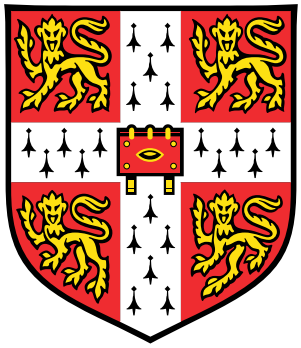
Automatic Model Construction with Gaussian Processes



David Kristjanson Duvenaud

Department of Engineering

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 65,000 words including appendices, bibliography, footnotes, tables and equations and has less than 150 figures.

David Kristjanson Duvenaud
June 2014

Acknowledgements

I'd like to thank my supervisor, Carl Edward Rasmussen, for his excellent advice and encouragement, as well as for giving me the freedom to make my own mistakes. It was wonderful having an advisor who had spent many years thinking deeply about the business of modeling. As he sometimes says, "The devil is in the details". I'd also like to thank my advisor, Zoubin Ghahramani, for providing much encouragement and support.

Thanks to Michael Osborne, whose thesis was an inspiration, and to Roman Garnett for making the research fun. Sinead Williamson and Peter Orbanz provided valuable advice in my early years in the lab. Philipp Hennig was a mentor to me during my time in Tübingen.

I'd like to thank Andrew McHutchon, Konstantina Palla, Alex Davies and Neil Houlsby for making the lab feel like a family. I'd like to thank James Lloyd for the many endless discussions that helped my understanding of a lot of things, and for keeping a level head even during the depths of deadline death-marches.

Christian Steinruecken constantly reminded me that amazing things are hidden all around.

Abstract

Gaussian processes are a broad class of models, useful for learning and forecasting in such domains as time series, geological formations, and physical dynamics. The types of generalization these models can perform depends on the kernel, whose choice typically requires trial-and-error by experts, limiting the power of Gaussian processes in practice. In this thesis, we show how to automate this task.

We first detail the many types of structure that can be encoded through kernels, giving recipes to produce models with a wide variety of properties. For example, we show how kernels can be combined to produce priors over manifolds having the topologies of cylinders, torii, and Möbius strips, as well as their higher-dimensional analogues.

The main contribution of this thesis is to show how this open-ended space of models can be explored automatically. We define a simple grammar over kernels, a search criterion (marginal likelihood), and a breadth-first search procedure. Combining these, we present a procedure which takes a dataset, and outputs a detailed report with graphs and automatically-generated text illustrating the qualitatively different, and potentially novel, types of structure discovered in that dataset. This system automates parts of the model-building and analysis currently performed by expert statisticians.

Kernel learning can also be seen as feature learning, a problem which has received intense interest in the last decade. In particular, the practical success of deep neural networks has demonstrated the potential of automatic feature extraction. Building on earlier work relating Gaussian processes and neural nets, we explore the natural extensions of these models to *deep kernels* and *deep Gaussian processes*.

We also explore several more extensions to Gaussian process models. First, we examine the model class consisting of the sum of functions of all possible combinations of input variables. We show a close connection between this model class and the recently-developed regularization method of *dropout*. Second, we combine Gaussian processes with the Dirichlet process to produce the *warped mixture model* - an unsupervised clustering model with nonparametric cluster shapes, and a corresponding interpretable, low-dimensional latent space.

Contents

List of Figures

List of Tables

Notation

Unbolded x represents a real number, $\mathbf{x}$ represents a vector, and $\mathbf{X}$ represents a matrix. The ith element of a vector $\mathbf{x}$ is denoted as x_i. A bold lower-case number with an index such as $\mathbf{x}_j$ represents a particular row of matrix $\mathbf{X}$.

Symbol	Description
$\mathbf{h}$	The implicit feature vector corresponding to a kernel.
$\mathcal{O}(\cdot)$	The big-O asymptotic complexity of an algorithm.
$A \otimes B$	The Kronecker product of matrices A and B.
$\boldsymbol{f}$	A function represented as an infinite-dimensional vector.
SE	Squared-exponential kernel, also known as the radial-basis function kernel, or Gaussian kernel.
RQ	Rational-quadratic kernel.
Per	Periodic kernel.
Lin	Linear kernel.
WN	White noise kernel.
C	constant kernel.
$k_1 + k_2$	Addition of kernels, shorthand for: $k_1(x, x') + k_2(x, x')$
$k_1 \times k_2$	Multiplication of kernels, shorthand for: $k_1(x, x') \times k_2(x, x')$

Chapter 1

Introduction

> "I only work on intractable nonparametrics - Gaussian processes don't count."
>
> Sinead Williamson, personal communication

> "All models are wrong, but yours are stupid too."
>
> **?**

Regression is usually stated as the problem of learning a function. From a human's point of view, the problem is closer to: Which method should I use? What model will work well on my problem? Is my current method silently failing?

Ideally, we would like an expressive language of models, which can represent both simple parametric forms of $f(\mathbf{x})$ such as linear or polynomial, and also complex non-parametric functions specified in terms of properties such as smoothness or periodicity. Fortunately, Gaussian processes (GPs) provide a very general and analytically tractable way of capturing many different classes of functions.

1.1 Gaussian Process Models

Gaussian processes are distributions over functions such that any finite subset of function evaluations $f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots f(\mathbf{x}_N)$, have a joint Gaussian distribution (**?**). A GP is completely specified by its mean function,

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

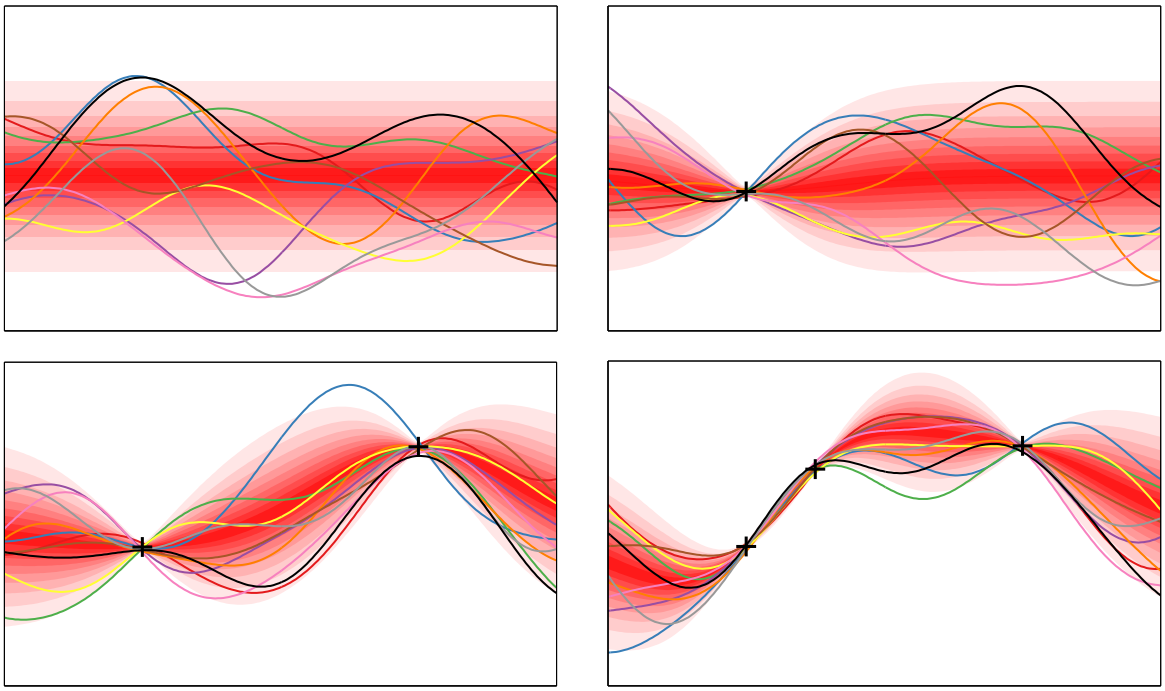


Fig. 1.1 A visual representation of a one-dimensional Gaussian process posterior. Different shades of red correspond to deciles of the marginal density at each input location. Coloured lines show samples from the process. Top left: A GP prior, not conditioned on any datapoints. Other plots show the posterior as more

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

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

It is common practice to assume zero mean, since marginalizing over an unknown mean function can be equivalently expressed as a zero-mean GP with a new kernel.

The kinds of structure which can be captured by a GP model is mainly determined by its kernel, which in turn determines how the model generalizes or extrapolates to new data. One of the main difficulties in using GPs lies in constructing a kernel which can represent the structure present in the data. There are many possible choices of covariance function. In fact, any symmetric positive-definite function can be used to construct a GP model. The benefit to this flexibility is that we can specify a wide range of models just by specifying the kernel.

The class of models that could be called Gaussian processes is extremely broad. Examples of commonly used models not usually cast as GPs are linear regression, splines,

some forms of generalized additive models, and Kalman filters.

Gaussian processes can be seen as a dual representation of Bayesian linear regression. (⋆) Cite By moving into this dual space, we pay a price in computational complexity, but gain the ability to model functions to any desired level of detail. Crucially, the marginal likelihood allows us to automatically discover the appropriate amount of detail to use, by Bayesian Occam's razor (**??**).

For concreteness, we give the marginal likelihood of the data under a GP:

$$\begin{aligned} p(\mathbf{y}|\mathbf{X}, \mu(\cdot), k(\cdot,\cdot)) &= \mathcal{N}(\mathbf{y}|\mu(\mathbf{X}), k(\mathbf{X},\mathbf{X})) \\ &= (2\pi)^{-\frac{N}{2}} |k(\mathbf{X},\mathbf{X})|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\left(\mathbf{y}-\boldsymbol{\mu}(\mathbf{X})\right)^{\mathsf{T}} k(\mathbf{X},\mathbf{X})^{-1}\left(\mathbf{y}-\boldsymbol{\mu}(\mathbf{X})\right)\right\} \end{aligned} \tag{1.3}$$

The Gaussian likelihood is referred to as the *marginal* likelihood, since it implicitly integrates over all possible functions $\mathbf{y}$.

Since the marginal likelihood depends on the kernel, we can select the form of the covariance function, or its parameters, by maximum likelihood, or through inference in a Bayesian model.

The predictive distributions also have a simple form:

$$\begin{aligned} p(y(\mathbf{x}^*)|\mathbf{y}, \mathbf{X}, \mu(\mathbf{x}), k(\mathbf{x},\mathbf{x})) = \mathcal{N}\Big(y(\mathbf{x}^*)|\mu(\mathbf{x}^*) + k(\mathbf{x}^*,\mathbf{X})k(\mathbf{X},\mathbf{X})^{-1}\left(\mathbf{y}-\mu(\mathbf{X})\right) \\ k(\mathbf{x}^*,\mathbf{x}^*) - k(\mathbf{x}^*,\mathbf{X})k(\mathbf{X},\mathbf{X})^{-1}k(\mathbf{X},\mathbf{x}^*)\Big) \end{aligned} \tag{1.4}$$

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

Sampling from a GP is also straightforward: a sample from a GP is just a single sample from a single multivariate Normal distribution (1.4). Figure 1.1 shows prior and posterior samples from a GP.

1.1.1 Useful Properties of Gaussian Processes

- **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** By choosing different covariance functions, we can express a very wide range of modeling assumptions.

- **Integration over hypotheses** The fact that a GP posterior lets us exactly integrate over a wide range of hypotheses means that overfitting is less of an issue than in comparable model classes. It also removes the need for sophisticated optimization schemes. In contrast, much of the neural network literature is devoted to techniques for regularization and optimization.

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

- **Closed-form posterior** The posterior predictive distribution of a GP is another GP. This means that GPs can easily be composed with other models or decision procedures. For example, in reinforcement learning applications,

- **Easy to Analyze** It may seem unsatisfying to restrict ourselves to a limited model class. Shouldn't we instead learn to use some more flexible model class, such as the set of all computible functions? The answer is: simple models can be used as well-understood building blocks for constructing more interesting models in diverse settings.

  Consider linear models. Although they form an extremely limited model class, they are fast, simple, and easy to analyze. This makes them easy to incorporate into other models or procedures. Linear models may seem like a hopelessly simple model class, but they're arguably the most useful modeling tools in existence.

1.1.2 Limitations of Gaussian Processes

- **Slow inference** Computing the matrix inverse in (1.3) and (1.4) takes $\mathcal{O}(N^3)$ time. Fortunately, this problem can be addressed by approximate inference schemes, and most GP software packages implement several of these.

- **Light tails** We may wish to use non-Gaussian noise models, for instance in order to be robust to outliers, or to perform classification, or some other form of structured prediction. Using non-Gaussian noise models requires approximate inference schemes; fortunately, mature software packages exist which can automatically perform approximate inference for a wide variety of likelihoods.

- **The need to choose a kernel** In practice, the extreme flexibility of GP models means that we are also faced with the difficult task of choosing a kernel. In fact,

choosing a useful kernel is equivalent to the problem of learning good features for the data. Typically, human experts choose from among a small set of standard kernels. In this thesis, we hope to go some way towards automating the construction and selection of useful kernels.

1.2 Outline and Contributions of Thesis

This thesis presents a set of related results about how the probabilistic nature of Gaussian process models allows them to be easily extended or composed with other models. Furthermore, the fact that the marginal likelihood is often available (or easily approximable) means that we can evaluate how much evidence the data provides for one structure over another.

Chapter 2 contains a wide-ranging overview of many of the types of structured priors on functions that can be easily expressed by constructing appropriate covariance functions. For example, in chapter 2, we'll see how GPs can be combined with latent variable models to produce models of nonparametric manifolds. By introducing structure into the kernels of those GPs, we can create manifolds with diverse topological structures, such as cylinders, torii and Möbius strips.

Given a wide variety of structures, plus the ability to evaluate the suitability of each one, it's straightforward to automatically search over models. **Chapter 3** shows how to construct a general, open-ended language over kernels - which implies a corresponding language over models. In chapter 3, we'll see how the marginal likelihood can guide us into automatically building structured models, and how capturing structure allows us to extrapolate, rather than simply interpolating.

Another benefit of using a relatively simple model class is that the resulting models are easy to understand. **Chapter 4** demonstrates how easy-to-understand the resulting models are, by demonstrating a simple system which automatically describes the structure discovered in a dataset by a search over GP models. This system automatically generates reports with graphs and english-language descriptions of GP models. Chapter 4 shows that, for the particular language of models constructed in chapter 3, it's relatively easy to automatically generate english-language descriptions of the models discovered. Augmented with interpretable plots decomposing the predictive posterior, we demonstrate how to automatically generate useful analyses of time-series. Combined with the automatic model search developed in chapter 3, this system represents the be-

ginnings of an "automatic statistician". We discuss the advantages and potential pitfalls of automating the modeling process in this way.

Chapter 6 examines the model class obtained by performing dropout in GPs, finding them to have equivalent covariance to *additive Gaussian processes*, a model summing over exponentially-many GP models, each depending on a different subset of the input variables. An polynomial-time algorithm for doing inference in this model class is given, and the resulting model class is characterized and related to existing model classes.

Chapter 7 develops an extension of the GP-LVM in which the latent distribution is a mixture of Gaussians. This model gives rise to a Bayesian clustering model in the clusters have nonparametric shapes. Like the density manifolds learned by the GP-LVM, the shapes of the clusters learned by the iWMM follow the contours of the data density.

Besides having a dual representation as linear regression, GPs can also be derived as the limit of an infinitely-wide neural network. As an example of using GPs as a simple-to-understand building block, **Chapter 5** analyzes deep network models by characterizing the prior over functions obtained by composing GP priors to form *deep Gaussian processes*. We find that, as the number of layers in such models increases, the amount of information retained about the original input diminshes to a single degree of freedom. We show that a simple change to the network architecture fixes this pathology.

1.3 Attribution

This thesis was made possible (and enjoyable to produce) by the substantial contributions of the many co-authors I was fortunate to work with. In this section, I attempt to give proper credit to my tireless co-authors.

Structure through kernels Section 5.6 of chapter 2, describing how symmetries in the kernel of a GP-LVM give rise to priors on manifolds with interesting topologies, is based on a collaboration with David Reshef, Roger Grosse, Josh Tenenbaum, and Zoubin Ghahramani.

Structure Search The research upon which Chapter 3 is based was done in collaboration with James Robert Lloyd, Roger Grosse, Joshua B. Tenenbaum, and Zoubin Ghahramani, and was published in (**?**), where James Lloyd was joint first author. Myself, James Lloyd and Roger Grosse jointly developed the idea of searching through a grammar-based language of GP models, inspired by **?**, and wrote the first versions of

the code together. James Lloyd ran almost all of the experiments. Carl Rasmussen, Zoubin Ghahramani and Josh Tenenbaum provided many conceptual insights, as well as suggestions about how the resulting procedure could be most fruitfully applied.

Automatic Statistician The work appearing in chapter 4 was written in collaboration with James Robert Lloyd, Roger Grosse, Joshua B. Tenenbaum, Zoubin Ghahramani, and was published in (**?**). The idea of the correspondence between kernels and adjectives grew out of discussions between James and myself. James Lloyd wrote most of the code to automatically generate reports, and ran all of the experiments. The text was written mainly by myself, James Lloyd, and Zoubin Ghahramani, with many helpful contributions and suggestions from Roger Grosse and Josh Tenenbaum.

Additive Gaussian processes The work in chapter 6 discussing additive GPs was done in collaboration with Hannes Nickisch and Carl Rasmussen, who developed a richly parameterized kernel which efficiently sums all possible products of input dimensions. My role in the project was to examine the properties of the resulting model, clarify the connections to existing methods, and to create all figures and run all experiments. This work was previously published in (**?**).

Warped Mixtures The work comprising the bulk of chapter 7 was done in collaboration with Tomoharu Iwata and Zoubin Ghahramani, and appeared in (**?**). Specifically, the main idea was borne out of a conversation between Tomo and myself, and together we wrote almost all of the code together as well as the paper. Tomo ran most of the experiments. Zoubin Ghahramani provided initial guidance, as well as many helpful suggestions throughout the project.

Deep Gaussian Processes The ideas contained in chapter 5 were developed through discussions with Oren Rippel, Ryan Adams and Zoubin Ghahramani, and appear in (**?**). The derivations, experiments and writing were done mainly by myself, with many helpful suggestions by my co-authors.

Chapter 2

Expressing Structure with Kernels

In this chapter, we'll show how to use kernels (also called *covariance functions*) to build many different kinds of models of functions. By combining a few simple kernels through addition and multiplication, we'll be able to express many different kinds of structure: additivity, symmetry, periodicity, interactions between variables, and many types of invariances. Combining kernels in a few simple ways gives us a rich language with which to build appropriate models.

2.1 Definition

Since we'll be discussing covariance functions at length, we now give a precise definition. A kernel $k(\mathbf{x}, \mathbf{x}') : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is any positive-definite function between two points x, x' in some space $\mathcal{X}$. In this chapter, $\mathcal{X}$ is usually taken to be Euclidian space, it can also be the space of images, documents, categories or points on a sphere, for example.

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 the context of GPs, since what is actually being specified is the similarity between two values of a *function* over objects. The kernel specifies which functions are likely under the GP prior, which in turn determines the generalization properties of the model.

2.2 Basic Kernels

Commonly used kernels families include the squared exponential (SE), periodic (Per), linear (Lin), and rational quadratic (RQ). These kernels are defined in Table 2.1. There is nothing special about these kernels in particular, except that they represent a

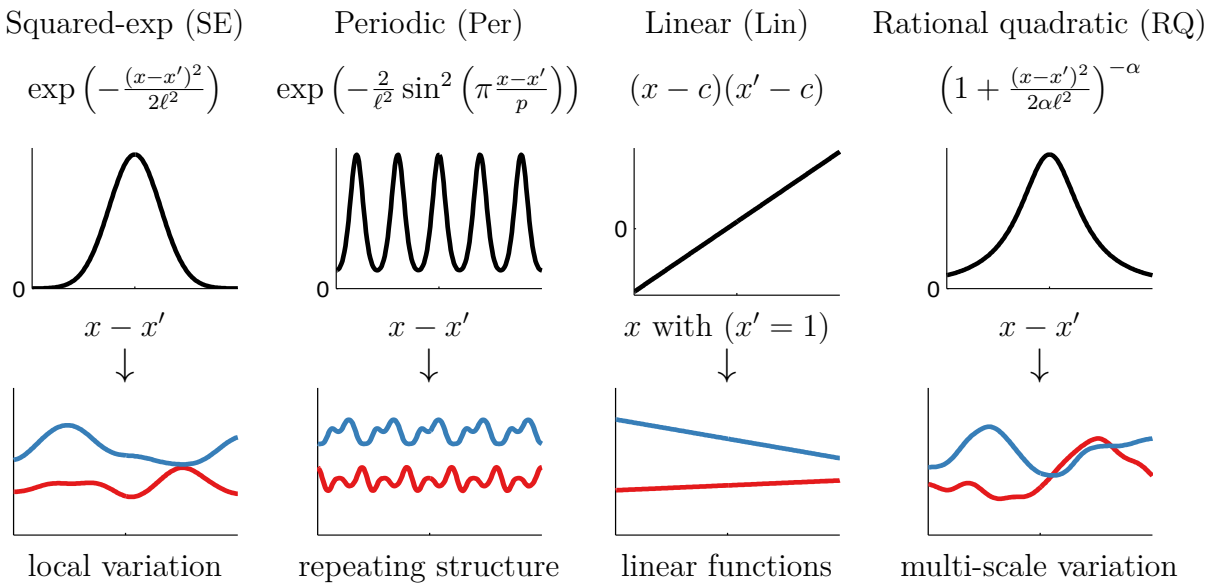


Table 2.1 Examples of structures expressible by base kernels. Left and third columns: base kernels $k(\cdot, 0)$. Second and fourth columns: draws from a GP with each repective kernel. The x-axis has the same range on all plots.

diverse set.

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.

2.3 Combining Kernels

The rest of this chapter will explore ways in which kernels can be combined to create new ones with different properties. Using a few simple base kernels and operations for combining them will give us a rich language with which to build models with diverse sorts of properties.

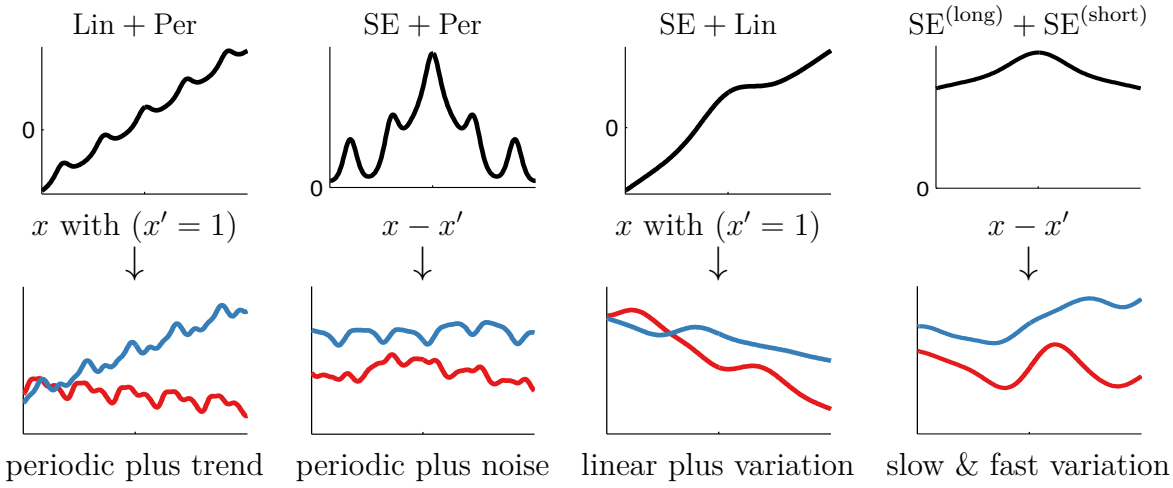


Fig. 2.1 Examples of one-dimensional structures expressible by adding kernels. Plots have same meaning as in figure 2.1.

2.3.1 Notation

In this chapter, we'll focus on two ways of combining kernels: addition and multiplication, which we'll often write in shorthand without arguments:

$$k_1 + k_2 = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \tag{2.2}$$

$$k_1 \times k_2 = k_1(\mathbf{x}, \mathbf{x}') \times k_2(\mathbf{x}, \mathbf{x}') \tag{2.3}$$

All of the base kernels we'll consider in this chapter are one-dimensional. Kernels over multidimensional inputs are constructed by adding and multiplying kernels over individual dimensions. These dimensions are represented using subscripts, e.g. SE_2 represents an SE kernel over the second dimension of x.

Each kernel has a number of parameters which specify the precise shape of the covariance function. These are sometimes referred to as *hyperparameters*, since they can be viewed as specifying a distribution over function parameters, instead of specifying the function directly. To remove clutter, we'll usually refer to a kernels without specifying their parameters.

2.4 Modeling Additive Functions

Additivity is a very useful modeling assumption in a wide variety of contexts. It often aids in building interpretable models, as well as enabling extrapolation in high-dimensional settings. Fortunately, this assumption is easy to encode in GP models. By summing kernels, we can model the data as a sum of independent functions, each possibly representing a different type of structure. Suppose functions f_1, f_2 are drawn independently from GP priors:

$$f_1 \sim \mathcal{GP}(\mu_1, k_1) \tag{2.4}$$
$$f_2 \sim \mathcal{GP}(\mu_2, k_2) \tag{2.5}$$

Then we can model the sum of those functions through another GP:

$$f_1 + f_2 \sim \mathcal{GP}(\mu_1 + \mu_2, k_1 + k_2). \tag{2.6}$$

Kernels k_1 and k_2 can be kernels of any type, and we can sum up any number of kernels 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 only of a subset of input dimensions, then the implied prior over functions decomposes in the same way. For example, if

$$f(x_1, x_2) \sim \mathcal{GP}(\mathbf{0}, k_1(x_1, x_1') + k_2(x_2, x_2')) \tag{2.7}$$

Then this is equivalent to the model

$$f_1(x_1) \sim \mathcal{GP}(\mathbf{0}, k_1(x_1, x_1')) \tag{2.8}$$
$$f_2(x_2) \sim \mathcal{GP}(\mathbf{0}, k_2(x_2, x_2')) \tag{2.9}$$
$$f(x_1, x_2) = f_1(x_1) + f_2(x_2) \tag{2.10}$$

Figure 2.2 illustrates this decomposition. Note that the product of two kernels does not have an analogous interpretation as the product of two functions.

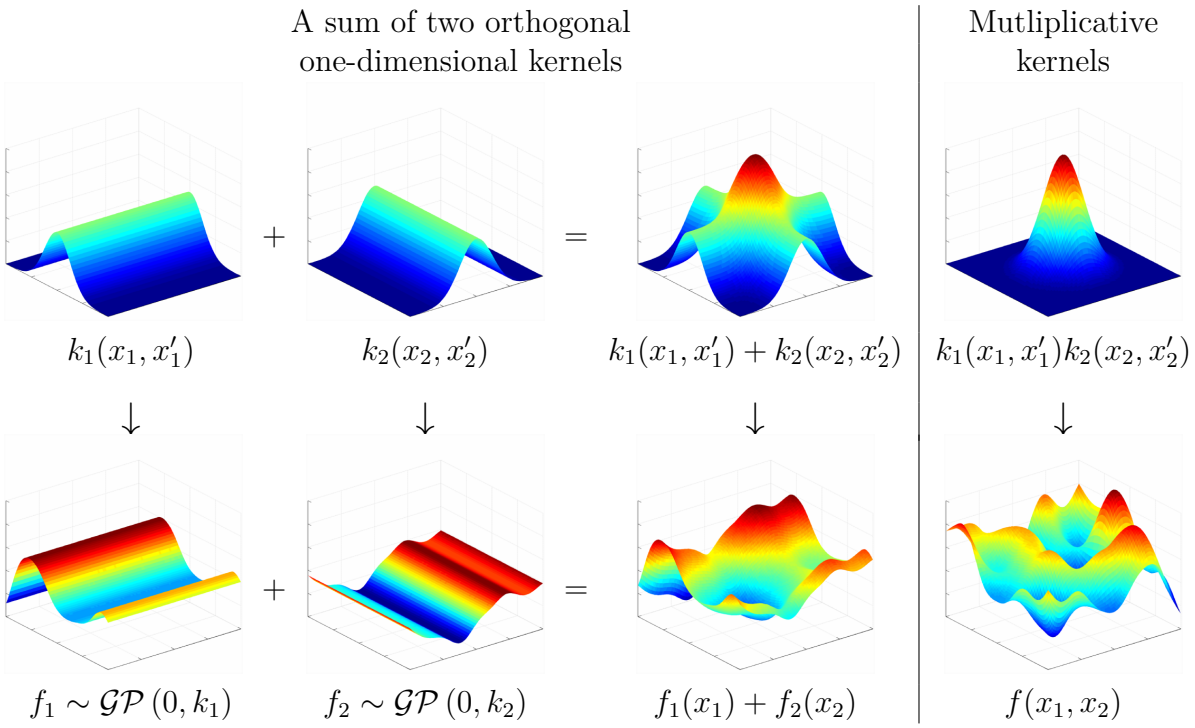


Fig. 2.2 Top left: An additive kernel is a sum of kernels. Bottom left: A draw from an additive kernel corresponds to a sum of draws from independent GP priors with the corresponding kernels. Top right: A product kernel. Bottom right: A GP prior with a product of kernels does not correspond to a product of draws from GPs.

2.4.2 Example: Additive Model of Concrete Compressive Strength

We now give an example of how additive kernels give rise to interpetable models. We model measurements of the compressive strength of concrete, as a function of the concentration of 7 ingredients, plus the age of the concrete (**?**).

We build an additive model

$$\begin{aligned} f(\mathbf{x}) = f_1(\text{cement}) + f_2(\text{slag}) + f_3(\text{fly ash}) + f_4(\text{water}) \\ + f_5(\text{plasticizer}) + f_6(\text{coarse}) + f_7(\text{fine}) + f_8(\text{age}) + \epsilon \end{aligned} \tag{2.11}$$

where $\epsilon \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma_n)$. After learning the kernel parameters by maximizing the marginal likelihood of the data, we can visualize the posterior distribution of each component of the model.

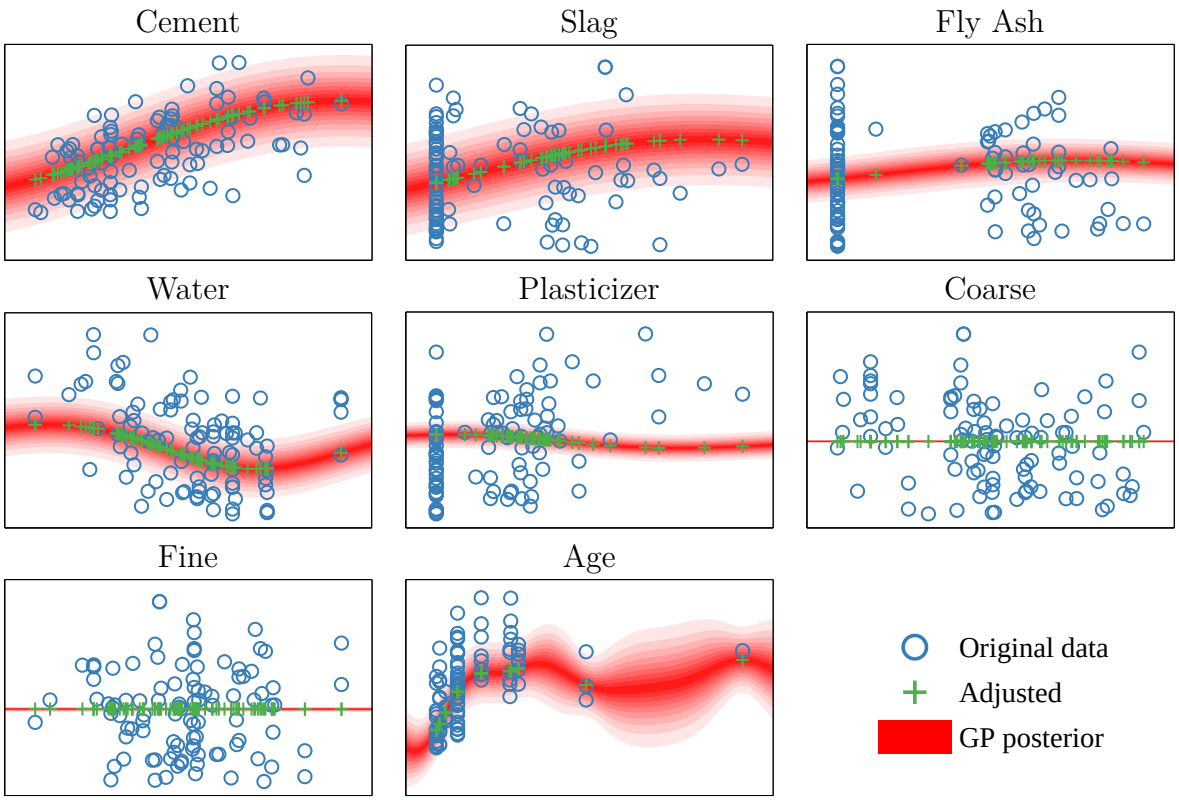


Fig. 2.3 By considering only one-dimensional terms of the additive kernel, we recover a form of Generalized Additive Model, and can plot the corresponding 1-dimensional functions. Blue points indicate the original data, green points are data after the contribution from all other terms has been subtracted. The vertical axis is the same for all plots.

Figure 2.3 shows the marginal posterior distribution of each of the 8 components in Equation (2.11). We can see that 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, also known as automatic relevance determination (ARD) (**?**).

2.4.3 Marginal Covariance of Components

How to compute the posterior distributions shown in Figure 2.3? Here we derive the posterior marginal variance and covariance of each of the additive components of a GP.

First, we'll write down the joint prior over the sum of two functions. We'll 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).

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

$$\begin{bmatrix} \boldsymbol{f}_1(\mathbf{x}) \\ \boldsymbol{f}_1(\mathbf{x}^\star) \\ \boldsymbol{f}_2(\mathbf{x}) \\ \boldsymbol{f}_2(\mathbf{x}^\star) \\ \boldsymbol{f}_1(\mathbf{x}) + \boldsymbol{f}_2(\mathbf{x}) \\ \boldsymbol{f}_1(\mathbf{x}^\star) + \boldsymbol{f}_2(\mathbf{x}^\star) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_1^\star \\ \boldsymbol{\mu}_2 \\ \boldsymbol{\mu}_2^\star \\ \boldsymbol{\mu}_1 + \boldsymbol{\mu}_2 \\ \boldsymbol{\mu}_1^\star + \boldsymbol{\mu}_2^\star \end{bmatrix}, \begin{bmatrix} \mathbf{K}_1 & \mathbf{K}_1^\star & 0 & 0 & \mathbf{K}_1 & \mathbf{K}_1^\star \\ \mathbf{K}_1^\star & \mathbf{K}_1^{\star\star} & 0 & 0 & \mathbf{K}_1^\star & \mathbf{K}_1^{\star\star} \\ 0 & 0 & \mathbf{K}_2 & \mathbf{K}_2^\star & \mathbf{K}_2 & \mathbf{K}_2^\star \\ 0 & 0 & \mathbf{K}_2^\star & \mathbf{K}_2^{\star\star} & \mathbf{K}_2^\star & \mathbf{K}_2^{\star\star} \\ \mathbf{K}_1 & \mathbf{K}_1^\star & \mathbf{K}_2 & \mathbf{K}_2^\star & \mathbf{K}_1 + \mathbf{K}_2 & \mathbf{K}_1^\star + \mathbf{K}_2^\star \\ \mathbf{K}_1^\star & \mathbf{K}_1^{\star\star} & \mathbf{K}_2^\star & \mathbf{K}_2^{\star\star} & \mathbf{K}_1^\star + \mathbf{K}_2^\star & \mathbf{K}_1^{\star\star} + \mathbf{K}_2^{\star\star} \end{bmatrix} \right) \tag{2.12}$$

where we represent the Gram matrices, evaluated at all pairs of vectors in bold capitals as $\mathbf{K}_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$. So

$$\mathbf{K}_1 = k_1(\mathbf{X}, \mathbf{X}) \tag{2.13}$$

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

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

By the formula for Gaussian conditionals: equation (A.1), we get that the conditional distribution of a GP-distributed function conditioned on its sum with another

GP-distributed function is given by

$$\boldsymbol{f}_1^\star|\boldsymbol{f} \sim \mathcal{N}\Big(\boldsymbol{\mu}_1^\star + \mathbf{K}_1^{\star\mathsf{T}}(\mathbf{K}_1+\mathbf{K}_2)^{-1}(\boldsymbol{f}-\boldsymbol{\mu}_1-\boldsymbol{\mu}_2) \\ \mathbf{K}_1^{\star\star} - \mathbf{K}_1^{\star\mathsf{T}}(\mathbf{K}_1+\mathbf{K}_2)^{-1}\mathbf{K}_1^\star\Big) \tag{2.16}$$

These formulae express the posterior model uncertainty about different components of the signal, integrating over the possible configurations of the other components. If we wish to condition on the 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 Components

One of the advantages of using a generative, model-based approach is that we can examine any aspect of the model we wish to. For instance, we can compute the posterior covariance between any two components, conditioned on their sum:

$$\operatorname{cov}\left[\boldsymbol{f}_1^\star, \boldsymbol{f}_2^\star|\boldsymbol{f}\right] = -\mathbf{K}_1^{\star\mathsf{T}}(\mathbf{K}_1+\mathbf{K}_2)^{-1}\mathbf{K}_2^\star \tag{2.17}$$

If this quantity is negative, it means that there is ambiguity about which of the two components explains the data at that location.

For example, Figure 2.4 plots the posterior correlation between all non-zero components of the concrete model. We can see that there is 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.

2.4.4 Long-range Extrapolation through Additivity

Finding additive structure is useful, because it allows us to make predictions far from the training data. Figure 2.5 compares the extrapolations made by additive versus non-additive GP models, conditioned on data from a sum of two axis-aligned sine functions, evaluated in a small, L-shaped area. In this example, the additive model is able to correctly predict the value of the function at unseen combinations of inputs. The product-kernel model is more flexible, and so remains uncertain about the function value at unseen combinations of inputs.

These types of additive models have been well-explored in the statistics literature. For example, generalized additive models (**?**) have seen wide adoption. In high dimen-

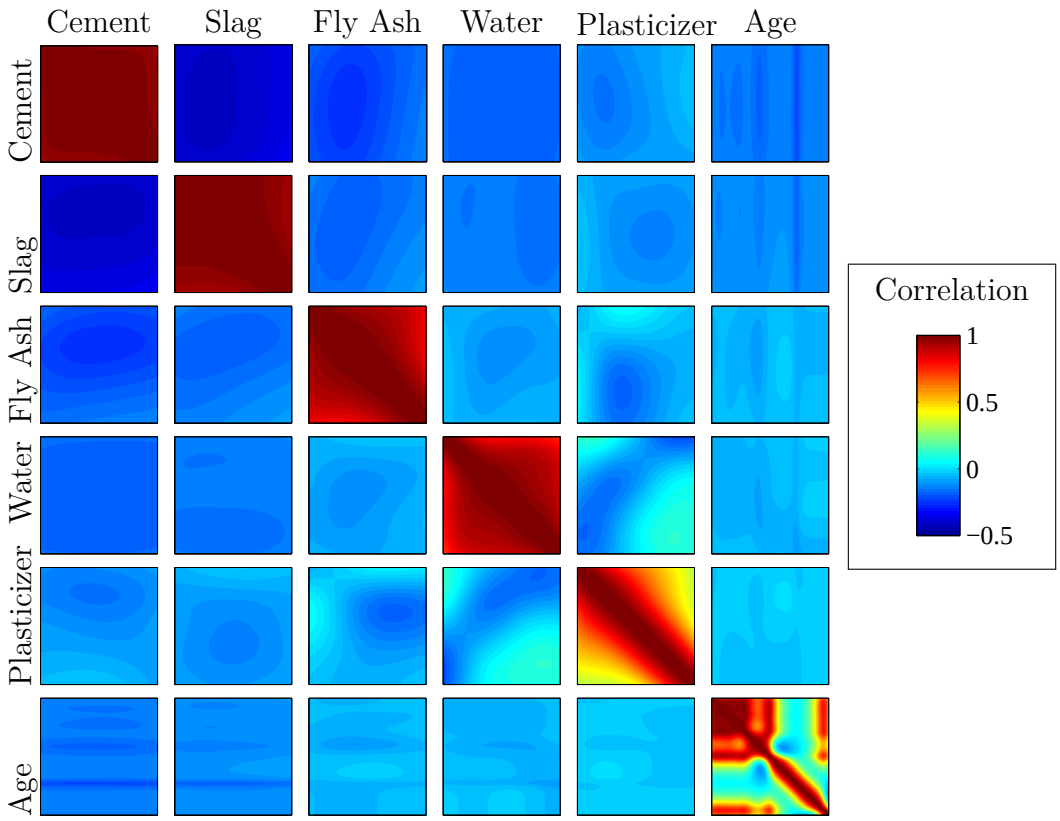


Fig. 2.4 Visualizing posterior correlations between the components explaining the concrete dataset. Each plot shows the additive model's posterior correlation between two components, plotted over the domain of the data ±5%. Red indicates high correlation, teal indicates no correlation, and blue indicates negative correlation. There is negative correlation between the "Cement" and "Slag" variables, meaning that - one of these functions is high and the other low, but which one is uncertain under the model. Dimensions 'Coarse' and 'Fine' are not shown, because their variance was zero.

sions, we can also consider sums of functions of more than one dimension. Chapter 6 considers this model class in more detail.

2.5 Modifying Kernels through Multiplication

Multiplying kernels allows us to account for interactions between different input dimensions or different notions of similarity. In univariate data, multiplying a kernel by SE gives a way of converting global structure to local structure. For example, Per corresponds to globally periodic structure, whereas Per × SE corresponds to locally periodic structure, as shown in row 1 of figure 2.2.

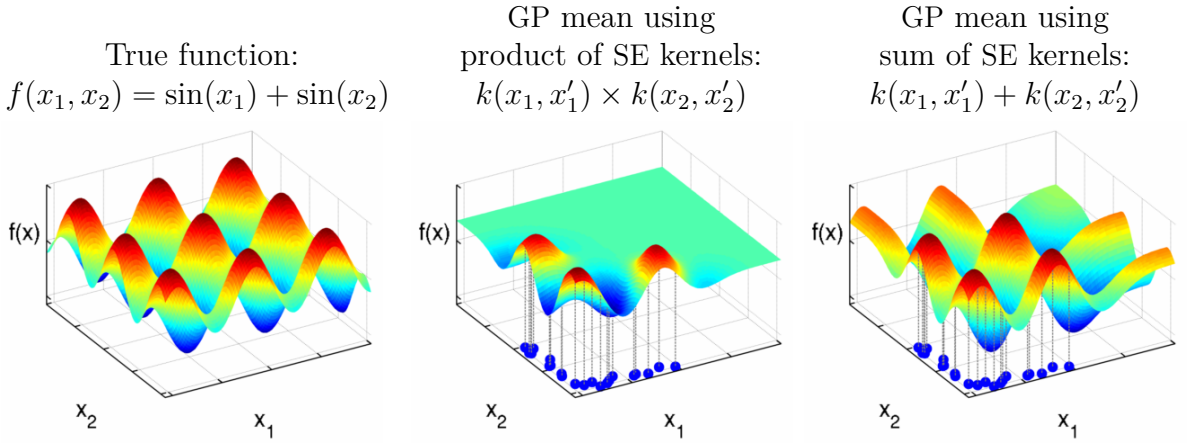


Fig. 2.5 Inference in functions with additive structure. When the function being modeled has additive structure, we can exploit this fact to extrapolate far from the training data. The product kernel allows a different function value for every combination of inputs, and so is uncertain about function values away from the training data.

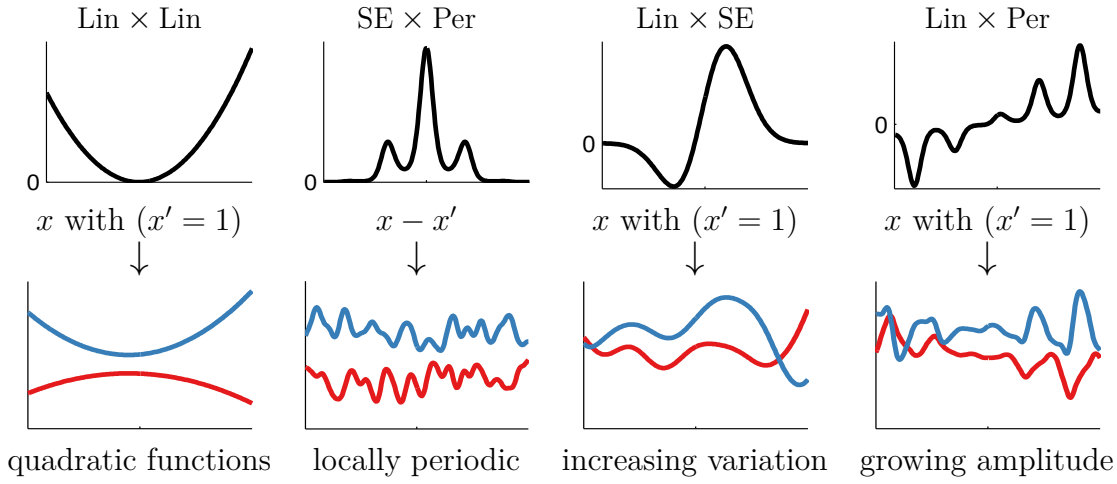


Fig. 2.6 Examples of one-dimensional structures expressible by multiplying kernels. Plots have same meaning as in figure 2.1.

Products of SE kernels, having a different lengthscale parameter for each dimension, are equivalent to the SE-ARD kernel:

$$\text{SE-ARD}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}\sum_{d=1}^{D}\frac{(x_d - x'_d)^2}{\ell_d^2}\right) \tag{2.18}$$

Where ARD stands for Automatic Relevance Determination .

Since we are applying these operations to the similarity functions rather than the regression functions themselves, compositions of even a few base kernels are able to capture complex relationships in data which do not have a simple parametric form. [TODO: flesh this section out]

2.6 Changepoints

An example of aChangepoints can be defined through addition and multiplication with sigmoidal functions:

$$\text{CP}(k_1, k_2) = k_1 \times \boldsymbol{\sigma} + k_2 \times \bar{\boldsymbol{\sigma}} \tag{2.19}$$

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. We can also express a function which changes structure within some interval - a change window - by replacing $\sigma(x)$ with a product of two sigmoids, one increasing and one decreasing.

2.6.1 Multiplication by a Known Function

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

$$f(\mathbf{x}) = a(\mathbf{x})g(\mathbf{x}), \quad g \sim \mathcal{GP}(g \,|\, \mathbf{0}, k(\mathbf{x}, \mathbf{x}')) \quad \iff \quad f \sim \mathcal{GP}(f \,|\, \mathbf{0}, a(\mathbf{x})k(\mathbf{x}, \mathbf{x}')a(\mathbf{x}')) \tag{2.20}$$

2.7 Feature Representation

By Mercer's theorem (**?**), any positive-definite kernel can be represented as the inner product between a fixed set of features, evaluated at x and at x':

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{h}(\mathbf{x})^\mathsf{T}\mathbf{h}(\mathbf{x}') \tag{2.21}$$

As a simple example, the squared-exponential kernel (SE) on the real line has a representation in terms of infinitely many radial-basis functions. Any stationary kernel (one which only depends on the distance between its inputs) on the real line can be represented by a set of sines and cosines - a Fourier representation (**?**). In general, the feature representation of a kernel is not unique, and depends on which space $\mathcal{X}$ is being considered (**?**).

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*, a topic explored in chapter 5.

2.7.1 Relation to Linear Regression

Suprisingly, GP regression is equivalent to Bayesian linear regression on $\mathbf{h}(\mathbf{x})$:

$$f(\mathbf{x}) = \mathbf{w}^\mathsf{T}\mathbf{h}(\mathbf{x}), \quad \mathbf{w} \sim \mathcal{N}(\mathbf{w}|\mathbf{0}, \Sigma) \quad \iff \quad f \sim \mathcal{GP}\Big(f \,\Big|\, \mathbf{0}, \mathbf{h}(\mathbf{x})^\mathsf{T}\Sigma\mathbf{h}(\mathbf{x})\Big) \tag{2.22}$$

The link between Gaussian processes, linear regression, and neural networks is explored further in chapter 5.

2.7.2 Feature-space view of Combining Kernels

Many architectures for learning complex functions, such as convolutional networks **?** and sum-product networks **?**, include units which compute `and`-like and `or`-like operations. Composite kernels can be viewed in this way too. A sum of kernels can be understood as an `or`-like operation: two points are considered similar if either kernel has a high value. Similarly, multiplying kernels is an `and`-like operation, since two points are considered similar only if both kernels have high values.

We can also view kernel addition and multiplication as a combination of the features

of the original kernels. For example, if we have two kernels

$$k_a(\mathbf{x}, \mathbf{x}') = \mathbf{a}(\mathbf{x})^\mathsf{T}\mathbf{a}(\mathbf{x}') \tag{2.23}$$

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

and we consider their addition, then

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

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

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:

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

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

$$= \sum_i \sum_j a_i(\mathbf{x})a_i(\mathbf{x}')b_j(\mathbf{x})b_j(\mathbf{x}') \tag{2.29}$$

$$= \sum_{i,j} \left[a_i(\mathbf{x})b_j(\mathbf{x})\right]\left[a_i(\mathbf{x}')b_j(\mathbf{x}')\right] \tag{2.30}$$

$$= \operatorname{vec}\left(\mathbf{a}(\mathbf{x}) \otimes \mathbf{b}(\mathbf{x}')\right)^\mathsf{T} \operatorname{vec}\left(\mathbf{a}(\mathbf{x}) \otimes \mathbf{b}(\mathbf{x}')\right) \tag{2.31}$$

In other words, the features of $k_a \times k_b$ are just the cartesian product (all possible combinations) of the original two sets of features. For example, a set of infinitely many Gaussian bumps spread evenly along the real line gives rise to one-dimensional SE kernel. The cartesian product of these features with another set spread along a different dimension gives a tiling of the plane with two-dimensional Gaussian bumps. This tiling corresponds to a two-dimensional SE kernel.

2.8 Expressing Symmetries and Invariances

When modeling functions, encoding known symmetries improves both learning and prediction. Many types of symmetry can be enforced through operations on the covariance function. In this section, we'll look at different ways in which we can encode a given symmetry into a prior on functions.

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

? and **?** characterized the set of GP priors on functions invariant to given transformations. They showed that the only way to construct a prior on functions which respect a given invariance is to construct a kernel which respects the same invariance w.r.t. each of its two inputs.

Formally, given a finite group of operations G to which we wish our function to remain invariant, and $f \sim \mathcal{GP}(0, k(\mathbf{x}, \mathbf{x}'))$, f is invariant under G if and only if $k(\cdot, \cdot)$ is argumentwise invariant:

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}, \forall g, g' \in G, \quad k(g(\mathbf{x}), g(\mathbf{x}')) = k(\mathbf{x}, \mathbf{x}') \tag{2.32}$$

As a simple example, consider the symmetry $f(x, y) = f(y, x)$, or the set of functions invariant to switching their two arguments. The elements of the group G_{switch} are

$$g_1(f(x, y)) = f(x, y) \qquad \text{(identity)} \tag{2.33}$$
$$g_2(f(x, y)) = f(y, x) \qquad \text{(switch)} \tag{2.34}$$

It might not be clear how to come up with a kernel obeying these symmetries. Fortunately, for finite groups, there are a few simple ways to transform any kernel into one which is argumentwise invariant to actions under the group:

1. **Sum over the Orbit ?** and **?** suggest a double sum over the orbits of $\mathbf{x}$ and $\mathbf{x}'$ w.r.t. 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 our example group G_{switch}, this operation results in:

$$k_{\text{switch}}(\mathbf{x}, \mathbf{x}') = \sum_{g, \in G_{\text{switch}}} \sum_{g' \in G_{\text{switch}}} k(g(\mathbf{x}), g'(\mathbf{x}')) \tag{2.36}$$
$$= k(x, y, x', y') + k(x, y, y', x') + k(y, x, x', y') + k(y, x, y', x') \tag{2.37}$$

For stationary kernels, some pairs of elements in this sum will be identical, and can be ignored. Figure 2.7a shows a draw from this prior using the SE kernel. This

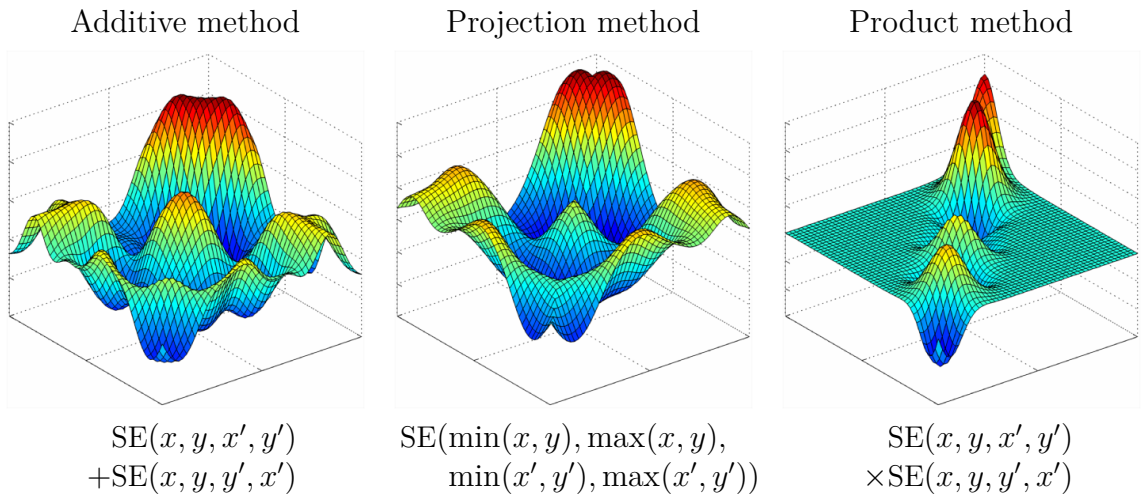


Fig. 2.7 Three methods of introducing symmetry, illustrated through draws from the corresponding priors. All three methods introduce a different type of nonstationarity.

construction has the property that the marginal variance is doubled near $x = y$, which may or may not be desirable.

2. **Projection to a Fundamental Domain ?** also explore the possibility of projecting each datapoint into a fundamental domain of the group:

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

For our example group G_{switch}, a fundamental domain is $\{x, y : x < y\}$, which can be mapped to using $A_{G_{\text{switch}}}(x, y) = \Big[\min(x, y), \max(x, y)\Big]$. Constructing a kernel using this method introduces a non-differentiable "seam" along $x = y$.

The projection method also works for infinite groups, as we shall see below.

3. **Product over the Orbit** Ryan P. Adams (personal communication) suggests a construction using 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 is not recommended, since it will often produce GP priors with zero variance in some regions of the space, as in Figure 2.7c.

There are many possible ways to achive 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.

2.8.1 Example: Periodicity

Periodicity in a one-dimensional function corresponds to the invariance

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

where τ is the period.

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

$$\mathrm{Per}(x, x') = \mathrm{SE}(\Big[\sin(x), \cos(x)\Big], \Big[\sin(x'), \cos(x')\Big]) \tag{2.41}$$

achieves the invariance in Equation (2.40). Some simple algebra reduces this kernel to the form shown in Table 2.1.

We could also build a periodic kernel by the mapping $A(x) = \mathrm{mod}(x, \tau)$. However, samples from this prior would be discontinuous at every multiple of τ.

2.8.2 Example: Reflective Symmetry Along an Axis

We can enforce symmetry about zero

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

using the sum over orbits method, by the transform

$$k_{\text{symm axis}}(x, x') = k(x, x') + k(x, -x') + k(-x, x') + k(-x, -x') \tag{2.43}$$

2.8.3 Example: Translation Invariance in Images

Most models of images are invariant to spatial translations (**?**). Similarily, most models of sounds are also invariant to translation through time.

Note that this sort of translational 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, we are discussing here a discretized input space (into pixels or the audio equivalent), where the input vectors have one dimension for every pixel. We are interested in creating priors on functions that are invariant to shifting a signal along its pixels:

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

Translational invariance in this setting is equivalent to invariance two swapping sets of dimensions in the input space.

This invariance can be achieved in one dimension by using the transformation

$$k_{\text{invariant}}\left(\mathbf{x}, \mathbf{x}'\right) = \sum_{i=1}^{D} \sum_{j=1}^{D} k(\text{shift}(\mathbf{x}, i), \text{shift}(\mathbf{x}, j)) \tag{2.45}$$

where

$$\text{shift}(\mathbf{x}, i) = \left[x_{\text{mod}(i+1,D)}, x_{\text{mod}(i+2,D)}, \ldots, x_{\text{mod}(D+i,D)}\right] \tag{2.46}$$

The extension to two dimensions is straightforward, but notationally cumbersome. We are simply defining the covariance between two images to be the sum of all covariances between all translations of the two images.

? built a kernel between images approximately invariant to translation and rotation by using the projection method.

2.9 Generating Topological Manifolds

We now give a geometric illustration of the symmetries encoded by GPs with different sorts of kernels.

Priors on functions exhibiting symmetries can be used to create a prior on topological manifolds, by warping a latent surface $\mathbf{x}$ to an observed surface $\mathbf{y} = f(\mathbf{x})$. To doso, first create a mesh in 2D. Next, draw 3 independent functions from a GP prior. Then, map the 2D mesh through those functions to get 3D coordinates of each point on the mesh.

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

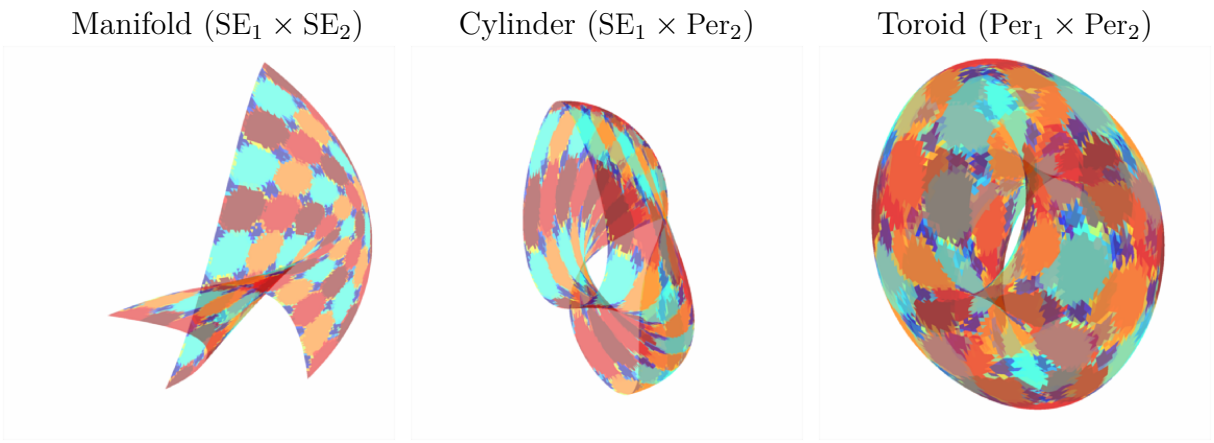


Fig. 2.8 Generating 2D manifolds with different topological structures. By enforcing that the functions mapping from $\mathbb{R}^2$ to $\mathbb{R}^3$ obey the appropriate symmetries, the surfaces created have the corresponding topologies (ignoring self-intersections).

2.9.1 Surfaces, Cylinders and Torii

Figure 2.8 shows 2D meshes warped into 3D by functions drawn from GP priors with different kernels. The different kernels give rise to different topologies.

2.9.2 Möbius Strips

A prior on functions on Möbius strips can be achieved by enforcing the symmetries:

$$f(x, y) = f(x, y + \tau_y) \tag{2.47}$$

$$f(x, y) = f(x + \tau_x, y) \tag{2.48}$$

$$f(x, y) = f(y, x) \tag{2.49}$$

If we imagine moving along the edge of a Möbius strip, that is equivalent to moving along a diagonal in the function generated. Figure 2.9a shows an example of a function drawn from such a prior. Figure 2.9b shows an example of a 2D mesh mapped to 3D by functions drawn from such a prior. This surface doesn't resemble a typical Möbius strip, because the edge of the mobius strip is in roughly circular shape, as opposed to the double-loop that one obtains by gluing a strip of paper with a single twist. The surface shown resembles the Sudanese Möbius strip (**?**), shown in Figure 2.9c.

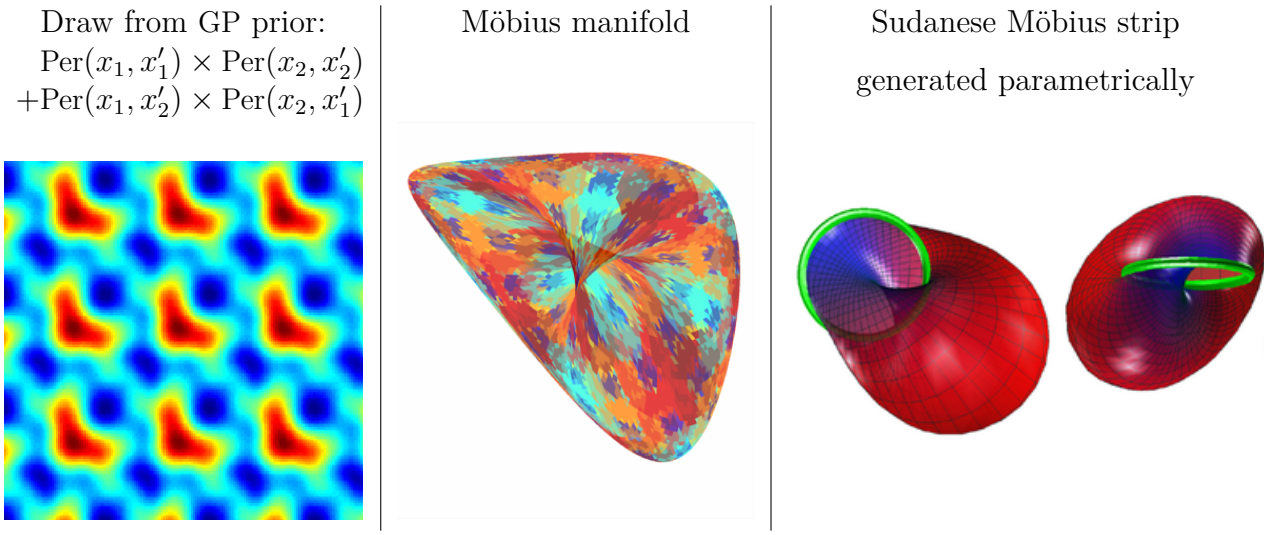


Fig. 2.9 Generating Möbius strips. Left: By enforcing that the functions mapping from $\mathbb{R}^2$ to $\mathbb{R}^3$ obey the appropriate symmetries, surfaces sampled from the prior have topology corresponding to a Möbius strip. Möbius strips generated this way do not have the familiar shape of a circular flat surface with a half-twist; rather they tend to look like *Sudanese* Möbius strips (**?**), whose edge has a circular shape. Right: A Sudanese projection of a Möbius strip. Image adapted from (**?**).

2.10 Kernels on Categorical Variables

One flexible way to build a kernel over categorical variables is simply to represent your categorical variable by a one-of-k encoding. For example, if $x \in \{1, 2, 3, 4, 5\}$, one-of-k$(3) = [0, 0, 1, 0, 0]$.

$$k_{\text{categorical}}(x, x') = \text{SE-ARD}(\text{one-of-k}(x), \text{one-of-k}(x')) \tag{2.50}$$

Short lengthscales for any particular dimension in the SE-ARD kernel indicate that that category is dissimilar to all 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 the fully-parameterized SE-full:

$$\text{SE-full}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{x}'\right) \tag{2.51}$$

where L is symmetric. This kernel individually parameterizes the covariance between each pair of categories.

2.11 Learning Kernel Parameters

One difficulty in building GP models is choosing, or integrating over, the kernel parameters. Fortunately, typical kernels only have $\mathcal{O}(D)$ parameters, meaning that if N is reasonably large, these parameters can be estimated by maximum marginal likelihood. For a fixed kernel form, these parameters can be optimized by gradient-based methods.

2.12 Automatically Choosing a Kernel

The marginal likelihood can also be used to select the form of the kernel.

For example, we might not know whether a particular structure or symmetry is present in the function we are trying to model. By building kernels with and without such structure, we can compute the marginal likelihoods of the corresponding GP models. The quantities represent the relative amount of evidence that the data provide for each of these possibilities, providing the assumptions of the model are correct.

2.13 Conclusion

We've seen that kernels are a flexible and powerful language for building models of different types of functions. However, for a given problem, it can difficult to specify an appropriate kernel, even after looking at the data. A better procedure would be to compare the predictive performance, or marginal likelihood, of a few different kernels. However, it might be difficult to enumerate all plausible kernels, and tedious to search over them.

Analogously, we usually don't expect to simply guess the best value of some parameter. Rather, we specify a search space and an objective, and ask the computer to the search this space for us. In the next chapter, we'll see how to perform such a search an automatic search over the open-ended, discrete space of kernel expressions.

Chapter 3

Automatically Building Structured Covariance Functions

> "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."
>
> ?

In chapter 2, we saw that the choice of kernel determines the type of structure that can be learnt by a GP model, and that a wide variety of models could be constructed through simply adding and multiplying a few base kernels together. We didn't answer the question, however, of how to tell which kernel to use for a given problem. Even for experts, choosing the kernel in nonparametric regression remains a black art.

In this chapter, we'll automate the process of building kernels for GP models. To do so, all we need to do is define an open-ended space of kernels, which we can do by simply adding and multiplying together simple kernels from a fixed set. We can then simply search over this space to find a kernel which captures as much structure in the data as possible.

Searching over such a large, structured model class has two benefits. First, this method has very good predictive accuracy, since it effectively tries out a huge number of different regression models. Second, we end up discovering interpretable structure in our dataset. Because GP posteriors can easily be decomposed (as in section 2.4.2), we can examine the resulting structures graphically. In chapter 4, we'll even show how to automatically generate english-language descriptions of the resulting models.

3.1 Ingredients of an Automatic Statistician

? 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."

In this section, we discuss in more detail the elements we believe are required to build an artificial intelligence that can do statistics.

1. An open-ended language of models Many statistical procedures consider all models in a class of fixed size - for example, graphical model construction algorithms [(1)] search over connectivity graphs for a given set of nodes. While these methods can be powerful, human statisticians are capable of deriving novel model classes when required by the modelling task. An automatic search through an open-ended class of models can achieve some of this flexibility, growing the complexity of the model to fit the task at hand, and possibly combining existing structures in novel ways.

(1) Cite

2. Searching 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 model search in an open-ended space will likely resemble a model-building procedure.

3. Model comparison and checking model fit An automatic statistician should be able to question the models it has constructed, and formal procedures from model checking provide a way for it to do this. **?** review the literature on model checking. In this work, we use approximate marginal likelihood to compare models, penalizing complexity using the Bayesian Information Criterion as a heuristic.

4. Describing models Part of the value of statistical models comes from enabling 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.

3.2 A Language of Regression Models

As shown in Chapter 2, we can construct a wide variety of kernel structures compositionally by adding and multiplying a small number of base kernels. In particular, we consider the four base kernel families discussed in Section **??**: SE, Per, Lin, and RQ. Any algebraic expression combining these kernels using the operations + and × defines a kernel family, whose parameters are the concatenation of the parameters for the base kernel families.

We would like an expressive language which can represent both simple parametric forms of f such as linear, polynomial, etc. and also complex nonparametric functions specified in terms of properties such as smoothness, periodicity, etc. Fortunately, Gaussian processes (GPs) provide a very general and analytically tractable way of capturing both simple and complex functions.

We can therefore define a language of regression models by specifying a language of kernels.

The elements of this language are a set of base kernels capturing different function properties, and a set of composition rules which combine kernels to yield other valid kernels. Our base kernels are white noise (WN), constant (C), linear (Lin), squared exponential (SE) and periodic (Per), which on their own encode for uncorrelated noise, constant functions, linear functions, smooth functions and periodic functions respectively. The composition rules are addition and multiplication:

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

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

We have found that incorporating changepoints into the language is essential for realistic models of time series (e.g. figure 4.3).

Table 3.1 lists common regression models that can be expressed by our language.

Regression model	Kernel
Linear regression	$\mathrm{C} + \mathrm{Lin} + \mathrm{WN}$
Kernel ridge regression	$\mathrm{SE} + \mathrm{WN}$
Multiple kernel learning	$\sum \mathrm{SE} + \mathrm{WN}$
Trend, cyclical, irregular	$\sum \mathrm{SE} + \sum \mathrm{Per} + \mathrm{WN}$
Fourier decomposition	$\mathrm{C} + \sum \cos + \mathrm{WN}$
Sparse spectrum GPs	$\sum \cos + \mathrm{WN}$
Spectral mixture	$\sum \mathrm{SE} \times \cos + \mathrm{WN}$
Changepoints	e.g. $\mathrm{CP}(\mathrm{SE}, \mathrm{SE}) + \mathrm{WN}$
Heteroscedasticity	e.g. $\mathrm{SE} + \mathrm{Lin} \times \mathrm{WN}$

Table 3.1 Common regression models expressible in our language. cos is a special case of our reparametrised Per.

3.3 Model Search

We explore the space of regression models using a greedy search. We use the same search operators, but also include additional operators to incorporate changepoints; a complete list is contained in the supplementary material.

Our search procedure begins by proposing all base kernel families applied to all input dimensions. We allow the following search operators over our set of expressions:

(1) Any subexpression $\mathcal{S}$ can be replaced with $\mathcal{S}+\mathcal{B}$, where $\mathcal{B}$ is any base kernel family.

(2) Any subexpression $\mathcal{S}$ can be replaced with $\mathcal{S}\times\mathcal{B}$, where $\mathcal{B}$ is any base kernel family.

(3) Any base kernel $\mathcal{B}$ may be replaced with any other base kernel family $\mathcal{B}'$.

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

These operators can generate all possible algebraic expressions. To see this, observe that if we restricted the $+$ and $\times$ rules to only apply to base kernel families, we would obtain a context-free grammar which generates the set of algebraic expressions. However, the more general versions of these rules allow more flexibility in the search procedure, which is useful because the context-free grammar derivation may not be the most straightforward way to arrive at a kernel family.

Our algorithm searches over this space using a greedy search: at each stage, we choose the highest scoring kernel and expand it by applying all possible operators.

Our search operators are motivated by strategies researchers often use to construct kernels. In particular,

- One can look for structure, e.g. periodicity, in the residuals of a model, and then extend the model to capture that structure. This corresponds to applying rule (1).
- One can start with structure, e.g. linearity, which is assumed to hold globally, but find that it only holds locally. This corresponds to applying rule (2) to obtain the structure shown in rows 1 and 3 of figure 2.2.
- One can add features incrementally, analogous to algorithms like boosting, backfitting, or forward selection. This corresponds to applying rules (1) or (2) to dimensions not yet included in the model.

Hyperparameter initialization

Unfortunately, optimizing 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. To alleviate this difficulty, we take advantage of our search procedure to provide reasonable initializations: all of the parameters which were part of the previous kernel are initialized to their previous values. All parameters are then optimized using conjugate gradients, randomly restarting the newly introduced parameters. This procedure is not guaranteed to find the global optimum, but it implements the commonly used heuristic of iteratively modeling residuals.

3.4 Model Evaluation

Choosing kernel structures requires a criterion for evaluating structures. We choose marginal likelihood as our criterion, since it balances the fit and complexity of a model (**?**). Conditioned on kernel parameters, the marginal likelihood of a GP can be computed analytically. However, to evaluate a kernel family we must integrate over kernel parameters. We approximate this intractable integral with the Bayesian information criterion (**?**) after first optimizing to find the maximum-likelihood kernel parameters.

In a fully Bayesian approach, we would put priors over the parameters and compute the marginal likelihood of the models with all the parameters integrated out. However, as this would be difficult to do across our space of models, we approximate this integral

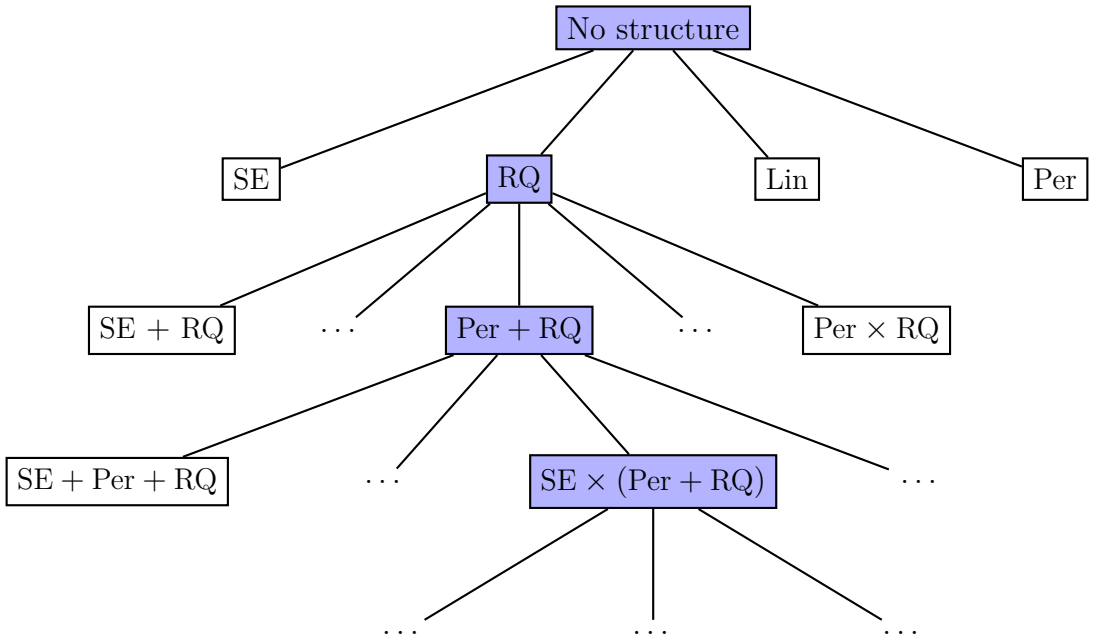


Fig. 3.1 An example of a search tree over kernel expressions. Figure 3.3 shows the model increasing in sophistication as the kernel expression grows.

by choosing the parameters to optimize the marginal likelihood, and then apply the Bayesian information criterion (BIC) to penalize model complexity.

Of course, other model selection criteria could be used with our search procedure. For instance, cross-validation could be used when the goal is interpolation.

We optimized kernel parameters at each step using conjugate gradients, randomly restarting any newly introduced kernel parameters. To approximate the marginal likelihood of a kernel without integrating over parameters, used the Bayesian Information Criterion (BIC) (**?**). We also experimented with using the Laplace approximation to the marginal likelihood, but this was found to be less numerically stable and was not meaningful in cases when the optimiser failed to reach an optimum. Because BIC is a function of the number of parameters in a model, we adjusted for cases where two parameters were only serving the role of one. e.g. when two kernels are multiplied, one of the variance parameters becomes redundant.

After each model is proposed its kernel parameters are optimised by conjugate gradient descent. We evaluate each optimized model, M, using the Bayesian Information Criterion (BIC) (**?**):

$$\mathrm{BIC}(M) = -2 \log p(D \mid M) + p \log n \tag{3.3}$$

where p is the number of kernel parameters, $\log p(D|M)$ is the marginal likelihood of

the data, D, and n is the number of data points. BIC trades off model fit against model complexity and implements what is known as "Bayesian Occam's Razor" (**??**).

3.5 Structure Discovery in Time Series

To investigate our method's ability to discover structure, we ran the kernel search on several time-series.

As discussed in Chapter 2, a GP whose kernel is a sum of kernels can be viewed as a sum of functions drawn from component GPs. This provides another method of visualizing the learned structures. In particular, all kernels in our search space can be equivalently written as sums of products of base kernels by applying distributivity. For example,

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

We visualize the decompositions into sums of components using the formulae given in the appendix. The search was run to depth 10, using the base kernels from Section **??**.

Mauna Loa atmospheric CO_2 Using our method, we analyzed records of carbon dioxide levels recorded at the Mauna Loa observatory. Since this dataset was analyzed in detail by **?**, 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 single base kernel model, the extrapolations improve dramatically as the increased search depth allows more structure to be included.

Figure 3.2 shows the final model chosen by our method, together with its decomposition into additive components. The final model exhibits both plausible extrapolation and interpretable components: a long-term trend, annual periodicity and medium-term deviations; the same components chosen by **?**. We also plot the residuals, showing that there is little obvious structure left in the data.

Airline passenger data Figure 3.4 shows the decomposition produced by applying our method to monthly totals of international airline passengers (**?**). We observe similar components to the previous dataset: a long term trend, annual periodicity and medium-

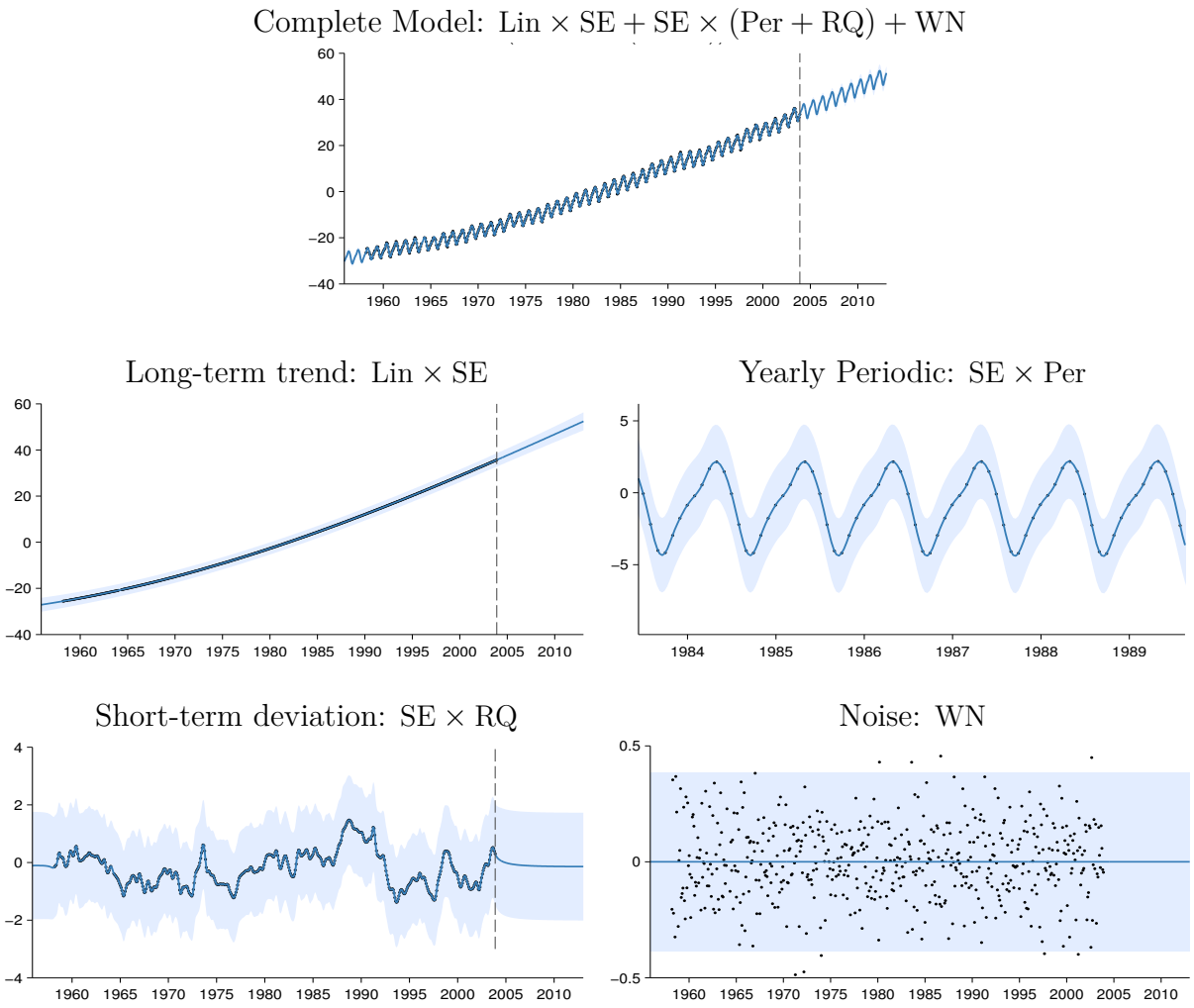


Fig. 3.2 First row: The posterior on the Mauna Loa dataset, after a search of depth 10. Subsequent rows show the automatic decomposition of the time series. The decompositions shows long-term, yearly periodic, medium-term anomaly components, and residuals, respectively. In the third row, the scale has been changed in order to clearly show the yearly periodic structure.

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.

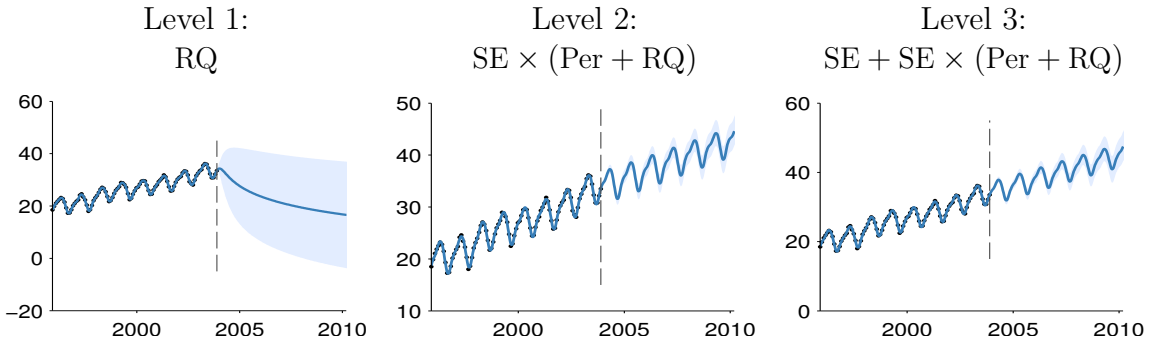


Fig. 3.3 Posterior mean and variance for different depths of kernel search. The dashed line marks the extent of the dataset. In the first column, the function is only modeled as a locally smooth function, and the extrapolation is poor. Next, a periodic component is added, and the extrapolation improves. At depth 3, the kernel can capture most of the relevant structure, and is able to extrapolate reasonably.

3.6 Related Work

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 the basic versions of these methods 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 (GAM) 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 analogous structure through sums of base kernels along different dimensions.

It is possible to add more flexibility to additive models by considering higher-order interactions between different dimensions. Additive Gaussian processes **?** are a GP model whose kernel implicitly sums over all possible products of one-dimensional base kernels. **?** constructs a GP with a composite kernel, summing an SE kernel along each dimension, with an SE-ARD kernel (i.e. a product of SE over all dimensions). Both of these models can be expressed in our grammar.

A closely related procedure is smoothing-splines ANOVA **??**. This model is a linear

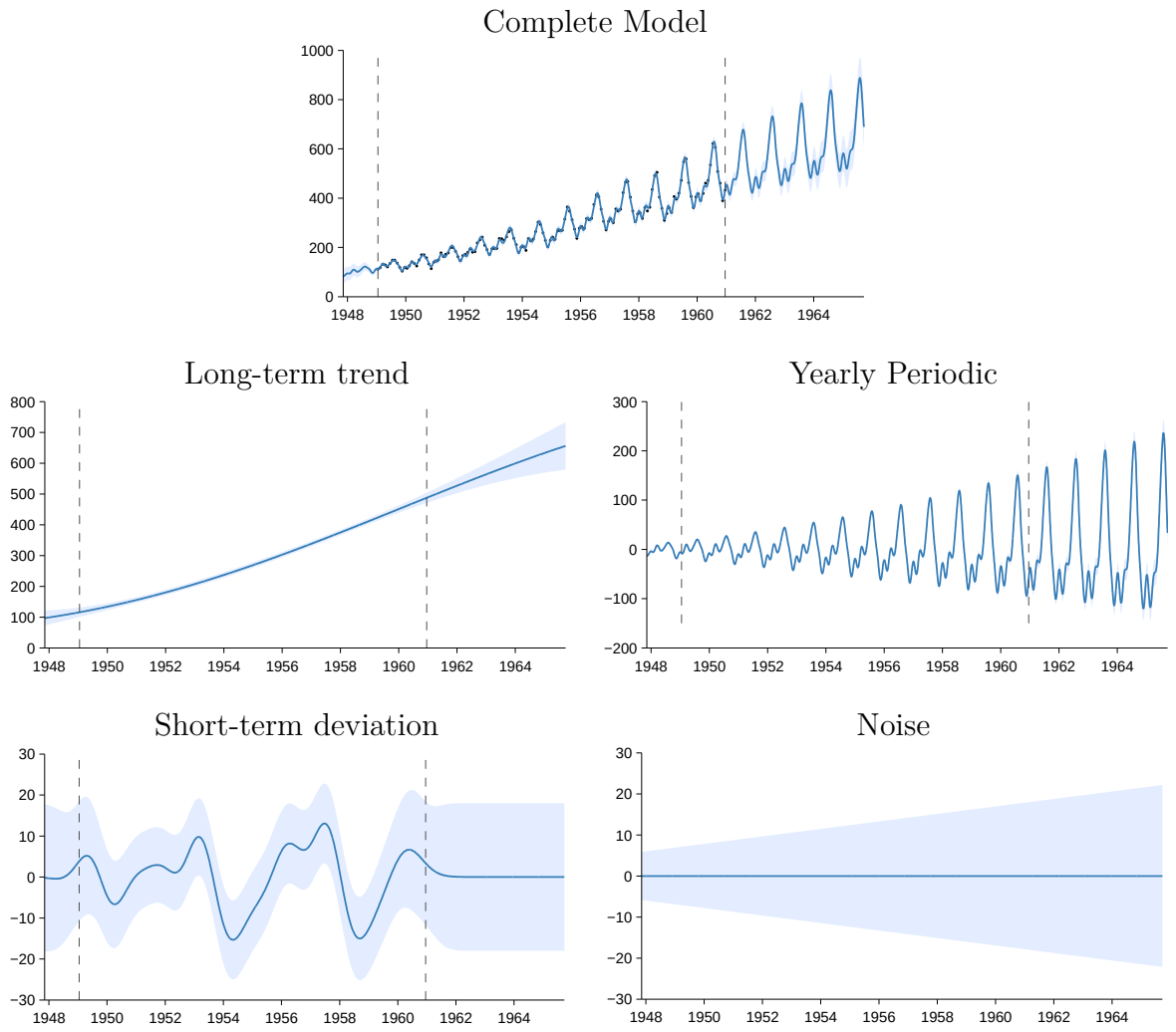


Fig. 3.4 First row: The airline dataset and posterior after a search of depth 10. Subsequent rows: Additive decomposition of posterior into long-term smooth trend, yearly variation, and short-term deviations. Due to the linear kernel, the marginal variance grows over time, making this a heteroskedastic model.

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.

Semiparametric regression (e.g. **?**) 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). In our approach, this can be represented as a sum of SE and Lin.

Kernel learning

There is a large body of work attempting to construct a rich kernel through a weighted sum of base kernels (e.g. **??**). While these approaches find the optimal solution in polynomial time, speed comes at a cost: the component kernels, as well as their parameters, must be specified in advance.

Another approach to kernel learning is to learn an embedding of the data points. **?** learns an embedding of the data into a low-dimensional space, and constructs a fixed kernel structure over that space. This model is typically used in unsupervised tasks and requires an expensive integration or optimisation over potential embeddings when generalizing to test points. **?** use a deep neural network to learn an embedding; this is a flexible approach to kernel learning but relies upon finding structure in the input density, p(x). Instead we focus on domains where most of the interesting structure is in f(x).

? and **?** learn composite kernels for support vector machines and relevance vector machines, using genetic search algorithms. Our work employs a Bayesian search criterion, and goes beyond this prior work by demonstrating the interpretability of the structure implied by composite kernels, and how such structure allows for extrapolation.

Structure discovery

There have been several attempts to uncover the structural form of a dataset by searching over a grammar of structures. For example, **?**, **?** and **?** attempt to learn parametric forms of equations to describe time series, or relations between quantities. Because we learn expressions describing the covariance structure rather than the functions themselves, we are able to capture structure which does not have a simple parametric form.

? learned the structural form of a graph used to model human similarity judgments. Examples of graphs included planes, trees, and cylinders. Some of their discrete graph structures have continous analogues in our own space; e.g. $\mathrm{SE}_1 \times \mathrm{SE}_2$ and $\mathrm{SE}_1 \times \mathrm{Per}_2$ can be seen as mapping the data to a plane and a cylinder, respectively.

? performed a greedy search over a compositional model class for unsupervised learning, using a grammar and a search procedure which parallel our own. This model class contained a large number of existing unsupervised models as special cases and was able to discover such structure automatically from data. Our work is tackling a similar problem, but in a supervised setting.

Building Kernel Functions

? devote 4 pages to manually constructing a composite kernel to model a time series of carbon dioxode concentrations. In the supplementary material, we include a report automatically generated by ABCD for this dataset; our procedure chose a model similar to the one they constructed by hand. Other examples of papers whose main contribution is to manually construct and fit a composite GP kernel are **?** and **?**.

?? and **?** search over a similar space of models as ABCD using genetic algorithms but do not interpret the resulting models.

Kernel Learning

Sparse spectrum GPs (**?**) approximate the spectral density of a stationary kernel function using delta functions which corresponds to kernels of the form $\sum \cos$. Similarly, **?** introduce spectral mixture kernels which approximate the spectral density using a scale-location mixture of Gaussian distributions corresponding to kernels of the form $\sum \mathrm{SE} \times \cos$. Both demonstrate, using Bochner's theorem (**?**), that these kernels can approximate any stationary covariance function. Our language of kernels includes both of these kernel classes (see table 3.1).

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. **?**). These approaches find the optimal solution in polynomial time but only if the component kernels and parameters are pre-specified. We compare to a Bayesian variant of MKL in section 3.7 which is expressed as a restriction of our language of kernels.

Equation learning

?, **?** and **?** learn 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 without a simple parametric form.

nonparametric nonstationary covariance learning

TODO: Add a lit review of nonparametric nonstationary covariance learning.

Discovering interpretable structure

Besides allowing faster learning and extrapolation, learning a more structured kernel sometimes has the added benefit of making the resulting model more intepretable. This is a similar motivation as for the use of sparsity-inducing methods: on many real datasets, the signal can be well-predicted by some small subset of the inputs. Identifying this subset allows both better generalization, and a more interpretable model.

Searching over open-ended model spaces

This work was inspired by previous successes at searching over open-ended model spaces: matrix decompositions (**?**) and graph structures (**?**). In both cases, the model spaces were defined compositionally through a handful of components and operators, and models were selected using criteria which trade off model complexity and goodness of fit. Our work differs in that our procedure automatically interprets the chosen model, making the results accessible to non-experts.

Natural-language output

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 built in the areas of video interpretation (**?**) and automated theorem proving (**?**).

3.6.1 Comparison to Equation Learning

We now compare the descriptions generated by ABCD to parametric functions produced by an equation learning system. We show equations produced by Eureqa (**?**) for the data sets shown above, using the default mean absolute error performance metric.

The learned function for the solar irradiance data is

$$\text{Irradiance}(t) = 1361 + \alpha \sin(\beta + \gamma t) \sin(\delta + \epsilon t^2 - \zeta t)$$

where t is time and constants are replaced with symbols for brevity. This equation captures the constant offset of the data, and models the long-term trend with a product of sinusoids, but fails to capture the solar cycle or the Maunder minimum.

The learned function for the airline passenger data is

$$\text{Passengers}(t) = \alpha t + \beta \cos(\gamma - \delta t)\text{logistic}(\epsilon t - \zeta) - \eta$$

which captures the approximately linear trend, and the periodic component with approximately linearly (logistic) increasing amplitude. However, the annual cycle is heavily approximated by a sinusoid and the model does not capture heteroscedasticity.

3.7 Predictive Accuracy

3.7.1 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. While these models have good predictive performance the large number of components can make them less interpretable. We experimented with distributing all products over addition during the search, causing models with many additive components to be more heavily penalized by BIC. We call this procedure ABCD-interpretability, in contrast to the unrestricted version of the search, ABCD-accuracy.

3.7.2 Datasets

We evaluate the performance of the algorithms listed below on 13 real time-series from various domains from the time series data library (**?**); plots of the data can be found at the beginning of the reports in the supplementary material.

3.7.3 Algorithms

We compare ABCD to equation learning using Eureqa (**?**) and 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. **?**), change point modeling (e.g. **???**), spectral mixture kernels (**?**) (spectral kernels) and trend-cyclical-irregular models (e.g. **?**).

We use the default mean absolute error criterion when using Eureqa. All other algorithms can be expressed as restrictions of our modeling language (see table 3.1) so

we perform inference using the same search methodology and selection criterion[1] with appropriate restrictions to the language. For MKL, trend-cyclical-irregular and spectral kernels, the greedy search procedure of ABCD corresponds to a forward-selection algorithm. For squared exponential and linear regression the procedure corresponds to marginal likelihood optimisation. More advanced inference methods are typically used for changepoint modeling but we use the same inference method for all algorithms for comparability.

We restricted to regression algorithms for comparability; this excludes models which regress on previous values of times series, such as autoregressive or moving-average models (e.g. **?**). Constructing a language for this class of time-series model would be an interesting area for future research.

3.7.4 Extrapolation

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

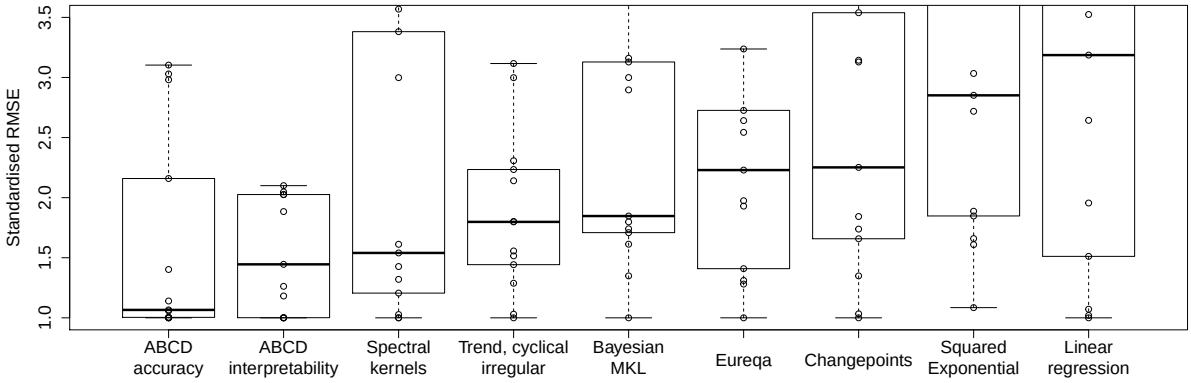


Fig. 3.5 Box plot (showing median and quartiles) of standardised extrapolation RMSE (best performance = 1) on 13 time-series. The methods are ordered by median.

Figure 3.5 shows the standardised RMSEs across algorithms. ABCD-accuracy outperforms ABCD-interpretability but both versions have lower quartiles than all other methods.

[1] We experimented with using unpenalised marginal likelihood as the search criterion but observed overfitting, as is to be expected.

Overall, the model construction methods with greater capacity 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, since very few datasets are parsimoniously explained by a parametric equation.

Not shown on the plot are large outliers for spectral kernels, Eureqa, squared exponential and linear regression with values of 11, 493, 22 and 29 respectively. All of these outliers occurred on a data set with a large discontinuity (see the call centre data in the supplementary material).

Interpolation To test the ability of the methods to interpolate, we randomly divided each data set into equal amounts of training data and testing data. The results are similar to those for extrapolation and are included in the appendix.

3.8 High-dimensional Prediction

ABCD can also be applied to multidimensional regression problems. To evaluate the predictive accuracy of our method in a high-dimensional setting, we extended the comparison of **?** to include our method. We performed 10-fold cross validation on 5 datasets, comparing 5 methods in terms of MSE and predictive likelihood. The data sets had dimensionalities ranging from 4 to 13, and the number of data points ranged from 150 to 450. Our structure search was run up to depth 10, using the SE and RQ base kernel families. All GP parameter optimisation was performed by automated calls to the GPML toolbox available at `http://www.gaussianprocess.org/gpml/code/`.

The comparison included three methods with fixed kernel families: Additive GPs, Generalized Additive Models (GAM), and a GP with a standard SE kernel using Automatic Relevance Determination (GP SE-ARD). Also included was the related kernel-search method of Hierarchical Kernel Learning (HKL).

Results are presented in table 6.2. Our method with all base kernels was always the best performing method, or the difference in performance was not statiscally significant at the 5% level.

3.8.1 Validation on Synthetic Data

Because it is difficult to visualize the structures discovered in multiple dimensions, it is difficult to tell from predictive accuracy alone if the search procedure is finding all

Table 3.2 Comparison of multidimensional regression performance. Bold results are not significantly different from the best-performing method in each experiment, in a paired t-test with a p-value of 5%.

	Mean Squared Error (MSE)					Negative Log-Likelihood				
Method	bach	concrete	puma	servo	housing	bach	concrete	puma	servo	housing
Linear reg.	1.031	0.404	0.641	0.523	0.289	3.430	1.403	1.881	2.678	1.052
GAM	1.259	0.149	0.598	0.281	0.161	2.708	0.467	1.195	1.800	0.457
HKL	**0.199**	0.147	0.346	0.199	0.151	-	-	-	-	-
GP SE-ARD	**0.045**	0.157	**0.317**	**0.126**	**0.092**	**0.869**	0.398	**0.843**	1.429	0.207
Additive GP	**0.045**	**0.089**	**0.316**	**0.110**	0.102	**0.869**	**0.114**	**0.841**	1.309	0.194
SE, RQ Search	**0.044**	**0.087**	**0.315**	**0.102**	**0.082**	**0.859**	**0.065**	**0.840**	1.265	**0.059**
SE, RQ, Lin, Per	**0.509**	**0.079**	**0.321**	**0.094**	**0.112**	**1.357**	**0.114**	**0.837**	**0.573**	**0.151**

structure present. To address this questions, we validated 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 3.3 Kernels chosen by our method on synthetic data generated using known kernel structures. D denotes the dimension of the functions being modeled. SNR indicates the signal-to-noise ratio. Dashes - indicate no structure was found.

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	$\mathrm{SE}_1 + \mathrm{SE}_2$	$\mathrm{Lin}_1 + \mathrm{SE}_2$	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	$\mathrm{SE}_1 \times \mathrm{SE}_2$	$\mathrm{Lin}_1 \times \mathrm{SE}_2$	Lin_2
$\mathrm{SE}_1 \times \mathrm{SE}_2 + \mathrm{SE}_2 \times \mathrm{SE}_3$	4	$\mathrm{SE}_1 \times \mathrm{SE}_2 + \mathrm{SE}_2 \times \mathrm{SE}_3$	$\mathrm{SE}_1 + \mathrm{SE}_2 \times \mathrm{SE}_3$	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 3.3 shows the results. The first column lists the true kernels we used to generate the data. Subscripts indicate which dimension each kernel was applied to. Subsequent columns show the dimensionality D of the input space, and the kernels chosen by our search for different SNRs. Dashes - indicate that no kernel had a higher

marginal likelihood than modeling the data as i.i.d. Gaussian noise.

For the highest SNR, the method finds all relevant structure in all but one case. The reported additional linear structure is explainable by 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 overfitting.

3.9 Discussion

Towards the goal of automating statistical modeling we have presented a system which constructs an appropriate model from an open-ended language and automatically generates detailed reports that describe patterns in the data captured by the model. We have demonstrated that our procedure can discover and describe a variety of patterns on several time series. Our procedure's extrapolation and interpolation performance on time-series are state-of-the-art compared to existing model construction techniques. We believe this procedure has the potential to make powerful statistical model-building techniques accessible to non-experts.

Towards the goal of automating the choice of kernel family, we introduced a space of composite kernels defined compositionally as sums and products of a small number of base kernels. The set of models included in this space includes many standard regression models. We proposed a search procedure for this space of kernels which parallels the process of scientific discovery.

We found that the learned structures are often capable of accurate extrapolation in complex time-series datasets, and are competitive with widely used kernel classes and kernel combination methods on a variety of prediction tasks. The learned kernels often yield decompositions of a signal into diverse and interpretable components, enabling model-checking by humans. We believe that a data-driven approach to choosing kernel structures automatically can help make nonparametric regression and classification methods accessible to non-experts.

We hope that the ABCD algorithm will help replace the current and often opaque art of kernel engineering with a more transparent science of automated kernel construction.

We demonstrate that the properties of Gaussian processes allow for an automatic, modular description generation procedure, through graphs illustrating interpretable decomposition of the posterior.

Whether or not such modularity and interpretability is present in other open-ended

model classes is an open question.

3.10 Future work

While we focus on Gaussian process regression, we believe our kernel search method can be extended to other supervised learning frameworks such as classification or ordinal regression, or to other kinds of kernel architectures such as kernel SVMs.

Chapter 4

Automatically Describing Structured Covariance Functions

The compositional structure of the language allows us to develop a method for automatically translating components of the model into natural-language descriptions of patterns in the data. In this chapter, we describe how ABCD generates natural-language descriptions of the models found by the search procedure. We also show examples of automatically generated reports which highlight interpretable features discovered in a variety of data sets (e.g. figure 4.3).

4.1 Building Noun Phrases

There are two main features of our language of GP models that allow description to be performed automatically. First, the sometimes complicated kernel expressions found can be simplified into a sum of products. A sum of kernels corresponds to a sum of functions so each product can be described separately. Second, each kernel in a product modifies the resulting model in a consistent way. Therefore, we can choose one kernel to be described as a noun, with all others described using adjectives.

Simplification Rules

We convert each kernel expression into a standard, simplified form. We do this by first distributing all products of sums into a sum of products. Next, we apply several simplifications to the kernel expression: The product of two SE kernels is another SE with different parameters. Multiplying WN by any stationary kernel (C, WN, SE, or

Per) gives another WN kernel. Multiplying any kernel by C only changes the parameters of the original kernel.

After applying these rules, the kernel can as 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 WN, C, 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.

Because sums of kernels correspond to sums of functions, we can describe each product of kernels separately.

Each kernel in a product modifies a model in a consistent way

This allows us 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 kernel modifies a model and how this can be described in natural language:

- **Multiplication by SE** removes long range correlations from a model since $\mathrm{SE}(x, x')$ decreases monotonically to 0 as $|x - x'|$ increases. This can be described as making an existing model's correlation structure 'local' or 'approximate'.

- **Multiplication by Lin** is equivalent to multiplying the function being modeled by a linear function. If $f(x) \sim \mathrm{GP}(0, k)$, then $xf(x) \sim \mathrm{GP}(0, k \times \mathrm{Lin})$. This causes the standard deviation of the model to vary linearly without affecting the correlation and can be described as e.g. 'with linearly increasing standard deviation'.

- **Multiplication by $\boldsymbol{\sigma}$** is equivalent to multiplying the function being modeled by a sigmoid which means that the function goes to zero before or after some point. This can be described as e.g. 'from [time]' or 'until [time]'.

- **Multiplication by Per** modifies the correlation structure in the same way as multiplying the function by an independent periodic function. Formally, if $f_1(x) \sim \mathrm{GP}(0, k_1)$ and $f_2(x) \sim \mathrm{GP}(0, k_2)$ then

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

This can be loosely described as e.g. 'modulated by a periodic function with a period of [period] [units]'.

Constructing a complete description of a product of kernels

We choose one kernel to act as a noun which is then described by the functions it encodes for when unmodified e.g. 'smooth function' for SE. Modifiers corresponding to the other kernels in the product are then appended to this description, forming a noun phrase of the form:

$$\text{Determiner} + \text{Premodifiers} + \text{Noun} + \text{Postmodifiers}$$

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

$$\underbrace{\text{SE}}_{\text{approximately}} \times \underbrace{\text{Per}}_{\text{periodic function}} \times \underbrace{\text{Lin}}_{\text{with linearly growing amplitude}} \times \underbrace{\boldsymbol{\sigma}}_{\text{until 1700.}}$$

where Per has been selected as the head noun.

In principle, any assignment of kernels in a product to these different phrasal roles is possible, but in practice we found certain assignments to produce more interpretable phrases than others. The head noun is chosen according to the following ordering:

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

i.e. Per is always chosen as the head noun when present.

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

4.1.1 Worked Example

Suppose we start with a kernel of the form

$$\text{SE} \times (\text{WN} \times \text{Lin} + \text{CP}(\text{C}, \text{Per})).$$

This is converted to a sum of products:

$$\mathrm{SE} \times \mathrm{WN} \times \mathrm{Lin} + \mathrm{SE} \times \mathrm{C} \times \boldsymbol{\sigma} + \mathrm{SE} \times \mathrm{Per} \times \bar{\boldsymbol{\sigma}}.$$

which is simplified to

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

To describe the first component, the head noun description for WN, 'uncorrelated noise', is concatenated with a modifier for Lin, 'with linearly increasing standard deviation'. The second component is described as 'A smooth function with a lengthscale of [lengthscale] [units]', corresponding to the SE, 'which applies until [changepoint]', which corresponds to the $\boldsymbol{\sigma}$. Finally, the third component is described as 'An approximately periodic function with a period of [period] [units] which applies from [changepoint]'.

We demonstrate the ability of our procedure to discover and describe a variety of patterns on two time series.

4.2 Example: Summarizing 400 Years of Solar Activity

We show excerpts from the report automatically generated on annual solar irradiation data from 1610 to 2011 (figure 4.1). This time series has two pertinent features: a

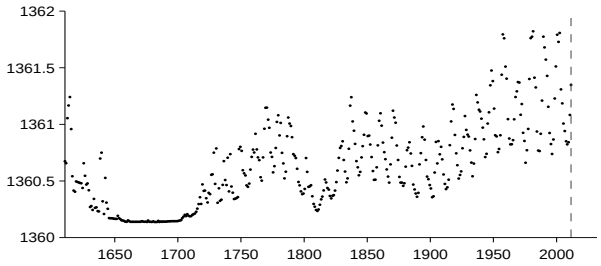


Fig. 4.1 Solar irradiance data.

roughly 11-year cycle of solar activity, and a period lasting from 1645 to 1715 with much smaller variance than the rest of the dataset. This flat region corresponds to the Maunder minimum, a period in which sunspots were extremely rare (**?**). ABCD clearly identifies these two features, as discussed below.

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.

This component explains 71.5% of the residual variance; this increases the total variance explained from 72.8% to 92.3%. The addition of this component reduces the cross validated MAE by 16.82% from 0.18 to 0.15.

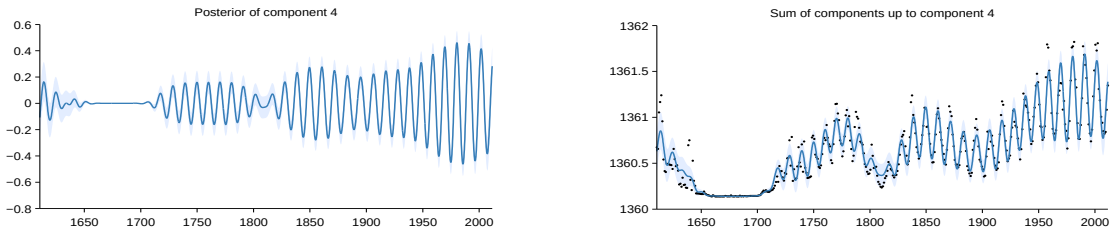


Figure 8: Pointwise posterior of component 4 (left) and the posterior of the cumulative sum of components with data (right)

Fig. 4.2 Extract from an automatically-generated report describing the model components discovered by automatic model search. This part of the report isolates and describes the approximately 11-year sunspot cycle, also noting its disappearance during the 16th century, a time known as the Maunder minimum (**?**).

Figure 4.4 shows the natural-language summaries of the top four components chosen by ABCD. From these short summaries, we can see that our system has identified the Maunder minimum (second component) and 11-year solar cycle (fourth component). These components are visualized in figures 4.5 and 4.3, respectively. The third component corresponds to long-term trends, as visualized in figure 4.6.

4.3 Example: Describing Heteroscedasticity in Air Traffic Data

Next, we present the analysis generated by our procedure on international airline passenger data (figure 4.7). The model constructed by ABCD has four components: Lin + SE × Per × Lin + SE + WN × Lin, with descriptions given in figure 4.8.

The second component (figure 4.9) is accurately described as approximately (SE) periodic (Per) with linearly growing amplitude (Lin). By multiplying a white noise kernel by a linear kernel, the model is able to express heteroscedasticity (figure 4.10).

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.

This component explains 71.5% of the residual variance; this increases the total variance explained from 72.8% to 92.3%. The addition of this component reduces the cross validated MAE by 16.82% from 0.18 to 0.15.

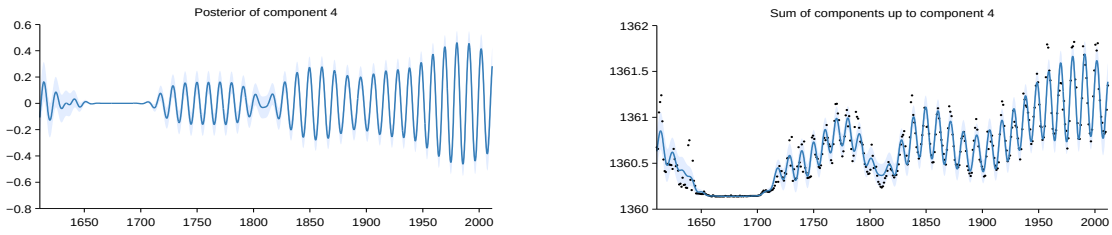


Figure 8: Pointwise posterior of component 4 (left) and the posterior of the cumulative sum of components with data (right)

Fig. 4.3 Extract from an automatically-generated report describing the model components discovered by automatic model search. This part of the report isolates and describes the approximately 11-year sunspot cycle, also noting its disappearance during the 16th century, a time known as the Maunder minimum (**?**).

The structure search algorithm has identified eight 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 6 additive components explain 99.7% 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 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.

Fig. 4.4 Automatically generated descriptions of the components discovered by ABCD on the solar irradiance data set. The dataset has been decomposed into diverse structures with simple descriptions.

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

This component explains 37.4% of the residual variance; this increases the total variance explained from 0.0% to 37.4%. The addition of this component reduces the cross validated MAE by 31.97% from 0.33 to 0.23.

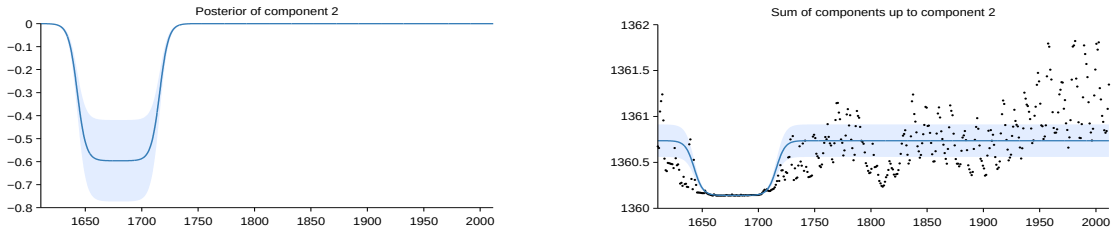


Figure 4: Pointwise posterior of component 2 (left) and the posterior of the cumulative sum of components with data (right)

Fig. 4.5 One of the learned components corresponds 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.

This component explains 56.6% of the residual variance; this increases the total variance explained from 37.4% to 72.8%. The addition of this component reduces the cross validated MAE by 21.08% from 0.23 to 0.18.

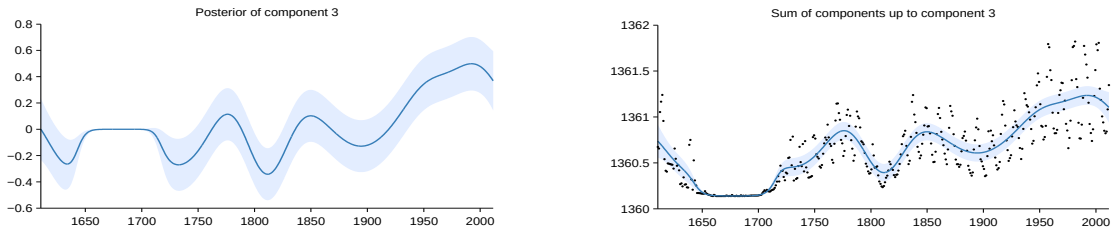


Figure 6: Pointwise posterior of component 3 (left) and the posterior of the cumulative sum of components with data (right)

Fig. 4.6 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.

4.4 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 built in the areas of video interpretation (**?**) and automated theorem proving

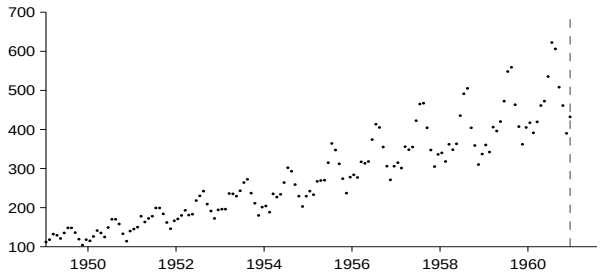


Fig. 4.7 International airline passenger monthly volume (e.g. **?**).

The structure search algorithm has identified four additive components in the data. The first 2 additive components explain 98.5% of the variation in the data as shown by the coefficient of determination (R^2) values in table 1. The first 3 additive components explain 99.8% of the variation in the data. After the first 3 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 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.

#	R^2 (%)	ΔR^2 (%)	Residual R^2 (%)	Cross validated MAE	Reduction in MAE (%)
-	-	-	-	280.30	-
1	85.4	85.4	85.4	34.03	87.9
2	98.5	13.2	89.9	12.44	63.4
3	99.8	1.3	85.1	9.10	26.8
4	100.0	0.2	100.0	9.10	0.0

Fig. 4.8 Short descriptions and summary statistics for the four components of the airline model.

(**?**).

4.5 Conclusion

Towards the goal of automating statistical modeling we have presented a system which constructs an appropriate model from an open-ended language and automatically generates detailed reports that describe patterns in the data captured by the model. We have demonstrated that our procedure can discover and describe a variety of patterns

2.2 Component 2 : An approximately periodic function with a period of 1.0 years and with linearly increasing amplitude

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.

This component explains 89.9% of the residual variance; this increases the total variance explained from 85.4% to 98.5%. The addition of this component reduces the cross validated MAE by 63.45% from 34.03 to 12.44.

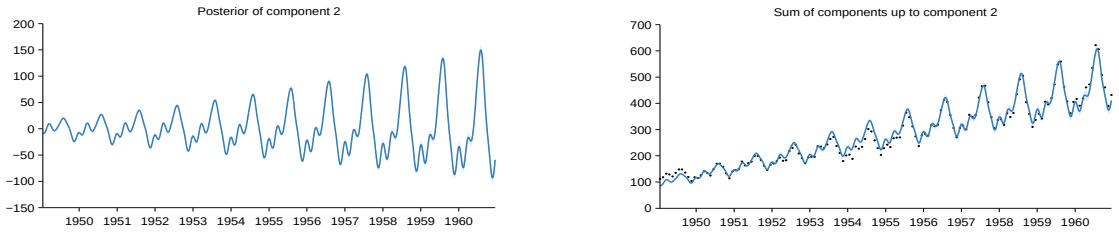


Figure 4: Pointwise posterior of component 2 (left) and the posterior of the cumulative sum of components with data (right)

Fig. 4.9 Capturing non-stationary periodicity in the airline data

2.4 Component 4 : Uncorrelated noise with linearly increasing standard deviation

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

This component explains 100.0% of the residual variance; this increases the total variance explained from 99.8% to 100.0%. The addition of this component reduces the cross validated MAE by 0.00% from 9.10 to 9.10. 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.

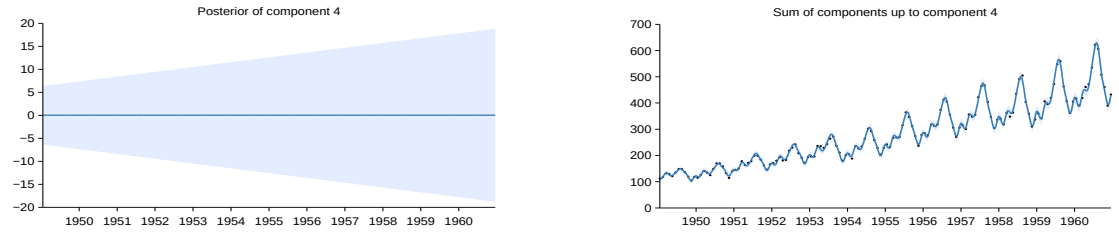


Figure 8: Pointwise posterior of component 4 (left) and the posterior of the cumulative sum of components with data (right)

Fig. 4.10 Modeling heteroscedasticity in the airline dataset.

on several time series. We believe this procedure has the potential to make powerful statistical model-building techniques accessible to non-experts.

Chapter 5

Characterizing Deep Gaussian Process Models

> "On a 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 crucial to good predictive performance. To shed light on this problem, we analyze the analogous problem of constructing useful priors on compositions of functions. Specifically, we study the deep Gaussian process, a type of infinitely-wide, deep neural network. We show that in standard architectures, the representational capacity of the network tends to capture fewer degrees of freedom as the number of layers increases, retaining only a single degree of freedom in the limit. We propose an alternate network architecture which does not suffer from this pathology. We also examine deep covariance functions, obtained by composing infinitely many feature transforms. Lastly, we characterize the class of models obtained by performing dropout on Gaussian processes.

5.1 Introduction

Much recent work on deep networks has focused on weight initialization (**?**), regularization (**?**) and network architecture (**?**). However, the interactions between these different design decisions can be complex and difficult to characterize. We propose to approach the design of deep architectures by examining the problem of assigning priors to nested compositions of functions. Well-defined priors allow us to explicitly examine

the assumptions being made about functions we may wish to learn. If we can identify classes of priors that give our models desirable properties, these in turn may suggest regularization, initialization, and architecture choices that also provide such properties.

Fundamentally, a multilayer neural network implements a composition of vector-valued functions, one per layer. Hence, understanding properties of such function compositions helps us gain insight into deep networks. In this paper, we examine a simple and flexible class of priors on compositions of functions, namely deep Gaussian processes (**?**). Deep GPs are simply priors on compositions of vector-valued functions, where each output of each layer is drawn independently from a GP prior:

$$\boldsymbol{f}^{(1:L)}(\mathbf{x}) = \boldsymbol{f}^{(L)}(\boldsymbol{f}^{(L-1)}(\dots \boldsymbol{f}^{(2)}(\boldsymbol{f}^{(1)}(\mathbf{x}))\dots)) \tag{5.1}$$

$$\boldsymbol{f}_d^{(\ell)} \overset{\text{ind}}{\sim} \text{GP}\left(0, k_d^{\ell}(\mathbf{x}, \mathbf{x}')\right) \tag{5.2}$$

These models correspond to a certain type of infinitely-wide multi-layer perceptron (MLP), and as such make canonical candidates for generative models of functions that closely relate to neural networks.

By characterizing these models, this paper shows that representations based on repeated composition of independently-initialized functions exhibit a pathology where the representation becomes invariant to all but one direction of variation. This corresponds to an eventual debilitating decrease in the information capacity of networks as a function of their number of layers. However, we will demonstrate that a simple change in architecture — namely, connecting the input to each layer — fixes this problem.

We also present two related analyses: first, we examine the properties of a arbitrarily deep fixed feature transforms ("deep kernels"). Second, we characterise the prior obtained by performing dropout on GPs, showing equivalences to existing models.

5.2 Relating Deep Neural Nets and Deep Gaussian Processes

5.2.1 Single-layer Models

In the typical definition of an MLP, the hidden units of the first layer are defined as:

$$\mathbf{h}^{(1)}(\mathbf{x}) = \sigma\left(\mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}\right) \tag{5.3}$$

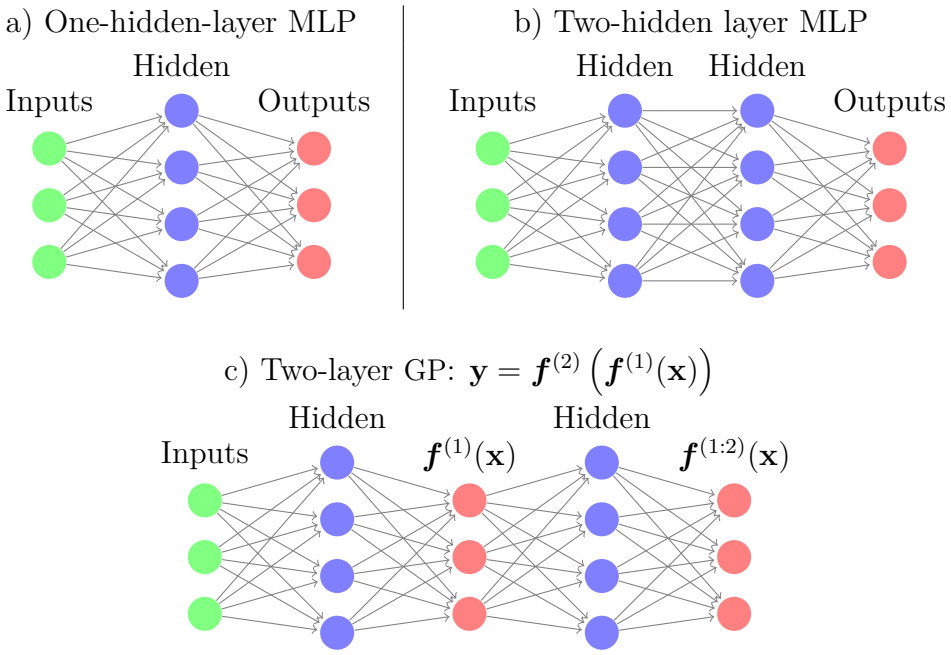


Fig. 5.1 a): GPs can be understood as a one-hidden-layer MLP with infinitely many hidden units. There are two interpretations of deep GPs as neural networks: b): As a neural network with a finite number of hidden units, each with a different non-parametric activation function. c) Alternatively, we can consider every second layer to be a random linear combination of an infinite number of fixed, parametric hidden units.

where $\mathbf{h}$ are the hidden unit activations, $\mathbf{b}$ is a bias vector, $\mathbf{W}$ is a weight matrix and σ is a one-dimensional nonlinear function applied element-wise. The output vector $f(\mathbf{x})$ is simply a weighted sum of these hidden unit activations:

$$f(\mathbf{x}) = \mathbf{V}^{(1)}\sigma\left(\mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}\right) = \mathbf{V}^{(1)}\mathbf{h}^{(1)}(\mathbf{x}) \tag{5.4}$$

where $\mathbf{V}^{(1)}$ is another weight matrix.

There exists a correspondence between one-layer MLPs and GPs (**?**). GPs can be viewed as a prior on neural networks with infinitely many hidden units, and unknown weights. More precisely, for any model of the form

$$f(\mathbf{x}) = \frac{1}{K}\boldsymbol{\alpha}^{\mathsf{T}}\mathbf{h}(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K}\alpha_i h_i(\mathbf{x}), \tag{5.5}$$

with fixed features $[h_1(\mathbf{x}), \ldots, h_K(\mathbf{x})]^{\mathsf{T}} = \mathbf{h}(\mathbf{x})$ and i.i.d. α's with zero mean and finite variance σ^2, the central limit theorem implies that as the number of features

K grows, any two function values $f(\mathbf{x}), f(\mathbf{x}')$ have a joint distribution approaching $\mathcal{N}\left(0, \frac{\sigma^2}{K}\sum_{i=1}^{K} h_i(\mathbf{x})h_i(\mathbf{x}')\right)$. 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 α 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 features $\mathbf{h}(\mathbf{x})$ of the kernel correspond to the hidden units of the MLP.

5.2.2 Multiple Hidden Layers

In an MLP, the ℓth layer units are given by the recurrence

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

This architecture is shown in figure 5.1b. For example, if we extend the model given by (5.4) to have two layers of feature mappings, the resulting model is

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

If the features $\mathbf{h}(\mathbf{x})$ are considered fixed with only the last layer weights $\boldsymbol{\alpha}$ unknown, this model corresponds to a GP with a "deep kernel":

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

These models, examined in section 5.6, imply a fixed representation as opposed to a prior over representations, which is what we wish to analyze in this paper.

To construct a neural network with fixed nonlinearities corresponding to a deep GP, one must introduce a second layer in between each infinitely-wide set of fixed basis functions, as in figure 5.1c. The D_ℓ outputs $\boldsymbol{f}^{(\ell)}(\mathbf{x})$ in between each layer are weighted sums (with unknown weights) of the fixed hidden units of the layer below, and the next layer's hidden units depend only on these D_ℓ outputs.

This alternating-layer architecture has an interpretation as a series of linear infor-

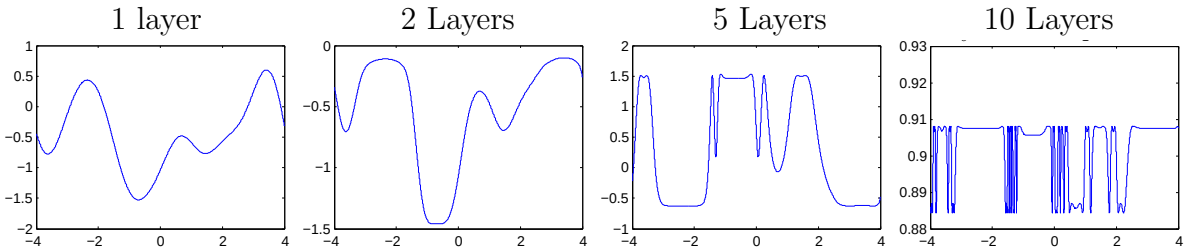


Fig. 5.2 One-dimensional draws from a deep gp prior. After a few layers, the functions begin to be either nearly flat, or highly varying, everywhere. This is a consequence of the distribution on derivatives becoming heavy-tailed.

mation bottlenecks. We can simply substitute (5.4) into (5.6) to get

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

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_ℓ matrices $\mathbf{W}^{(\ell)}\mathbf{V}^{(\ell-1)}$.

A more direct way to construct a network architecture corresponding to a deep GP is to integrate out all $\mathbf{V}^{(\ell)}$, 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 ℓ^{th} layer. This second view lets us compare deep GP models to standard neural net architectures more directly.

5.3 Characterizing Deep Gaussian Processes

In this section, we develop several theoretical results that explore the behavior of deep GPs as a function of their depth. This will allow us in section 5.4 to formally identify a pathology that emerges in very deep networks.

Specifically, we will show that the size of the derivative of a one-dimensional deep GP becomes log-normal distributed as the network becomes deeper. We'll also show that the Jacobian of a multivariate deep GP is a product of independent Gaussian matrices with independent entries.

5.3.1 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:

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

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

By the chain rule, the derivative of a one-dimensional deep GP is simply a product of its (independent) derivatives. 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 log of the magnitude of the derivatives has 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:

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

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 5.2 shows this behavior in a draw from a 1D deep GP prior, at varying depths. 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.

5.3.2 Distribution of the Jacobian

We now derive the distribution on Jacobians of multivariate functions drawn from a deep GP prior.

Lemma 5.3.1. *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 w.r.t. input dimensions d_1 and d_2 of vector $\mathbf{x}$ are given by **?**:

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

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

In the case of the multivariate squared-exp kernel, the covariance between derivatives has the form:

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

Lemma 5.3.2. *The Jacobian of a set of D functions $\mathbb{R}^D \to \mathbb{R}$ drawn independently from a GP prior with a 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 $J_{ij} = \frac{\partial \boldsymbol{f}_i(\mathbf{x})}{\partial x_j}$. Because we've assumed that the GPs on each output dimension $\boldsymbol{f}_d(\mathbf{x})$ are independent (5.2), it follows that each row of J is independent. Lemma 5.3.1 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 5.3.3. *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.*

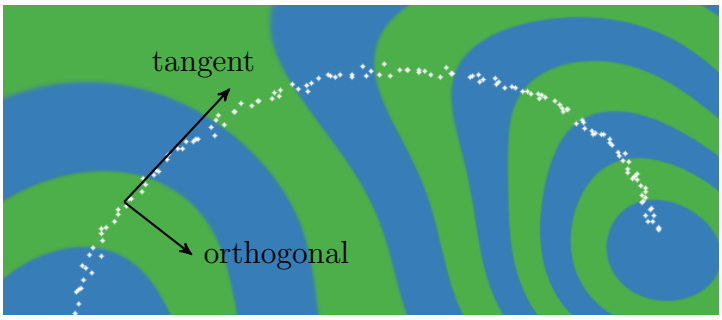


Fig. 5.3 Representing a 1-D data manifold. Colors correspond to the computed representation as a function of the input space. The representation (blue & green) varies in directions tangent to the data manifold (white), preserving information for later layers. The representation is also invariant to directions orthogonal to the manifold, making it robust to noise in those directions.

Proof. When composing L different functions, we'll denote the *immediate* Jacobian of the function mapping from layer $\ell - 1$ to layer ℓ 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 $\boldsymbol{f}^{(L)}(\boldsymbol{f}^{(L-1)}(\dots \boldsymbol{f}^{(3)}(\boldsymbol{f}^{(2)}(\boldsymbol{f}^{(1)}(\mathbf{x})))\dots))$ is

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

By lemma 5.3.2, each $J^{\ell}_{i,j} \overset{\text{ind}}{\sim} \mathcal{N}$, so the complete Jacobian is a product of independent Gaussian matrices, with each entry of each matrix drawn independently. □

Theorem 5.3.3 allows us to analyze the representational properties of a deep Gaussian process by simply examining the properties of products of independent Gaussian matrices, a well-studied object.

5.4 Formalizing a Pathology

? argue that a good latent representation is invariant in directions orthogonal to the manifold on which the data lie. Conversely, a good latent representation must also change in directions tangent to the data manifold, in order to preserve relevant information. Figure 5.3 visualizes this idea. As in **?**, we characterize the representational properties of a function by the singular value spectrum of the Jacobian. In their experiments, the Jacobian was computed at the training points. Because the priors we

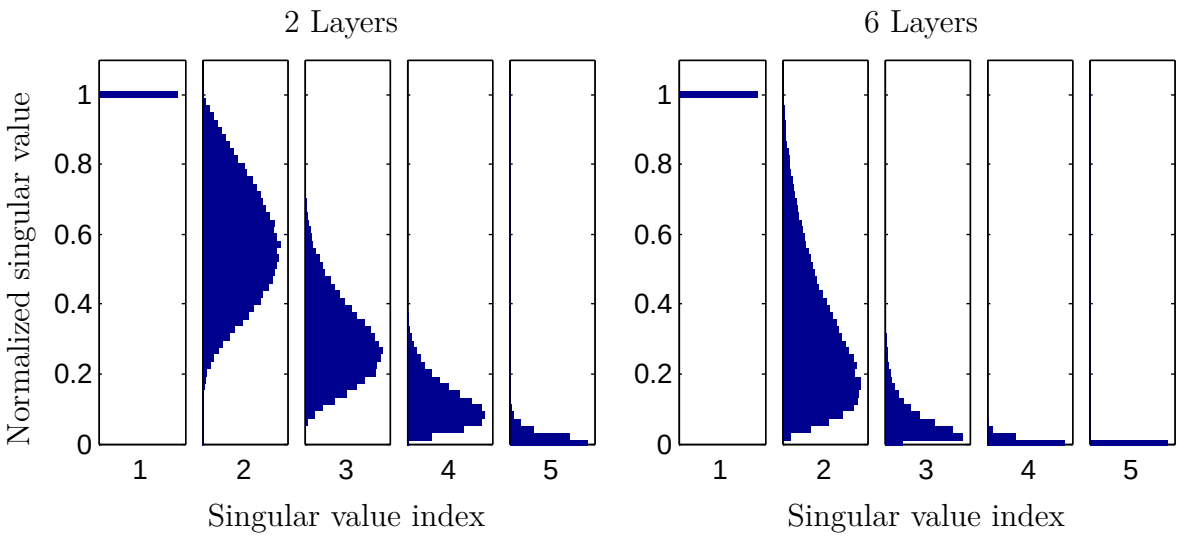


Fig. 5.4 The distribution of singular values relative to the largest, of the Jacobian of a function drawn from 5-dimensional deep GP prior, at 25 and 50 layers deep. As the net gets deeper, the largest singular value becomes much larger than the others. This implies that with high probability, there is only one effective degree of freedom in the representation being computed.

are examining are stationary, the distribution of the Jacobian is identical everywhere. Figure 5.4 shows the singular value spectrum for 5-dimensional deep GPs of different depths. As the net gets deeper, the largest singular value dominates, implying there is usually only one effective degree of freedom in representation being computed.

Figure 5.5 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*, implying that such models are insuitable to model manifolds whose underlying dimensionality is greater than one.

To visualize this pathology in another way, figure 5.6 illustrates a colour-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. 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 nets being used today? In simulations, we found that for deep functions with a fixed latent dimension D, the singular value spectrum remained relatively flat for hundreds of layers as long as $D > 100$. Thus,

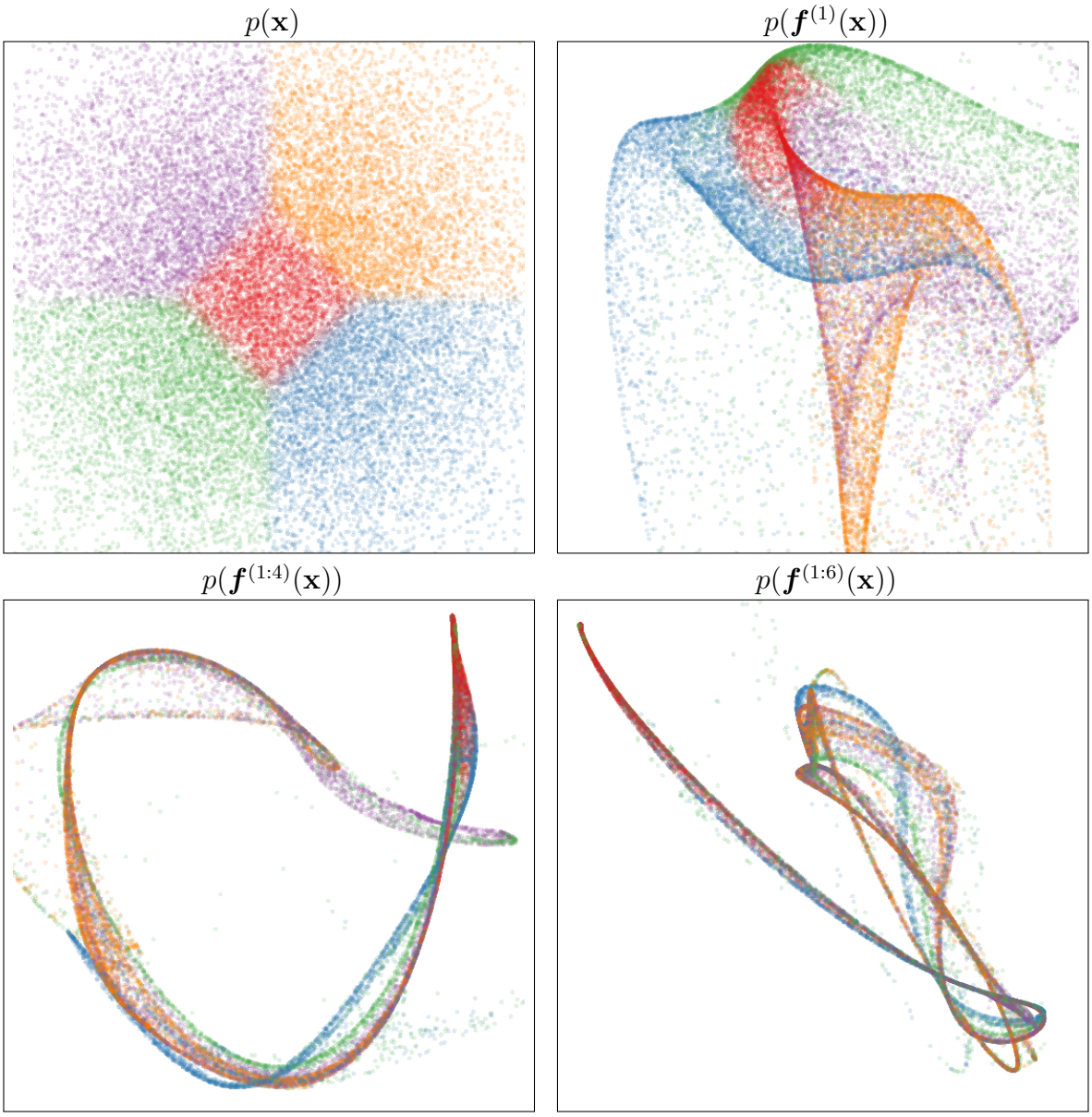


Fig. 5.5 Visualization of draws from a deep GP. A 2-dimensional Gaussian distribution (top left) is warped by successive functions drawn from a GP prior. As the number of layers increases, the density concentrates along one-dimensional filaments.

these pathologies are unlikely to severely affect relatively shallow, wide networks.

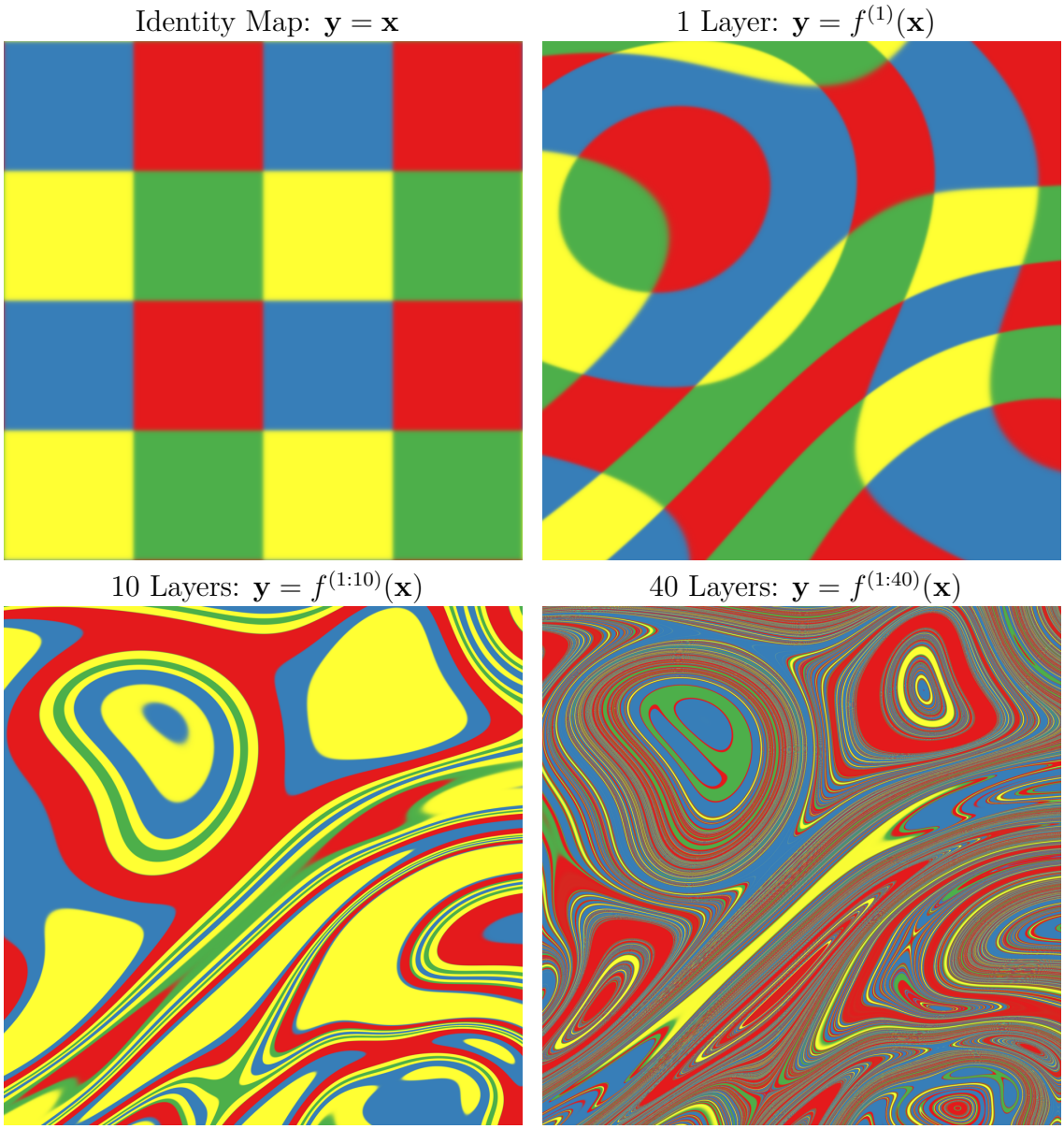


Fig. 5.6 Feature mapping of a deep GP. Colors correspond to the location $\mathbf{y} = \boldsymbol{f}(\mathbf{x})$ that each point is mapped to after being warped by a deep GP. The number of directions in which the color changes rapidly corresponds to the number of large singular values in the Jacobian. Just as the densities in figure 5.5 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 $\mathbf{x}$.

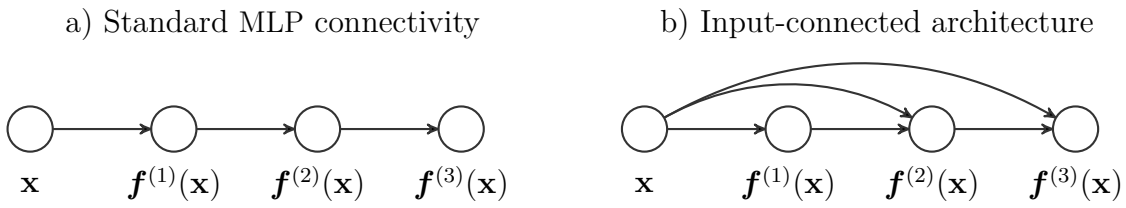


Fig. 5.7 Two different architectures for deep neural networks. The standard architecture connects each layer's outputs to the next layer's inputs. The input-connected architecture also connects the original input $\mathbf{x}$ to each layer.

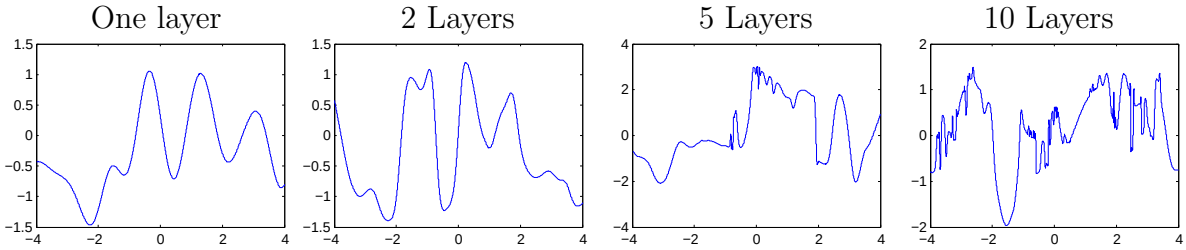


Fig. 5.8 Draws from a 1D deep GP prior with each layer connected to the input. Even after many layers, the functions remain smooth in some regions, while varying rapidly in other regions. Compare to standard-connectivity deep GP draws shown in figure 5.2.

5.5 Fixing the Pathology

Following a suggestion from **?**, we can fix the pathologies exhibited in figures 5.5 and 5.6 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. Figure 5.7 shows a graphical representation of the two connectivity architectures. Similar connections between non-adjacent layers can also be found the primate visual cortex (**?**). Formally, this functional dependence can be written as

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

Draws from the resulting prior are shown in figures 5.8, 5.9 and 5.11. The Jacobian of the composed, input-connected deep function is defined by the recurrence

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

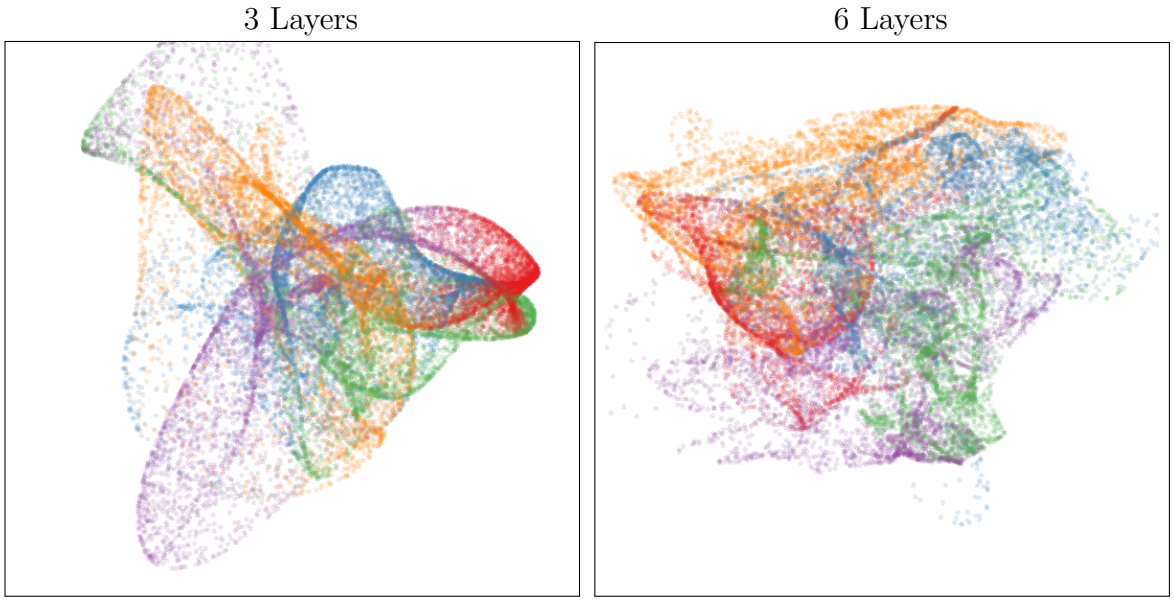


Fig. 5.9 Left: Densities defined by a draw from a deep GP, with each layer connected to the input $\mathbf{x}$. As depth increases, the density becomes more complex without concentrating along filaments.

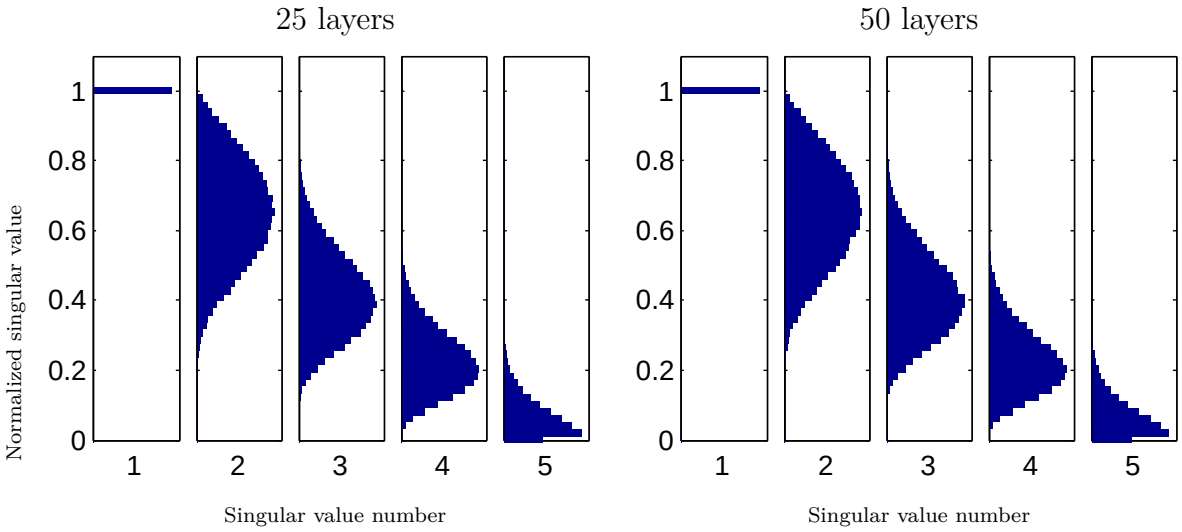


Fig. 5.10 The distribution of singular values drawn from 5-dimensional input-connected deep GP priors, 25 and 50 layers deep. The singular values remain roughly the same scale as one another.

Figure 5.10 shows that with this architecture, even 50-layer deep GPs have well-behaved singular value spectra.

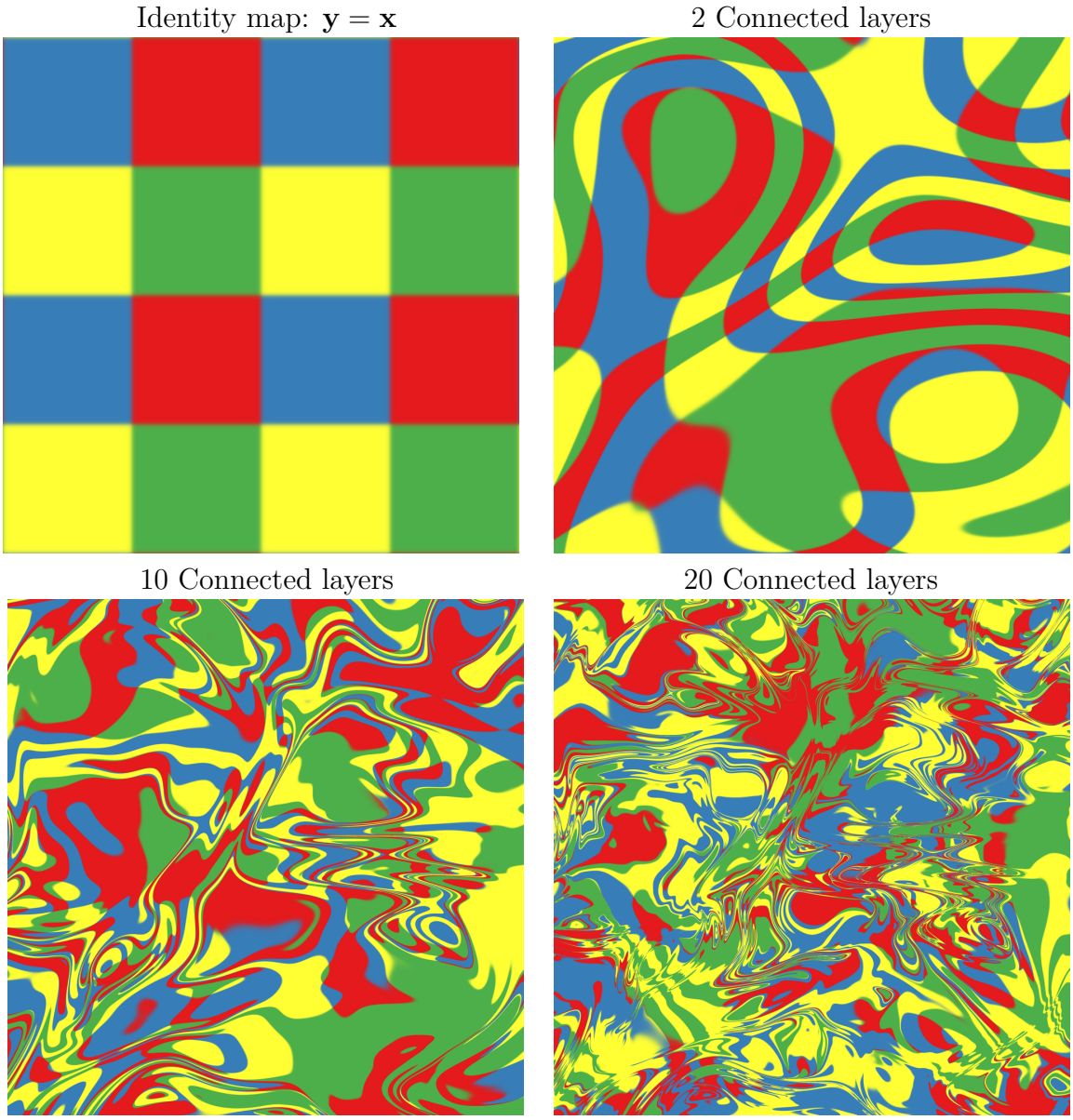


Fig. 5.11 Feature mapping of a deep GP with each layer connected to the input $\mathbf{x}$. Just as the densities in figure 5.9 remained locally two-dimensional even after many transformations, in this mapping there are often two directions that one can move locally in $\mathbf{x}$ to in order to change the values of $\boldsymbol{f}(\mathbf{x})$. This means that the prior puts mass on representations which sometimes depend on all aspects of the input. Compare to figure 5.6.

5.6 Deep Kernels

? showed that kernel machines have limited generalization ability when they use a local kernel such as the squared-exp. However, many interesting non-local kernels can be constructed which allow non-trivial extrapolation. For example, 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.

Can we construct other useful kernels by composing fixed feature maps several times, creating deep kernels? **?** constructed kernels of this form, repeatedly applying multiple layers of feature mappings. We can compose the feature mapping of two kernels:

$$k_1(\mathbf{x}, \mathbf{x}') = \mathbf{h}_1(\mathbf{x})^\mathsf{T}\mathbf{h}_1(\mathbf{x}') \tag{5.18}$$

$$k_2(\mathbf{x}, \mathbf{x}') = \mathbf{h}_2(\mathbf{x})^\mathsf{T}\mathbf{h}_2(\mathbf{x}') \tag{5.19}$$

$$(k_1 \circ k_2)(\mathbf{x}, \mathbf{x}') = k_2(\mathbf{h}_1(\mathbf{x}), \mathbf{h}_1(\mathbf{x}')) \tag{5.20}$$

$$= [\mathbf{h}_2(\mathbf{h}_1(\mathbf{x}))]^\mathsf{T} \mathbf{h}_2(\mathbf{h}_1(\mathbf{x}')) \tag{5.21}$$

Composing the squared-exp kernel with any implicit mapping $\mathbf{h}(\mathbf{x})$ has a simple closed form:

$$\begin{aligned} k_{L+1}(\mathbf{x}, \mathbf{x}') &= k_{SE}(\mathbf{h}(\mathbf{x}), \mathbf{h}(\mathbf{x}')) = \\ &= \exp\left(-\frac{1}{2}||\mathbf{h}(\mathbf{x}) - \mathbf{h}(\mathbf{x}')||_2^2\right) \\ &= \exp\left(-\frac{1}{2}\left[\mathbf{h}(\mathbf{x})^\mathsf{T}\mathbf{h}(\mathbf{x}) - 2\mathbf{h}(\mathbf{x})^\mathsf{T}\mathbf{h}(\mathbf{x}') + \mathbf{h}(\mathbf{x}')^\mathsf{T}\mathbf{h}(\mathbf{x}')\right]\right) \\ &= \exp\left(-\frac{1}{2}\left[k_L(\mathbf{x}, \mathbf{x}) - 2k_L(\mathbf{x}, \mathbf{x}') + k_L(\mathbf{x}', \mathbf{x}')\right]\right) \end{aligned} \tag{5.22}$$

Thus, we can express k_{L+1} exactly in terms of k_L.

5.6.1 Infinitely Deep Kernels

What happens when we repeat this composition of feature maps many times, starting with the squared-exp kernel? In the infinite limit, this recursion converges to $k(\mathbf{x}, \mathbf{x}') = 1$ for all pairs of inputs, which corresponds to a prior on constant functions $f(\mathbf{x}) = c$.

A non-degenerate construction As before, we can overcome this degeneracy by connecting the inputs $\mathbf{x}$ to each layer. To do so, we simply augment the feature vector

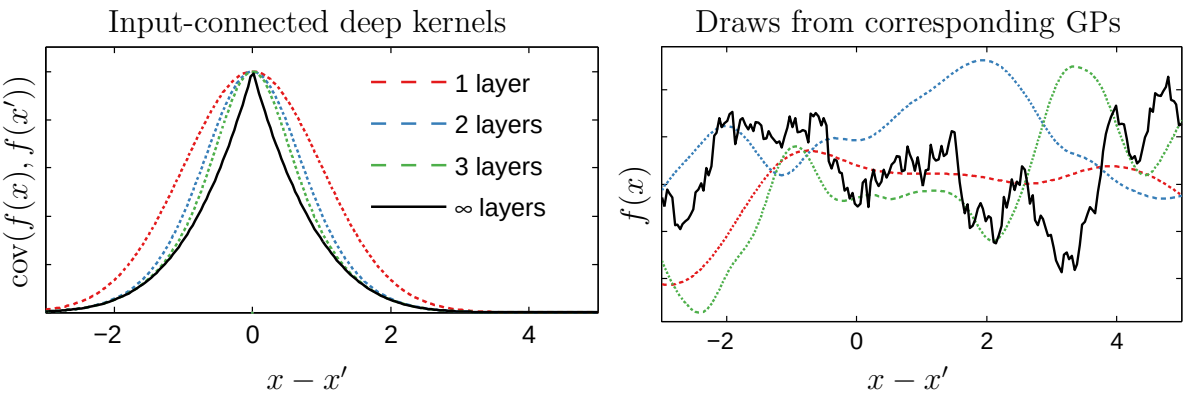


Fig. 5.12 Left: Input-connected deep kernels. By connecting the inputs $\mathbf{x}$ to each layer, the kernel can still depend on its input even after arbitrarily many layers of computation. Right: GP draws using deep input-connected kernels.

$\mathbf{h}_L(\mathbf{x})$ with $\mathbf{x}$ at each layer:

$$k_{L+1}(\mathbf{x},\mathbf{x}') = \exp\left(-\frac{1}{2}\left\|\begin{bmatrix}\mathbf{h}_L(\mathbf{x})\\ \mathbf{x}\end{bmatrix} - \begin{bmatrix}\mathbf{h}_L(\mathbf{x}')\\ \mathbf{x}'\end{bmatrix}\right\|_2^2\right)$$
$$= \exp\left(-\frac{1}{2}\left[k_L(\mathbf{x},\mathbf{x}) - 2k_L(\mathbf{x},\mathbf{x}') + k_L(\mathbf{x}',\mathbf{x}') - ||\mathbf{x}-\mathbf{x}'||_2^2\right]\right) \quad (5.23)$$

For the SE kernel, this repeated mapping satisfies

$$k_\infty(\mathbf{x},\mathbf{x}') - \log\left(k_\infty(\mathbf{x},\mathbf{x}')\right) = 1 + \frac{1}{2}||\mathbf{x}-\mathbf{x}'||_2^2 \quad (5.24)$$

The solution to this recurrence has no closed form, but has a similar shape to the Ornstein-Uhlenbeck covariance $k_{\mathrm{OU}}(x,x') = \exp(-|x-x'|)$ with lighter tails. Samples from a GP prior with this kernel are not differentiable, and are locally fractal.

5.6.2 When are Deep Kernels Useful Models?

Kernels correspond to fixed feature maps, and so kernel learning is an example of implicit representation learning. Such feature maps can capture rich structure (**?**), and can enable many types of generalization, such as affine invariance in images (**?**). **?** used a deep neural network to learn feature transforms for kernels, which learn invariances in an unsupervised manner. The relatively uninteresting properties of the kernels derived in this section simply reflect the fact that an arbitrary deep computation is not usually 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.

5.7 Related Work

Prior work on deep GPs **?** discussed the properties of arbitrarily deep random networks, including those that would give rise to deep GPs. However, deep GPs were first clearly defined and developed by **?**, as a model of hierarchical generative relationships. Deep GPs were first referred to as such in **?**, who developed a variational inference scheme and analyzed the effect of automatic relevance determination in that model.

Nonparametric neural networks **?** proposed a prior on arbitrarily deep Bayesian networks. Their architecture has connections only between adjacent layers, and may also be expected to have similar pathologies to those of deep GPs as the number of layers increases.

Deep Density Networks (**?**) were constructed with invertibility in mind, with penalty terms encouraging the preservation of information about lower layers. These models are a promising alternative approach to alleviating the pathology discussed in this paper.

? introduce GP Regression Networks (GPRN), which define a matrix product of GPs, rather than a composition. The form of the GPRN is

$$\mathbf{y}(\mathbf{x}) = \mathbf{W}(\mathbf{x})\boldsymbol{f}(\mathbf{x}) \qquad \text{with each } f_d, W_{d,j} \overset{\text{iid}}{\sim} \mathcal{GP}(\mathbf{0}, \text{SE} + \text{WN}) \tag{5.25}$$

We can easily define a deep GPRN:

$$\mathbf{y}(\mathbf{x}) = \mathbf{W}^{(3)}(\mathbf{x})\mathbf{W}^{(2)}(\mathbf{x})\mathbf{W}^{(1)}(\mathbf{x})\boldsymbol{f}(\mathbf{x}) \tag{5.26}$$

which might be amenable to a similar analysis to that of section 5.3.

Recurrent networks **?** and **?** analyze a related problem with gradient-based learning in recurrent nets, the "exploding-gradients" problem. Specifically, they note that in recurrent neural networks, the size of the training gradient can grow or shrink exponentially as it is back-propagated, making gadient-based training difficult.

Deep kernels **?** construct deep kernels in a time-series setting, constructing kernels corresponding to infinite-size recurrent neural networks (also called Echo State Net-

works). They also propose concatenating the implicit feature vectors from previous time steps with the current inputs, resulting in an architecture analogous to the input-connected architure proposed by **?**.

Analyses of deep learning **?** perform a layer-wise analysis of deep networks, and note that the performance of MLPs degrades as the number of layers with random weights increases. The importance of network architectures relative to learning has been examined by **?**. **?** looked at the dynamics of learning in deep linear models, as a tractable approximation to deep neural networks.

5.8 Conclusions

In this chapter, we attempted to gain insight into the properties of deep neural networks by characterizing the sorts of functions likeley 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 directions of variation but one. 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.

Chapter 6

Higher-order Additive GPs

In this chapter, we introduce a Gaussian process model of functions which are *additive*. An additive function is one which decomposes into a sum of low-dimensional functions, each depending on only a subset of the input variables. Additive GPs generalize both Generalized Additive Models, and the standard GP models which use squared-exponential kernels. We introduce an expressive but tractable parameterization of the kernel function, which allows efficient evaluation of all input interaction terms, whose number is exponential in the input dimension. The additional structure discoverable by this model results in state-of-the-art predictive power in regression tasks, as well as increased interpretability.

Many popular statistical regression models are of the form:

$$g(y) = f(x_1) + f(x_2) + \cdots + f(x_D). \tag{6.1}$$

Popular examples include logistic regression, linear regression, and Generalized Linear Models (**?**). This family of functions, known as Generalized Additive Models (GAM) (**?**), are typically easy to fit and interpret. Some extensions of this family, such as smoothing-splines ANOVA (**?**), add terms depending on more than one variable. However, such models generally become intractable and difficult to fit as the number of terms increases.

At the other end of the spectrum are kernel-based models, which typically allow the response to depend on all input variables simultaneously. These have the form: $y = f(x_1, x_2, \ldots, x_D)$. A popular example would be a Gaussian process model using a squared-exponential (or Gaussian) kernel. We denote this model as SE-GP. This model is much more flexible than the GAM, but its flexibility makes it difficult to generalize to new combinations of input variables.

In this chapter, we introduce a Gaussian process model that generalizes both GAMs and the SE-GP. This is achieved through a kernel which allow additive interactions of all orders, ranging from first order interactions (as in a GAM) all the way to Dth-order interactions (as in a SE-GP). Although this kernel amounts to a sum over an exponential number of terms, we show how to compute this kernel efficiently, and introduce a parameterization which limits the number of parameters to $\mathcal{O}(D)$. A Gaussian process with this kernel function (an additive GP) constitutes a powerful model that allows one to automatically determine which orders of interaction are important. We show that this model can significantly improve modeling efficacy, and has major advantages for model interpretability.

We can see that a GP with a first-order additive kernel is an example of a GAM: Each function drawn from this model is a sum of orthogonal one-dimensional functions. Compared to functions drawn from the higher-order GP, draws from the first-order GP have more long-range structure.

We can expect many natural functions to depend only on sums of low-order interactions. For example, the price of a house or car will presumably be well approximated by a sum of prices of individual features, such as a sun-roof. Other parts of the price may depend jointly on a small set of features, such as the size and building materials of a house. Capturing these regularities will mean that a model can confidently extrapolate to unseen combinations of features.

6.1 Additive Kernels

We now give a precise definition of additive kernels. We first assign each dimension $i \in \{1 \dots D\}$ a one-dimensional *base kernel* $k_i(x_i, x_i')$. We then define the first order, second order and nth order additive kernel as:

$$k_{add_1}(\mathbf{x}, \mathbf{x}') = \sigma_1^2 \sum_{i=1}^{D} k_i(x_i, x_i') \tag{6.2}$$

$$k_{add_2}(\mathbf{x}, \mathbf{x}') = \sigma_2^2 \sum_{i=1}^{D} \sum_{j=i+1}^{D} k_i(x_i, x_i') k_j(x_j, x_j') \tag{6.3}$$

$$k_{add_n}(\mathbf{x}, \mathbf{x}') = \sigma_n^2 \sum_{1 \le i_1 < i_2 < \ldots < i_n \le D} \left[\prod_{d=1}^{n} k_{i_d}(x_{i_d}, x_{i_d}') \right] \tag{6.4}$$

$$k_{add_D}(\mathbf{x}, \mathbf{x}') = \sigma_n^2 \sum_{1 \le i_1 < i_2 < \ldots < i_D \le D} \left[\prod_{d=1}^{D} k_{i_d}(x_{i_d}, x_{i_d}') \right] = \sigma_D^2 \prod_{d=1}^{D} k_d(x_d, x_d') \tag{6.5}$$

where D is the dimension of our input space, and σ_n^2 is the variance assigned to all nth order interactions. The nth covariance function is a sum of $\binom{D}{n}$ terms. In particular, the Dth order additive covariance function has $\binom{D}{D} = 1$ term, a product of each dimension's covariance function:

$$k_{add_D}(\mathbf{x}, \mathbf{x}') = \sigma_D^2 \prod_{d=1}^{D} k_d(x_d, x_d') \tag{6.6}$$

In the case where each base kernel is a one-dimensional squared-exponential kernel, the Dth-order term corresponds to the multivariate squared-exponential kernel:

$$k_{add_D}(\mathbf{x}, \mathbf{x}') = \sigma_D^2 \prod_{d=1}^{D} k_d(x_d, x_d') = \sigma_D^2 \prod_{d=1}^{D} \exp\Big(-\frac{(x_d - x_d')^2}{2l_d^2} \Big) = \sigma_D^2 \exp\Big(-\sum_{d=1}^{D} \frac{(x_d - x_d')^2}{2l_d^2} \Big) \tag{6.7}$$

also commonly known as the Gaussian kernel. The full additive kernel is a sum of the additive kernels of all orders.

6.1.1 Parameterization

The only design choice necessary in specifying an additive kernel is the selection of a one-dimensional base kernel for each input dimension. Any parameters (such as length-scales) of the base kernels can be learned as usual by maximizing the marginal likelihood of the training data.

In addition to the parameters of each dimension-wise kernel, additive kernels are equipped with a set of D parameters $\sigma_1^2 \dots \sigma_D^2$ controlling how much variance we assign to each order of interaction. These "order variance" parameters have a useful interpretation: the dth order variance hyperparameter controls how much of the target function's variance comes from interactions of the dth order. Table 6.1 shows examples of the variance contributed by the different orders of interaction, learned on real datasets.

On different datasets, the dominant order of interaction estimated by the additive model varies widely. An additive GP with all of its variance coming from the 1st order is equivalent to a GAM; an additive GP with all its variance coming from the Dth order is equivalent to a SE-GP.

Because the variance parameters can specify which degrees of interaction are important, the additive GP can capture many different types of structure. If the function we are modeling is decomposable into a sum of low-dimensional functions, our model can

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

	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

discover this fact and exploit it. Discovering such structure allows long-range extrapolation, as we saw in section 2.4.4. If low-dimensional additive structure is not present, the kernel parameters can specify a suitably flexible model, with interactions between as many variables as necessary.

6.1.2 Efficient Evaluation of Additive Kernels

An additive kernel over D inputs with interactions up to order n has $O(2^n)$ terms. Naïvely summing over these terms quickly becomes intractable. In this section, we show how one can evaluate the sum over all terms in $O(D^2)$.

To efficiently compute the additive kernel, we exploit the fact that the nth order additive kernel corresponds to the nth *elementary symmetric polynomial* (**?**) 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

$$k_{\text{add}_0}(\mathbf{x}, \mathbf{x}') = e_0(k_1, k_2, k_3, k_4) = 1 \tag{6.8}$$

$$k_{\text{add}_1}(\mathbf{x}, \mathbf{x}') = e_1(k_1, k_2, k_3, k_4) = k_1 + k_2 + k_3 + k_4 \tag{6.9}$$

$$k_{\text{add}_2}(\mathbf{x}, \mathbf{x}') = e_2(k_1, k_2, k_3, k_4) = k_1k_2 + k_1k_3 + k_1k_4 + k_2k_3 + k_2k_4 + k_3k_4 \tag{6.10}$$

$$k_{\text{add}_3}(\mathbf{x}, \mathbf{x}') = e_3(k_1, k_2, k_3, k_4) = k_1k_2k_3 + k_1k_2k_4 + k_1k_3k_4 + k_2k_3k_4 \tag{6.11}$$

$$k_{\text{add}_4}(\mathbf{x}, \mathbf{x}') = e_4(k_1, k_2, k_3, k_4) = k_1k_2k_3k_4 \tag{6.12}$$

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

nomials:

$$k_{\text{add}_n}(\mathbf{x}, \mathbf{x}') = e_n(k_1, \ldots, k_D) = \frac{1}{n} \sum_{a=1}^{n} (-1)^{(a-1)} e_{n-a}(k_1, \ldots, k_D) \sum_{i=1}^{D} k_i^a \tag{6.13}$$

The Newton-Girard formulae have time complexity $\mathcal{O}(D^2)$, while computing a sum over an exponential number of terms. Note that the same sum can be computed in time

Conveniently, we can use the same trick to efficiently compute all of the necessary derivatives of the additive kernel with respect to the base kernels. We merely need to remove the kernel of interest from each term of the polynomials:

$$\frac{\partial k_{add_n}}{\partial k_j} = e_{n-1}(k_1, \ldots, k_{j-1}, k_{j+1}, \ldots k_D) \tag{6.14}$$

This trick allows us to optimize the base kernel parameters with respect to the marginal likelihood.

6.1.3 Computational Cost

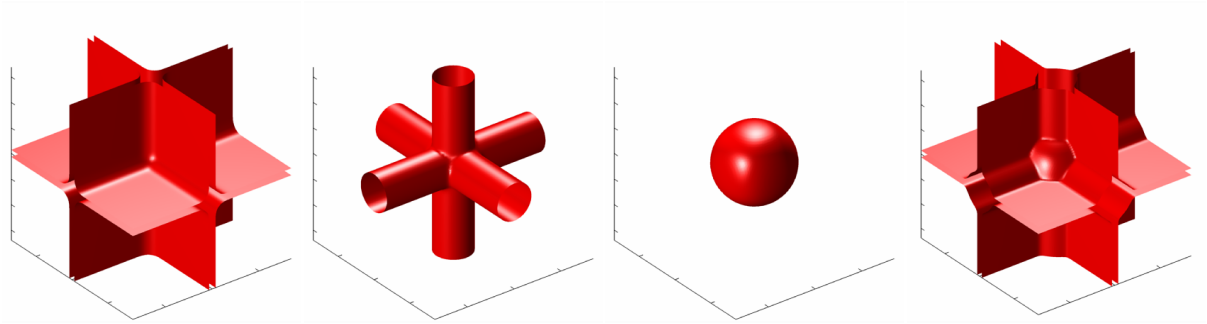
The computational cost of evaluating the Gram matrix of a product kernel (such as the SE kernel) is $\mathcal{O}(N^2 D)$, while the cost of evaluating the Gram matrix of the additive kernel is $\mathcal{O}(N^2 DR)$, where R is the maximum degree of interaction allowed (up to D). In higher dimensions, this can be a significant cost, even relative to the fixed $\mathcal{O}(N^3)$ cost of inverting the Gram matrix. However, as our experiments show, typically only the first few orders of interaction are important for modeling a given function; hence if one is computationally limited, one can simply limit the maximum degree of interaction without losing much accuracy.

6.2 Non-local Interactions

By far the most popular kernels for regression and classification tasks on continuous data are the SE kernel, and the Matérn kernels. These kernels depend only on the scaled Euclidean distance between two points, both having the form:

$$k(\mathbf{x}, \mathbf{x}') = g\left(\sum_{d=1}^{D} \left(\frac{x_d - x'_d}{l_d} \right)^2 \right) \tag{6.15}$$

? argue that models based on squared-exponential kernels are particularly susceptible to the *curse of dimensionality*. They emphasize that the locality of the kernels means that these models cannot capture non-local structure. They argue that many functions that we care about have such structure. Methods based solely on local kernels will require training examples at all combinations of relevant inputs.



1st-order terms $k_1 + k_2 + k_3$ | 2nd-order terms $k_1k_2 + k_2k_3 + k_1k_3$ | 3rd-order terms $k_1k_2k_3$ Squared-exp kernel | All interactions Additive kernel

Fig. 6.1 Isocontours of additive kernels in 3 dimensions. The third-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.

Additive kernels have a much more complex structure, and allow extrapolation based on distant parts of the input space, without spreading the mass of the kernel over the whole space. For example, additive kernels of the second order allow strong non-local interactions between any points which are similar in any two input dimensions. Figure 6.1 provides a geometric comparison between squared-exponential kernels and additive kernels in 3 dimensions.

6.3 Dropout in Gaussian Processes

Dropout is a method for regularizing neural networks (**??**). Training with dropout entails randomly and independently "dropping" (setting to zero) 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, weights are multiplied by $1/p$ at test time. Alternatively, feature activations are multiplied by $1/p$ during training. At test time, the set of models defined by all possible ways of dropping-out neurons is averaged over, usually in an approximate way.

? and **?** 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 examine the priors on functions that result from performing dropout in the one-hidden-layer neural network implicitly defined by a GP (equation (5.4)).

6.3.1 Dropout on Feature Activations

First, we examine the prior that results from randomly dropping features from $\mathbf{h}(\mathbf{x})$ with probability p. If these features have a weight distribution with finite moments $\mathbb{E}[\alpha_i] = \mu, \mathbb{V}[\alpha_i] = \sigma^2$, then the distribution of weights after each one has been dropped out with probability p is:

$$r_i \overset{\text{iid}}{\sim} \text{Ber}(p) \qquad \mathbb{E}[r_i\alpha_i] = p\mu, \quad \mathbb{V}[r_i\alpha_i] = p^2\sigma^2 \; . \tag{6.16}$$

However, after we increase the remaining activations to maintain the same expected activation by multiplying them by $1/p$, the resulting moments are again:

$$\mathbb{E}\left[\frac{1}{p}r_i\alpha_i\right] = \frac{p}{p}\mu = \mu, \quad \mathbb{V}\left[\frac{1}{p}r_i\alpha_i\right] = \frac{p^2}{p^2}\sigma^2 = \sigma^2 \; . \tag{6.17}$$

Thus, dropping out features of an infinitely-wide MLP does not change the model at all, since no individual feature can have more than infinitesimal contribution to the network output.

6.3.2 Dropping Out Inputs

In a GP with kernel $k(\mathbf{x}, \mathbf{x}') = \prod_{d=1}^{D} k_d(x_d, x'_d)$, exact averaging over all possible ways of dropping out inputs with probability $1/2$ results in a mixture of GPs, each depending on only a subset of the inputs:

$$p(f(\mathbf{x})) = \frac{1}{2^D} \sum_{\mathbf{r}\in\{0,1\}^D} \text{GP}\left(0, \prod_{d=1}^{D} k_d(x_d, x'_d)^{r_d}\right) \tag{6.18}$$

We present two results ways to gain intuition about this model.

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

Another way to understand the resulting prior is to note that the dropout mixture (6.18) has the same covariance as

$$f(\mathbf{x}) \sim \mathrm{GP}\left(0, \frac{1}{2^{-2D}} \sum_{\mathbf{r}\in\{0,1\}^D} \prod_{d=1}^{D} k_d(x_d, x'_d)^{r_d}\right) \tag{6.19}$$

A GP whose covariance is a sum of kernels corresponds to a sum of functions, each distributed according to a GP. Therefore, (6.19) describes a sum of 2^D functions, each depending on a different subset of the inputs.

6.4 Related Work

Since additive models are a relatively natural and easy-to-analyze model class, the literature on similar model classes is extensive. We try to give a broad overview in this section.

Additive GPs

Since the non-local structure capturable by additive kernels is necessarily axis-aligned, we can naturally consider that an initial transformation of the input space might allow us to recover non-axis aligned additivity in functions. This avenue was explored by **?**, who developed a linearly-transformed first-order additive GP model, called projection-pursuit GP regression. They further showed that inference in this model was possible in $\mathcal{O}(N)$ time.

? also examined the properties of additive GPs, and proposed a layer-wise optimization strategy for kernel hyperparameters in these models.

? constructed an additive GP using only the first-order and Dth-order terms. This model is motivated by the desire to trade off the interpretability of first-order models with the flexibility of full-order models. Our experiments show that often, the intermediate degrees of interaction contribute most of the variance.

A related functional ANOVA GP model (**?**) decomposes the *mean* function into a weighted sum of GPs. However, the effect of a particular degree of interaction cannot be quantified by that approach. Also, computationally, the Gibbs sampling approach used in (**?**) is disadvantageous.

? previously showed how mixtures of kernels can be learnt by gradient descent in the Gaussian process framework. They call this *Bayesian localized multiple kernel learning*.

However, their approach learns a mixture over a small, fixed set of kernels, while our method learns a mixture over all possible products of those kernels.

Hierarchical Kernel Learning

A similar model class has been recently explored by **?** called Hierarchical Kernel Learning (HKL). HKL uses a regularized optimization framework to learn a weighted sum over an exponential number of kernels which can be computed in polynomial time. This method

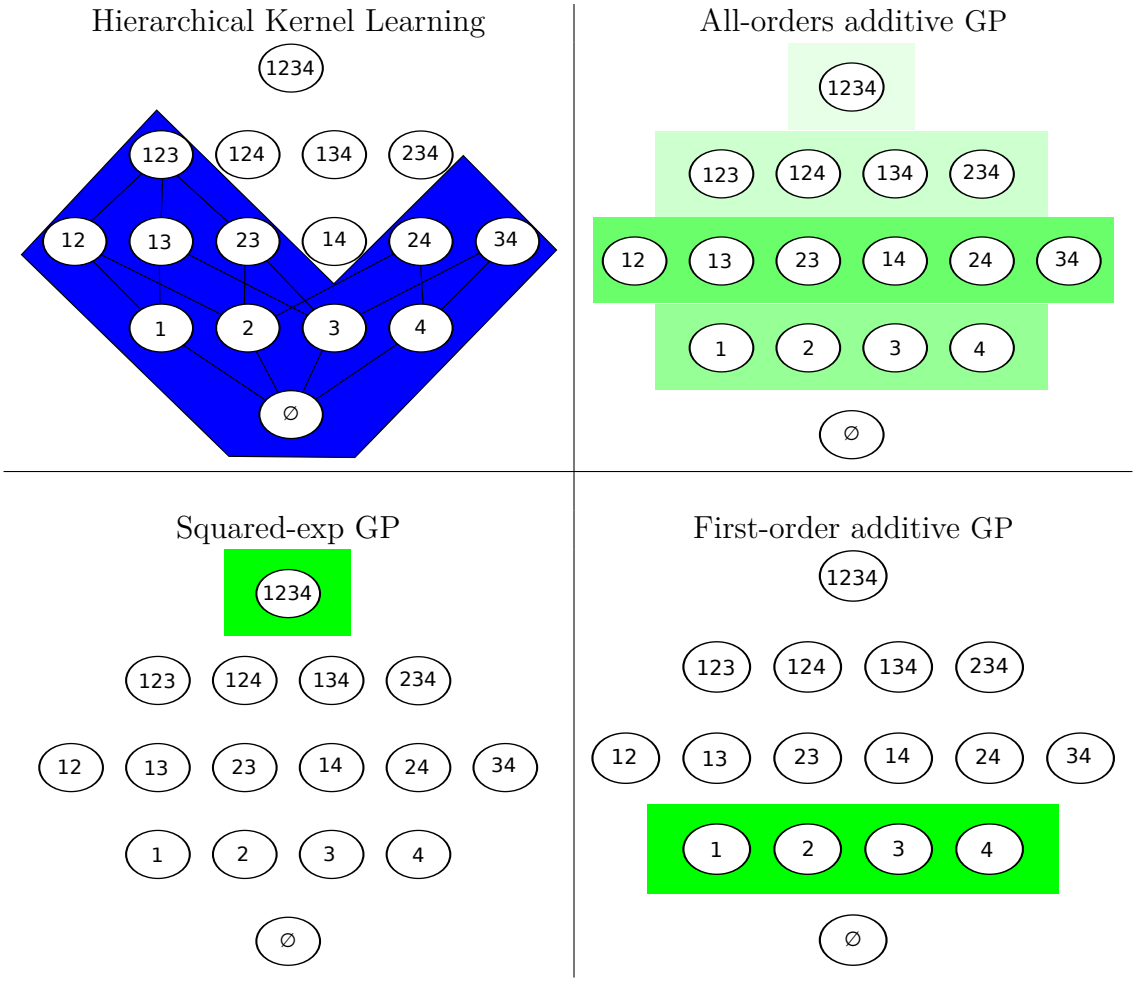


Fig. 6.2 A comparison of different additive model classes. Nodes represent different interaction terms, ranging from first-order to fourth-order interactions. Coloured boxes represent the weightings of different terms. Top left: HKL can select a hull of interaction terms, but must use a pre-determined weighting over those terms. Top right: the additive GP model can weight each order of interaction separately. Bottom row: Neither HKL nor the additive model dominate one another in terms of flexibility, however the SE-GP and first-order additive GP models are special cases of the all-orders additive GP.

chooses among *hull* of kernels, defined as a subset of all terms such that if $\prod_{j \in J} k_j(\mathbf{x}, \mathbf{x}')$ is included in the subset, then so are all terms $\prod_{j \in J/i} k_j(\mathbf{x}, \mathbf{x}')$, for all $i \in J$. HKL computes the sum over all orders in $\mathcal{O}(D)$ time by the formula:

$$k_a(\mathbf{x}, \mathbf{x}') = v^2 \prod_{d=1}^{D} (1 + \alpha k_d(x_d, x'_d)) \tag{6.20}$$

which forces the weight of all nth order terms to be weighted by α^n.

Figure 6.2 contrasts the HKL model class with the additive GP model. Neither method is strictly more flexibile than the other. The main difficulty with the approach of **?** is that the kernel parameters are hard to set, other than by cross-validation.

Support Vector Machines

? introduced the support vector ANOVA decomposition, which has the same form as our additive kernel. They recommend approximating the sum over all D orders with only one term "of appropriate order", presumably because of the difficulty of setting the parameters of an SVM. **?** 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.

Generalizations

A closely related procedure from **?** is smoothing-splines ANOVA (SS-ANOVA). An SS-ANOVA model is a weighted sum of splines along each dimension, plus a sum of 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, in practice, only terms of first and second order are usually considered.

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). **?** review some properties and applications of this model clas.

The main benefits of the relatively Bayesian treatment of this model class 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.

6.5 Experiments

Parameterization

A D-dimensional SE-ARD kernel has D lengthscale parameters and the output variance. An first-order additive SE model has $2 \times D$ parameters. A fully-parametrized model including all orders of interaction, with a seperate output variance for each scale will have $3 \times D$ parameters. Because each additional parameter increases the tendency to overfit, we recommend allowing only one kernel parameter per input dimension. In our experiments, we parametrized each one-dimensional kernel with only the lengthscale, fixing the output variance to be 1.

Methods

We compare five 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 in the additive kernels to 10. GP-GAM denotes an additive GP model with only first-order interactions. GP Squared-Exp is a GP model with a squared-exponential ARD kernel. HKL was run using the all-subsets kernel, which corresponds to the same set of kernels 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 (**?**) 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 done using Expectation Propagation (**?**).

Bach Synthetic Dataset

In addition to standard UCI repository datasets, we generated a synthetic dataset following the same recipe as **?**. This dataset was designed by **?** to demonstrate the advantages of HKL over GP-ARD. It is generated by passing correlated Gaussian-distributed inputs $x_1, x_2, \ldots, x_8$ through the quadratic function

$$f(\mathbf{x}) = \sum_{i=1}^{4} \sum_{j=1+1}^{4} x_i x_j + \epsilon \qquad \epsilon \sim \mathcal{N}(0, \sigma_\epsilon) \tag{6.21}$$

This dataset is 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 irrelevant inputs.

If the dataset is large enough, HKL can construct a hull around only those subsets of cross terms that are optimal for predicting the output. GP-ARD, in contrast, can learn to ignore the noisy copy variables, but cannot learn to ignore the higher-term interactions between the predictive variables. However, a GP with an additive kernel can learn both to ignore irrelevant variables, and to ignore missing orders of interaction. With enough data, the marginal likelihood will favour hyperparameters specifying such a model.

6.5.1 Results

Tables 6.2, 6.3, 6.4 and 6.5 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.

Table 6.2 Regression Mean Squared Error

Method	bach	concrete	pumadyn-8nh	servo	housing
Linear Regression	1.031	0.404	0.641	0.523	0.289
GP GAM	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**

Table 6.3 Regression Negative Log Likelihood

Method	bach	concrete	pumadyn-8nh	servo	housing
Linear Regression	2.430	1.403	1.881	1.678	1.052
GP GAM	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**

The model with best performance on each dataset is in bold, along with all other models that were not significantly different under a paired t-test. The additive model

Table 6.4 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 GAM	**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 6.5 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 GAM	**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**

never performs significantly worse than any other model, and sometimes performs significantly better than all other models. The difference between all methods is larger in the case of regression experiments. The performance of HKL is consistent with the results in (**?**), performing competitively but slightly worse than SE-GP.

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-GAM model and the SE-GP 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 ccould be expected to benefit from integrating over the kernel parameters.

6.5.2 Source Code

Additive Gaussian processes are particularly appealing in practice because their use requires only the specification of the base kernel; all other aspects of GP inference remain the same. Note that we are also free to choose a different covariance function along each dimension.

All of the experiments in this chapter were performed using the standard GPML

toolbox, available at http://www.gaussianprocess.org/gpml/code/. Code to perform all experiments is available at http://github.com/duvenaud/dropout-gps

6.6 Conclusion

In this chapter, we presented a parameterization of higher-order additive GPs allowing for tractable inference. Our experiments indicate that, to varying degrees, additive structure is present in 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, as well.

The additive GP also affords extra interpretability: the variance parameters on each order of interaction indicate which sorts of structure are present the data.

Chapter 7

Warped Mixture Models

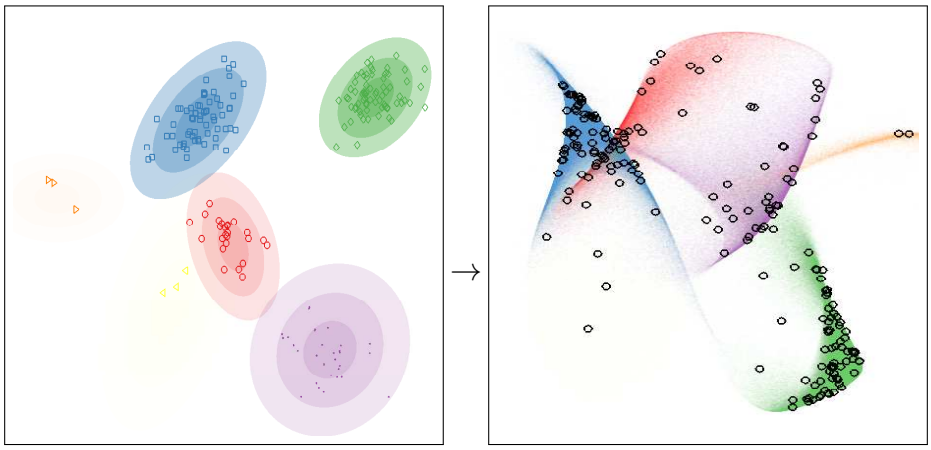
"What, exactly, is a cluster?"

- Bernhard Schölkopf, personal communication

A mixture of Gaussians fit to a single curved or heavy-tailed cluster will report that the data contains many clusters. To produce more appropriate clusterings, we introduce a model which warps a latent mixture of Gaussians to produce nonparametric cluster shapes. The possibly low-dimensional latent mixture model allows us to summarize the properties of the high-dimensional clusters (or density manifolds) describing the data. The number of manifolds, as well as the shape and dimension of each manifold is automatically inferred. We derive a simple inference scheme for this model which analytically integrates out both the mixture parameters and the warping function. We show that our model is effective for density estimation, performs better than infinite Gaussian mixture models at recovering the true number of clusters, and produces interpretable summaries of high-dimensional datasets.

Probabilistic mixture models are often used for clustering. However, if the mixture components are parametric (e.g. Gaussian), then the clustering obtained can be heavily dependent on how well each actual cluster can be modeled by a Gaussian. For example, a heavy tailed or curved cluster may need many components to model it. Thus, although mixture models are widely used for probabilistic clustering, their assumptions are generally inappropriate if the primary goal is to discover clusters in data. Dirichlet process mixture models can alleviate the problem of an unknown number of clusters, but this does not address the problem that real clusters may not be well matched by any parametric density.

In this paper, we propose a nonparametric Bayesian model that can find nonlinearly separable clusters with complex shapes. The proposed model assumes that each



Latent space Observed space

Fig. 7.1 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 non-Gaussian manifolds in the observed space.

observation has coordinates in a latent space, and is generated by warping the latent coordinates via a nonlinear function from the latent space to the observed space. By this warping, complex shapes in the observed space can be modeled by simpler shapes in the latent space. In the latent space, we assume an infinite Gaussian mixture model (**?**), which allows us to automatically infer the number of clusters. For the prior on the nonlinear mapping function, we use Gaussian processes (**?**), which enable us to flexibly infer the nonlinear warping function from the data. We call the proposed model the *infinite warped mixture model* (iWMM). Figure 7.1 shows a set of manifolds and datapoints sampled from the prior defined by this model.

To our knowledge this is the first probabilistic generative model for clustering with flexible nonparametric component densities. Since the proposed model is generative, it can be used for density estimation as well as clustering. It can also be extended to handle missing data, integrate with other probabilistic models, and use other families of distributions for the latent components.

We derive an inference procedure for the iWMM based on Markov chain Monte Carlo (MCMC). In particular, we sample the cluster assignments using Gibbs sampling, sample the latent coordinates using Hamiltonian Monte Carlo, and analytically integrate out both the mixture parameters (weights, means and covariance matrices), and the nonlinear warping function.

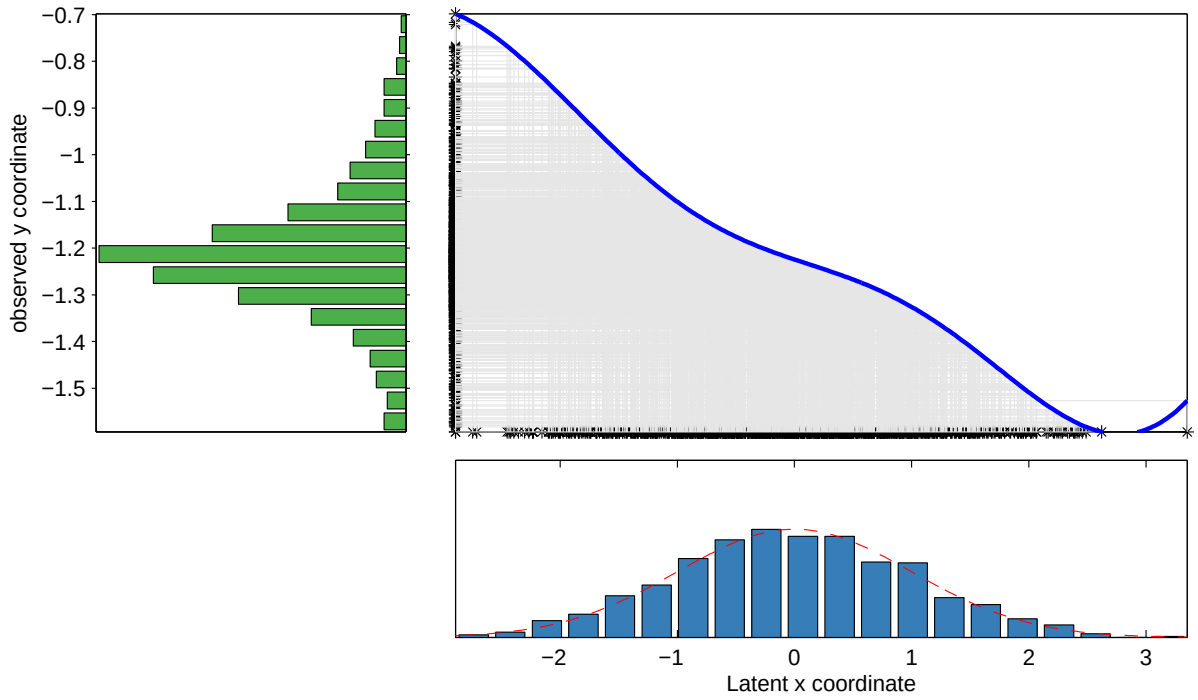


Fig. 7.2 A visual representation of the Gaussian process latent variable model. Bottom: density and samples from a 1D Gaussian, specifying the distribution $p(\mathbf{X})$ in the latent space. Top Right: A function drawn from a GP prior. Left: A nonparametric density defined by warping the latent density through the function drawn from a GP prior.

7.1 Gaussian Process Latent Variable Model

Besides being useful for modeling functions, a simple extension allows GPs to be useful for general density modeling.

Unfortunately, this extension causes many of the useful properties of the GP not to hold.

The GP-LVM can also be thought of as a method for modeling the covariance matrix between all rows of Y using a number of parameters which grows linearly with N.

In this section, we give a brief introduction to the GP-LVM, which can be viewed as a special case of the iWMM. The GP-LVM is a probabilistic model of nonlinear manifolds. While not typically thought of as a density model, the GP-LVM does in fact define a posterior density over observations (**?**). It does this by smoothly warping a single, isotropic Gaussian density in the latent space into a more complicated distribution in the observed space.

Suppose that we have a set of observations $\mathbf{Y} = (\mathbf{y}_1, \cdots, \mathbf{y}_N)^\mathsf{T}$, where $\mathbf{y}_n \in \mathbb{R}^D$, and

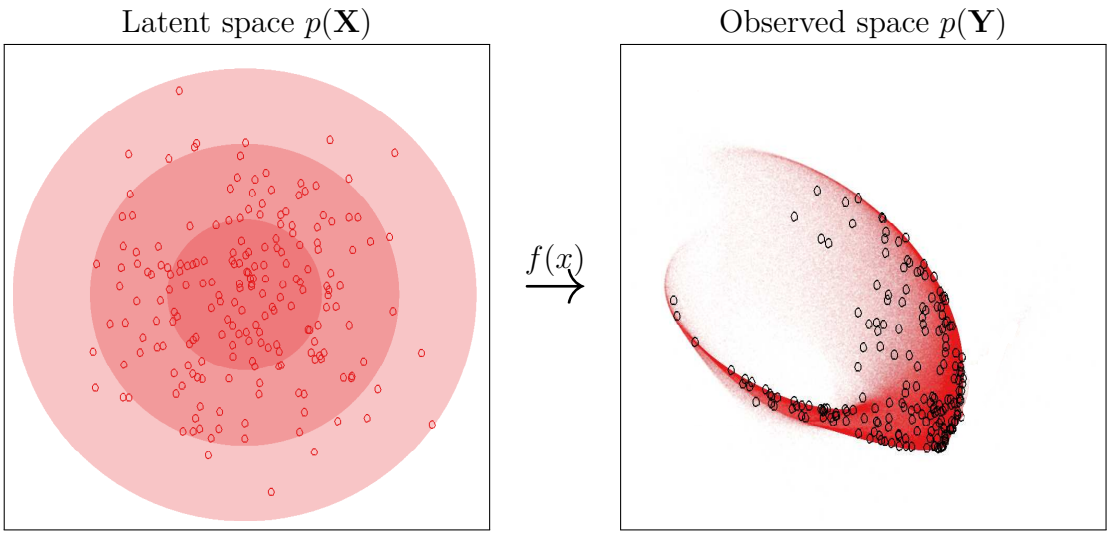


Fig. 7.3 A visual representation of the Gaussian process latent variable model. Left: Isocontours and samples from a 2D Gaussian, specifying the distribution $p(\mathbf{X})$ in the latent space. Right: Density and samples from a nonparametric density defined by warping the latent density through a function drawn from a GP prior.

they are associated with a set of latent coordinates $\mathbf{X} = (\mathbf{x}_1, \cdots, \mathbf{x}_N)^\mathsf{T}$, where $\mathbf{x}_n \in \mathbb{R}^Q$. The GP-LVM assumes that observations are generated by mapping the latent coordinates through a set of smooth functions, over which Gaussian process priors are placed. Under the GP-LVM, the probability of observations given the latent coordinates, integrating out the mapping functions, is

$$p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta}) = (2\pi)^{-\frac{DN}{2}} |\mathbf{K}|^{-\frac{D}{2}} \exp\left(-\frac{1}{2}\mathrm{tr}(\mathbf{Y}^\mathsf{T}\mathbf{K}^{-1}\mathbf{Y})\right), \tag{7.1}$$

where $\mathbf{K}$ is the $N \times N$ covariance matrix defined by the kernel function $k(\mathbf{x}_n, \mathbf{x}_m)$, and $\boldsymbol{\theta}$ is the kernel hyperparameter vector. In this paper, we use an RBF kernel with an additive noise term:

$$k(\mathbf{x}_n, \mathbf{x}_m) = \alpha \exp\left(-\frac{1}{2\ell^2}(\mathbf{x}_n - \mathbf{x}_m)^\mathsf{T}(\mathbf{x}_n - \mathbf{x}_m)\right) + \delta_{nm}\beta^{-1}. \tag{7.2}$$

This likelihood is simply the product of D independent Gaussian process likelihoods, one for each output dimension.

Typically, the GP-LVM is used for dimensionality reduction or visualization, and the latent coordinates are determined by maximizing the posterior probability of the latent coordinates, while integrating out the warping function. In that setting, the

Gaussian prior density on $\mathbf{x}$ is essentially a regularizer which keeps the latent coordinates from spreading arbitrarily far apart. In contrast, we instead integrate out the latent coordinates as well as the warping function, and place a more flexible parameterization on $p(\mathbf{x})$ than a single isotropic Gaussian.

Just as the GP-LVM can be viewed as a manifold learning algorithm, the iWMM can be viewed as learning a set of manifolds, one for each cluster.

7.2 Infinite Warped Mixture Model

In this section, we define in detail the infinite warped mixture model (iWMM). In the same way as the GP-LVM, the iWMM assumes a set of latent coordinates and a smooth, nonlinear mapping from the latent space to the observed space. In addition, the iWMM assumes that the latent coordinates are generated from a Dirichlet process mixture model. In particular, we use the following infinite Gaussian mixture model,

$$p(\mathbf{x}|\{\lambda_c, \boldsymbol{\mu}_c, \mathbf{R}_c\}) = \sum_{c=1}^{\infty} \lambda_c \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_c, \mathbf{R}_c^{-1}), \tag{7.3}$$

where λ_c, $\boldsymbol{\mu}_c$ and $\mathbf{R}_c$ is the mixture weight, mean, and precision matrix of the c^{th} mixture component. We place Gaussian-Wishart priors on the Gaussian parameters $\{\boldsymbol{\mu}_c, \mathbf{R}_c\}$,

$$p(\boldsymbol{\mu}_c, \mathbf{R}_c) = \mathcal{N}(\boldsymbol{\mu}_c|\mathbf{u}, (r\mathbf{R}_c)^{-1})\mathcal{W}(\mathbf{R}_c|\mathbf{S}^{-1}, \nu), \tag{7.4}$$

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 ν is the number of degrees of freedom for $\mathbf{R}_c$. The Wishart distribution is defined as follows:

$$\mathcal{W}(\mathbf{R}|\mathbf{S}^{-1}, \nu) = \frac{1}{G}|\mathbf{R}|^{\frac{\nu-Q-1}{2}} \exp\left(-\frac{1}{2}\text{tr}(\mathbf{SR})\right), \tag{7.5}$$

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 latent assignment of the n^{th} point. The probability of latent coordinates $\mathbf{X}$ given latent assignments $\mathbf{Z} = (z_1, z_2, \ldots, z_N)$ is obtained by integrating out the Gaussian parameters

$\{\boldsymbol{\mu}_c, \mathbf{R}_c\}$ as follows:

$$p(\mathbf{X}|\mathbf{Z}, \mathbf{S}, \nu, r) = \prod_{c=1}^{\infty} \pi^{-\frac{N_c Q}{2}} \frac{r^{Q/2}|\mathbf{S}|^{\nu/2}}{r_c^{Q/2}|\mathbf{S}_c|^{\nu_c/2}} \times \prod_{q=1}^{Q} \frac{\Gamma(\frac{\nu_c+1-q}{2})}{\Gamma(\frac{\nu+1-q}{2})}, \tag{7.6}$$

where N_c is the number of data points assigned to the c^{th} component, $\Gamma(\cdot)$ is the Gamma function, and

$$r_c = r + N_c, \qquad \nu_c = \nu + N_c, \quad \mathbf{u}_c = \frac{r\mathbf{u} + \sum_{n:z_n=c} \mathbf{x}_n}{r + N_c},$$

$$\mathbf{S}_c = \mathbf{S} + \sum_{n:z_n=c} \mathbf{x}_n \mathbf{x}_n^{\mathsf{T}} + r\mathbf{u}\mathbf{u}^{\mathsf{T}} - r_c \mathbf{u}_c \mathbf{u}_c^{\mathsf{T}}, \tag{7.7}$$

are the posterior Gaussian-Wishart parameters of the c^{th} component. We use a Dirichlet process with concentration parameter η for infinite mixture modeling (**?**) in the latent space. Then, the probability of $\mathbf{Z}$ is given as follows:

$$p(\mathbf{Z}|\eta) = \frac{\eta^C \prod_{c=1}^{C}(N_c - 1)!}{\eta(\eta+1)\cdots(\eta+N-1)}, \tag{7.8}$$

where C is the number of components for which $N_c > 0$. The joint distribution is given by

$$p(\mathbf{Y}, \mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}, \boldsymbol{S}, \nu, \mathbf{u}, r, \eta) = p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta}) p(\mathbf{X}|\mathbf{Z}, \boldsymbol{S}, \nu, \mathbf{u}, r) p(\mathbf{Z}|\eta), \tag{7.9}$$

where factors in the right hand side can be calculated by (7.1), (7.6) and (7.8), respectively.

In summary, the infinite warped mixture model generates observations $\mathbf{Y}$ according to the following generative process:

1. Draw mixture weights $\boldsymbol{\lambda} \sim \text{GEM}(\eta)$
2. For each component $c = 1, 2, \ldots, \infty$
   (a) Draw precision $\mathbf{R}_c \sim \mathcal{W}(\mathbf{S}^{-1}, \nu)$
   (b) Draw mean $\boldsymbol{\mu}_c \sim \mathcal{N}(\mathbf{u}, (r\mathbf{R}_c)^{-1})$
3. For each observed dimension $d = 1, 2, \ldots, D$
   (a) Draw function $f_d(\mathbf{x}) \sim \text{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$

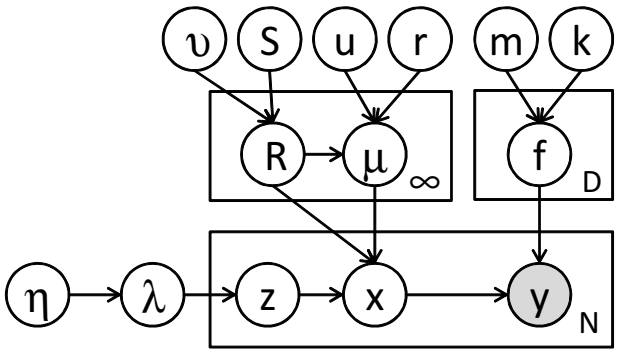


Fig. 7.4 A graphical model representation of the infinite warped mixture model, where the shaded and unshaded nodes indicate observed and latent variables, respectively, and plates indicate repetition.

4. For each observation $n = 1, 2, \ldots, N$

   (a) Draw latent assignment $z_n \sim \text{Mult}(\boldsymbol{\lambda})$

   (b) Draw latent coordinates $\mathbf{x}_n \sim \mathcal{N}(\boldsymbol{\mu}_{z_n}, \mathbf{R}_{z_n}^{-1})$

   (c) For each observed dimension $d = 1, 2, \ldots, D$

      i. Draw feature $y_{nd} \sim \mathcal{N}(f_d(\mathbf{x}_n), \beta^{-1})$

Here, GEM(η) is the stick-breaking process (**?**) that generates mixture weights for a Dirichlet process with parameter η, Mult($\boldsymbol{\lambda}$) represents a multinomial distribution with parameter $\boldsymbol{\lambda}$, $m(\mathbf{x})$ is the mean function of the Gaussian process, and $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^Q$. Figure 7.4 shows the graphical model representation of the proposed model. Here, we assume a Gaussian for the mixture component, although we could in principle use other distributions such as Student's t-distribution or the Laplace distribution.

The iWMM can be seen as a generalization of either the GP-LVM or the infinite Gaussian mixture model (iGMM). To be precise, the iWMM with a single fixed spherical Gaussian density on the latent coordinates corresponds to the GP-LVM, while the iWMM with fixed direct mapping function $f_d(\mathbf{x}) = x_d$ and $Q = D$ corresponds to the iGMM.

The iWMM offers attractive properties that do not exist in other probabilistic models; principally, the ability to model clusters with nonparametric densities, and to infer a seperate dimension for manifold.

7.3 Inference

We infer the posterior distribution of the latent coordinates $\mathbf{X}$ and cluster assignments $\mathbf{Z}$ using Markov chain Monte Carlo (MCMC). In particular, we alternate collapsed Gibbs sampling of $\mathbf{Z}$, and Hamiltonian Monte Carlo sampling of $\mathbf{X}$. Given $\mathbf{X}$, we can efficiently sample $\mathbf{Z}$ using collapsed Gibbs sampling, integrating out the mixture parameters. Given $\mathbf{Z}$, we can calculate the gradient of the unnormalized posterior distribution of $\mathbf{X}$, integrating over warping functions. This gradient allows us to sample $\mathbf{X}$ using Hamiltonian Monte Carlo.

First, we explain collapsed Gibbs sampling for $\mathbf{Z}$. 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:

$$p(z_n = c|\mathbf{X}, \mathbf{Z}_{\backslash n}, \boldsymbol{S}, \nu, \mathbf{u}, r, \eta) \propto \begin{cases} N_{c\backslash n} \cdot p(\mathbf{x}_n|\mathbf{X}_{c\backslash n}, \boldsymbol{S}, \nu, \mathbf{u}, r) & \text{existing components} \\ \eta \cdot p(\mathbf{x}_n|\boldsymbol{S}, \nu, \mathbf{u}, r) & \text{a new component} \end{cases} \tag{7.10}$$

where $\mathbf{X}_c = \{\mathbf{x}_n|z_n = c\}$ is the set of latent coordinates assigned to the c^{th} component, and $\backslash n$ represents the value or set when excluding the n^{th} data point. We can analytically calculate $p(\mathbf{x}_n|\mathbf{X}_{c\backslash n}, \boldsymbol{S}, \nu, \mathbf{u}, r)$ as follows:

$$p(\mathbf{x}_n|\mathbf{X}_{c\backslash n}, \boldsymbol{S}, \nu, \mathbf{u}, r) = \pi^{-\frac{N_{c\backslash n}Q}{2}} \frac{r_{c\backslash n}^{Q/2}|\mathbf{S}_{c\backslash n}|^{\nu_{c\backslash n}/2}}{r_{c\backslash n}'^{Q/2}|\mathbf{S}'_{c\backslash n}|^{\nu'_{c\backslash n}/2}} \prod_{d=1}^{Q} \frac{\Gamma(\frac{\nu'_{c\backslash n}+1-d}{2})}{\Gamma(\frac{\nu_{c\backslash n}+1-d}{2})}, \tag{7.11}$$

where r'_c, ν'_c, $\mathbf{u}'_c$ and $\mathbf{S}'_c$ represent the posterior Gaussian-Wishart parameters of the c^{th} component when the n^{th} data point is assigned to the c^{th} component. We can efficiently calculate the determinant by using the rank one Cholesky update. In the same way, we can analytically calculate the likelihood for a new component $p(\mathbf{x}_n|\boldsymbol{S}, \nu, \mathbf{u}, r)$.

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 of the unnormalized posterior

$$\log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta}) + \log p(\mathbf{X}|\mathbf{Z}, \boldsymbol{S}, \nu, \mathbf{u}, r) \tag{7.12}$$

The first term of the gradient can be calculated by

$$\frac{\partial \log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta})}{\partial \mathbf{K}} = -\frac{1}{2} D\mathbf{K}^{-1} + \frac{1}{2}\mathbf{K}^{-1}\mathbf{Y}\mathbf{Y}^{T}\mathbf{K}^{-1}, \tag{7.13}$$

and

$$\frac{\partial k(\mathbf{x}_n, \mathbf{x}_m)}{\partial \mathbf{x}_n} = -\frac{\alpha}{\ell^2} \exp\left(-\frac{1}{2\ell^2}(\mathbf{x}_n - \mathbf{x}_m)^{\mathsf{T}}(\mathbf{x}_n - \mathbf{x}_m)\right)(\mathbf{x}_n - \mathbf{x}_m), \tag{7.14}$$

using the chain rule. The second term can be calculated as follows:

$$\frac{\partial \log p(\mathbf{X}|\mathbf{Z}, \boldsymbol{S}, \nu, \mathbf{u}, r)}{\partial \mathbf{x}_n} = -\nu_{z_n} \boldsymbol{S}_{z_n}^{-1}(\mathbf{x}_n - \mathbf{u}_{z_n}). \tag{7.15}$$

We also infer kernel parameters $\boldsymbol{\theta} = \{\alpha, \beta, \ell\}$ via HMC, using the gradient of the log unnormalized posterior with respect to the kernel parameters. 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 such as (**??**).

In summary, we obtain samples from the posterior $p(\mathbf{X}, \mathbf{Z}|\mathbf{Y}, \boldsymbol{\theta}, \mathbf{S}, \nu, \mathbf{u}, r, \eta)$ by iterating the following steps:

1. For each observation $n = 1, \cdots, N$, sample the component assignment z_n by collapsed Gibbs sampling (7.10).

2. Sample latent coordinates $\mathbf{X}$ and kernel parameters $\boldsymbol{\theta}$ using Hamiltonian Monte Carlo.

7.3.1 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 which is most likely to be mapped to $\mathbf{y}_\star$, then using the density of $p(\mathbf{x}_\star)$ and the Jacobian of the warping at that point to approximately compute the density at $\mathbf{y}_\star$. When inference is done by simply optimizing the location of the latent points, this estimation method simply 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 but rather sample them, we would need to perform optimizations for each $p(\mathbf{y}_\star)$ seperately for each sample. Our method

gives estimates for all $p(\mathbf{y}_\star)$ at once, but may not be accurate in very high dimensions.

The posterior density in the observed space given the training data is simply:

$$\begin{aligned} p(\mathbf{y}_\star|\mathbf{Y}) &= \iint p(\mathbf{y}_\star, \mathbf{x}_\star, \mathbf{X}|\mathbf{Y}) d\mathbf{x}_\star d\mathbf{X} \\ &= \iint p(\mathbf{y}_\star|\mathbf{x}_\star, \mathbf{X}, \mathbf{Y}) p(\mathbf{x}_\star|\mathbf{X}, \mathbf{Y}) p(\mathbf{X}|\mathbf{Y}) d\mathbf{x}_\star d\mathbf{X}. \end{aligned} \tag{7.16}$$

We approximate $p(\mathbf{X}|\mathbf{Y})$ using the samples from the Gibbs and Hamiltonian Monte Carlo samplers. We approximate $p(\mathbf{x}_\star|\mathbf{X}, \mathbf{Y})$ by sampling points from the latent mixture and warping them, using the following procedure:

1. Draw latent assignments $z_\star \sim \text{Mult}(\frac{N_1}{N+\eta}, \cdots, \frac{N_C}{N+\eta}, \frac{\eta}{N+\eta})$
2. Draw precision matrix $\mathbf{R}_\star \sim \mathcal{W}(\mathbf{S}_{z_\star}^{-1}, \nu_{z_\star})$
3. Draw mean $\boldsymbol{\mu}_\star \sim \mathcal{N}(\mathbf{u}_{z_\star}, (r_{z_\star}\mathbf{R}_\star)^{-1})$
4. Draw latent coordinates $\mathbf{x}_\star \sim \mathcal{N}(\boldsymbol{\mu}_\star, \mathbf{R}_\star^{-1})$

When a new component $C+1$ is assigned to $z_\star$, the prior Gaussian-Wishart distribution is used for sampling in steps 2 and 3. The first factor of (7.16) can be calculated by

$$p(\mathbf{y}_\star|\mathbf{x}_\star, \mathbf{X}, \mathbf{Y}) = \mathcal{N}(\mathbf{k}_\star^\mathsf{T}\mathbf{K}^{-1}\mathbf{Y}, k(\mathbf{x}_\star, \mathbf{x}_\star) - \mathbf{k}_\star^\mathsf{T}\mathbf{K}^{-1}\mathbf{k}_\star), \tag{7.17}$$

where $\mathbf{k}_\star = (k(\mathbf{x}_\star, \mathbf{x}_1), \cdots, k(\mathbf{x}_\star, \mathbf{x}_N))^\mathsf{T}$. Each step of this sampling procedure draws from the exact conditional distribution, so the Monte Carlo estimate of the predictive density $p(\mathbf{y}_\star|\mathbf{X}, \mathbf{Y})$ will converge to the true marginal distribution. 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. This procedure was used to generate the plots of posterior density in figures 7.1, 7.6, and 7.8.

7.4 Related work

The GP-LVM is effective as a nonlinear latent variable model in a wide variety of applications (**???**). The latent positions $\mathbf{X}$ in the GP-LVM are typically obtained by maximum a posteriori estimation or variational Bayesian inference (**?**), placing a single fixed spherical Gaussian prior on $\mathbf{x}$. A prior which penalizes a high-dimensional latent

space was introduced by **?**, in which the latent variables and their intrinsic dimensionality are simultaneously optimized. The iWMM can also infer the intrinsic dimensionality of nonlinear manifolds: inferring the Gaussian covariance for each latent cluster allows the variance of irrelevant dimensions to become small. Because each latent cluster has a different set of parameters, the effective dimension of each cluster can vary, allowing manifolds of differing dimension in the observed space. This ability is demonstrated in figure 7.6b.

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

There exist non-probabilistic clustering methods which can find clusters with complex shapes, such as spectral clustering (**?**) and nonlinear manifold clustering (**??**). Spectral clustering finds clusters by first forming a similarity graph, then finding a low-dimensional latent representation using the graph, and finally, 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. In contrast, the iWMM infers such parameters automatically. One of the main advantages of the iWMM over these methods is that there is no need to construct a similarity graph.

The kernel Gaussian mixture model (**?**) can also find non-Gaussian shaped clusters. This model estimates a GMM in the implicit high-dimensional feature space defined by the kernel mapping of the observed space. However, the kernel GMM uses a fixed nonlinear mapping function, with no guarantee that the latent points will be well-modeled by a GMM. In contrast, the iWMM infers the mapping function such that the latent co-ordinates will be well-modeled by a mixture of Gaussians.

7.5 Experimental results

7.5.1 Clustering Faces

We first 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 (**?**). Both series of images capture a person turning his head to the right.

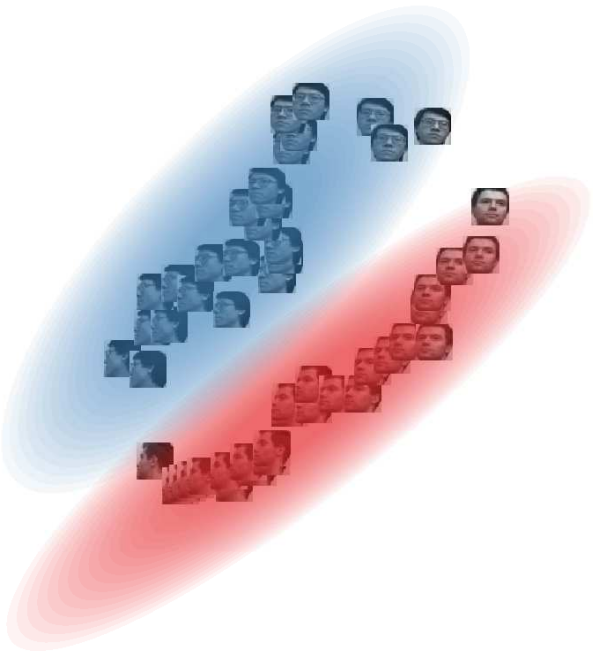
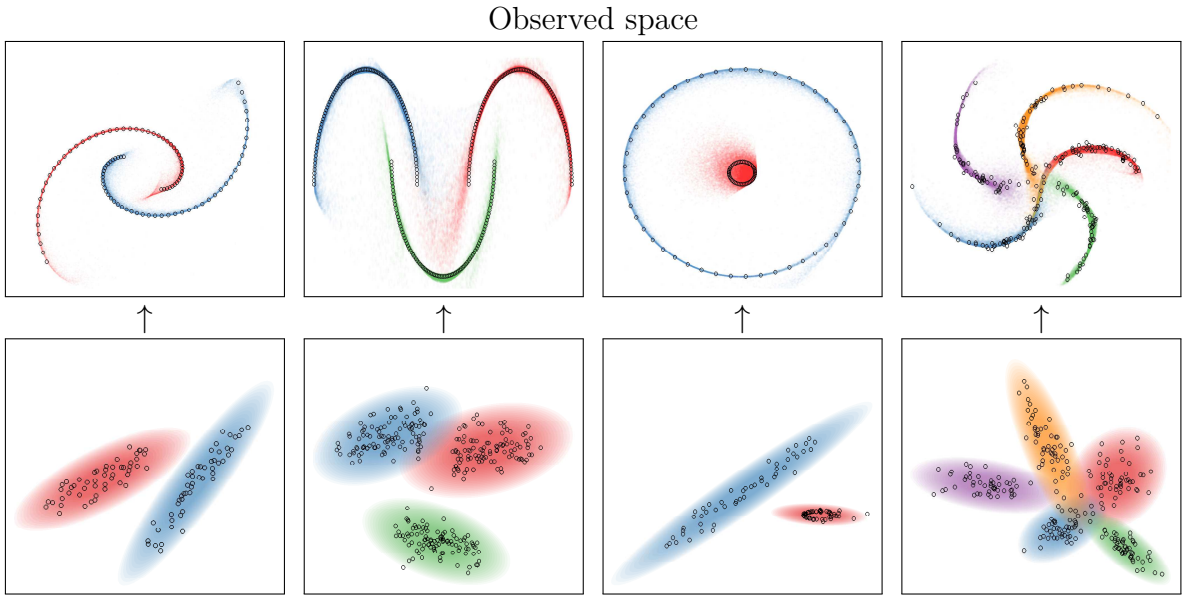


Fig. 7.5 A sample from the 2-dimensional latent space when modeling a series of 32x32 face images. Our model correctly discovers that the data consists of two seperate manifolds, both approximately one-dimensional, which share the same head-turning structure.

Figure 7.5 shows a sample from the posterior over the latent coordinates, as well as the density model. The model has recovered three relevant, 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.

7.5.2 Synthetic Datasets

Next, we demonstrate the proposed model on the four synthetic datasets shown in Figure 7.6. None of these four datasets can be appropriately clustered by Gaussian mixture models (GMM). For example, consider the 2-curve data shown in Figure 7.6 (a), where 100 data points lie in one 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, with the iWMM, the two non-Gaussian-shaped clusters in the observed space were represented by two Gaussian-shaped clusters in the latent



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

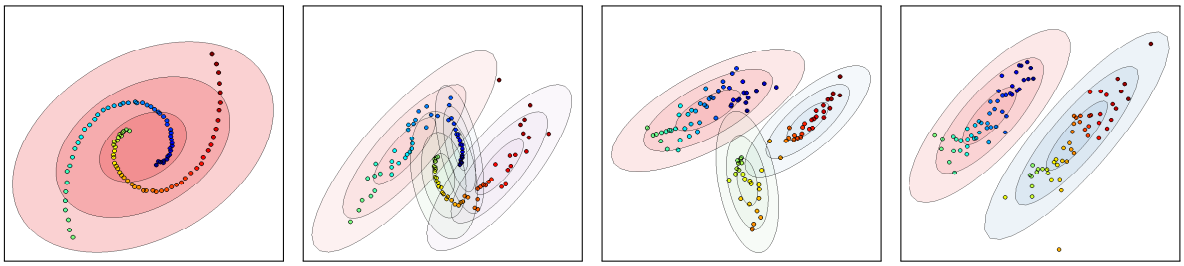
Fig. 7.6 Top row: The observed, unlabeled data points, and the clusters inferred by the iWMM. Bottom row: Latent coordinates and Gaussian components, shown for a single sample from the posterior. Each point in the latent space corresponds to a point in the observed space. This figure is best viewed in color.

space, as shown at the bottom row of Figure 7.6 (a). The iWMM separated the two curved lines by nonlinearly warping two Gaussians from the latent space to the observed space.

Figure 7.6 (c) shows an interesting manifold learning challenge: a dataset consisting of two circles. The outer circle is modeled in the latent space by a Gaussian with effectively one degree of freedom. This linear topology fits the outer circle in the observed space by bending the two ends until they overlap. 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 manifold can have a different effective dimension.

7.5.3 Mixing

An interesting side-effect of learning the number of latent clusters is that this added flexibility can help the sampler escape local minima, helping the sampler to mix properly. Figure 7.7 shows the samples of the latent coordinates and clusters of the iWMM over



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

Fig. 7.7 The inferred infinite GMMs over iterations in the two-dimensional latent space with the iWMM using the 2-curve data. Labels indicate the number of iterations of the sampler, and the color of each point represents its ordering in the observed coordinates.

time, when modeling the 2-curve data. 7.7(a) shows the latent coordinates initialized at the observed coordinates, starting with one latent component. At the 500th iteration 7.7(b), each curved line is modeled by two components. At the 1800th iteration 7.7(c), the left curved line is modeled by a single component. At the 3000th iteration 7.7(d), the right curved line is also modeled by a single component, and the dataset is appropriately clustered. This configuration was relatively stable, and a similar state was found at the 5000th iteration.

7.5.4 Density Estimation

(a) iWMM (b) GP-LVM

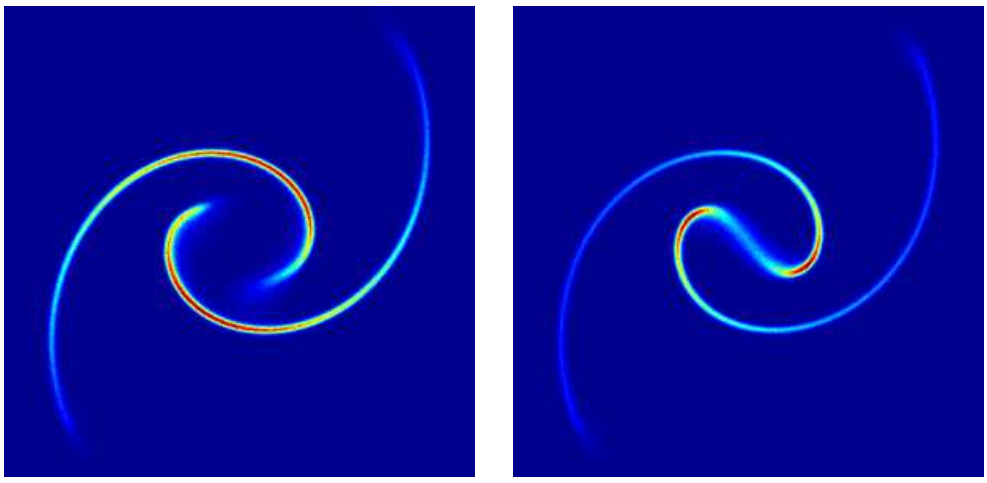
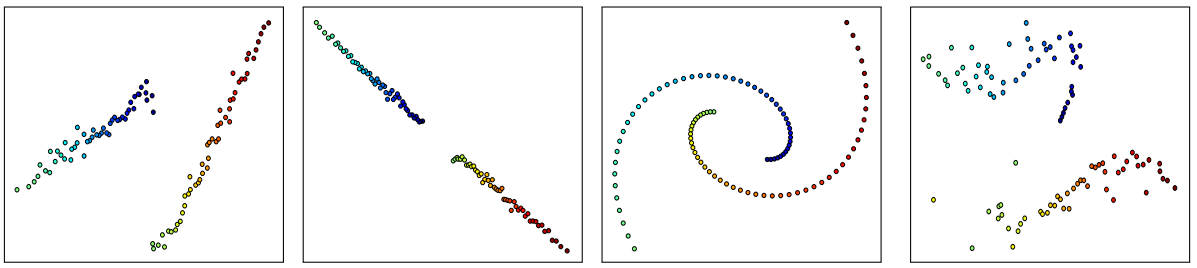


Fig. 7.8 Left: The posterior density in the observed space with the 2-curve data inferred by the iWMM. Rirght: The posterior inferred by the iWMM with one component, a model equivalent to the GP-LVM.

Figure 7.8 (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 two separate manifolds of high density implied by the two curved lines was recovered by the iWMM. Note also that the density along the manifold varies with the density of data shown in Figure 7.6 (a). This result can be compared to a special case of our model, which uses only a single Gaussian to model the latent coordinates instead of an infinite GMM. Figure 7.8 (b) shows that the result of the iWMM with $C = 1$, where posterior is forced to place significant density connecting the two clusters. Figure 7.8 (b) shows that the single-cluster variant of the iWMM posterior is forced to place significant density connecting the two clusters.

7.5.5 Visualization

Next, we briefly investigate the potential of the iWMM for low-dimensional visualization of data. Figure 7.9 (a) shows the latent coordinates obtained by averaging over 1000



(a) iWMM (b) iWMM ($C = 1$) (c) GP-LVM (d) Bayesian GP-LVM

Fig. 7.9 Latent coordinates of the 2-curve data, estimated by four different methods.

samples from the posterior of the iWMM. Because rotating the latent coordinates does not change their probability, averaging may not be an adequate way to summarize the posterior. However, we show this result in order to show the characteristics of latent coordinates obtained by the iWMM. The estimated latent coordinates are clearly separated, and they form two straight lines. This result indicates that in some cases, the iWMM can recover the topology of the data before it has been warped into a manifold. For comparison, Figure 7.9 (b) shows the latent coordinates estimated by the iWMM when forced to use a single cluster: the latent coordinates lie in two sections of a single straight line. Figure 7.9 (c) and (d) show the latent coordinates estimated by the GP-LVM when optimizing or integrating out the latent coordinates, respectively. Recall that the iWMM ($C = 1$) is a more flexible model than the GP-LVM, since the GP-LVM

enforces a spherical covariance in the latent space. These methods did not unfold the two curved lines, since the effective dimension of their latent representation is fixed beforehand. In contrast, the iWMM effectively formed a low-dimensional representation in the latent space.

Regardless of the dimension of the latent space, the iWMM will tend to model each cluster with as low-dimensional a Gaussian as possible. This is because, if the data in a cluster can be made to lie in a low-dimensional plane, a narrowly-shaped Gaussian will assign the latent coordinates much higher likelihood than a spherical Gaussian.

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 (**?**), in addition to the four synthetic datasets shown above: 2-curve, 3-semi, 2-circle and Pinwheel (**?**). The statistics of these datasets are summarized in Table 7.1. In each experiment, we show the results of ten-fold cross-validation.

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

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 (**?**), which measures the correspondence between inferred clusters and true clusters. The iGMM is another probabilistic generative model commonly used for clustering, which can be seen as a special case of the iWMM in which the Gaussian

clusters are not warped. These experiments demonstrate the extent to which nonparametric cluster shapes allow a mixture model to recover more meaningful clusters.

Table 7.3 lists average test log likelihood, comparing the proposed models with kernel density estimation (KDE), and the infinite Gaussian mixture model (iGMM). In KDE, the kernel width is estimated by maximizing the leave-one-out log densities. 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 in iWMMs. We also include the $Q = 2$ case in an attempt to characterize how much modeling power is lost by forcing the latent representation to be visualizable. The proposed models achieved high test likelihoods compared with the KDE and the iGMM.

Table 7.3 Average test log-likelihood 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
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

7.5.7 Source code

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

7.6 Future work

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 (**?**), or a Dirichlet diffusion tree (**?**). Another straightforward extension of our model would be making inference more scalable by using sparse Gaussian processes (**??**) or more advanced Hamiltonian Monte Carlo methods (**?**). An interesting but more complex 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 those regions were stretched along low-dimensional manifolds in the observed space. Another natural extension would be to allow a separate warping for each cluster, which would also improve inference speed.

7.7 Conclusion

In this chapter, we 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 data contours. We then introduced an efficient sampler for this model which integrates out both the cluster parameters and the warping function exactly. We further demonstrated that allowing non-parametric cluster shapes improves clustering performance over the Dirichlet process Mixture of Gaussians.

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.

Chapter 8

Discussion

8.1 Why Assume Zero Mean?

In literature, as well as in practice, it is common to construct GP priors with a zero mean function. This might seem strange, since it is presumably a good place to put prior information, or if we are comparing models, to express since marginalizing over an unknown mean function can be equivalently expressed as a different GP with zero-mean, and another term added to the kernel.

A known mean function can be moved into the covariance function Specifically, if we wish to model an unknown function $f(\mathbf{x})$ with known mean $m(\mathbf{x})$, (with unknown magnitude $c \sim \mathcal{N}(0, \sigma_c^2)$), we can equivalently express this model using another GP with zero mean:

$$f \sim \mathcal{GP}(cm(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')), \quad c \sim \mathcal{N}\left(0, \sigma_c^2\right) \iff f \sim \mathcal{GP}\left(\mathbf{0}, c^2 m(\mathbf{x}) m(\mathbf{x}') + k(\mathbf{x}, \mathbf{x}')\right) \tag{8.1}$$

By moving the mean function into the covariance function, we get the same model, but we can integrate over the magnitude of the mean function at no additional cost. This is one advantage of moving as much structure as possible into the covariance function.

An unknown mean function can be moved into the covariance function If we wish to express our ignorance about the mean function, one way would be by putting a

GP prior on it.

$$m \sim \mathcal{GP}(\mathbf{0}, k_1(\mathbf{x}, \mathbf{x}'))\,, \quad f \sim \mathcal{GP}(m(\mathbf{x}), k_2(\mathbf{x}, \mathbf{x}')) \iff f \sim \mathcal{GP}(\mathbf{0}, k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')) \tag{8.2}$$

8.2 Why Not Learn the Mean Function Instead?

One might ask: besides integrating over the magnitude, what is the advantage of moving the mean function into the covariance function? After all, mean functions are certainly more interpretable than a posterior distribution over functions.

Instead of searching over a large class of covariance functions, which seems strange and unnatural, we might consider simply searching over a large class of structured mean functions, assuming a simple i.i.d. noise model. This is the approach taken by practically every other regression technique: neural networks, decision trees, boosting, etc. . If we could integrate over a wide class of possible mean functions, we would have a very powerful learning an inference method. The problem faced by all of these methods is the well-known problem *overfitting*. If we are forced to choose just a single function with which to make predictions, we must carefully control the flexibility of the model we learn, generally prefering "simple" functions, or to choose a function from a restricted set.

If, on the other hand, we are allowed to keep in mind many possible explanations of the data, *there is no need to penalize complexity*. [cite Occam's razor paper?] The power of putting structure into the covariance function is that doing so allows us to implictly integrate over many functions, maintaining a posterior distribution over infinitely many functions, instead of choosing just one. In fact, each of the functions being considered can be infinitely complex, without causing any form of overfitting. For example, each of the samples shown in figure 1.1 varies randomly over the whole real line, never repeating, each one requiring an infinite amount of information to describe. Choosing the one function which best fits the data will almost certainly cause overfitting. However, if we integrate over many such functions, we will end up with a posterior putting mass on only those functions which are compatible with the data. In other words, the parts of the function that we can determine from the data will be predicted with certainty, but the parts which are still uncertain will give rise to a wide range of predictions.

To repeat: *there is no need to assume that the function being modeled is simple, or to prefer simple explanations* in order to avoid overfitting, if we integrate over many possible explanations rather than choosing just the one.

In Chapter 3, we will compare a system which estimates a parametric covariance function against one which estimates a parametric mean function.

8.3 Signal versus Noise

In most derivations of Gaussian processes, the model is given as

$$y = f(x) + \epsilon, \quad \text{where} \quad \epsilon \overset{\text{iid}}{\sim} \mathcal{N}\left(0, \sigma^2_{\text{noise}}\right) \tag{8.3}$$

However, ϵ can equivalently be thought of as another Gaussian process, and so this model can be written as $y(x) \sim \mathcal{GP}(0, k + \delta)$. The lack of a hard distinction between the noise model and the signal model raises the larger question: Which part of a model represents signal, and which represents noise?

We believe that it depends on what you want to do with the model - there is no hard distinction between signal and noise in general.

For example: often, we don't care about the short-term variations in a function, and only in the long-term trend. However, in many other cases, we with to de-trend our data to see more clearly how much a particular part of the signal deviated from normal.

8.4 Why does everyone use squared-exp?

In some instances, using a 'default' kernel yields acceptable performance. Many frequentist methods assume a characteristic kernel, such as the squared-exp. This choice is motivated by the fact that, in the limit of infinite data, and a shrinking lengthscale, the estimate of the function will converge asymptotically to the truth. [citation needed]

8.5 Why haven't structured kernels been built for SVMs?

Because without marginal likelihood to tell you which structure is present in your data, it's not clear how to choose which kernel to use without cross-validation.

8.6 Automating Statistics

Automating the process of statistical modeling would have a tremendous impact on fields that currently rely on expert statisticians, machine learning researchers, and data scientists. While fitting simple models (such as linear regression) is largely automated by standard software packages, there has been little work on the automatic construction of flexible but interpretable models.

Appendix A

Appendix for Kernels

A.1 Gaussian Conditionals

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

Appendix B

Appendix for Grammars

B.1 Kernels

B.1.1 Base kernels

For scalar-valued inputs, the white noise (WN), constant (C), linear (Lin), squared exponential (SE), and periodic kernels (Per) are defined as follows:

$$\mathrm{WN}(x, x') = \sigma^2 \delta_{x,x'} \tag{B.1}$$

$$\mathrm{C}(x, x') = \sigma^2 \tag{B.2}$$

$$\mathrm{Lin}(x, x') = \sigma^2 (x - \ell)(x' - \ell) \tag{B.3}$$

$$\mathrm{SE}(x, x') = \sigma^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right) \tag{B.4}$$

$$\mathrm{Per}(x, x') = \sigma^2 \frac{\exp\left(\frac{\cos\frac{2\pi(x-x')}{p}}{\ell^2}\right) - I_0\left(\frac{1}{\ell^2}\right)}{\exp\left(\frac{1}{\ell^2}\right) - I_0\left(\frac{1}{\ell^2}\right)} \tag{B.5}$$

where $\delta_{x,x'}$ is the Kronecker delta function, I_0 is the modified Bessel function of the first kind of order zero and other symbols are parameters of the kernel functions.

B.1.2 Changepoints and changewindows

The changepoint, $\mathrm{CP}(\cdot, \cdot)$ operator is defined as follows:

$$\begin{aligned} \mathrm{CP}(k_1, k_2)(x, x') = {} & \sigma(x) k_1(x, x') \sigma(x') \\ & + (1 - \sigma(x)) k_2(x, x') (1 - \sigma(x')) \end{aligned} \tag{B.6}$$

where $\sigma(x) = 0.5 \times (1 + \tanh(\frac{\ell - x}{s}))$. This can also be written as

$$\mathrm{CP}(k_1, k_2) = \boldsymbol{\sigma} k_1 + \bar{\boldsymbol{\sigma}} k_2 \tag{B.7}$$

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

Changewindow, $\mathrm{CW}(\cdot, \cdot)$, operators are defined similarly by replacing the sigmoid, $\sigma(x)$, with a product of two sigmoids.

B.1.3 Properties of the periodic kernel

A simple application of l'Hôpital's rule shows that

$$\mathrm{Per}(x, x') \to \sigma^2 \cos\left(\frac{2\pi(x - x')}{p}\right) \quad \text{as} \quad \ell \to \infty. \tag{B.8}$$

This limiting form is written as the cosine kernel (cos).

B.2 Model construction / search

B.2.1 Overview

The model construction phase of ABCD starts with the kernel equal to the noise kernel, WN. New kernel expressions are generated by applying search operators to the current kernel. When new base kernels are proposed by the search operators, their parameters are randomly initialised with several restarts. Parameters are then optimized by conjugate gradients to maximise the likelihood of the data conditioned on the kernel parameters. The kernels are then scored by the Bayesian information criterion and the top scoring kernel is selected as the new kernel. The search then proceeds by applying the search operators to the new kernel i.e. this is a greedy search algorithm.

In all experiments, 10 random restarts were used for parameter initialisation and the search was run to a depth of 10.

B.2.2 Search operators

ABCD is based on a search algorithm which used the following search operators

$$\mathcal{S} \rightarrow \mathcal{S} + \mathcal{B} \tag{B.9}$$
$$\mathcal{S} \rightarrow \mathcal{S} \times \mathcal{B} \tag{B.10}$$
$$\mathcal{B} \rightarrow \mathcal{B}' \tag{B.11}$$

where $\mathcal{S}$ represents any kernel subexpression and $\mathcal{B}$ is any base kernel within a kernel expression i.e. the search operators represent addition, multiplication and replacement.

To accommodate changepoint/window operators we introduce the following additional operators

$$\mathcal{S} \rightarrow \mathrm{CP}(\mathcal{S}, \mathcal{S}) \tag{B.12}$$
$$\mathcal{S} \rightarrow \mathrm{CW}(\mathcal{S}, \mathcal{S}) \tag{B.13}$$
$$\mathcal{S} \rightarrow \mathrm{CW}(\mathcal{S}, \mathrm{C}) \tag{B.14}$$
$$\mathcal{S} \rightarrow \mathrm{CW}(\mathrm{C}, \mathcal{S}) \tag{B.15}$$

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

Based on experience with typical paths followed by the search algorithm we introduced the following operators

$$\mathcal{S} \rightarrow \mathcal{S} \times (\mathcal{B} + \mathrm{C}) \tag{B.16}$$
$$\mathcal{S} \rightarrow \mathcal{B} \tag{B.17}$$
$$\mathcal{S} + \mathcal{S}' \rightarrow \mathcal{S} \tag{B.18}$$
$$\mathcal{S} \times \mathcal{S}' \rightarrow \mathcal{S} \tag{B.19}$$

where $\mathcal{S}'$ represents any other kernel expression. Their introduction is currently not rigorously justified.

B.3 Predictive accuracy

Interpolation To test the ability of the methods to interpolate, we randomly divided each data set into equal amounts of training data and testing data. We trained each algorithm on the training half of the data, produced predictions for the remaining half

and then computed the root mean squared error (RMSE). The values of the RMSEs are then standardised by dividing by the smallest RMSE for each data set i.e. the best performance on each data set will have a value of 1.

Figure B.1 shows the standardised RMSEs for the different algorithms. The box plots show that all quartiles of the distribution of standardised RMSEs are lower for both versions of ABCD. The median for ABCD-accuracy is 1; it is the best performing algorithm on 7 datasets. The largest outliers of ABCD and spectral kernels are similar in value.

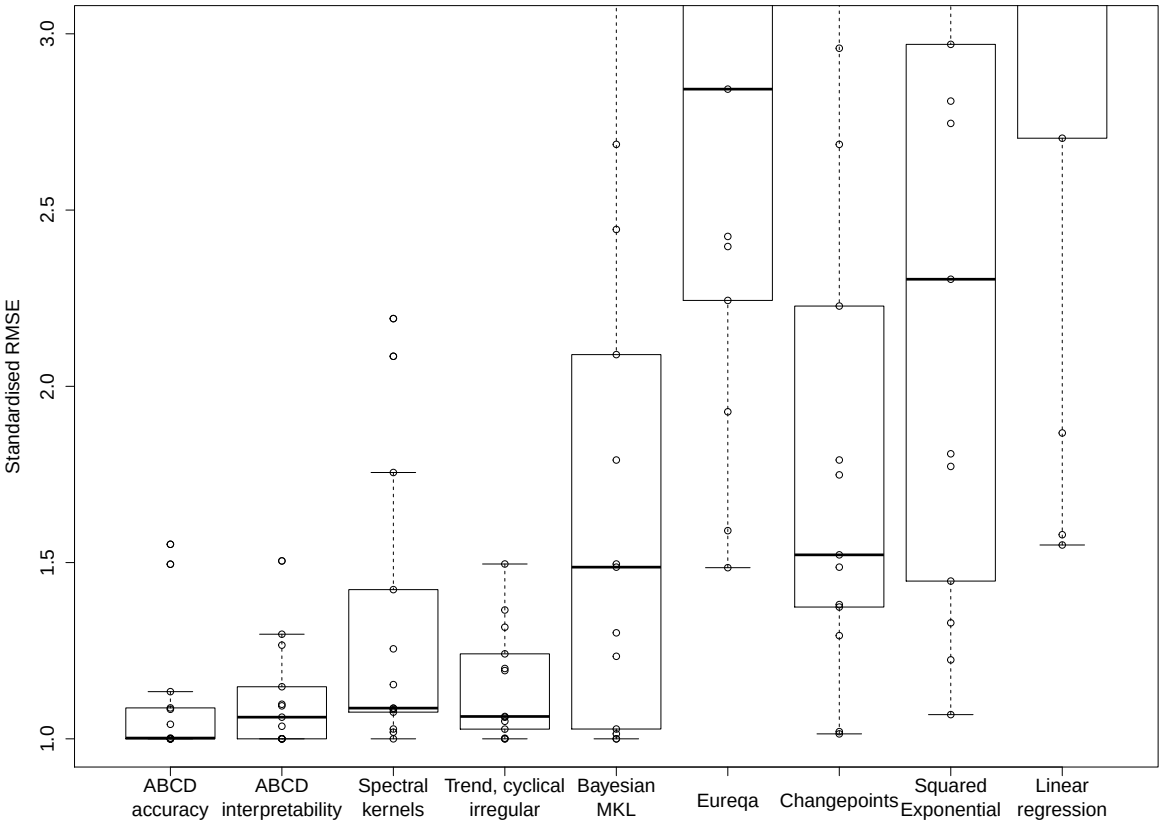


Fig. B.1 Box plot of standardised RMSE (best performance = 1) on 13 interpolation tasks.

Changepoints performs slightly worse than MKL despite being strictly more general than Changepoints. The introduction of changepoints allows for more structured models, but it introduces parametric forms into the regression models (i.e. the sigmoids expressing the changepoints). This results in worse interpolations at the locations of the

change points, suggesting that a more robust modeling language would require a more flexible class of changepoint shapes or improved inference (e.g. fully Bayesian inference over the location and shape of the changepoint).

Eureqa is not suited to this task and performs poorly. The models learned by Eureqa tend to capture only broad trends of the data since the fine details are not well explained by parametric forms.

B.3.1 Tabels of standardised RMSEs

See table B.1 for raw interpolation results and table B.2 for raw extrapolation results. The rows follow the order of the datasets in the rest of the supplementary material. The following abbreviations are used: ABCD-accuracy (ABCD-acc), ABCD-interpretability ((ABCD-int), Spectral kernels (SP), Trend-cyclical-irregular (TCI), Bayesian MKL (MKL), Eureqa (EL), Changepoints (CP), Squared exponential (SE) and Linear regression (Lin).

ABCD-acc	ABCD-int	SP	TCI	MKL	EL	CP	SE	Lin
1.04	1.00	2.09	1.32	3.20	5.30	3.25	4.87	5.01
1.00	1.27	1.09	1.50	1.50	3.22	1.75	2.75	3.26
1.00	1.00	1.09	1.00	2.69	26.20	2.69	7.93	10.74
1.09	1.04	1.00	1.00	1.00	1.59	1.37	1.33	1.55
1.00	1.06	1.08	1.06	1.01	1.49	1.01	1.07	1.58
1.50	1.00	2.19	1.37	2.09	7.88	2.23	6.19	7.36
1.55	1.50	1.02	1.00	1.00	2.40	1.52	1.22	6.28
1.00	1.30	1.26	1.24	1.49	2.43	1.49	2.30	3.20
1.00	1.09	1.08	1.06	1.30	2.84	1.29	2.81	3.79
1.08	1.00	1.15	1.19	1.23	42.56	1.38	1.45	2.70
1.13	1.00	1.42	1.05	2.44	3.29	2.96	2.97	3.40
1.00	1.15	1.76	1.20	1.79	1.93	1.79	1.81	1.87
1.00	1.10	1.03	1.03	1.03	2.24	1.02	1.77	9.97

Table B.1 Interpolation standardised RMSEs

ABCD-acc	ABCD-int	SP	TCI	MKL	EL	CP	SE	Lin
1.14	2.10	1.00	1.44	4.73	3.24	4.80	32.21	4.94
1.00	1.26	1.21	1.03	1.00	2.64	1.03	1.61	1.07
1.40	1.00	1.32	1.29	1.74	2.54	1.74	1.85	3.19
1.07	1.18	3.00	3.00	3.00	1.31	1.00	3.03	1.02
1.00	1.00	1.03	1.00	1.35	1.28	1.35	2.72	1.51
1.00	2.03	3.38	2.14	4.09	6.26	4.17	4.13	4.93
2.98	1.00	11.04	1.80	1.80	493.30	3.54	22.63	28.76
3.10	1.88	1.00	2.31	3.13	1.41	3.13	8.46	4.31
1.00	2.05	1.61	1.52	2.90	2.73	3.14	2.85	2.64
1.00	1.45	1.43	1.80	1.61	1.97	2.25	1.08	3.52
2.16	2.03	3.57	2.23	1.71	2.23	1.66	1.89	1.00
1.06	1.00	1.54	1.56	1.85	1.93	1.84	1.66	1.96
3.03	4.00	3.63	3.12	3.16	1.00	5.83	5.35	4.25

Table B.2 Extrapolation standardised RMSEs

References