k_D)\sum_{i=1}^{D} k_i^a \qquad (6.14)$$

These formulas have time complexity $\mathcal{O}(D^2)$, while computing a sum over an exponential number of terms.

#### Evaluation of derivatives

Conveniently, we can use the same trick to efficiently compute the necessary derivatives of the additive kernel with respect to the base kernels. This can be done by removing the base kernel of interest from each term of the polynomials:

$$\frac{\partial k_{add_n}}{\partial k_j} = \frac{\partial e_n(k_1, k_2, \ldots, k_D)}{\partial k_j} = e_{n-1}(k_1, k_2, \ldots, k_{j-1}, k_{j+1}, \ldots k_D) \qquad (6.15)$$

These derivatives allow optimization of the base kernel parameters with respect to the marginal likelihood using gradient-based methods.

#### Computational cost

The computational cost of evaluating the Gram matrix $k(\mathbf{X}, \mathbf{X})$ of a product kernel such as the SE-ARD scales as $\mathcal{O}(N^2 D)$, while the cost of evaluating the Gram matrix of the additive kernel scales as $\mathcal{O}(N^2 DR)$, where $R$ is the maximum degree of interaction allowed (up to $D$). In high dimensions this can be a significant cost, even relative to the $\mathcal{O}(N^3)$ cost of inverting the Gram matrix. However, table 6.1 shows that sometimes only the first few orders of interaction contribute much variance. In those cases, one may be able to limit the maximum degree of interaction to save time, without losing much accuracy.

## 6.3 Additive models allow non-local interactions

Commonly-used kernels such as the SE, RQ or Matérn kernels are *local* kernels, depending only on the scaled Euclidean distance between two points, having the form:

$$k(\mathbf{x}, \mathbf{x}') = g\left(\sum_{d=1}^{D}\left(\frac{x_d - x'_d}{\ell_d}\right)^2\right) \qquad (6.16)$$

for some function $g(\cdot)$. Bengio et al. (2006) argue that models based on local kernels are particularly susceptible to the *curse of dimensionality* (Bellman, 1956), and are generally unable to extrapolate away from the training data. Thus, methods based solely on local kernels sometimes require training examples at exponentially-many combinations of inputs.

Additive kernels can allow extrapolation far from the training data. For example, additive kernels of the second order give high covariance between function values at points which are similar in any two input dimensions.

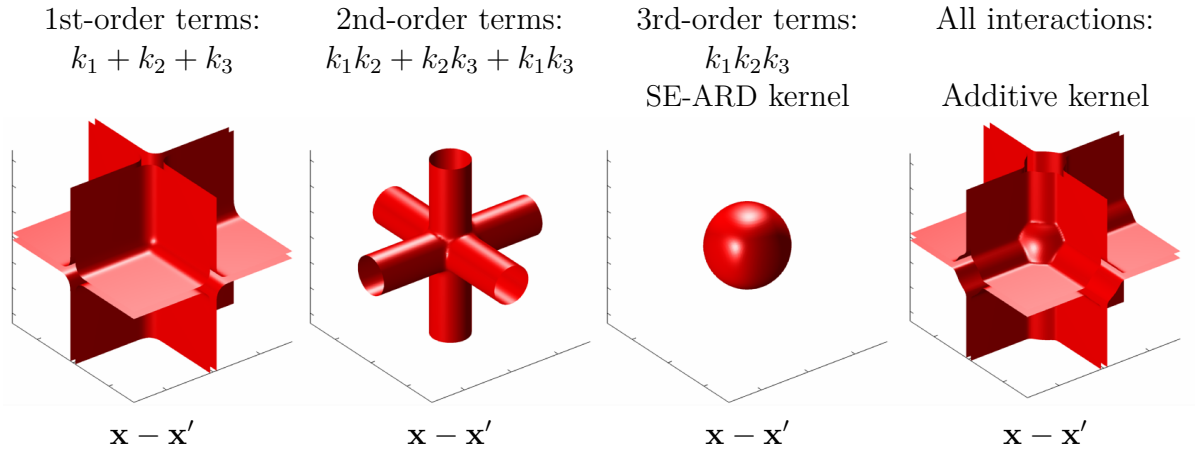



Figure 6.1 Isocontours of additive kernels in $D = 3$ dimensions. The $D$th-order kernel only considers nearby points relevant, while the lower-order kernels allow the output to depend on distant points, as long as they share one or more input value.

Figure 6.1 provides a geometric comparison between squared-exponential kernels and additive kernels in 3 dimensions. Section 2.4.1 contains an example of how additive kernels extrapolate differently than local kernels.

## 6.4 Dropout in Gaussian processes

*Dropout* is a recently-introduced method for regularizing neural networks (Hinton et al., 2012; Srivastava, 2013). Training with dropout entails randomly and independently setting to zero ("dropping") some proportion $p$ of features or inputs, in order to improve the robustness of the resulting network, by reducing co-dependence between neurons. To maintain similar overall activation levels, the remaining weights are divided by $p$. Predictions are made by approximately averaging over all possible ways of dropping out neurons.

Baldi and Sadowski (2013) and Wang and Manning (2013) analyzed dropout in terms of the effective prior induced by this procedure in several models, such as linear and logistic regression. In this section, we perform a similar analysis for GPs, examining the priors on functions that result from performing dropout in the one-hidden-layer neural network implicitly defined by a GP.

Recall from section 5.1 that some GPs can be derived as infinitely-wide one-hidden-layer neural networks, with fixed activation functions $\mathbf{h}(\mathbf{x})$ and independent random weights $\mathbf{w}$ having zero mean and finite variance $\sigma_{\mathbf{w}}^2$:

$$f(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K} w_i h_i(\mathbf{x}) \implies f \overset{K\to\infty}{\sim} \mathcal{GP}\Big(0, \sigma_{\mathbf{w}}^2 \mathbf{h}(\mathbf{x})^\mathsf{T}\mathbf{h}(\mathbf{x}')\Big) \tag{6.17}$$

where $k(\mathbf{x}, \mathbf{x}') = \sigma_{\mathbf{w}}^2 \mathbf{h}(\mathbf{x})^\mathsf{T}\mathbf{h}(\mathbf{x}')$. Having expressed a GP as a neural network, we can examine the prior obtained by performing dropout in this network.

### 6.4.1 Dropout on infinitely-wide hidden layers has no effect

First, we examine the prior obtained by dropping features from $\mathbf{h}(\mathbf{x})$ by setting weights $\mathbf{w}$ independently to zero with probability $p$. For simplicity, we assume that $\mathbb{E}[\mathbf{w}] = \mathbf{0}$. If the weights initially have finite variance $\sigma_w^2$ before dropout, then after dropout they will have variance

$$r_i \overset{\text{iid}}{\sim} \text{Ber}(p) \qquad \mathbb{V}[r_i w_i] = p\sigma_w^2 \,. \tag{6.18}$$

Because equation (6.17) is a result of the central limit theorem, it does not depend on the exact form of the distribution on $\mathbf{w}$, but only on its mean and variance. Thus dropping out features of an infinitely-wide MLP does not change the model at all, except to rescale the output variance. Indeed, dividing all weights by $\sqrt{p}$ restores the initial variance:

$$\mathbb{V}\left[\frac{1}{p^{\frac{1}{2}}} r_i w_i\right] = \frac{p}{p}\sigma_w^2 = \sigma_w^2 \,. \tag{6.19}$$

In which case dropout on the hidden units has no effect at all. Intuitively, this is because no individual feature can have more than an infinitesimal contribution to the network output.

This result does not hold in neural networks having a finite number of hidden features with Gaussian-distributed weights, another model class that gives rise to GPs.

### 6.4.2 Dropout on inputs gives additive covariance

We can also perform dropout on the $D$ inputs to the GP. For simplicity, we'll consider a stationary product kernel $k(\mathbf{x}, \mathbf{x}') = \prod_{d=1}^{D} k_d(x_d, x'_d)$ which has been normalized such that $k(\mathbf{x}, \mathbf{x}') = 1$, and dropout probability of $p = 1/2$. In this case, the generative model can be written as:

$$r_i \overset{\text{iid}}{\sim} \text{Ber}\left(\frac{1}{2}\right), \quad f(\mathbf{x})|\mathbf{r} \sim \mathcal{GP}\left(0, \prod_{d=1}^{D} k_d(x_d, x'_d)^{r_d}\right) \tag{6.20}$$

This is a mixture of $2^D$ GPs, each depending on a different subset of the inputs:

$$p\left(f(\mathbf{x})\right) = \sum_{\mathbf{r}} p\left(f(\mathbf{x})|\mathbf{r}\right) p(\mathbf{r}) = \frac{1}{2^D} \sum_{\mathbf{r}\in\{0,1\}^D} \mathcal{GP}\left(f(\mathbf{x}) \,\middle|\, 0, \prod_{d=1}^{D} k_d(x_d, x'_d)^{r_d}\right) \tag{6.21}$$

We present two results which might give intuition about this model.

First, if the kernel on each dimension has the form $k_d(x_d, x'_d) = g\left(\frac{x_d - x'_d}{\ell_d}\right)$, as does the SE kernel, then any input dimension can be dropped out by setting its lengthscale $\ell_d$ to $\infty$. Thus, performing dropout on the inputs of a GP corresponds to putting independent spike-and-slab priors on the lengthscales, with each dimension's distribution independently having "spikes" at $\ell_d = \infty$ with probability mass of $1/2$.

Another way to understand the resulting prior is to note that the dropout mixture (equation (6.21)) has the same covariance as an additive GP, scaled by a factor of $2^{-D}$:

$$\text{cov}\left(\begin{bmatrix} f(\mathbf{x}) \\ f(\mathbf{x}') \end{bmatrix}\right) = \frac{1}{2^D} \sum_{\mathbf{r}\in\{0,1\}^D} \prod_{d=1}^{D} k_d(x_d, x'_d)^{r_d} \tag{6.22}$$

For dropout rates $p \neq 1/2$, the $d$th order terms will be weighted by $p^{(1-D)}(1-p)^D$. Therefore, performing dropout on the inputs of a GP gives a distribution with the same first two moments as an additive GP. This suggests an interpretation of additive GPs as an approximation to a mixture of models where each model only depends on a subset of the input variables.

## 6.5 Related work

Since additive models are a relatively natural and easy-to-analyze model class, the literature on similar model classes is extensive. This section attemps to provide a broad

overview.

### Previous examples of additive GPs

The additive models considered in this chapter are axis-aligned, but transforming the input space would allow one to recover non-axis aligned additivity. This model was explored by Gilboa et al. (2013), who developed a linearly-transformed first-order additive GP model, called projection-pursuit GP regression. They showed that inference in this model was possible in $\mathcal{O}(N)$ time.

Durrande et al. (2011) also examined properties of additive GPs, and proposed a layer-wise optimization strategy for kernel hyperparameters in these models.

Plate (1999) constructed an additive GP having only first-order and $D$th-order terms, motivated by the desire to trade off the interpretability of first-order models with the flexibility of full-order models. Table 6.1 shows that sometimes intermediate degrees of interaction contribute most of the variance.

Kaufman and Sain (2010) used a closely related procedure called Gaussian process ANOVA to perform a Bayesian analysis of meteorological data using 2nd and 3rd-order interactions. They introduced a weighting scheme to ensure that each order's total contribution sums to zero. It is not clear if this weighting scheme permits the use of the Newton-Girard formulas to speed computation of the Gram matrix.

### Hierarchical kernel learning

A similar model class was recently explored by Bach (2009) called hierarchical kernel learning (HKL). HKL uses a regularized optimization framework to build a weighted sum of an exponential number of kernels that can be computed in polynomial time. This method chooses among a *hull* of kernels, defined as a set of terms such that if $\prod_{j \in J} k_j(\mathbf{x}, \mathbf{x}')$ is included in the set, then so are all products of subsets containing the same elements: $\prod_{j \in J/i} k_j(\mathbf{x}, \mathbf{x}')$, for all $i \in J$. HKL does not estimate a seperate weighting parameter for each order.

Figure 6.2 contrasts the HKL model class with the additive GP model. Neither model class encompasses the other. The main difficulty with this approach is that the kernel parameters are hard to set other than by cross-validation.

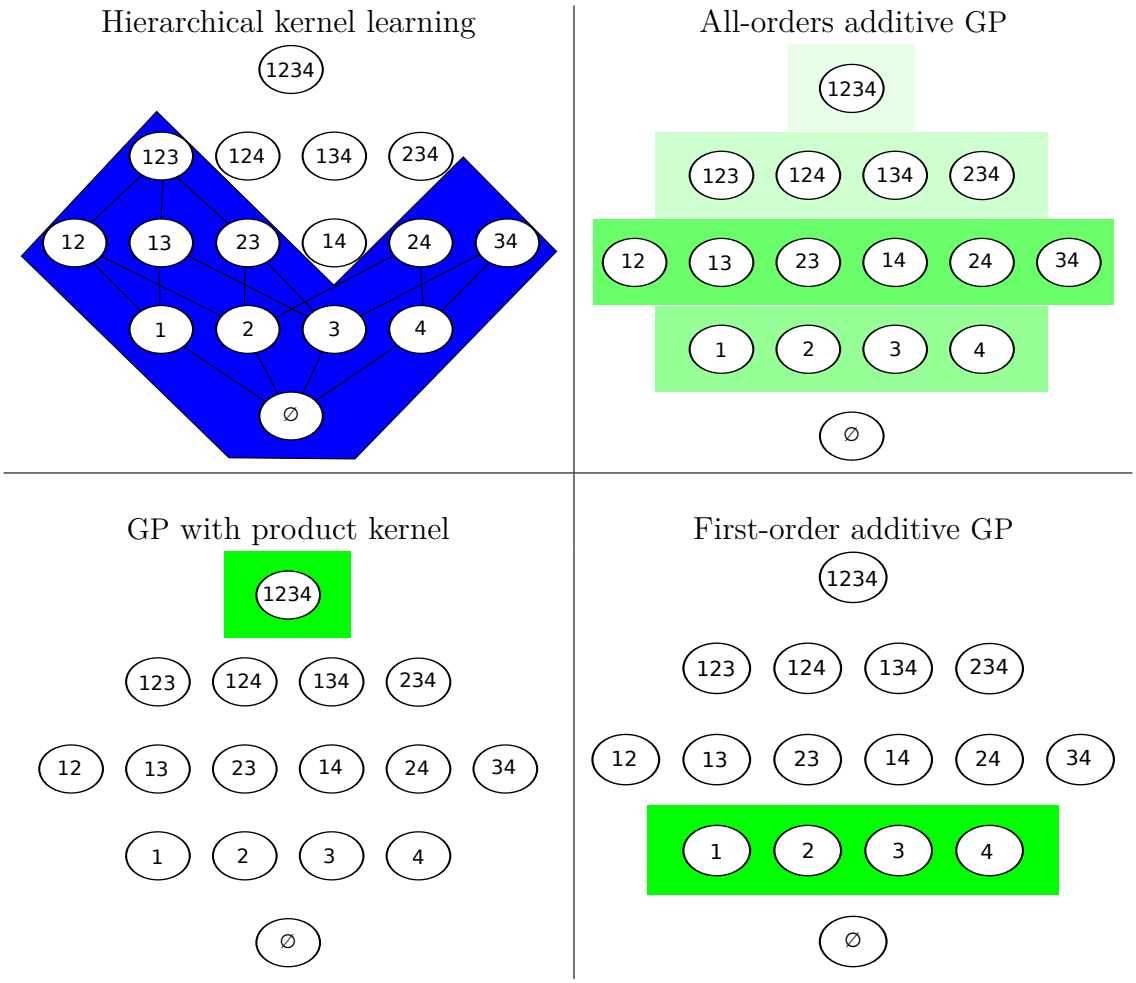



Figure 6.2 A comparison of different additive model classes of 4-dimensional functions. Circles represent different interaction terms, ranging from first-order to fourth-order interactions. Coloured boxes represent the weightings of different terms. *Top left:* HKL can select a hull of interaction terms, but must use a pre-determined weighting over those terms. *Top right:* the additive GP model can weight each order of interaction separately, but weights all terms equally within each order. *Bottom row:* GPs with product kernels (such as the SE-ARD kernel) and first-order additive GP models are special cases of the all-orders additive GP, with all variance assigned to a single order of interaction.

### Support vector machines

Vapnik (1998) introduced the support vector ANOVA decomposition, which has the same form as our additive kernel. They recommend approximating the sum over all interactions with only one set of interactions "of appropriate order", presumably because of the difficulty of setting the parameters of an SVM. This is an example of a model choice which can be automated in the GP framework.

Stitson et al. (1999) performed experiments which favourably compared the predictive accuracy of the support vector ANOVA decomposition against polynomial and spline kernels. They too allowed only one order to be active, and set parameters by cross-validation.

### Other related models

A closely related procedure from Wahba (1990) is smoothing-splines ANOVA (SS-ANOVA). An SS-ANOVA model is a weighted sum of splines along each dimension, splines over all pairs of dimensions, all triplets, etc, with each individual interaction term having a separate weighting parameter. Because the number of terms to consider grows exponentially in the order, only terms of first and second order are usually considered in practice.

This more general model class, in which each interaction term is estimated separately, is known in the physical sciences as high dimensional model representation (HDMR). Rabitz and Aliş (1999) review some properties and applications of this model class.

The main benefits of the model setup and parameterization proposed in this chapter are the ability to include all $D$ orders of interaction with differing weights, and the ability to learn kernel parameters individually per input dimension, allowing automatic relevance determination to operate.

## 6.6 Regression and classification experiments

### Choosing the base kernel

An additive GP using a seperate SE kernel on each input dimension will have $3 \times D$ effective parameters. Because each additional parameter increases the tendency to overfit, in our experiments we fixed each one-dimensional kernel's output variance to be 1, and only estimated the lengthscale of each kernel.

### Methods

We compared six different methods. In the results tables below, GP Additive refers to a GP using the additive kernel with squared-exp base kernels. For speed, we limited the maximum order of interaction to 10. GP-1st denotes an additive GP model with only first-order interactions - a sum of one-dimensional kernels. GP Squared-exp is a GP using a SE-ARD kernel. HKL was run using the all-subsets kernel, which corresponds to the same set of interaction terms as considered by the additive GP with a squared-exp base kernel.

For all GP models, we fit kernel parameters by the standard method of maximizing training-set marginal likelihood, using L-BFGS (Nocedal, 1980) for 500 iterations, allowing five random restarts. In addition to learning kernel parameters, we fit a constant mean function to the data. In the classification experiments, approximate GP inference was performed using expectation propagation (Minka, 2001).

The regression experiments also compared against the structure search method from chapter 3, run up to depth 10, using only the SE and RQ base kernels.

## 6.6.1 Datasets

We compared these methods on regression and classification datasets from the UCI repository (Bache and Lichman, 2013). Their size and dimension are given in tables 6.2 and 6.3:

Table 6.2 Regression dataset statistics

| Method | bach | concrete | pumadyn | servo | housing |
|---|---|---|---|---|---|
| Dimension | 8 | 8 | 8 | 4 | 13 |
| Number of datapoints | 200 | 500 | 512 | 167 | 506 |

Table 6.3 Classification dataset statistics

| Method | breast | pima | sonar | ionosphere | liver | heart |
|---|---|---|---|---|---|---|
| Dimension | 9 | 8 | 60 | 32 | 6 | 13 |
| Number of datapoints | 449 | 768 | 208 | 351 | 345 | 297 |

Table 6.4 Regression mean squared error

| Method | bach | concrete | pumadyn-8nh | servo | housing |
|---|---|---|---|---|---|
| Linear Regression | 1.031 | 0.404 | 0.641 | 0.523 | 0.289 |
| GP-1st | 1.259 | 0.149 | 0.598 | 0.281 | 0.161 |
| HKL | **0.199** | 0.147 | 0.346 | 0.199 | 0.151 |
| GP Squared-exp | **0.045** | 0.157 | 0.317 | 0.126 | **0.092** |
| GP Additive | **0.045** | **0.089** | **0.316** | **0.110** | 0.102 |
| Structure Search | **0.044** | **0.087** | **0.315** | **0.102** | **0.082** |

Table 6.5 Regression negative log-likelihood

| Method | bach | concrete | pumadyn-8nh | servo | housing |
|---|---|---|---|---|---|
| Linear Regression | 2.430 | 1.403 | 1.881 | 1.678 | 1.052 |
| GP-1st | 1.708 | 0.467 | 1.195 | 0.800 | 0.457 |
| GP Squared-exp | **−0.131** | 0.398 | 0.843 | 0.429 | 0.207 |
| GP Additive | **−0.131** | **0.114** | **0.841** | **0.309** | 0.194 |
| Structure Search | **−0.141** | **0.065** | **0.840** | **0.265** | **0.059** |

#### Bach synthetic dataset

In addition to standard UCI repository datasets, we generated a synthetic dataset using the same recipe as Bach (2009). This dataset was presumably designed to demonstrate the advantages of HKL over a GP using an SE-ARD kernel. It is generated by passing correlated Gaussian-distributed inputs $x_1, x_2, \ldots, x_8$ through the quadratic function

$$f(\mathbf{x}) = \sum_{i=1}^{4} \sum_{j=1+1}^{4} x_i x_j + \epsilon \qquad \epsilon \sim \mathcal{N}(0, \sigma_\epsilon) \tag{6.23}$$

This dataset will presumably be well-modeled by an additive kernel which includes all two-way interactions over the first 4 variables, but does not depend on the extra 4 correlated nuisance inputs or the higher-order interactions.

### 6.6.2 Results

Tables 6.4 to 6.7 show mean performance across 10 train-test splits. Because HKL does not specify a noise model, it could not be included in the likelihood comparisons.

On each dataset, the best performance is in boldface, along with all other perfor-

Table 6.6 Classification percent error

| Method | breast | pima | sonar | ionosphere | liver | heart |
|---|---|---|---|---|---|---|
| Logistic Regression | 7.611 | 24.392 | 26.786 | 16.810 | 45.060 | **16.082** |
| GP-1st | **5.189** | **22.419** | **15.786** | **8.524** | **29.842** | **16.839** |
| HKL | **5.377** | 24.261 | **21.000** | 9.119 | **27.270** | **18.975** |
| GP Squared-exp | **4.734** | **23.722** | **16.357** | **6.833** | **31.237** | **20.642** |
| GP Additive | **5.566** | **23.076** | **15.714** | **7.976** | **30.060** | **18.496** |

Table 6.7 Classification negative log-likelihood

| Method | breast | pima | sonar | ionosphere | liver | heart |
|---|---|---|---|---|---|---|
| Logistic Regression | 0.247 | 0.560 | 4.609 | 0.878 | 0.864 | 0.575 |
| GP-1st | **0.163** | **0.461** | **0.377** | **0.312** | **0.569** | **0.393** |
| GP Squared-exp | **0.146** | 0.478 | **0.425** | **0.236** | **0.601** | 0.480 |
| GP Additive | **0.150** | **0.466** | **0.409** | **0.295** | **0.588** | **0.415** |

mances not significantly different under a paired $t$-test. The additive and structure search methods usually outperformed the other methods, especially on regression problems.

The structure search outperforms the additive GP at the cost of a slow search over kernels. Structure search was on the order of 10 times slower than the additive GP, which was on the order of 10 times slower than GP-SE.

The additive GP performed best on datasets well-explained by low orders of interaction, and approximately as well as the SE-GP model on datasets which were well explained by high orders of interaction (see table 6.1). Because the additive GP is a superset of both the GP-1st model and the GP-SE model, instances where the additive GP performs slightly worse are presumably due to over-fitting, or due to the hyperparameter optimization becoming stuck in a local maximum. Performance of all GP models could be expected to benefit from approximately integrating over the kernel parameters.

The performance of HKL is consistent with the results in Bach (2009), performing competitively but slightly worse than SE-GP.

### Source code

All of the experiments in this chapter were performed using the standard GPML toolbox, available at `http://wwww.gaussianprocess.org/gpml/code`. The additive kernel de-

scribed in this chapter is included in GPML as of version 3.2. Code to perform all experiments in this chapter is available at http://www.github.com/duvenaud/additive-gps.

## 6.7 Conclusions

This chapter presented a tractable GP model consisting of a sum of exponentially-many functions, each depending on a different subset of the inputs. Our experiments indicate that, to varying degrees, such additive structure is useful for modeling real datasets. When it is present, modeling this structure allows our model to perform better than standard GP models. In the case where no such structure exists, the higher-order interaction terms present in the kernel can recover arbitrarily flexible models. The additive GP also affords some degree of interpretability: the variance parameters on each order of interaction indicate which sorts of structure are present the data, although they do not indicate which particular interactions explain the dataset.

The model class considered in this chapter is a subset of that explored by the structure search presented in chapter 3. Thus additive GPs can be considered a quick-and-dirty structure search, being strictly more limited in the types of structure that it can discover, but much faster and simpler to implement.

Closely related model classes have previously been explored, most notably smoothing-splines ANOVA and the support vector ANOVA decomposition. However, these models can be difficult to apply in practice because their kernel parameters, regularization penalties, and the relevant orders of interaction must be set by hand or by cross-validation. This chapter illustrates that the GP framework allows these model choices to be performed automatically.

# Chapter 7

# Warped Mixture Models

> "What, exactly, is a cluster?"
>
> - Bernhard Schölkopf, personal communication

Previous chapters showed how the probabilistic nature of GPs sometimes lets us automatically include the appropriate amount of structure, and the correct kind, when building models of functions. One can also take advantage of this property when composing GPs with other models, automatically trading-off complexity between the GP and the other parts of the model.

This chapter considers a simple example: a Gaussian mixture model warped by a draw from a GP. This novel model produces clusters (density manifolds) with arbitrary nonparametric shapes. We call the proposed model the *infinite warped mixture model* (iWMM). Figure 7.3 shows a set of manifolds and datapoints sampled from the prior defined by this model. The probabilistic nature of the iWMM lets us automatically infer the number, dimension, and shape of a set of nonlinear manifolds, and summarize those manifolds in a low-dimensional latent space.

The work comprising the bulk of this chapter was done in collaboration with Tomoharu Iwata and Zoubin Ghahramani, and appeared in Iwata et al. (2013). The main idea was born out of a conversation between Tomoharu and myself, and together we wrote almost all of the code as well as the paper. Tomoharu ran most of the experiments. Zoubin Ghahramani provided guidance and many helpful suggestions throughout the project.

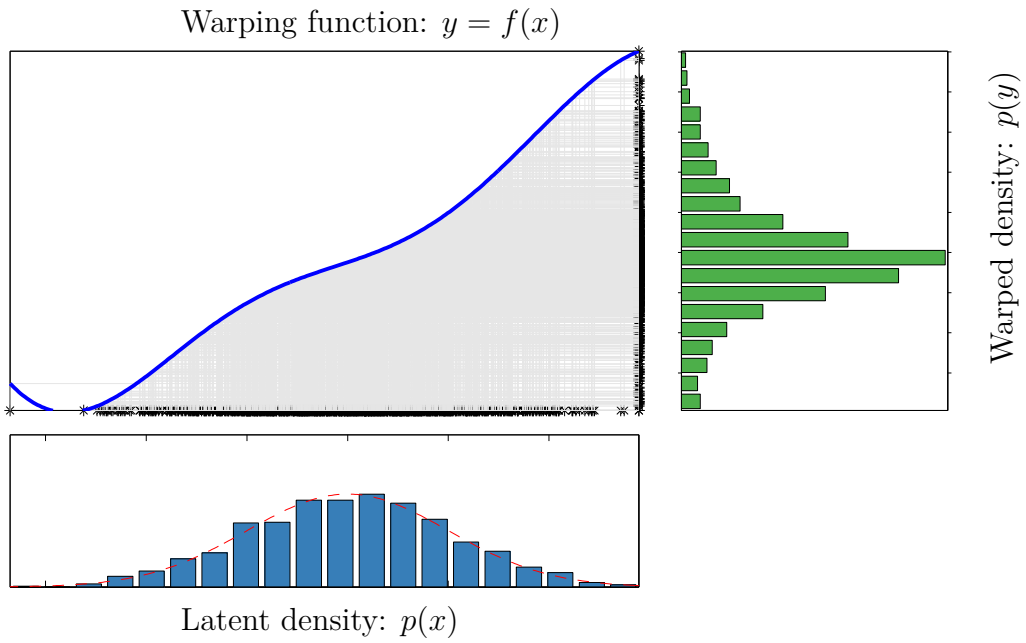



Figure 7.1 A draw from a one-dimensional Gaussian process latent variable model. *Bottom:* the density of a set of samples from a 1D Gaussian, specifying the distribution $p(x)$ in the latent space. *Top left:* A function $y = f(x)$ drawn from a GP prior. Grey lines show points being mapped through $f$. *Right:* A nonparametric density $p(y)$ defined by warping the latent density through the sampled function.

## 7.1 The Gaussian process latent variable model

The iWMM can be viewed as an extension of the Gaussian process latent variable model (GP-LVM) (Lawrence, 2004), a probabilistic model of nonlinear manifolds. The GP-LVM smoothly warps a Gaussian density into a more complicated distribution, using a draw from a GP. Usually, we think of the Gaussian density as living in a "latent space" having $Q$ dimensions, and the warped density living in the observed space having $D$ dimensions.

A generative definition of the GP-LVM is:

$$\text{latent coordinates } \mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)^\mathsf{T} \overset{\text{iid}}{\sim} \mathcal{N}(\mathbf{x}|0, \mathbf{I}_Q) \tag{7.1}$$

$$\text{warping functions } \boldsymbol{f} = (f_1, f_2, \ldots, f_D)^\mathsf{T} \overset{\text{iid}}{\sim} \mathcal{GP}(0, \text{SE-ARD} + \text{WN}) \tag{7.2}$$

$$\text{observed datapoints } \mathbf{Y} = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N)^\mathsf{T} = \boldsymbol{f}(\mathbf{X}) \tag{7.3}$$

Under the GP-LVM, the probability of observations $\mathbf{Y}$ given the latent coordinates

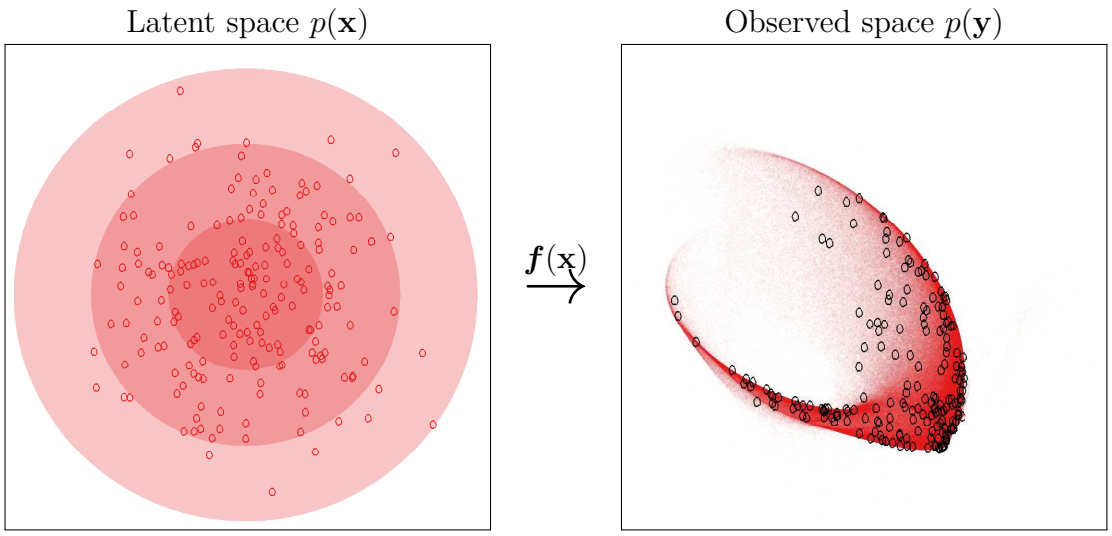



Figure 7.2 A draw from a multi-dimensional Gaussian process latent variable model. *Left:* Isocontours and samples from a 2D Gaussian, specifying the distribution $p(\mathbf{x})$ in the latent space. *Right:* Observed density $p(\mathbf{y})$ has a nonparametric shape, defined by warping the latent density through a function drawn from a GP prior.

$\mathbf{X}$ integrating out the mapping functions $f$ is simply a product of GP likelihoods:

$$p(\mathbf{Y}|\mathbf{X},\boldsymbol{\theta}) = \prod_{d=1}^{D} p(\mathbf{Y}_{:,d}|\mathbf{X},\boldsymbol{\theta}) = \prod_{d=1}^{D} \mathcal{N}(\mathbf{Y}_{:,d}|0,\mathbf{K}_{\boldsymbol{\theta}}) \tag{7.4}$$

$$= (2\pi)^{-\frac{DN}{2}} |\mathbf{K}|^{-\frac{D}{2}} \exp\left(-\frac{1}{2}\mathrm{tr}(\mathbf{Y}^{\mathsf{T}}\mathbf{K}^{-1}\mathbf{Y})\right), \tag{7.5}$$

where $\boldsymbol{\theta}$ are the kernel parameters and $\mathbf{K}_{\boldsymbol{\theta}}$ is the Gram matrix $k_{\boldsymbol{\theta}}(\mathbf{X},\mathbf{X})$.

Typically, the GP-LVM is used for dimensionality reduction or visualization, and the latent coordinates are set by maximizing (7.5). In that setting, the Gaussian prior density on $\mathbf{x}$ is essentially a regularizer which keeps the latent coordinates from spreading arbitrarily far apart. One can also integrate out $\mathbf{X}$, which is the approach taken in this chapter.

## 7.2 The infinite warped mixture model

This section defines the infinite warped mixture model (iWMM). Like the GP-LVM, the iWMM assumes a smooth nonlinear mapping from a latent density to an observed density. The only difference is that the iWMM assumes that the latent density is an

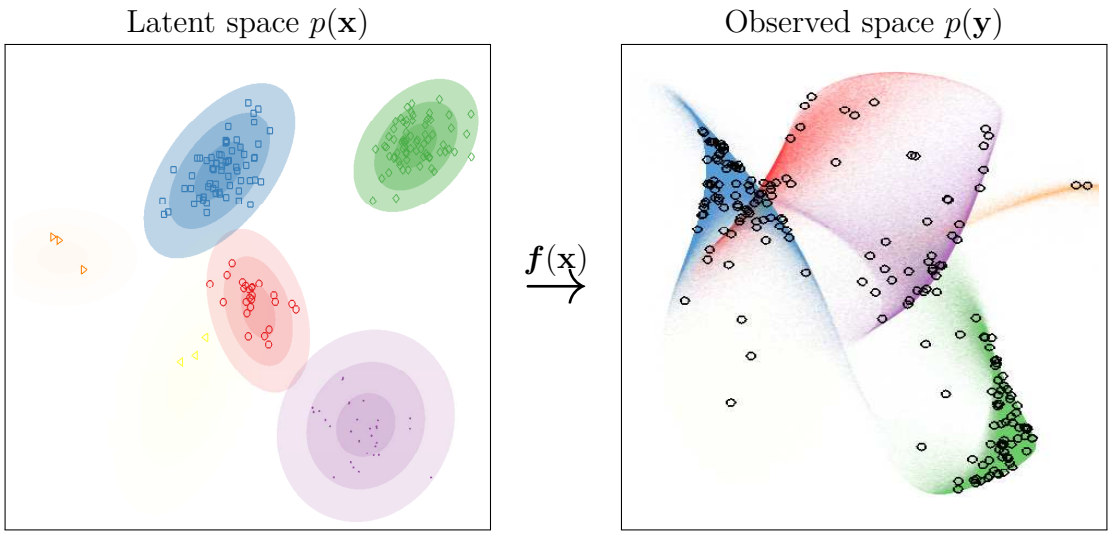



Figure 7.3 A sample from the iWMM prior. *Left:* In the latent space, a mixture distribution is sampled from a Dirichlet process mixture of Gaussians. *Right:* The latent mixture is smoothly warped to produce a set of non-Gaussian manifolds in the observed space.

infinite Gaussian mixture model (iGMM) (Rasmussen, 2000):

$$p(\mathbf{x}) = \sum_{c=1}^{\infty} \lambda_c \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_c, \mathbf{R}_c^{-1}) \tag{7.6}$$

where $\lambda_c$, $\boldsymbol{\mu}_c$ and $\mathbf{R}_c$ is the mixture weight, mean, and precision matrix of the $c^{\text{th}}$ mixture component.

The iWMM can be seen as a generalization of either the GP-LVM or the iGMM: The iWMM with a single fixed spherical Gaussian density on the latent coordinates $p(\mathbf{x})$ corresponds to the GP-LVM, while the iWMM with fixed mapping $\mathbf{y} = \mathbf{x}$ and $Q = D$ corresponds to the iGMM.

A flexible model of cluster shapes is required to correctly estimate the number of clusters, if those clusters do not happen to be Gaussian. For example, a mixture of Gaussians fit to a single non-Gaussian cluster (such as one that is curved or heavy-tailed) will report that the data contains many Gaussian clusters.

## 7.3 Inference

As discussed in section 1.1.3, one of the main advantages of GP priors is that, given inputs $\mathbf{X}$, outputs $\mathbf{Y}$ and kernel parameters $\boldsymbol{\theta}$, one can analytically integrate over functions

mapping $\mathbf{X}$ to $\mathbf{Y}$. However, inference becomes more difficult if we introduce uncertainty about the kernel parameters or the input locations $\mathbf{X}$. This section outlines how to compute approximate posterior distributions over all parameters in the iWMM given only a set of observations $\mathbf{Y}$. Details can be found in appendix E.

We first place conjugate priors on the parameters of the Gaussian mixture components, allowing analytic integration over latent cluster shapes, given the assignments of points to clusters. The only remaining variables to infer are the latent points $\mathbf{X}$, the cluster assignments $\mathbf{z}$, and the kernel parameters $\boldsymbol{\theta}$. We can obtain samples from their posterior $p(\mathbf{X}, \mathbf{z}, \boldsymbol{\theta}|\mathbf{Y})$ by iterating two steps:

1. Given the latent points $\mathbf{X}$, sample the discrete cluster memberships $\mathbf{z}$ using collapsed Gibbs sampling, integrating out the iGMM parameters (E.9).

2. Given the cluster assignments $\mathbf{z}$, sample the continuous latent coordinates $\mathbf{X}$ and kernel parameters $\boldsymbol{\theta}$ using Hamiltonian Monte Carlo (HMC) (MacKay, 2003, chapter 30). The relevant equations are given by equations (E.11) to (E.14).

The complexity of each iteration of HMC is dominated by the $\mathcal{O}(N^3)$ computation of $\mathbf{K}^{-1}$. This complexity could be improved by making use of an inducing-point approximation (Quiñonero-Candela and Rasmussen, 2005; Snelson and Ghahramani, 2006).

#### Posterior predictive density

One disadvantage of the GP-LVM is that its predictive density has no closed form. To approximate the predictive density, we sample latent points and warpings from the posterior and map the sampled points through the sampled warpings. The Gaussian noise added to each observation by the WN kernel component means that each sample adds a Gaussian to the Monte Carlo estimate of the predictive density. Details can be found in chapter E. This procedure was used to generate the plots of posterior density in figures 7.3, 7.4 and 7.6.

## 7.4 Related work

The literature on manifold learning, clustering and dimensionality reduction is extensive. This section highlights some of the most relevant related work.

### Extensions of the GP-LVM

The GP-LVM has been used effectively in a wide variety of applications (Lawrence, 2004; Lawrence and Urtasun, 2009; Salzmann et al., 2008). The latent positions $\mathbf{X}$ in the GP-LVM are typically obtained by maximum a posteriori estimation or variational Bayesian inference (Titsias and Lawrence, 2010), placing a single fixed spherical Gaussian prior on $\mathbf{x}$.

A regularized extension of the GP-LVM which allows estimation of the dimension of the latent space was introduced by Geiger et al. (2009), in which the latent variables and their intrinsic dimensionality were simultaneously optimized. The iWMM can also infer the intrinsic dimensionality of nonlinear manifolds: the Gaussian covariance parameters for each latent cluster allow the variance of irrelevant dimensions to become small. The marginal likelihood of the latent Gaussian mixture will favor using as few dimensions as possible to describe each cluster. Because each latent cluster has a different set of parameters, each cluster can have a different effective dimension in the observed space, as demonstrated in figure 7.4(c).

Nickisch and Rasmussen (2010) considered several modifications of the GP-LVM which model the latent density using a mixture of Gaussians centered around the latent points. They approximated the observed density $p(\mathbf{Y})$ by another mixture of Gaussians, obtained by moment-matching the density obtained by warping each latent Gaussian into the observed space. Because their model was not generative, training was done by maximizing a leave-some-out predictive density. This method had poor predictive performance compared to simple baselines.

### Related linear models

The iWMM can also be viewed as a generalization of the mixture of probabilistic principle component analyzers (Tipping and Bishop, 1999), or the mixture of factor analyzers (Ghahramani and Beal, 2000), where the linear mapping of the mixtures is generalized to a nonlinear mapping by Gaussian processes, and the number of components is infinite.

### Non-probabilistic methods

There exist non-probabilistic clustering methods which can find clusters with complex shapes, such as spectral clustering (Ng et al., 2002) and nonlinear manifold clustering (Cao and Haralick, 2006; Elhamifar and Vidal, 2011). Spectral clustering finds clus-

ters by first forming a similarity graph, finding a low-dimensional latent representation using the graph, and clustering the latent coordinates via k-means. The performance of spectral clustering depends on parameters which are usually set manually, such as the number of clusters, the number of neighbors, and the variance parameter used for constructing the similarity graph. The iWMM infers such parameters automatically, and has no need to construct a similarity graph.

The kernel Gaussian mixture model (Wang et al., 2003) can also find non-Gaussian shaped clusters. This model estimates a GMM in the implicit infinite-dimensional feature space defined by the kernel mapping of the observed space. However, the kernel parameters must be set by cross-validation. In contrast, the iWMM infers the mapping function such that the latent coordinates will be well-modeled by a mixture of Gaussians.

#### Nonparametric cluster shapes

To the best of our knowledge, the only other Bayesian clustering method with nonparametric cluster shapes is that of Rodríguez and Walker (2012), who for one-dimensional data introduce a nonparametric model of *unimodal* clusters, where each cluster's density function strictly decreases away from its mode.

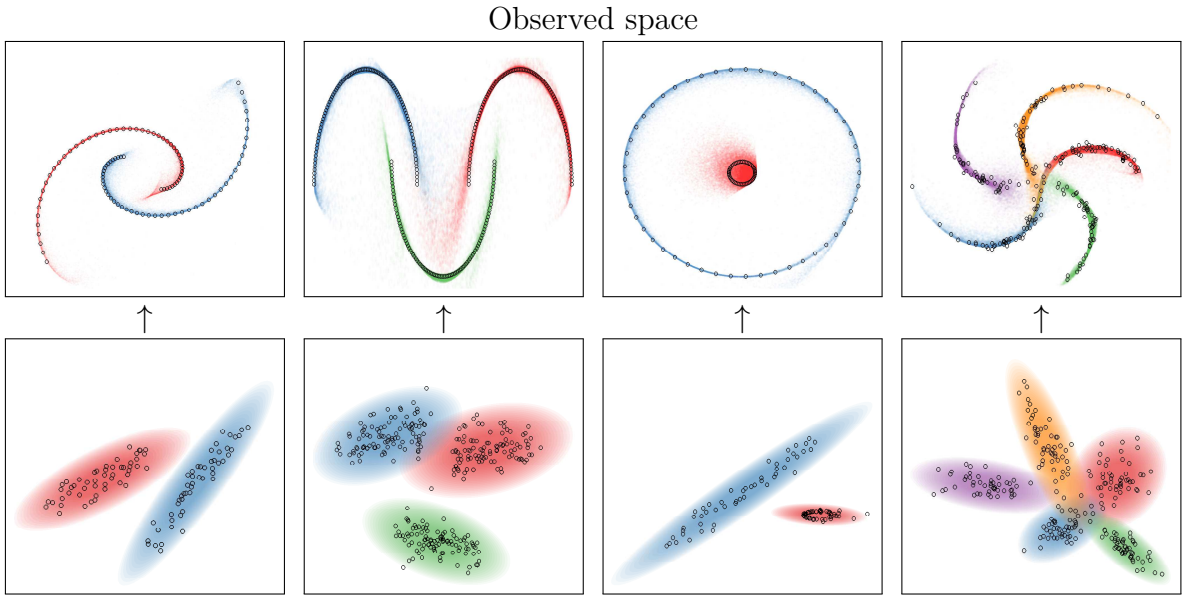
#### Deep Gaussian processes

An elegant way to construct a GP-LVM with a more structured latent density $p(\mathbf{x})$ is to use another GP-LVM to model the latent coordinates $\mathbf{X}$. This latent GP-LVM can have another GP-LVM modeling its latent space, etc. This model class was considered by Damianou and Lawrence (2013), who also tested to what extent each layer's latent representation grouped points having the same label. They found that when modeling MNIST hand-written digits, nearest-neighbour classification performed best in the 4th layer of a 5-layer deep nested GP-LVM, suggesting that the latent density might have been implicitly forming clusters at that level.

## 7.5 Experimental results

### 7.5.1 Synthetic datasets

Figure 7.4 demonstrates the proposed model on four synthetic datasets. None of these datasets can be appropriately clustered by Gaussian mixture models (GMM). For example, consider the 2-curve data shown in Figure 7.4(a), where 100 data points lie in each




(a) 2-curve (b) 3-semi (c) 2-circle (d) Pinwheel

Figure 7.4 *Top row:* Observed unlabeled data points (black), and cluster densities inferred by the iWMM (colors). *Bottom row:* Latent coordinates and Gaussian components from a single sample from the posterior. Each point in the latent space corresponds to a point in the observed space.

of two curved lines in a two-dimensional observed space. A GMM with two components cannot separate the two curved lines, while a GMM with many components could separate the two lines only by breaking each line into many clusters. In contrast, the iWMM represents the two non-Gaussian clusters in the observed space by two Gaussian-shaped clusters in the latent space. Figure 7.4(b) shows a similar three-cluster example.

Figure 7.4(c) shows an interesting manifold learning challenge: a dataset consisting of two concentric circles. The outer circle is modeled in the latent space by a Gaussian with one effective degree of freedom. This linear topology is fit to the outer circle in the observed space by bending the two ends until they cross over. In contrast, the sampler fails to discover the 1D topology of the inner circle, modeling it with a 2D manifold instead. This example demonstrates that each cluster in the iWMM can have a different effective dimension.

Figure 7.4(d) shows a five-armed variant of the pinwheel dataset of Adams and Ghahramani (2009), generated by warping a mixture of Gaussians into a spiral. This generative process closely matches the assumptions of the iWMM. Unsurprisingly, the iWMM is able to recover anologous latent structure, and its predictive density follows

the observed data manifolds.

### 7.5.2 Clustering face images

We also examined our model's ability to model images without pre-processing. We constructed a dataset consisting of 50 greyscale 32x32 pixel images of two individuals from the UMIST faces dataset (Graham and Allinson, 1998). Both series of images show a person turning his head to the right.

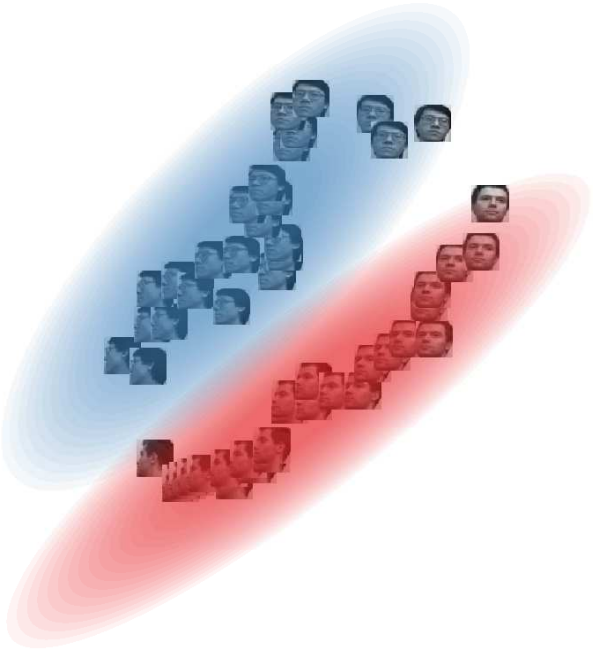


Figure 7.5 A sample from the 2-dimensional latent space modeling a series of face images. Images are rendered at their latent 2D coordinates. The iWMM reports that the data consists of two separate manifolds, both approximately one-dimensional, which both share the same head-turning structure.

Figure 7.5 shows a sample from the posterior over latent coordinates and density, with each image rendered at its location in the latent space. The model has recovered three interpretable features of the dataset: First, that there are two distinct faces. Second, that each set of images lies approximately along a smooth one-dimensional manifold. Third, that the two manifolds share roughly the same structure: the front-facing images of both individuals lie close to one another, as do the side-facing images.

(a) iWMM (b) GP-LVM

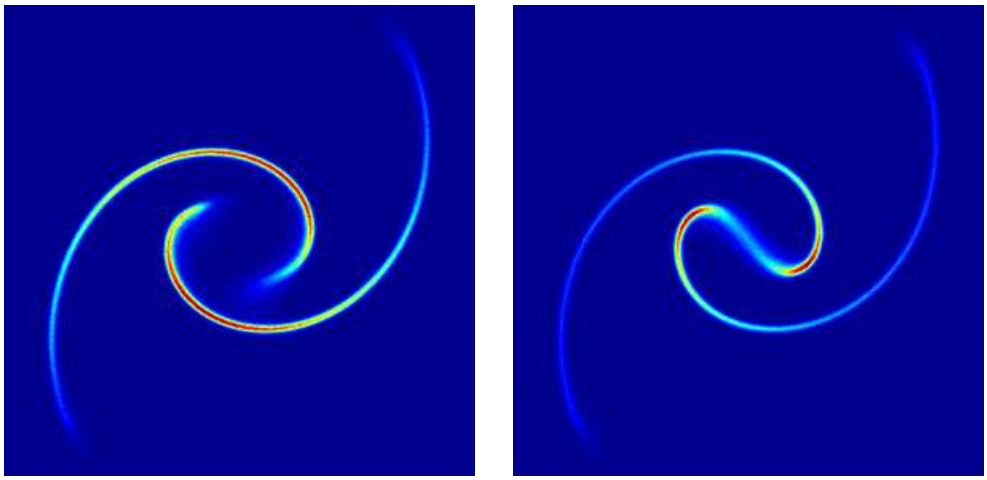


Figure 7.6 *Left:* Posterior density inferred by the iWMM in the observed space, on the 2-curve data. *Right:* Posterior density inferred by the iWMM with one component, a model equivalent to a fully-Bayesian GP-LVM.

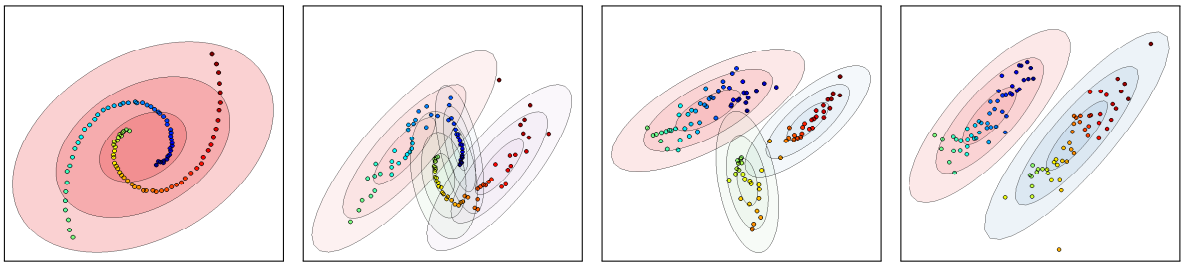
### 7.5.3 Density estimation

Figure 7.6(a) shows the posterior density in the observed space inferred by the iWMM on the 2-curve data, computed using 1000 samples from the Markov chain. The seperation of the density into two unconnected manifolds was recovered by the iWMM. Also note that the density along the manifold varies along with the density of data, shown in figure 7.4(a).

This result can be compared to a fully-Bayesian GP-LVM, equivalent to a special case of our model having only a single Gaussian in the latent space. Figure 7.6(b) shows that the GP-LVM places significant density connecting the two clusters, since it has to reproduce the observed density manifold by warping a single Gaussian.

### 7.5.4 Mixing

An interesting side-effect of learning the number of latent clusters is that this added flexibility can help the sampler escape local minima. Figure 7.7 shows the samples of the latent coordinates and clusters of the iWMM over a single Markov chain modeling the 2-curve data. Figure 7.7(a) shows the latent coordinates initialized at the observed coordinates, starting with one latent component. After 500 iterations, each curved line was modeled by two components. After 1800 iterations, the left curved line was modeled by a single component. After 3000 iterations, the right curved line was also modeled



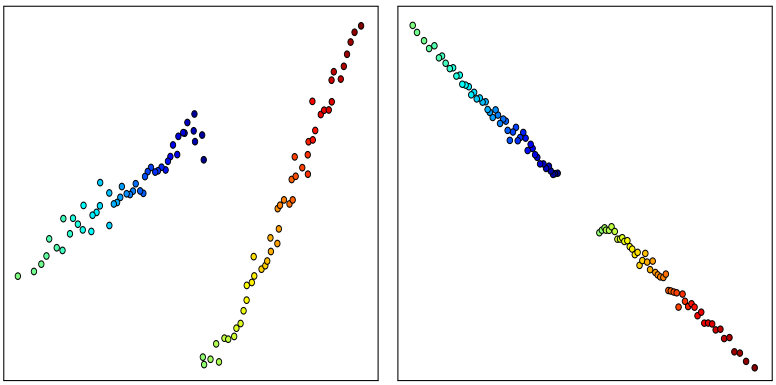
(a) Initialization (b) Iteration 500 (c) Iteration 1800 (d) Iteration 3000

Figure 7.7 Latent coordinates and densities of the iWMM, plotted throughout one run of a Markov chain.

by a single component, and the dataset was appropriately clustered. This configuration was relatively stable, and a similar state was found at the 5000th iteration.

### 7.5.5 Visualization

Next, we briefly investigate the utility of the iWMM for low-dimensional visualization of data. Figure 7.8(a) shows the latent coordinates obtained by averaging over 1000



(a) iWMM (b) iWMM ($C = 1$)

Figure 7.8 Latent coordinates of the 2-curve data, estimated by two different methods.

samples from the posterior of the iWMM. The estimated latent coordinates are clearly separated, forming two straight lines. This result is an example of the iWMM recovering the original topology of the data before it was warped.

For comparison, figure 7.8(b) shows the latent coordinates estimated by the fully-Bayesian GP-LVM, in which case the latent coordinates lie in two sections of a single straight line.

### 7.5.6 Clustering performance

We more formally evaluated the density estimation and clustering performance of the proposed model using four real datasets: iris, glass, wine and vowel, obtained from LIBSVM multi-class datasets (Chang and Lin, 2011), in addition to the four synthetic datasets shown above: 2-curve, 3-semi, 2-circle and pinwheel (Adams and Ghahramani, 2009). The statistics of these datasets are summarized in Table 7.1. For each experiment,

Table 7.1 Statistics of the datasets used for evaluation.

| | 2-curve | 3-semi | 2-circle | pinwheel | iris | glass | wine | vowel |
|---|---|---|---|---|---|---|---|---|
| samples: $N$ | 100 | 300 | 100 | 250 | 150 | 214 | 178 | 528 |
| dimension: $D$ | 2 | 2 | 2 | 2 | 4 | 9 | 13 | 10 |
| num. clusters: $C$ | 2 | 3 | 2 | 5 | 3 | 7 | 3 | 11 |

we show the results of ten-fold cross-validation. Results in bold are not significantly different from the best performing method in each column according to a paired t-test.

Table 7.2 Average Rand index for evaluating clustering performance.

| | 2-curve | 3-semi | 2-circle | Pinwheel | Iris | Glass | Wine | Vowel |
|---|---|---|---|---|---|---|---|---|
| iGMM | 0.52 | 0.79 | 0.83 | 0.81 | 0.78 | 0.60 | 0.72 | **0.76** |
| iWMM(Q=2) | **0.86** | **0.99** | **0.89** | **0.94** | **0.81** | **0.65** | 0.65 | 0.50 |
| iWMM(Q=D) | **0.86** | **0.99** | **0.89** | **0.94** | 0.77 | 0.62 | **0.77** | **0.76** |

Table 7.2 compares the clustering performance of the iWMM with the iGMM, quantified by the Rand index (Rand, 1971), which measures the correspondence between inferred clusters and true clusters. Since the manifold on which the observed data lies can be at most $D$-dimensional, we set the latent dimension $Q$ equal to the observed dimension $D$. We also included the $Q = 2$ case in an attempt to characterize how much modeling power is lost by forcing the latent representation to be visualizable.

These experiments were designed to measure the extent to which nonparametric cluster shapes helped to estimate meaningful clusters. To eliminate any differences due to different inference procedures, we used identical code for the iGMM and iWMM, the only difference being that the warping function was set to the identity $\mathbf{y} = \mathbf{x}$. Both variants of the iWMM usually outperformed the iGMM on this measure.

### 7.5.7 Density estimation

Next, we compared the iWMM in terms of predictive density against kernel density estimation (KDE), the iGMM, and the fully-Bayesian GP-LVM. For KDE, the kernel width was estimated by maximizing the leave-one-out density. Table 7.3 lists average test log likelihoods.

Table 7.3 Average test log-likelihoods for evaluating density estimation performance.

| | 2-curve | 3-semi | 2-circle | Pinwheel | Iris | Glass | Wine | Vowel |
|---|---|---|---|---|---|---|---|---|
| KDE | −2.47 | −0.38 | −1.92 | −1.47 | **−1.87** | 1.26 | −2.73 | **6.06** |
| iGMM | −3.28 | −2.26 | −2.21 | −2.12 | −1.91 | 3.00 | **−1.87** | −0.67 |
| GP-LVM(Q=2) | −1.02 | −0.36 | **−0.78** | **−0.78** | −1.91 | **5.70** | **−1.95** | **6.04** |
| GP-LVM(Q=D) | −1.02 | −0.36 | **−0.78** | **−0.78** | −1.86 | **5.59** | −2.89 | −0.29 |
| iWMM(Q=2) | **−0.90** | **−0.18** | **−1.02** | **−0.79** | **−1.88** | **5.76** | **−1.96** | **5.91** |
| iWMM(Q=D) | **−0.90** | **−0.18** | **−1.02** | **−0.79** | **−1.71** | **5.70** | −3.14 | −0.35 |

The iWMM usually achieved higher test likelihoods than the KDE and the iGMM. The GP-LVM performed competitively with the iWMM, although it never significantly outperformed the iWMM having the same latent dimension.

The sometimes large differences between performance in the $D = 2$ case and the $D = Q$ case of these two methods may be attributed to the fact that when the observed dimension is high, many samples are required from the latent distribution to produce accurate estimates of the posterior predictive density at the test locations. This difficulty might be resolved by using a warping with back-constraints (Lawrence, 2006), which would allow a more direct evaluation of the density at a given point in the observed space.

#### Source code

Code to reproduce all the above figures and experiments is available at
`http://www.github.com/duvenaud/warped-mixtures`.

## 7.6 Conclusions

This chapter introduced a simple generative model of non-Gaussian density manifolds which can infer nonlinearly separable clusters, low-dimensional representations of varying dimension per cluster, and density estimates which smoothly follow the contours of each

cluster. We also introduced a sampler for this model which integrates out both the cluster parameters and the warping function exactly at each step.

Non-probabilistic methods such as spectral clustering can also produce nonparametric cluster shapes, but usually lack principled methods for setting kernel parameters, the number of clusters, and the implicit dimension of the learned manifolds, other than by cross-validation. This chapter showed that using a fully generative model allows most model choices to be determined automatically.

Many methods have been proposed which can perform some combination of clustering, manifold learning, density estimation and visualization. We demonstrated that a simple but flexible probabilistic generative model can perform well at all these tasks.

## 7.7 Future work

### More sophisticated latent density models

The Dirichlet process mixture of Gaussians in the latent space of our model could easily be replaced by a more sophisticated density model, such as a hierarchical Dirichlet process (Teh et al., 2006), or a Dirichlet diffusion tree (Neal, 2003). Another straightforward extension of our model would be making inference more scalable by using sparse Gaussian processes (Quiñonero-Candela and Rasmussen, 2005; Snelson and Ghahramani, 2006) or more advanced Hamiltonian Monte Carlo methods (Zhang and Sutton, 2011).

### A finite cluster count model

Miller and Harrison (2013) note that the Dirichlet process assumes infinitely many clusters, and that estimates of the number of clusters in a dataset based on Bayesian inference are inconsistent under this model. They propose a consistent alternative which still allows efficient Gibbs sampling, called the mixture of finite mixtures. Replacing the Dirichlet process with a mixture of finite mixtures could improve the consistency properties of the iWMM.

### Semi-supervised learning

A straightforward extension of the iWMM would be a semi-supervised version of the model. The iWMM could allow label propagation along regions of high density in the latent space, even if the individual points in those regions are stretched far apart along low-dimensional manifolds in the observed space. Another natural extension would be

to allow a separate warping for each cluster, producing a mixture of warped Gaussians, rather than a warped mixture of Gaussians.

### Learning the topology of data manifolds

Some datasets naturally live on manifolds which are not simply-connected. For example, motion capture data or video of a person walking in a circle naturally lives on a torus, with one coordinate specifying the phase of the person's step, and another specifying how far around the circle they are.

As shown in section 2.8, using structured kernels to specify the warping of a latent space gives rise to interesting topologies on the observed density manifold. If a suitable method for computing the marginal likelihood of a GP-LVM is available, an automatic search similar to that described in chapter 3 would be applicable, automatically discovering the topology of the data manifold.

# Chapter 8

# Discussion

This chapter summarizes the contributions of this thesis, articulates some of the questions raised by this work, and relates the kernel-based model-building procedure of chapters 2 to 4 to the larger project of automating statistics and machine learning.

## 8.1 Summary of contributions

The main contribution of this thesis was to develop a way to automate the construction of structured, interpretable nonparametric regression models using Gaussian processes. This was done in several parts: First, chapter 2 presented a systematic overview of kernel construction techniques, and examined the resulting GP priors. Next, chapter 3 showed the viability of a bread-first search over an open-ended space of kernels, and showed that the corresponding GP models could be automatically decomposed into diverse parts illustrating the structure found in the data. Chapter 4 showed that sometimes parts of kernels can be described modularly, allowing automatically written text to be included in detailed reports describing GP models. An example report is included in appendix D. Together, these chapters demonstrate a proof-of-concept of what could be called an "automatic statistician" capable of the performing some of the model construction and analysis currently performed by experts.

The second half of this thesis examined several extensions of Gaussian processes, all of which enabled the automatic determination of model choices that were previously set by trial and error or cross-validation. Chapter 5 characterized and visualized deep Gaussian processes, related them to existing deep neural networks, and derived novel deep kernels. Chapter 6 investigated additive GPs, and showed that they have the same covariance as a GP using dropout. Chapter 7 extended the GP latent variable model

into a Bayesian clustering model which automatically infers the nonparametric shape of each cluster, as well as the number of clusters.

## 8.2 Structured versus unstructured models

One question left unanswered by this thesis is: when to prefer the structured, kernel-based models described chapters 2 to 4 and 6 to the relatively unstructured deep GP models described in chapter 5? This section considers some advantages and disadvantages of the two approaches.

- **Difficulty of optimization.** The discrete nature of the space of composite kernel structures can be seen as both a blessing and a curse. Certainly, a mixed discrete-and-continuous search space requires more complex optimization procedures than the continuous-only optimization possible in deep GPs.

  However, the discrete nature of the space of composite kernels offers the possibility of learning heuristics to suggest which types of structure to add, based on features of the dataset, or previous model fits. For example, finding periodic structure or growing variance in the residuals suggests adding periodic or linear components to the kernel, respectively. It is not clear whether such heuristics could easily be constructed for optimizing the variational parameters of a deep GP.

- **Long-range extrapolation.** Another open question is whether the inductive bias of deep GPs can be made to allow the sorts of long-range extrapolation shown in chapters 2 and 3. As an example, consider the problem of extrapolating a periodic function. A deep GP could learn a latent representation similar to that of the periodic kernel, projecting into a basis equivalent to $[\sin(x), \cos(x)]$ in the first hidden layer. However, to extrapolate a periodic function, the sin and cos functions would have to repeat beyond the input range of the training data, which would not happen if each layer assumed only local smoothness.

  One obvious possibility is to marry the two approaches, building deep GPs with structured kernels. However, we may lose some of the advantages of interpretability by this approach.

  Another point to consider is that, in high dimensions, the distinction between interpolation and extrapolation becomes less meaningful. If the training and test data both live on a low-dimensional manifold, then learning a suitable representation of that manifold may be sufficient for obtaining high predictive accuracy.

- **Ease of interpretation.** Historically, the statistics community has put more emphasis on the interpretabilty and meaning of models than the machine learning community, which has focused more on predictive performance. To begin to automate the practice of statistics, developing model-description procedures for powerful open-ended model classes seems to be a necessary step.

  At first glance, automatic model description may seem to require a decomposition of the model being described into discrete components, as in the additive decomposition demonstrated in chapters 2 to 4.

  On the other hand, Damianou and Lawrence (2013) showed that deep GPs allow summarization of high-dimensional structure through sampling from the posterior, examining the dimension of each latent layer, visualizing latent coordinates, and examining how the predictive distribution changes as one moves in the latent space. Perhaps more sophisticated procedures could also allow intelligible text-based descriptions of such models.

The warped mixture model of chapter 7 represents a compromise between these two approaches, combining a discrete clustering model with an unstructured warping function. However, the explicit clusterting model may by unnecessary: the results of Damianou and Lawrence (2013) suggest that clustering can be automatically (but implicitly) accomplished by a sufficiently deep, unstructured GP.

## 8.3 Approaches to automating model construction

This thesis is a small part of a larger push to automate the practice of model building and inference. Broadly speaking, this problem is being attacked from two directions.

From the top-down, the probabilistic programming community is developing automatic inference engines for extremely broad classes of models (Goodman et al., 2008; Liang et al., 2010; Mansinghka et al., 2014; Wood et al., 2014) such as the class of all computable distributions (Li and Vitányi, 1997; Solomonoff, 1964). As discussed in section 3.1, model construction procedures can usually be seen as a search through an open-ended model class. Exhaustive search strategies have been constructed for the space of computable distributions (Hutter, 2002; Levin, 1973; Schmidhuber, 2002), but they remain impractically slow.

An alternative, bottom-up, approach is to design procedures which extend and combine existing model classes for which relatively efficient inference algorithms are already

known. The language of models proposed in chapter 3 is an example of this bottom-up approach. Another example is Grosse (2014), who built an open-ended language of matrix decomposition models and a corresponding compositional language of relatively efficient approximate inference algorithms. Similarily, Steinruecken (2014) showed how to compose inference algorithms for sequence models. These approaches have the advantage that inference is usually feasible for any model in the language, but extending the language may require developing new inference algorithms.

If sufficiently powerful building-blocks are composed, the line between the top-down and bottom-up approaches becomes blurred. For example, deep generative models (Adams et al., 2010; Bengio et al., 2013; Damianou and Lawrence, 2013; Rippel and Adams, 2013; Salakhutdinov and Hinton, 2009) could be considered an example of the bottom-up approach, since they compose individual model "layers" to produce more powerful models. However, large neural nets can capture enough different types of structure that they could also be seen as an example of the universalist, top-down approach.

## 8.4 End note

It seems clear that one way or another, large parts of the existing practice of model-building will eventually be automated. However, it remains to be seen which of the above model-building approaches will be most useful. I hope that this thesis will contribute to our understanding of the strengths and weaknesses of these different approaches, and towards the use of more powerful model classes by practitioners in other fields.

# Appendix A

# Gaussian Conditionals

A standard result shows how to condition on a subset of dimensions $\mathbf{y}_B$ of a vector $\mathbf{y}$ having a multivariate Gaussian distribution. If

$$\mathbf{y} = \begin{bmatrix} \mathbf{y}_A \\ \mathbf{y}_B \end{bmatrix} \sim \mathcal{N}\left( \begin{bmatrix} \boldsymbol{\mu}_A \\ \boldsymbol{\mu}_B \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{AA} & \boldsymbol{\Sigma}_{AB} \\ \boldsymbol{\Sigma}_{BA} & \boldsymbol{\Sigma}_{BB} \end{bmatrix} \right) \tag{A.1}$$

then

$$\mathbf{y}_A | \mathbf{y}_B \sim \mathcal{N}\Big( \boldsymbol{\mu}_A + \boldsymbol{\Sigma}_{AB}\boldsymbol{\Sigma}_{BB}^{-1}\left(\mathbf{x}_B - \boldsymbol{\mu}_B\right), \boldsymbol{\Sigma}_{AA} - \boldsymbol{\Sigma}_{AB}\boldsymbol{\Sigma}_{BB}^{-1}\boldsymbol{\Sigma}_{BA} \Big). \tag{A.2}$$

This result can be used in the context of Gaussian process regression, where $\mathbf{y}_B = [f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_N)]$ represents a set of function values observed at some subset of locations $[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]$, while $\mathbf{y}_A = [f(\mathbf{x}_1\star), f(\mathbf{x}_2\star), \ldots, f(\mathbf{x}_N\star)]$ represents test points whose predictive distribution we'd like to know. In this case, the necessary covariance matrices are given by:

$$\boldsymbol{\Sigma}_{AA} = k(\mathbf{X}^\star, \mathbf{X}^\star) \tag{A.3}$$

$$\boldsymbol{\Sigma}_{AB} = k(\mathbf{X}^\star, \mathbf{X}) \tag{A.4}$$

$$\boldsymbol{\Sigma}_{BA} = k(\mathbf{X}, \mathbf{X}^\star) \tag{A.5}$$

$$\boldsymbol{\Sigma}_{BB} = k(\mathbf{X}, \mathbf{X}) \tag{A.6}$$

and similarily for the mean vectors.

# Appendix B

# Kernel Definitions

This appendix gives the formulas for all one-dimensional base kernels used in the thesis. Each of these formulas is multiplied by a scale factor $\sigma_f^2$, which we omit for clarity.

$$\mathrm{C}(x, x') = 1 \tag{B.1}$$

$$\mathrm{SE}(x, x') = \exp\left(-\frac{(x - x')^2}{2\ell^2}\right) \tag{B.2}$$

$$\mathrm{Per}(x, x') = \exp\left(-\frac{2}{\ell^2}\sin^2\left(\pi\frac{x - x'}{p}\right)\right) \tag{B.3}$$

$$\mathrm{Lin}(x, x') = (x - c)(x' - c) \tag{B.4}$$

$$\mathrm{RQ}(x, x') = \left(1 + \frac{(x - x')^2}{2\alpha\ell^2}\right)^{-\alpha} \tag{B.5}$$

$$\cos(x, x') = \cos\left(\frac{2\pi(x - x')}{p}\right) \tag{B.6}$$

$$\mathrm{WN}(x, x') = \delta(x - x') \tag{B.7}$$

$$\mathrm{CP}(k_1, k_2)(x, x') = \sigma(x)k_1(x, x')\sigma(x') + (1 - \sigma(x))k_2(x, x')(1 - \sigma(x')) \tag{B.8}$$

$$\boldsymbol{\sigma}(x, x') = \sigma(x)\sigma(x') \tag{B.9}$$

$$\bar{\boldsymbol{\sigma}}(x, x') = (1 - \sigma(x))(1 - \sigma(x')) \tag{B.10}$$

where $\delta_{x,x'}$ is the Kronecker delta function, $\{c, \ell, p, \alpha\}$ represent kernel parameters, and $\sigma(x) = {}^{1}\!/_{1+\exp(-x)}$.

Equations (B.2) to (B.4) are plotted in figure 2.1, and equations (B.5) to (B.7) are plotted in figure 3.1. Draws from GP priors with changepoint kernels are shown in figure 2.9.

### The zero-mean periodic kernel

James Lloyd (personal communication) showed that the standard periodic kernel due to MacKay (1998) can be decomposed into a sum of a periodic and a constant component. He derived the equivalent periodic kernel without any constant component:

$$\mathrm{ZMPer}(x, x') = \sigma_f^2 \frac{\exp\left(\frac{1}{\ell^2}\cos 2\pi\frac{(x-x')}{p}\right) - I_0\left(\frac{1}{\ell^2}\right)}{\exp\left(\frac{1}{\ell^2}\right) - I_0\left(\frac{1}{\ell^2}\right)} \tag{B.11}$$

where $I_0$ is the modified Bessel function of the first kind of order zero.

He further showed that its limit as the lengthscale grows is the cosine kernel:

$$\lim_{\ell\to\infty} \mathrm{ZMPer}(x, x') = \cos\left(\frac{2\pi(x-x')}{p}\right). \tag{B.12}$$

Separating out the constant component allows us to express negative prior covariance, as well as increasing the interpretability of the resulting models. This covariance function is included in the GPML software package (Rasmussen and Nickisch, 2010), and its source can be viewed at `gaussianprocess.org/gpml/code/matlab/cov/covPeriodicNoDC.m`.

# Appendix C

# Search Operators

The model construction phase of ABCD starts with the noise kernel, WN. New kernel expressions are generated by applying search operators to the current kernel, which replace some part of the existing kernel expression with a new kernel expression.

The search used in the multidimensional regression experiments in sections 3.8.4 and 6.6 used only the following search operators:

$$\mathcal{S} \rightarrow \mathcal{S} + \mathcal{B} \tag{C.1}$$

$$\mathcal{S} \rightarrow \mathcal{S} \times \mathcal{B} \tag{C.2}$$

$$\mathcal{B} \rightarrow \mathcal{B}' \tag{C.3}$$

where $\mathcal{S}$ represents any kernel subexpression and $\mathcal{B}$ is any base kernel within a kernel expression. These search operators represent addition, multiplication and replacement. When the multiplication operator is applied to a subexpression which includes a sum of subexpressions, parentheses () are introduced. For instance, if rule (C.2) is applied to the subexpression $k_1 + k_2$, the resulting expression is $(k_1 + k_2) \times \mathcal{B}$.

Afterwards, we added several more search operators in order to speed up the search. This expanded set of operators was used in the experiments in sections 3.6 and 3.8.2 and chapter 4. These new operators do not change the set of possible models.

To accommodate changepoints and changewindows, we introduced the following ad-

ditional operators to our search:

$$\mathcal{S} \rightarrow \mathrm{CP}(\mathcal{S}, \mathcal{S}) \tag{C.4}$$
$$\mathcal{S} \rightarrow \mathrm{CW}(\mathcal{S}, \mathcal{S}) \tag{C.5}$$
$$\mathcal{S} \rightarrow \mathrm{CW}(\mathcal{S}, \mathrm{C}) \tag{C.6}$$
$$\mathcal{S} \rightarrow \mathrm{CW}(\mathrm{C}, \mathcal{S}) \tag{C.7}$$

where C is the constant kernel. The last two operators result in a kernel only applying outside, or within, a certain region.

To allow the search to simplify existing expressions, we introduced the following operators:

$$\mathcal{S} \rightarrow \mathcal{B} \tag{C.8}$$
$$\mathcal{S} + \mathcal{S}' \rightarrow \mathcal{S} \tag{C.9}$$
$$\mathcal{S} \times \mathcal{S}' \rightarrow \mathcal{S} \tag{C.10}$$

where $\mathcal{S}'$ represents any other kernel expression. We also introduced the operator

$$\mathcal{S} \rightarrow \mathcal{S} \times (\mathcal{B} + \mathrm{C}) \tag{C.11}$$

Which allows a new base kernel to be added along with the constant kernel, for cases when multiplying by a base kernel by itself would be overly restrictive.

# Appendix D

# Example Automatically Generated Report

The following pages of this appendix contain an automatically-generated report, run on a dataset measuring annual solar irradiation data from 1610 to 2011. This dataset was previously analyzed by Lean et al. (1995).

The structure search was run using the ABCD-interpretable variant, with base kernels SE, Lin, C, Per, $\boldsymbol{\sigma}$, and WN.

Other example reports can be found at `http://www.mlg.eng.cam.ac.uk/Lloyd/abcdoutput/`, including anlyses of wheat prices, temperature records, call centre volumes, radio interference, gas production, unemployment, number of births, and wages over time.

# 1 Executive summary

The raw data and full model posterior with extrapolations are shown in figure 1.

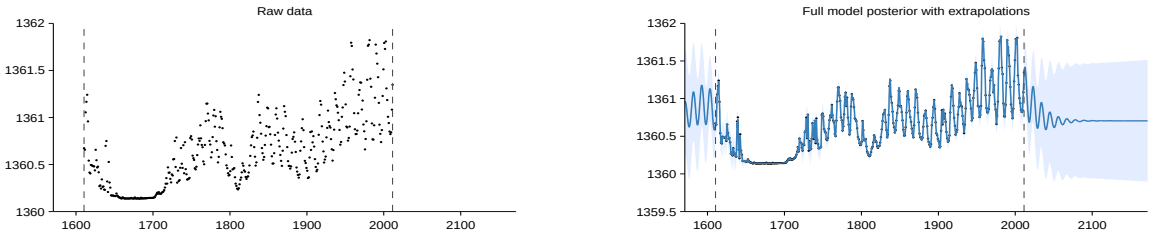



Figure 1: Raw data (left) and model posterior with extrapolation (right)

The structure search algorithm has identified nine additive components in the data. The first 4 additive components explain 92.3% of the variation in the data as shown by the coefficient of determination ($R^2$) values in table 1. The first 8 additive components explain 99.2% of the variation in the data. After the first 5 components the cross validated mean absolute error (MAE) does not decrease by more than 0.1%. This suggests that subsequent terms are modelling very short term trends, uncorrelated noise or are artefacts of the model or search procedure. Short summaries of the additive components are as follows:

- A constant.
- A constant. This function applies from 1644 until 1713.
- A smooth function. This function applies until 1644 and from 1719 onwards.
- An approximately periodic function with a period of 10.8 years. This function applies until 1644 and from 1719 onwards.
- A rapidly varying smooth function. This function applies until 1644 and from 1719 onwards.
- Uncorrelated noise.
- A rapidly varying smooth function with marginal standard deviation increasing linearly away from 1843. This function applies from 1751 onwards.
- A rapidly varying smooth function. This function applies until 1644 and from 1719 until 1751.
- A constant. This function applies from 1713 until 1719.

| # | $R^2$ (%) | $\Delta R^2$ (%) | Residual $R^2$ (%) | Cross validated MAE | Reduction in MAE (%) |
|---|---|---|---|---|---|
| - | - | - | - | 1360.65 | - |
| 1 | 0.0 | 0.0 | 0.0 | 0.33 | 100.0 |
| 2 | 35.3 | 35.3 | 35.3 | 0.23 | 29.4 |
| 3 | 72.5 | 37.2 | 57.5 | 0.18 | 20.7 |
| 4 | 92.3 | 19.9 | 72.2 | 0.15 | 16.4 |
| 5 | 97.8 | 5.5 | 71.4 | 0.15 | 0.4 |
| 6 | 97.8 | 0.0 | 0.2 | 0.15 | 0.0 |
| 7 | 98.4 | 0.5 | 24.8 | 0.15 | -0.0 |
| 8 | 99.2 | 0.8 | 50.7 | 0.15 | -0.0 |
| 9 | 100.0 | 0.8 | 100.0 | 0.15 | -0.0 |

Table 1: Summary statistics for cumulative additive fits to the data. The residual coefficient of determination ($R^2$) values are computed using the residuals from the previous fit as the target values; this measures how much of the residual variance is explained by each new component. The mean absolute error (MAE) is calculated using 10 fold cross validation with a contiguous block design; this measures the ability of the model to interpolate and extrapolate over moderate distances. The model is fit using the full data so the MAE values cannot be used reliably as an estimate of out-of-sample predictive performance.

## 2 Detailed discussion of additive components

### 2.1 Component 1 : A constant

This component is constant.

This component explains 0.0% of the total variance. The addition of this component reduces the cross validated MAE by 100.0% from 1360.6 to 0.3.

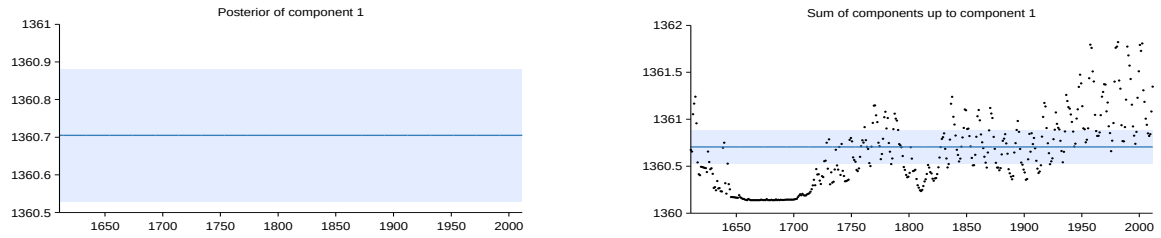



Figure 2: Posterior of component 1 (left) and the posterior of the cumulative sum of components with data (right)

### 2.2 Component 2 : A constant. This function applies from 1644 until 1713

This component is constant. This component applies from 1644 until 1713.

This component explains 35.3% of the residual variance; this increases the total variance explained from 0.0% to 35.3%. The addition of this component reduces the cross validated MAE by 29.42% from 0.33 to 0.23.

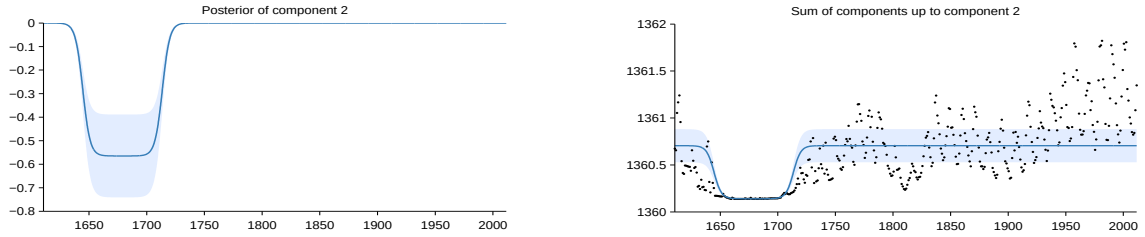



Figure 3: Posterior of component 2 (left) and the posterior of the cumulative sum of components with data (right)

### 2.3 Component 3 : A smooth function. This function applies until 1644 and from 1719 onwards

This component is a smooth function with a typical lengthscale of 21.9 years. This component applies until 1644 and from 1719 onwards.

This component explains 57.5% of the residual variance; this increases the total variance explained from 35.3% to 72.5%. The addition of this component reduces the cross validated MAE by 20.66% from 0.23 to 0.18.

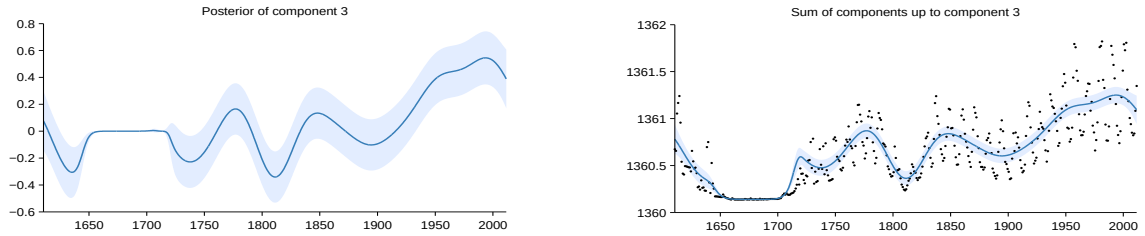



Figure 4: Posterior of component 3 (left) and the posterior of the cumulative sum of components with data (right)

### 2.4 Component 4 : An approximately periodic function with a period of 10.8 years. This function applies until 1644 and from 1719 onwards

This component is approximately periodic with a period of 10.8 years. Across periods the shape of the function varies smoothly with a typical lengthscale of 33.2 years. The shape of the function within each period has a typical lengthscale of 12.6 years. This component applies until 1644 and from 1719 onwards.

This component explains 72.2% of the residual variance; this increases the total variance explained from 72.5% to 92.3%. The addition of this component reduces the cross validated MAE by 16.42% from 0.18 to 0.15.

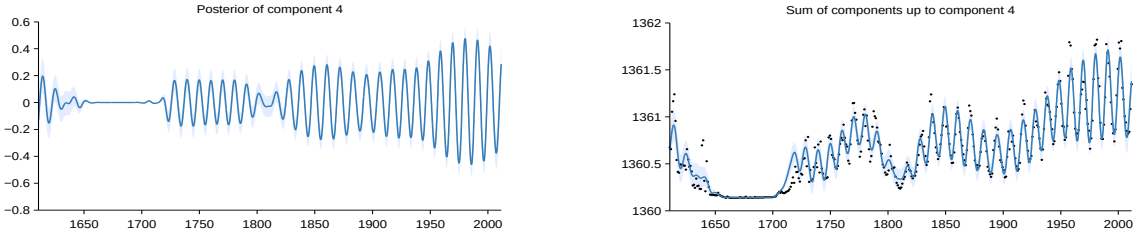



Figure 5: Posterior of component 4 (left) and the posterior of the cumulative sum of components with data (right)

### 2.5 Component 5 : A rapidly varying smooth function. This function applies until 1644 and from 1719 onwards

This function is a rapidly varying but smooth function with a typical lengthscale of 1.2 years. This component applies until 1644 and from 1719 onwards.

This component explains 71.4% of the residual variance; this increases the total variance explained from 92.3% to 97.8%. The addition of this component reduces the cross validated MAE by 0.41% from 0.15 to 0.15.

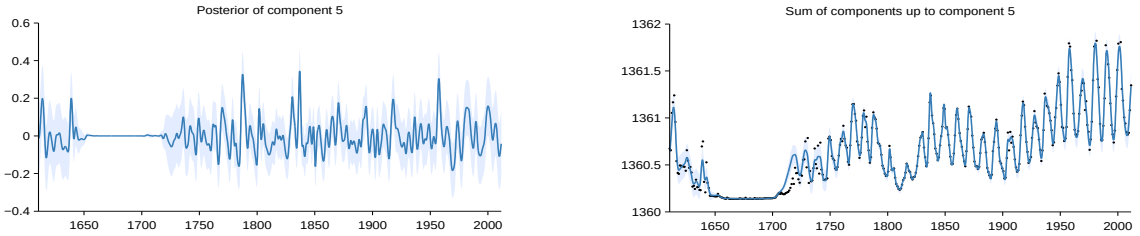



Figure 6: Posterior of component 5 (left) and the posterior of the cumulative sum of components with data (right)

### 2.6 Component 6 : Uncorrelated noise

This component models uncorrelated noise.

This component explains 0.2% of the residual variance; this increases the total variance explained from 97.8% to 97.8%. The addition of this component reduces the cross validated MAE by 0.00% from 0.15 to 0.15. This component explains residual variance but does not improve MAE which suggests that this component describes very short term patterns, uncorrelated noise or is an artefact of the model or search procedure.

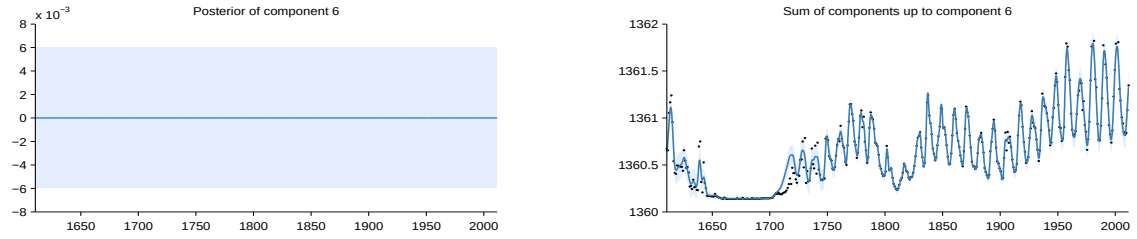



Figure 7: Posterior of component 6 (left) and the posterior of the cumulative sum of components with data (right)

### 2.7 Component 7 : A rapidly varying smooth function with marginal standard deviation increasing linearly away from 1843. This function applies from 1751 onwards

This function is a rapidly varying but smooth function with a typical lengthscale of 3.1 months. The marginal standard deviation of the function increases linearly away from 1843. This component applies from 1751 onwards.

This component explains 24.8% of the residual variance; this increases the total variance explained from 97.8% to 98.4%. The addition of this component increases the cross validated MAE by 0.00% from 0.15 to 0.15. This component explains residual variance but does not improve MAE which suggests that this component describes very short term patterns, uncorrelated noise or is an artefact of the model or search procedure.

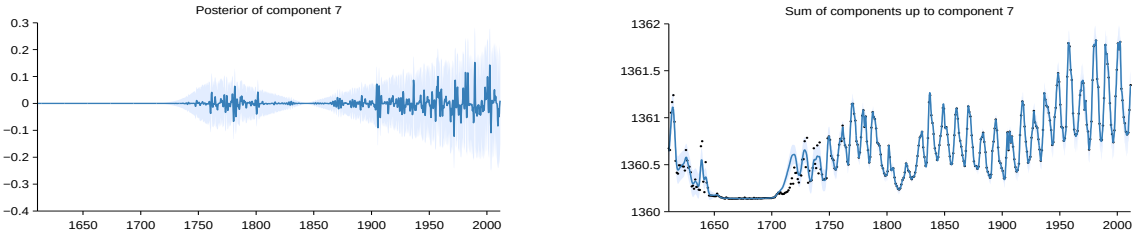



Figure 8: Posterior of component 7 (left) and the posterior of the cumulative sum of components with data (right)

## 2.8 Component 8 : A rapidly varying smooth function. This function applies until 1644 and from 1719 until 1751

This function is a rapidly varying but smooth function with a typical lengthscale of 3.1 months. This component applies until 1644 and from 1719 until 1751.

This component explains 50.7% of the residual variance; this increases the total variance explained from 98.4% to 99.2%. The addition of this component increases the cross validated MAE by 0.00% from 0.15 to 0.15. This component explains residual variance but does not improve MAE which suggests that this component describes very short term patterns, uncorrelated noise or is an artefact of the model or search procedure.

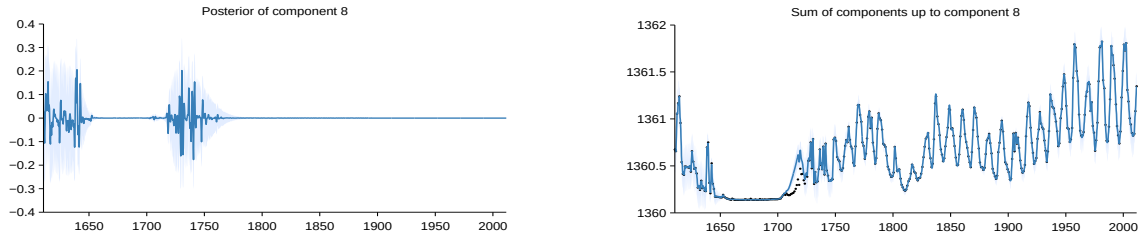



Figure 9: Posterior of component 8 (left) and the posterior of the cumulative sum of components with data (right)

## 2.9 Component 9 : A constant. This function applies from 1713 until 1719

This component is constant. This component applies from 1713 until 1719.

This component explains 100.0% of the residual variance; this increases the total variance explained from 99.2% to 100.0%. The addition of this component increases the cross validated MAE by 0.01% from 0.15 to 0.15. This component explains residual variance but does not improve MAE which suggests that this component describes very short term patterns, uncorrelated noise or is an artefact of the model or search procedure.

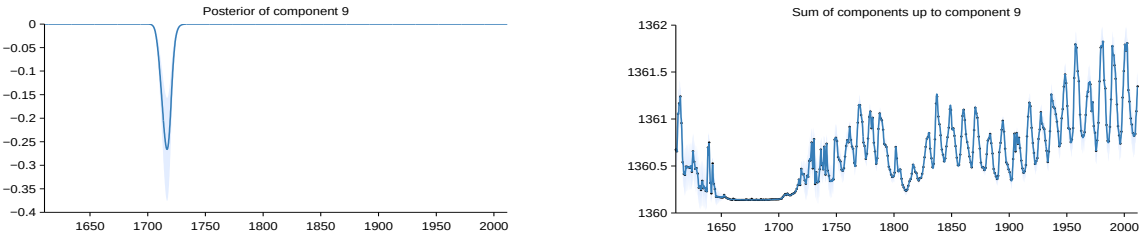



Figure 10: Posterior of component 9 (left) and the posterior of the cumulative sum of components with data (right)

## 3 Extrapolation

Summaries of the posterior distribution of the full model are shown in figure 11. The plot on the left displays the mean of the posterior together with pointwise variance. The plot on the right displays three random samples from the posterior.

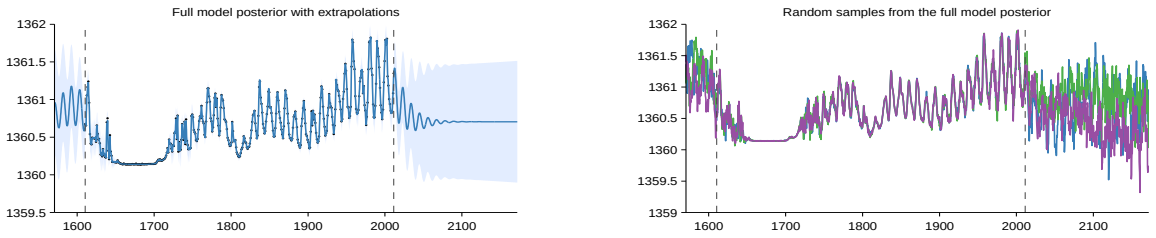



Figure 11: Full model posterior. Mean and pointwise variance (left) and three random samples (right)

# Appendix E

# Inference in the Warped Mixture Model

### Detailed definition of model

The iWMM assumes that the latent density is an infinite mixture of Gaussians:

$$p(\mathbf{x}) = \sum_{c=1}^{\infty} \lambda_c \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_c, \mathbf{R}_c^{-1}) \tag{E.1}$$

where $\lambda_c$, $\boldsymbol{\mu}_c$ and $\mathbf{R}_c$ is the mixture weight, mean, and precision matrix of the $c^{\text{th}}$ mixture component. We place a conjugate Gaussian-Wishart priors on the Gaussian parameters $\{\boldsymbol{\mu}_c, \mathbf{R}_c\}$:

$$p(\boldsymbol{\mu}_c, \mathbf{R}_c) = \mathcal{N}(\boldsymbol{\mu}_c|\mathbf{u}, (r\mathbf{R}_c)^{-1})\mathcal{W}(\mathbf{R}_c|\mathbf{S}^{-1}, \nu), \tag{E.2}$$

where $\mathbf{u}$ is the mean of $\boldsymbol{\mu}_c$, $r$ is the relative precision of $\boldsymbol{\mu}_c$, $\mathbf{S}^{-1}$ is the scale matrix for $\mathbf{R}_c$, and $\nu$ is the number of degrees of freedom for $\mathbf{R}_c$. The Wishart distribution is defined as:

$$\mathcal{W}(\mathbf{R}|\mathbf{S}^{-1}, \nu) = \frac{1}{G}|\mathbf{R}|^{\frac{\nu-Q-1}{2}} \exp\left(-\frac{1}{2}\text{tr}(\mathbf{SR})\right), \tag{E.3}$$

where $G$ is the normalizing constant.

Because we use conjugate Gaussian-Wishart priors for the parameters of the Gaussian mixture components, we can analytically integrate out those parameters given the assignments of points to components. Let $z_n$ be the assignment of the $n^{\text{th}}$ point. The prior probability of latent coordinates $\mathbf{X}$ given latent cluster assignments $\mathbf{z} = (z_1, z_2, \ldots, z_N)$

factorizes over clusters, and can be obtained in closed-form by integrating out the Gaussian parameters $\{\boldsymbol{\mu}_c, \mathbf{R}_c\}$ to give:

$$p(\mathbf{X}|\mathbf{z}, \mathbf{S}, \nu, r) = \prod_{c=1}^{\infty} \pi^{-\frac{N_c Q}{2}} \frac{r^{Q/2}}{r_c^{Q/2}} \frac{|\mathbf{S}|^{\nu/2}}{|\mathbf{S}_c|^{\nu_c/2}} \times \prod_{q=1}^{Q} \frac{\Gamma\left(\frac{\nu_c+1-q}{2}\right)}{\Gamma\left(\frac{\nu+1-q}{2}\right)}, \tag{E.4}$$

where $N_c$ is the number of data points assigned to the $c^{\text{th}}$ component, $\Gamma(\cdot)$ is the Gamma function, and

$$r_c = r + N_c, \qquad \nu_c = \nu + N_c, \qquad \mathbf{u}_c = \frac{r\mathbf{u} + \sum_{n:z_n=c} \mathbf{x}_n}{r + N_c}, \tag{E.5}$$

$$\text{and} \qquad \mathbf{S}_c = \mathbf{S} + \sum_{n:z_n=c} \mathbf{x}_n \mathbf{x}_n^{\mathsf{T}} + r\mathbf{u}\mathbf{u}^{\mathsf{T}} - r_c \mathbf{u}_c \mathbf{u}_c^{\mathsf{T}}, \tag{E.6}$$

are the posterior Gaussian-Wishart parameters of the $c^{\text{th}}$ component (Murphy, 2007).

To model the cluster assignments, we use a Dirichlet process (MacEachern and Müller, 1998) with concentration parameter $\eta$. Under a Dirichlet process prior, the probability of observing a particular cluster assignment $\mathbf{z}$ depends only on the partition induced, and is given by the Chinese restaurant process:

$$p(\mathbf{z}|\eta) = \frac{\Gamma(\eta)\eta^C}{\Gamma(\eta + N)} \prod_{c=1}^{C} \Gamma(N_c) \tag{E.7}$$

where $C$ is the number of components for which $N_c > 0$, and $N$ is the total number of datapoints.

The joint distribution of observed coordinates, latent coordinates, and cluster assignments is given by

$$p(\mathbf{Y}, \mathbf{X}, \mathbf{z}|\boldsymbol{\theta}, \boldsymbol{S}, \nu, \mathbf{u}, r, \eta) = p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta}) p(\mathbf{X}|\mathbf{z}, \boldsymbol{S}, \nu, \mathbf{u}, r) p(\mathbf{z}|\eta), \tag{E.8}$$

where the factors in the right hand side can be calculated by equations (7.5), (E.4) and (E.7), respectively.

### Details of inference

After analytically integrating out the parameters of the Gaussian mixture components, the only remaining variables to infer are the latent points $\mathbf{X}$, the cluster assignments

$\mathbf{z}$, and the kernel parameters $\boldsymbol{\theta}$. We'll estimate the posterior over these parameters using Markov chain Monte Carlo. In particular, we'll alternate between collapsed Gibbs sampling of each row of $\mathbf{z}$, and Hamiltonian Monte Carlo sampling of $\mathbf{X}$ and $\boldsymbol{\theta}$.

First, we explain collapsed Gibbs sampling for the cluster assignments $\mathbf{z}$. Given a sample of $\mathbf{X}$, $p(\mathbf{z}|\mathbf{X}, \mathbf{S}, \nu, \mathbf{u}, r, \eta)$ does not depend on $\mathbf{Y}$. This lets us resample cluster assignments, integrating out the iGMM likelihood in closed form. Given the current state of all but one latent component $z_n$, a new value for $z_n$ is sampled with the following probability:

$$p(z_n = c|\mathbf{X}, \mathbf{z}_{\backslash n}, \boldsymbol{S}, \nu, \mathbf{u}, r, \eta) \propto \begin{cases} N_{c\backslash n} \cdot p(\mathbf{x}_n|\mathbf{X}_{c\backslash n}, \boldsymbol{S}, \nu, \mathbf{u}, r) & \text{existing components} \\ \eta \cdot p(\mathbf{x}_n|\boldsymbol{S}, \nu, \mathbf{u}, r) & \text{a new component} \end{cases} \tag{E.9}$$

where $\mathbf{X}_c = \{\mathbf{x}_n|z_n = c\}$ is the set of latent coordinates assigned to the $c^{\text{th}}$ component, and $\backslash n$ represents the value or set when excluding the $n^{\text{th}}$ data point. We can analytically calculate $p(\mathbf{x}_n|\mathbf{X}_{c\backslash n}, \boldsymbol{S}, \nu, \mathbf{u}, r)$ as follows:

$$p(\mathbf{x}_n|\mathbf{X}_{c\backslash n}, \boldsymbol{S}, \nu, \mathbf{u}, r) = \pi^{-\frac{N_{c\backslash n}Q}{2}} \frac{r_{c\backslash n}^{Q/2} \left|\mathbf{S}_{c\backslash n}\right|^{\nu_{c\backslash n}/2}}{r_{c\backslash n}'^{Q/2} \left|\mathbf{S}'_{c\backslash n}\right|^{\nu'_{c\backslash n}/2}} \times \prod_{d=1}^{Q} \frac{\Gamma\left(\frac{\nu'_{c\backslash n}+1-d}{2}\right)}{\Gamma\left(\frac{\nu_{c\backslash n}+1-d}{2}\right)}, \tag{E.10}$$

where $r'_c$, $\nu'_c$, $\mathbf{u}'_c$ and $\mathbf{S}'_c$ represent the posterior on Gaussian-Wishart parameters of the $c^{\text{th}}$ component when the $n^{\text{th}}$ data point has been assigned to it. We can efficiently calculate the determinant $\left|\mathbf{S}'_{c\backslash n}\right|$ using the rank-one Cholesky update. A special case of equation (E.10) gives the likelihood for a new component, $p(\mathbf{x}_n|\boldsymbol{S}, \nu, \mathbf{u}, r)$.

### Gradients for Hamiltonian Monte Carlo

Hamiltonian Monte Carlo (HMC) sampling of $\mathbf{X}$ from posterior $p(\mathbf{X}|\mathbf{z}, \mathbf{Y}, \boldsymbol{\theta}, \boldsymbol{S}, \nu, \mathbf{u}, r)$, requires computing the gradient of the log-unnormalized-posterior with respect to $\mathbf{X}$:

$$\frac{\partial}{\partial \mathbf{X}}\Big[\log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta}) + \log p(\mathbf{X}|\mathbf{z}, \boldsymbol{S}, \nu, \mathbf{u}, r)\Big] \tag{E.11}$$

The first term of gradient (E.11) can be calculated by

$$\frac{\partial \log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta})}{\partial \mathbf{X}} = \frac{\partial \log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta})}{\partial \mathbf{K}} \frac{\partial \mathbf{K}}{\partial \mathbf{X}} = \left[-\frac{1}{2} D\mathbf{K}^{-1} + \frac{1}{2}\mathbf{K}^{-1}\mathbf{Y}\mathbf{Y}^T\mathbf{K}^{-1}\right]\left[\frac{\partial \mathbf{K}}{\partial \mathbf{X}}\right], \tag{E.12}$$

where for an SE + WN kernel with the same lengthscale $\ell$ on all dimensions,

$$\frac{\partial k(\mathbf{x}_n, \mathbf{x}_m)}{\partial \mathbf{x}_n} = -\frac{\sigma_f^2}{\ell^2} \exp\left(-\frac{1}{2\ell^2}(\mathbf{x}_n - \mathbf{x}_m)^\mathsf{T}(\mathbf{x}_n - \mathbf{x}_m)\right)(\mathbf{x}_n - \mathbf{x}_m). \tag{E.13}$$

The second term of (E.11) is given by

$$\frac{\partial \log p(\mathbf{X}|\mathbf{z}, \boldsymbol{S}, \nu, \mathbf{u}, r)}{\partial \mathbf{x}_n} = -\nu_{z_n} \boldsymbol{S}_{z_n}^{-1}(\mathbf{x}_n - \mathbf{u}_{z_n}). \tag{E.14}$$

We also infer kernel parameters $\boldsymbol{\theta}$ via HMC, using the gradient of the log unnormalized posterior with respect to the kernel parameters, using an improper uniform prior.

### Posterior predictive density

In the GP-LVM, the predictive density of at test point $\mathbf{y}_\star$ is usually computed by finding the point $\mathbf{x}_\star$ which has the highest probability of being mapped to $\mathbf{y}_\star$, then using the density of $p(\mathbf{x}_\star)$ and the Jacobian of the warping at that point to approximate the density at $\mathbf{y}_\star$. When inference is done this way, approximating the predictive density only requires solving a single optimization for each $\mathbf{y}_\star$.

For our model, we use approximate integration to estimate $p(\mathbf{y}_\star)$. This is done for two reasons: First, multiple latent points (possibly from different clusters) can map to the same observed point, meaning the standard method can underestimate $p(\mathbf{y}_\star)$. Second, because we do not optimize the latent coordinates of training points, but instead sample them, we would need to optimize each $p(\mathbf{x}_\star)$ seperately for each sample in the Markov chain. One advantage of our method is that it gives estimates for all $p(\mathbf{y}_\star)$ at once. However, it may not be as accurate in very high observed dimensions, when the volume to sample over is relatively large.

The posterior density in the observed space given the training data is

$$\begin{aligned} p(\mathbf{y}_\star|\mathbf{Y}) &= \iint p(\mathbf{y}_\star, \mathbf{x}_\star, \mathbf{X}|\mathbf{Y}) d\mathbf{x}_\star d\mathbf{X} \\ &= \iint p(\mathbf{y}_\star|\mathbf{x}_\star, \mathbf{X}, \mathbf{Y}) p(\mathbf{x}_\star|\mathbf{X}, \mathbf{Y}) p(\mathbf{X}|\mathbf{Y}) d\mathbf{x}_\star d\mathbf{X}. \end{aligned} \tag{E.15}$$

We first approximate $p(\mathbf{X}|\mathbf{Y})$ using samples from the Gibbs and Hamiltonian Monte Carlo chain. We then approximate $p(\mathbf{x}_\star|\mathbf{X}, \mathbf{Y})$ by sampling points from the posterior density in the latent space and warping them, using the following procedure:

1. Draw a latent cluster assignment $z_\star \sim \text{Mult}\left(\frac{N_1}{N+\eta}, \frac{N_2}{N+\eta}, \cdots, \frac{N_C}{N+\eta}, \frac{\eta}{N+\eta}\right)$

2. Draw a latent cluster precision matrix $\mathbf{R}_\star \sim \mathcal{W}(\mathbf{S}_{z_\star}^{-1}, \nu_{z_\star})$
3. Draw a latent cluster mean $\boldsymbol{\mu}_\star \sim \mathcal{N}(\mathbf{u}_{z_\star}, (r_{z_\star}\mathbf{R}_\star)^{-1})$
4. Draw latent coordinates $\mathbf{x}_\star \sim \mathcal{N}(\boldsymbol{\mu}_\star, \mathbf{R}_\star^{-1})$
5. For each observed dimension $d = 1, 2, \ldots, D$,
   draw observed coordinates $\mathbf{y}_d^\star \sim \mathcal{N}(\mathbf{k}_\star^\mathsf{T}\mathbf{K}^{-1}\mathbf{Y}_{:,d}, k(\mathbf{x}_\star, \mathbf{x}_\star) - \mathbf{k}_\star^\mathsf{T}\mathbf{K}^{-1}\mathbf{k}_\star)$

If $z_\star$ is assigned to a new component in step 1, the prior Gaussian-Wishart distribution (E.2) is used for sampling in steps 2 and 3. The density drawn from in step 5 is the predictive distribution of a GP, where $\mathbf{k}_\star = (k(\mathbf{x}_\star, \mathbf{x}_1), \cdots, k(\mathbf{x}_\star, \mathbf{x}_N))^\mathsf{T}$ and $\mathbf{Y}_{:,d}$ represents the $d$th column of $\mathbf{Y}$.

Each step of this sampling procedure draws from the exact conditional distribution, so the Monte Carlo estimate of the conditional predictive density $p(\mathbf{y}_\star|\mathbf{X}, \mathbf{Y})$ will converge to the true marginal distribution as the number of samples increases. Since the observations $\mathbf{y}_\star$ are conditionally normally distributed, each one adds a smooth contribution to the empirical Monte Carlo estimate of the posterior density, as opposed to a collection of point masses.

### Source code

A reference implementation of the above algorithms is available at
`http://www.github.com/duvenaud/warped-mixtures`.

# References


Ryan P. Adams and Zoubin Ghahramani. Archipelago: Nonparametric Bayesian semi-supervised learning. In *Proceedings of the 26th International Conference on Machine Learning*, pages 1–8. ACM, 2009. (pages 102 and 106)

Ryan P. Adams and David J.C. MacKay. Bayesian online changepoint detection. *arXiv preprint arXiv:0710.3742*, 2007. (page 41)

Ryan P. Adams, Hanna M. Wallach, and Zoubin Ghahramani. Learning the structure of deep sparse graphical models. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2010. (pages 77 and 113)

Mark A. Armstrong, Gérard Iooss, and Daniel D. Joseph. *Groups and symmetry*. Springer, 1988. (page 22)

Francis R. Bach. High-dimensional non-linear variable selection through hierarchical kernel learning. *arXiv preprint arXiv:0909.0844*, 2009. (pages 88, 92, and 93)

Francis R. Bach, Gert R.G. Lanckriet, and Michael I. Jordan. Multiple kernel learning, conic duality, and the SMO algorithm. In *Proceedings of the 21st International Conference on Machine learning*, 2004. (pages 40 and 43)

Kevin Bache and Moshe Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`. (page 91)

Pierre Baldi and Peter J. Sadowski. Understanding dropout. In *Advances in Neural Information Processing Systems*, pages 2814–2822, 2013. (page 85)

Andrei Barbu, Alexander Bridge, Zachary Burchill, Dan Coroian, Sven Dickinson, Sanja Fidler, Aaron Michaux, Sam Mussman, Siddharth Narayanaswamy, Dhaval Salvi, Lara Schmidt, Jiangnan Shangguan, Jeffrey M. Siskind, Jarrell Waggoner, Song Wang, Jinlian Wei, Yifan Yin, and Zhiqi Zhang. Video in sentences out. In *Conference on Uncertainty in Artificial Intelligence*, 2012. (page 55)

Richard Bellman. Dynamic programming and Lagrange multipliers. *Proceedings of the National Academy of Sciences of the United States of America*, 42(10):767, 1956. (pages 13 and 85)

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166, 1994. (page 78)

Yoshua Bengio, Olivier Delalleau, and Nicolas Le Roux. The curse of highly variable functions for local kernel machines. *Advances in Neural Information Processing Systems*, 18:107–114, 2006. ISSN 1049-5258. (pages 74 and 85)

Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized denoising auto-encoders as generative models. In *Advances in Neural Information Processing Systems*, pages 899–907, 2013. (page 113)

Wu Bing, Zhang Wen-qiong, Chen Ling, and Liang Jia-hong. A GP-based kernel construction and optimization method for RVM. In *International Conference on Computer and Automation Engineering (ICCAE)*, volume 4, pages 419–423, 2010. (page 42)

Salomon Bochner. *Lectures on Fourier integrals*, volume 42. Princeton University Press, 1959. (pages 21 and 41)

George E.P. Box, Gwilym M. Jenkins, and Gregory C. Reinsel. *Time series analysis: forecasting and control*. John Wiley & Sons, 1970. (pages 38 and 43)

Wenbo Cao and Robert Haralick. Nonlinear manifold clustering by dimensionality. In *International Conference on Pattern Recognition (ICPR)*, volume 1, pages 920–924. IEEE, 2006. (page 100)

Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27:1–27:27, 2011. (page 106)

Youngmin Cho and Lawrence K. Saul. Kernel methods for deep learning. In *Advances in Neural Information Processing Systems*, pages 342–350, 2009. (pages 74 and 78)

Corinna Cortes and Vladimir N. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995. (page 16)

Andreas Damianou and Neil D. Lawrence. Deep Gaussian processes. In *Artificial Intelligence and Statistics*, pages 207–215, 2013. (pages 58, 76, 101, 112, and 113)

Eyal Dechter, Jon Malmaud, Ryan P. Adams, and Joshua B. Tenenbaum. Bootstrap learning via modular concept discovery. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 1302–1309. AAAI Press, 2013. (page 42)

Laura Diosan, Alexandrina Rogozan, and Jean-Pierre Pecuchet. Evolving kernel functions for SVMs by genetic programming. In *Machine Learning and Applications, 2007*, pages 19–24. IEEE, 2007. (page 42)

Nicolas Durrande, David Ginsbourger, and Olivier Roustant. Additive kernels for Gaussian process modeling. *arXiv preprint arXiv:1103.4023*, 2011. (page 88)

Nicolas Durrande, James Hensman, Magnus Rattray, and Neil D. Lawrence. Gaussian process models for periodicity detection. *arXiv preprint arXiv:1303.7090*, 2013. (page 55)

David Duvenaud, Hannes Nickisch, and Carl E. Rasmussen. Additive Gaussian processes. In *Advances in Neural Information Processing Systems 24*, pages 226–234, Granada, Spain, 2011. (page 80)

David Duvenaud, James Robert Lloyd, Roger B. Grosse, Joshua B. Tenenbaum, and Zoubin Ghahramani. Structure discovery in nonparametric regression through compositional kernel search. In *Proceedings of the 30th International Conference on Machine Learning*, 2013. (page 30)

David Duvenaud, Oren Rippel, Ryan P. Adams, and Zoubin Ghahramani. Avoiding pathologies in very deep networks. In *17th International Conference on Artificial Intelligence and Statistics*, Reykjavik, Iceland, April 2014. (page 59)

Daniel Eaton and Kevin Murphy. Bayesian structure learning using dynamic programming and MCMC. In *Conference on Uncertainty in Artificial Intelligence*, 2007. (page 31)

Ehsan Elhamifar and René Vidal. Sparse manifold clustering and embedding. In *Advances in Neural Information Processing Systems*, pages 55–63, 2011. (page 100)

Emily B. Fox and David B. Dunson. Multiresolution Gaussian processes. In *Advances in Neural Information Processing Systems 25*. MIT Press, 2013. (page 43)

Nir Friedman and Daphne Koller. Being Bayesian about network structure: A Bayesian approach to structure discovery in Bayesian networks. *Machine Learning*, 50:95–126, 2003. (page 31)

M. Ganesalingam and Timonthy W. Gowers. A fully automatic problem solver with human-style output. *arXiv preprint arXiv:1309.4501*, 2013. (page 55)

Roman Garnett, Michael A. Osborne, Steven Reece, Alex Rogers, and Stephen J. Roberts. Sequential Bayesian prediction in the presence of changepoints and faults. *The Computer Journal*, 53(9):1430–1446, 2010. (pages 33, 41, and 43)

Andreas Geiger, Raquel Urtasun, and Trevor Darrell. Rank priors for continuous nonlinear dimensionality reduction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 880–887. IEEE, 2009. (page 100)

Andrew Gelman. Why waste time philosophizing?, 2013. URL `http://andrewgelman.com/2013/02/11/why-waste-time-philosophizing/`. (page 31)

Andrew Gelman and Cosma R. Shalizi. Philosophy and the practice of Bayesian statistics. *British Journal of Mathematical and Statistical Psychology*, 2012. (page 31)

Robert Gens and Pedro Domingos. Learning the structure of sum-product networks. In *Proceedings of the 30th International Conference on Machine learning*, 2013. (page 79)

Zoubin Ghahramani and M.J. Beal. Variational inference for Bayesian mixtures of factor analysers. *Advances in Neural Information Processing Systems*, 12:449–455, 2000. (page 100)

Elad Gilboa, Yunus Saatçi, and John Cunningham. Scaling multidimensional inference for structured Gaussian processes. In *Proceedings of the 30th International Conference on Machine Learning*, 2013. (page 88)

David Ginsbourger, Xavier Bay, Olivier Roustant, and Laurent Carraro. Argumentwise invariant kernels for the approximation of invariant functions. In *Annales de la Faculté de Sciences de Toulouse*, 2012. (page 23)

David Ginsbourger, Olivier Roustant, and Nicolas Durrande. Invariances of random fields paths, with applications in Gaussian process regression. *arXiv preprint arXiv:1308.1359 [math.ST]*, August 2013. (pages 23 and 24)

Mehmet Gönen and Ethem Alpaydın. Multiple kernel learning algorithms. *Journal of Machine Learning Research*, 12:2211–2268, 2011. (pages 33, 40, and 43)

Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, K. Bonawitz, and Joshua B. Tenenbaum. Church: A language for generative models. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, pages 220–229, 2008. (page 112)

Noah D. Goodman, Tomer D. Ullman, and Joshua B. Tenenbaum. Learning a theory of causality. *Psychological review*, 118(1):110, 2011. (page 13)

Daniel B Graham and Nigel M Allinson. Characterizing virtual eigensignatures for general purpose face recognition. *Face Recognition: From Theory to Applications*, 163:446–456, 1998. (page 103)

Roger B. Grosse. *Model Selection in Compositional Spaces*. PhD thesis, Massachusetts Institute of Technology, 2014. (page 113)

Roger B. Grosse, Ruslan Salakhutdinov, William T. Freeman, and Joshua B. Tenenbaum. Exploiting compositionality to explore a large space of model structures. In *Uncertainty in Artificial Intelligence*, 2012. (pages 31 and 42)

Chong Gu. *Smoothing spline ANOVA models*. Springer Verlag, 2002. ISBN 0387953531. (page 40)

Trevor J. Hastie and Robert J. Tibshirani. *Generalized additive models*. Chapman & Hall/CRC, 1990. (pages 15, 40, and 81)

James Hensman, Nicolo Fusi, and Neil D. Lawrence. Gaussian processes for big data. *arXiv preprint arXiv:1309.6835*, 2013. (page 5)

James Hensman, Andreas Damianou, and Neil D. Lawrence. Deep Gaussian processes for large datasets. In *Artificial Intelligence and Statistics Late-breaking Posters*, 2014a. (page 58)

James Hensman, Nicolo Fusi, Ricardo Andrade, Nicolas Durrande, Alan Saul, Max Zwiessele, and Neil D. Lawrence. GPy: A Gaussian process framework in python, 2014b. `https://github.com/SheffieldML/GPy`. (page 5)

Michiel Hermans and Benjamin Schrauwen. Recurrent kernel machines: Computing with infinite echo state networks. *Neural Computation*, 24(1):104–133, 2012. (page 78)

Geoffrey Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. (page 85)

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. (page 78)

Marcus Hutter. The fastest and shortest algorithm for all well-defined problems. *International Journal of Foundations of Computer Science*, 13(03):431–443, 2002. (page 112)

Rob J. Hyndman. Time series data library, accessed July 2013. URL `http://data.is/TSDLdemo`. (page 43)

Tomoharu Iwata, David Duvenaud, and Zoubin Ghahramani. Warped mixtures for nonparametric cluster shapes. In *29th Conference on Uncertainty in Artificial Intelligence*, Bellevue, Washington, July 2013. (page 95)

Edwin T. Jaynes. Highly informative priors. In *Proceedings of the Second International Meeting on Bayesian Statistics*, 1985. (page 30)

Cari G. Kaufman and Stephan R. Sain. Bayesian functional ANOVA modeling using Gaussian process prior distributions. *Bayesian Analysis*, 5(1):123–150, 2010. (page 88)

Charles Kemp and Joshua B. Tenenbaum. The discovery of structural form. *Proceedings of the National Academy of Sciences*, 105(31):10687–10692, 2008. (page 42)

Edward D. Klenske, Melanie N. Zeilinger, Bernhard Schölkopf, and Philipp Hennig. Nonparametric dynamics estimation for time periodic systems. In *51st Annual Allerton Conference on Communication, Control, and Computing*, pages 486–493, Oct 2013. (page 38)

Imre Risi Kondor. *Group theoretical methods in machine learning*. PhD thesis, Columbia University, 2008. (pages 23 and 26)

Gabriel Kronberger and Michael Kommenda. Evolution of covariance functions for Gaussian process regression using genetic programming. *arXiv preprint arXiv:1305.3794*, 2013. (page 42)

Neil D. Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. *Advances in Neural Information Processing Systems*, pages 329–336, 2004. (pages 96 and 100)

Neil D. Lawrence. Probabilistic non-linear principal component analysis with Gaussian process latent variable models. *Journal of Machine Learning Research*, 6:1783–1816, 2005. (page 27)

Neil D. Lawrence. Local distance preservation in the GP-LVM through back constraints. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 513–520, 2006. (page 107)

Neil D. Lawrence and Andrew J. Moore. Hierarchical Gaussian process latent variable models. In *Proceedings of the 24th International Conference on Machine learning*, pages 481–488, 2007. (page 76)

Neil D. Lawrence and Raquel Urtasun. Non-linear matrix factorization with Gaussian processes. In *Proceedings of the 26th International Conference on Machine Learning*, pages 601–608, 2009. (page 100)

Miguel Lázaro-Gredilla, Joaquin Quiñonero-Candela, Carl E. Rasmussen, and Aníbal R. Figueiras-Vidal. Sparse spectrum Gaussian process regression. *Journal of Machine Learning Research*, 99:1865–1881, 2010. (pages 33 and 41)

Judith Lean, Juerg Beer, and Raymond Bradley. Reconstruction of solar irradiance since 1610: Implications for climate change. *Geophysical Research Letters*, 22(23): 3195–3198, 1995. (pages 53 and 119)

Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361, 1995. (page 25)

Honglak Lee, Chaitanya Ekanadham, and Andrew Ng. Sparse deep belief net model for visual area v2. In *Advances in Neural Information Processing Systems*, pages 873–880, 2007. (page 79)

Doug Lerner and Dan Asimov. The Sudanese Möbius band. In *SIGGRAPH Electronic Theatre*, 1984. (page 28)

Leonid A. Levin. Universal sequential search problems. *Problemy Peredachi Informatsii*, 9(3):115–116, 1973. (page 112)

Ming Li and Paul M.B. Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer, 1997. (page 112)

Percy Liang, Michael I. Jordan, and Dan Klein. Learning programs: A hierarchical Bayesian approach. In *Proceedings of the 27th International Conference on Machine Learning*, pages 639–646, 2010. (pages 42 and 112)

Douglas A. Lind, William G. Marchal, and Samuel Adam Wathen. *Basic statistics for business and economics*. McGraw-Hill/Irwin Boston, 2006. (pages 33 and 43)

James Robert Lloyd. GEFCom2012 hierarchical load forecasting: Gradient boosting machines and Gaussian processes. *International Journal of Forecasting*, 2013. (page 38)

James Robert Lloyd, David Duvenaud, Roger B. Grosse, Joshua B. Tenenbaum, and Zoubin Ghahramani. Automatic construction and natural-language description of nonparametric regression models. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2014. (pages 30, 41, 47, and 54)

Ian G. Macdonald. *Symmetric functions and Hall polynomials*. Oxford University Press, USA, 1998. ISBN 0198504500. (page 83)

Steven N. MacEachern and Peter Müller. Estimating mixture of Dirichlet process models. *Journal of Computational and Graphical Statistics*, pages 223–238, 1998. (page 128)

David J.C. MacKay. *Bayesian methods for adaptive models*. PhD thesis, California Institute of Technology, 1992. (page 3)

David J.C. MacKay. Introduction to Gaussian processes. *NATO ASI Series F Computer and Systems Sciences*, 168:133–166, 1998. (pages 25 and 116)

David J.C. MacKay. *Information theory, inference, and learning algorithms*. Cambridge University press, 2003. (pages 3 and 99)

Vikash Mansinghka, Daniel Selsam, and Yura Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *arXiv preprint arXiv:1404.0099*, 2014. (page 112)

James Martens. Deep learning via Hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning*, pages 735–742, 2010. (page 79)

John H. R. Maunsell and David C. van Essen. The connections of the middle temporal visual area (MT) and their relationship to a cortical hierarchy in the macaque monkey. *Journal of neuroscience*, 3(12):2563–2586, 1983. (page 71)

James Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, pages 415–446, 1909. (page 21)

Charles A. Micchelli, Yuesheng Xu, and Haizhang Zhang. Universal kernels. *Journal of Machine Learning Research*, 7:2651–2667, 2006. (page 12)

Jeffrey W. Miller and Matthew T. Harrison. A simple example of Dirichlet process mixture inconsistency for the number of components. In *Advances in Neural Information Processing Systems 26*, pages 199–206. Curran Associates, Inc., 2013. (page 108)

Ha Quang Minh, Partha Niyogi, and Yuan Yao. Mercer's theorem, feature maps, and smoothing. In *Learning theory*, pages 154–168. Springer, 2006. (page 21)

Thomas P. Minka. Expectation propagation for approximate Bayesian inference. In *Uncertainty in Artificial Intelligence*, volume 17, pages 362–369, 2001. (page 91)

@ML_Hipster. "...essentially, all models are wrong, but yours are stupid too." – G.E.P. Box in a less than magnanimous mood., 2013. URL `https://twitter.com/ML_Hipster/status/394577463990181888`. (page 1)

Grégoire Montavon, Mikio L. Braun, and Klaus-Robert Müller. Layer-wise analysis of deep networks with Gaussian kernels. *Advances in Neural Information Processing Systems*, 23:1678–1686, 2010. (page 78)

Kevin P. Murphy. Conjugate Bayesian analysis of the Gaussian distribution. Technical report, Computer Science Department, University of British Columbia, 2007. (page 128)

Radford M. Neal. *Bayesian learning for neural networks*. PhD thesis, University of Toronto, 1995. (pages 16, 60, 68, 76, and 78)

Radford M. Neal. Density modeling and clustering using Dirichlet diffusion trees. *Bayesian Statistics*, 7:619–629, 2003. (page 108)

John Ashworth Nelder and Robert W.M. Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society. Series A (General)*, 135(3):370–384, 1972. (page 81)

Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems*, 2:849–856, 2002. (page 100)

Hannes Nickisch and Carl E. Rasmussen. Gaussian mixture modeling with Gaussian process latent variable models. *Pattern Recognition*, pages 272–282, 2010. (page 100)

Jorge Nocedal. Updating quasi-Newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980. (page 91)

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *arXiv preprint arXiv:1211.5063*, 2012. (page 78)

Tony A. Plate. Accuracy versus interpretability in flexible modeling: Implementing a tradeoff using Gaussian process models. *Behaviormetrika*, 26:29–50, 1999. ISSN 0385-7417. (pages 33, 40, and 88)

Daniel Preotiuc-Pietro and Trevor Cohn. A temporal model of text periodicities using Gaussian processes. In *Conference on Empirical Methods on Natural Language Processing*, pages 977–988. ACL, 2013. (page 38)

Joaquin Quiñonero-Candela and Carl E. Rasmussen. A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6: 1939–1959, 2005. (pages 5, 99, and 108)

Herschel Rabitz and Ömer F. Aliş. General foundations of high-dimensional model representations. *Journal of Mathematical Chemistry*, 25(2-3):197–233, 1999. (page 90)

William M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, pages 846–850, 1971. (page 106)

Carl E. Rasmussen. The infinite Gaussian mixture model. *Advances in Neural Information Processing Systems*, 2000. (page 98)

Carl E. Rasmussen and Zoubin Ghahramani. Occam's razor. *Advances in Neural Information Processing Systems*, pages 294–300, 2001. (pages 3 and 35)

Carl E. Rasmussen and Hannes Nickisch. Gaussian processes for machine learning (GPML) toolbox. *Journal of Machine Learning Research*, 11:3011–3015, December 2010. (pages 5 and 116)

Carl E. Rasmussen and Christopher K.I. Williams. *Gaussian Processes for Machine Learning*, volume 38. The MIT Press, Cambridge, MA, USA, 2006. (pages 1, 5, and 38)

Miles A. Reid and Balázs Szendrői. *Geometry and topology*. Cambridge University Press, 2005. (page 27)

Salah Rifai, Grégoire Mesnil, Pascal Vincent, Xavier Muller, Yoshua Bengio, Yann Dauphin, and Xavier Glorot. Higher order contractive auto-encoder. In *Machine Learning and Knowledge Discovery in Databases*, pages 645–660. Springer, 2011a. (page 66)

Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning*, pages 833–840, 2011b. (page 67)

Oren Rippel and Ryan P. Adams. High-dimensional probability estimation with deep density models. *arXiv preprint arXiv:1302.5125*, 2013. (pages 77 and 113)

Carlos E. Rodríguez and Stephen G. Walker. Univariate Bayesian nonparametric mixture modeling with unimodal kernels. *Statistics and Computing*, pages 1–15, 2012. (page 101)

Frank Rosenblatt. Principles of neurodynamics: Perceptrons and the theory of brain mechanisms. *Brain Mechanisms*, pages 555–559, 1962. (page 59)

David Ruppert, Matthew P. Wand, and Raymond J. Carroll. *Semiparametric regression*, volume 12. Cambridge University Press, 2003. (pages 33 and 40)

Yunus Saatçi, Ryan D. Turner, and Carl E. Rasmussen. Gaussian process change point models. In *Proceedings of the 27th International Conference on Machine Learning*, pages 927–934, 2010. (pages 41 and 43)

Ruslan Salakhutdinov and Geoffrey Hinton. Using deep belief nets to learn covariance kernels for Gaussian processes. *Advances in Neural information processing systems*, 20:1249–1256, 2008. (page 41)

Ruslan Salakhutdinov and Geoffrey E. Hinton. Deep boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*, pages 448–455, 2009. (page 113)

Mathieu Salzmann, Raquel Urtasun, and Pascal Fua. Local deformation models for monocular 3D shape recovery. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2008. (page 100)

Andrew Saxe, Pang W. Koh, Zhenghao Chen, Maneesh Bhand, Bipin Suresh, and Andrew Y. Ng. On random weights and unsupervised feature learning. In *Proceedings of the 28th International Conference on Machine Learning*, pages 1089–1096, 2011. (page 78)

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Dynamics of learning in deep linear neural networks. In *NIPS Workshop on Deep Learning*, 2013. (page 78)

Jürgen Schmidhuber. The speed prior: a new simplicity measure yielding near-optimal computable predictions. In *Computational Learning Theory*, pages 216–228. Springer, 2002. (page 112)

Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009. ISSN 1095-9203. doi: 10.1126/science.1165893. (page 41)

Michael Schmidt and Hod Lipson. Eureqa [software], Accessed February 2013. URL `http://www.eureqa.com`. (pages 41 and 43)

Gideon Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2): 461–464, 1978. (page 36)

Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. *Advances in Neural Information Processing Systems*, 2006. (pages 5, 99, and 108)

Edward Snelson, Carl E. Rasmussen, and Zoubin Ghahramani. Warped Gaussian processes. *Advances in Neural Information Processing Systems*, pages 337–344, 2004. (page 56)

Ercan Solak, Roderick Murray-Smith, William E. Leithead, Douglas J. Leith, and Carl E. Rasmussen. Derivative observations in Gaussian process models of dynamic systems. In *Advances in Neural Information Processing Systems*, 2003. (page 65)

Ray J. Solomonoff. A formal theory of inductive inference. Part I. *Information and control*, 7(1):1–22, 1964. (page 112)

Nitish Srivastava. Improving neural networks with dropout. Master's thesis, University of Toronto, 2013. (page 85)

Christian Steinruecken. *Lossless Data Compression*. PhD thesis, Cavendish Laboratory, University of Cambridge, 2014. (pages 42 and 113)

Mark O. Stitson, Alex Gammerman, Vladimir Vapnik, Volodya Vovk, Chris Watkins, and Jason Weston. Support vector regression with ANOVA decomposition kernels. *Advances in kernel methods: Support vector learning*, pages 285–292, 1999. (page 90)

Yee Whye Teh, Michael I. Jordan, Matthew J. Beal, and David M. Blei. Hierarchical Dirichlet processes. *Journal of the American Statistical Association*, 101(476):1566–1581, 2006. (page 108)

Michael E. Tipping and Christopher M. Bishop. Mixtures of probabilistic principal component analyzers. *Neural computation*, 11(2):443–482, 1999. (page 100)

Michalis Titsias and Neil D. Lawrence. Bayesian Gaussian process latent variable model. *International Conference on Artificial Intelligence and Statistics*, 2010. (page 100)

Ljupčo Todorovski and Sašo Džeroski. Declarative bias in equation discovery. In *Proceedings of the International Conference on Machine Learning*, pages 376–384, 1997. (page 41)

Jarno Vanhatalo, Jaakko Riihimäki, Jouni Hartikainen, Pasi Jylänki, Ville Tolvanen, and Aki Vehtari. GPstuff: Bayesian modeling with Gaussian processes. *Journal of Machine Learning Research*, 14(1):1175–1179, 2013. (page 5)

Vladimir N. Vapnik. *Statistical learning theory*, volume 2. Wiley New York, 1998. (page 90)

Grace Wahba. *Spline models for observational data*. Society for Industrial Mathematics, 1990. ISBN 0898712440. (pages 40 and 90)

Chunyi Wang and Radford M. Neal. Gaussian process regression with heteroscedastic or non-gaussian residuals. *arXiv preprint arXiv:1212.6246*, 2012. (page 77)

Jingdong Wang, Jianguo Lee, and Changshui Zhang. Kernel trick embedded Gaussian mixture model. In *Algorithmic Learning Theory*, pages 159–174. Springer, 2003. (page 101)

Sida Wang and Christopher Manning. Fast dropout training. In *Proceedings of the 30th International Conference on Machine Learning*, pages 118–126, 2013. (page 86)

Takashi Washio, Hiroshi Motoda, and Yuji Niwa. Discovering admissible model equations from observed data based on scale-types and identity constraints. In *International Joint Conference On Artificial Intelligence*, volume 16, pages 772–779, 1999. (page 41)

Wikimedia Commons. Stereographic projection of a Sudanese Möbius band, 2005. URL http://commons.wikimedia.org/wiki/File:MobiusSnail2B.png. (page 28)

Andrew G. Wilson and Ryan P. Adams. Gaussian process kernels for pattern discovery and extrapolation. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1067–1075, 2013. (pages 33, 41, and 43)

Andrew G. Wilson, David A. Knowles, and Zoubin Ghahramani. Gaussian process regression networks. In *Proceedings of the 29th International Conference on Machine Learning*, pages 599–606, 2012. (page 77)

Frank Wood, Jan Willem van de Meent, and Vikash Mansinghka. A new approach to probabilistic programming inference. In *Proceedings of the 17th International conference on Artificial Intelligence and Statistics*, 2014. (page 112)

I-Cheng Yeh. Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete research*, 28(12):1797–1808, 1998. (page 16)

Yichuan Zhang and Charles A. Sutton. Quasi-Newton methods for Markov chain Monte Carlo. *Advances in Neural Information Processing Systems*, pages 2393–2401, 2011. (page 108)