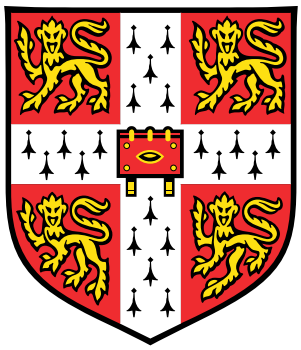
Automatic Model Construction with Gaussian Processes



David Kristjanson Duvenaud

University of Cambridge

This dissertation is submitted for the degree of

Doctor of Philosophy

Pembroke College

June 2014

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text. This dissertation contains less than 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 Rasmussen, for so much advice and encouragement. It was wonderful having an advisor who had spent many years thinking deeply about the business of modeling, who also allowed me the freedom to make my own mistakes. 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

This thesis develops a broad class of models, useful for learning and forecasting in such domains as time series, geological formations, and physical dynamics. These models are based on Gaussian processes, which can express a wide variety of statistical structure, depending on the choice of kernel. Historically, the type of kernel has been chosen by hand by experts. In this thesis, we show how to automate this task, creating an artificial statistician capable of systematically exploring a large space of models.

The introductory chapters show many types of structure, such as periodicity, change-points, additivity, and symmetries can be encoded by kernels, and that these structure can be combined by combining kernels. For example, compound kernels can produce priors over topological manifolds such as cylinders, toruses, 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. To do so, 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.

This thesis also explores several extensions to Gaussian process models. First, 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*. Second, 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*. Third, 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 latent space in which each cluster has an interpretable parametric form.

Contents

List of Figures

List of Tables

Notation

Unbolded x represents a single number, $\mathbf{x}$ represents a vector, and $\mathbf{X}$ represents a matrix. A individual element of a vector will be denoted with a subscript and without boldface. For example, 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
$y \sim p(y)$	y is drawn from, or distributed according to, distribution $p(y)$
$\mathcal{O}(\cdot)$	The big-O asymptotic complexity of an algorithm.
$A \otimes B$	The Kronecker product of matrices A and B.
SE	The squared-exponential kernel, also known as the radial-basis function (RBF) kernel, or the Gaussian kernel.
RQ	The rational-quadratic kernel.
Per	The periodic kernel.
Lin	The linear kernel.
WN	The white-noise kernel.
C	The constant kernel.
$\boldsymbol{\sigma}$	The changepoint kernel, $\boldsymbol{\sigma}(x, x') = \sigma(x)\sigma(x')$, where $\sigma(x)$ is a sigmoidal function such as the logistic function.
$k_1 + k_2$	Addition of kernels, shorthand for $k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')$
$k_1 \times k_2$	Multiplication of kernels, shorthand for $k_1(\mathbf{x}, \mathbf{x}') \times k_2(\mathbf{x}, \mathbf{x}')$
$k(\mathbf{X}, \mathbf{X})$	The Gram matrix, whose i, jth element is given by $k(\mathbf{x}_i, \mathbf{x}_j)$.
$\mathbf{K}$	Shorthand for the Gram matrix $k(\mathbf{X}, \mathbf{X})$
$\boldsymbol{f}(\mathbf{X})$	A vector of function values, whose ith element is given by $f(\mathbf{x}_i)$.
$\mathrm{vec}(\mathbf{X})$	The vectorization operator, which concatenates each column of $\mathbf{X}$ into a column vector.
$\mathrm{mod}(i, j)$	The modulo operator: the remainder after dividing i by j.

Chapter 1

Introduction

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

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

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

This thesis will show how to build such a language using Gaussian processes (GPs), a tractable set of models of very different types of functions. This chapter will introduce the basic properties of GPs. The next chapter will describe the many types of functions that we know how to model using GPs.

1.1 Gaussian Process Models

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

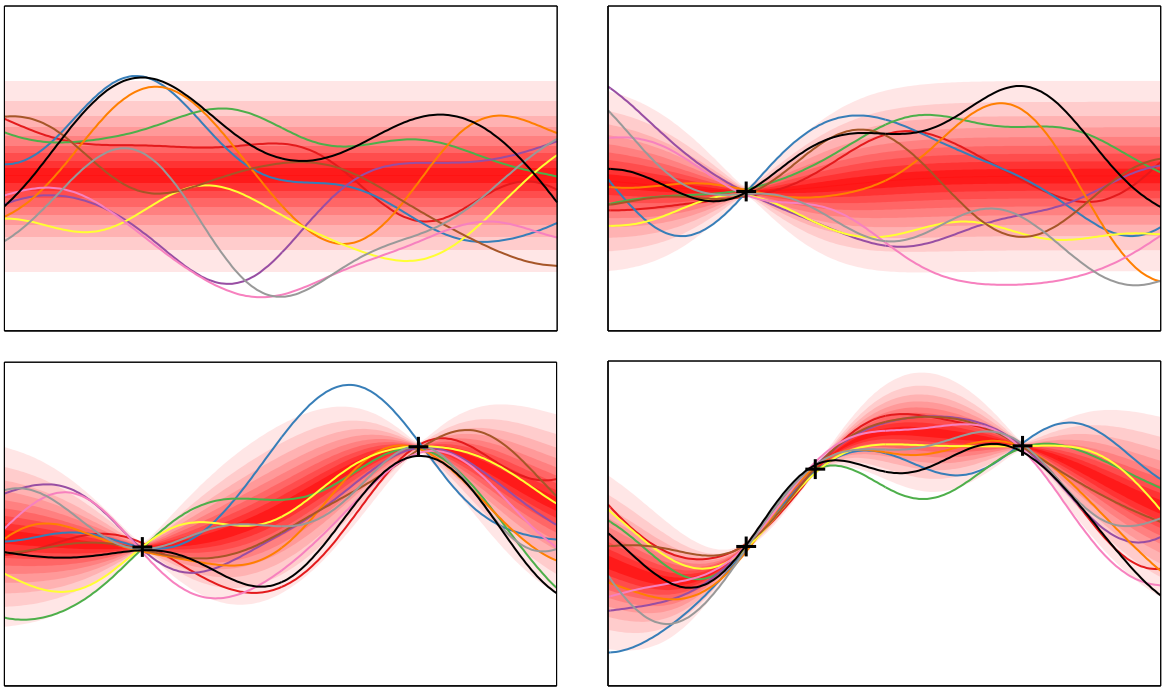


Fig. 1.1 A visual representation of a one-dimensional Gaussian process posterior. Different shades of red correspond to deciles of the predictive density at each input location. Coloured lines show samples from the process. *Top left:* A GP not conditioned on any datapoints. *Remaining plots:* The posterior after conditioning on different amounts of data.

ditioning on data, is completely specified by its mean function,

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

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

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

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

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

There are many possible choices of covariance function, and we can specify a wide

range of models just by specifying the kernel of a GP. For example, linear regression, splines, and Kalman filters are all examples of GPs with particular kernels. However, these model classes barely scratch the surface of the many possible models we can express through choosing a kernel. One of the main difficulties in using GPs is constructing a kernel which represents the particular structure present in the data being modelled.

1.1.1 Model Selection

The crucial property of GPs that allows us to automatically construct models is that we can compute the *marginal likelihood* of a particular model, also known as the *evidence* **?**. The marginal likelihood allows us to compare models and automatically discover the appropriate amount of detail to use, due to Bayesian Occam's razor (**??**). Choosing a kernel, or kernel parameters, by maximizing the marginal likelihood will typically select the *least* flexible model class which still captures all the structure in the data. For example, if a kernel has a parameter which controls the smoothness of the functions it models, the maximum-likelihood estimate of that parameter will usually correspond to the smoothest possible family of functions which still go through all the observed function values.

Here's the marginal likelihood under a GP of observing a set of function values $[f(\mathbf{x}_1), f(\mathbf{x}_2), \dots f(\mathbf{x}_N)] = \boldsymbol{f}(\mathbf{X})$ at locations $[\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N]^\mathsf{T} = \mathbf{X}$:

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

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

1.1.2 Prediction

Even though we don't need to consider locations other than at the data when computing the marginal likelihood, we can still ask the model which function values are likely to

occur at any location, given the observations we've seen. By the formula for Gaussian conditionals (described in appendix **??**), the predictive distribution of a function value $f(\mathbf{x}^\star)$ at a test point $\mathbf{x}^\star$ has a simple form:

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

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

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

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

1.1.3 Useful Properties of Gaussian Processes

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

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

- **Expressivity.** 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, such as neural networks. 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 hypotheses is that we can compute the *marginal likelihood* of the data given the model. This gives us a principled way of comparing different models.

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

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

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

1.1.4 Limitations of Gaussian Processes

There are, unfortunately, several issues which make usage of GPs sometimes difficult:

- **Slow inference.** Computing the matrix inverse in **????** takes $\mathcal{O}(N^3)$ time, making inference slow for more than about 1000 datapoints. However, this problem can be addressed by approximate inference schemes (**???**). Most GP software packages implement several of these methods (**??**).

- **Light tails of the predictive distribution.** The predictive distribution of a standard GP model is Gaussian. In order to be robust to outliers, or to perform classification, we may wish to use non-Gaussian noise models. Fortunately, mature software packages exist which can automatically perform approximate inference for a wide variety of non-Gaussian likelihoods.

- **The need to choose a kernel.** 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 a good representation of the input. Kernel parameters are usually set automatically by maximizing the marginal likelihood. However, until recently, human experts were still required to choose the parametric form of the kernel from a small set of

standard kernels. **??** will show how the entire construction of kernels can be automated.

1.2 Outline and Contributions of Thesis

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

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

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

One benefit of using a compositional model class is that the resulting models are interpretable. **Chapter ??** demonstrates a system which automatically describes the structure implied by a given kernel on a given dataset, generating reports with graphs and English-language text describing the resulting model. We'll show several automatic analyses of time-series. Combined with the automatic model search developed in chapter **??**, this system represents the beginnings of an "automatic statistician".

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

Chapter ?? examines a more limited, but much faster way of discovering structure using GPs. Specifying a kernel with many different types of structure, we use kernel parameters to discard whichever types of structure *aren't* found in the current dataset.

The model class we examine is called *additive Gaussian processes*, a model summing over exponentially-many GPs, each depending on a different subset of the input variables. We give a polynomial-time inference algorithm for this model class, and relate it to other model classes. For example, additive GPs are shown to have the same covariance as a GP that uses *dropout*, a recently discovered regularization technique for neural networks.

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

1.3 Attribution

This thesis was made possible 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, who made this research enjoyable.

Structure through kernels: Section **??** of chapter **??**, describing how kernel symmetries 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 **??** 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 most of the experiments. I produced all of the figures. 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 **??** was written in collaboration with James Robert Lloyd, Roger Grosse, Joshua B. Tenenbaum, Zoubin Ghahramani, and was published in (**?**). The procedure translating kernels into adjectives grew out of discussions between James and myself. James Lloyd wrote the code to automat-

ically 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.

Deep Gaussian processes: The ideas contained in chapter **??** 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.

Additive Gaussian processes: The work in chapter **??** was done in collaboration with Hannes Nickisch and Carl Rasmussen, who derived and coded up the initial model. My role in the project was to examine the properties of the resulting model, clarify the connections to existing methods, to create all figures and run all experiments. This work was published in (**?**). The connection to dropout regularization was my own contribution.

Warped mixtures: The work comprising the bulk of chapter **??** 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 guidance and many helpful suggestions throughout the project.

Chapter 2

Expressing Structure with Kernels

In this chapter, we'll show how to use kernels 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 changepoints. We'll also show several ways to encode group invariants into kernels. Combining kernels in these simple ways will give us a rich, open-ended language of models.

2.1 Definition

Since we'll be discussing kernels at length, we now give a precise definition. A kernel (also called a *covariance function*), is a positive-definite function between two points x, x' in some space $\mathcal{X}$. Formally, $k(\mathbf{x}, \mathbf{x}') : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. In this chapter, $\mathcal{X}$ is usually a Euclidean space, but it could just as easily corresponding to the space of images, document, categories or points on a sphere.

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 a slightly misleading statement in this context, 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 A Few Basic Kernels

To begin understanding the types of structures expressible by GPs, we'll start by briefly examining the types of structure encoded by a diverse set of commonly used kernels: the squared-exponential (SE), periodic (Per), and linear (Lin) kernels. These kernels are defined in Table **??**.

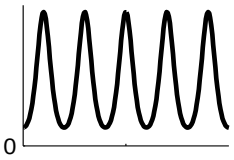
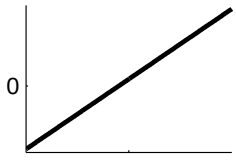
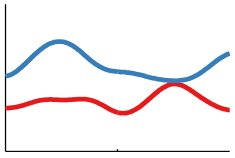
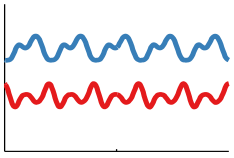
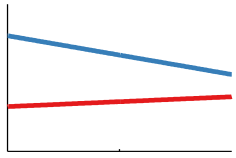
Kernel name:	Squared-exp (SE)	Periodic (Per)	Linear (Lin)
$k(x, x') =$	$\exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$	$\exp\left(-\frac{2}{\ell^2}\sin^2\left(\pi\frac{x-x'}{p}\right)\right)$	$(x-c)(x'-c)$
Plot of kernel:			
	$x - x'$	$x - x'$	x (with $x' = 1$)
Samples from prior:			
Type of structure:	local variation	repeating structure	linear functions

Table 2.1 Examples of structures expressible by some basic kernels.

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

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

Stationary and Non-stationary The SE and Per kernels are *stationary*, meaning that their value only depends on the difference $x - x'$. This implies that the probability of observing a particular dataset remains the same, even if we move all the $\mathbf{x}$ values

over by some amount. In contrast, the linear kernel Lin is non-stationary, meaning that the corresponding GP model will produce different answers if the data is moved around while the kernel parameters are kept fixed.

2.3 Combining Kernels

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

2.3.1 Notation

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

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

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

All of the basic kernels we considered in **??** are one-dimensional, but kernels over multidimensional inputs can be constructed by adding and multiplying between kernels on different dimensions. The dimension on which a kernel operates is denoted by a subscripted integer. For example, SE_2 represents an SE kernel over the second dimension of $\mathbf{x}$. To remove clutter, we'll usually refer to a kernels without specifying their parameters.

2.3.2 Combining Properties through Multiplication

Multiplying two positive-definite kernels together always results in another positive-definite kernel. But what properties do these new kernels have? **??** shows some interesting kernels that one can obtain by multiplying two basic kernels together.

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

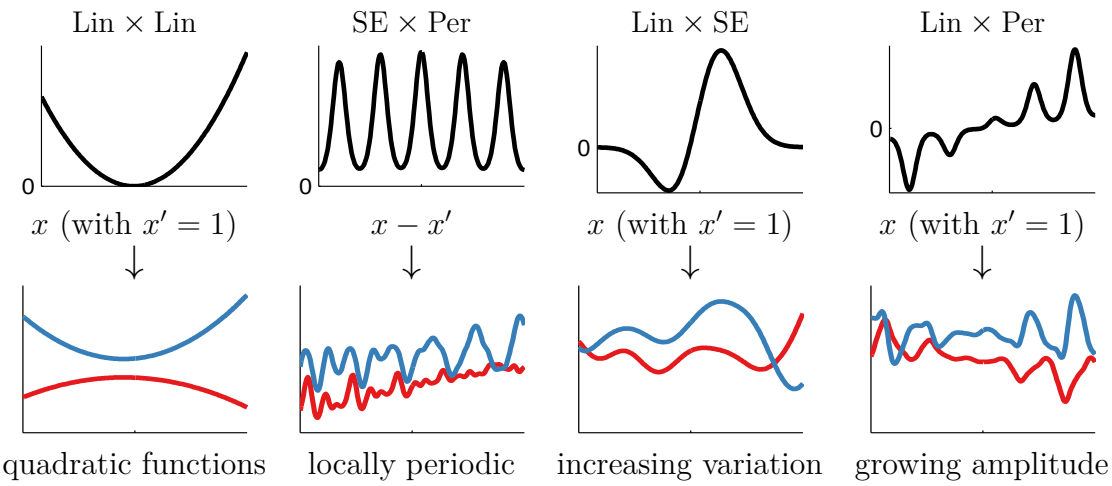


Fig. 2.1 Examples of one-dimensional structures expressible by multiplying kernels. Plots have same meaning as in figure **??**.

Polynomial Regression

By multiplying together T linear kernels, we obtain a prior on polynomials of degree T. Column 1 of **??** shows a quadratic kernel.

Locally Periodic Functions

In univariate data, multiplying a kernel by SE gives a way of converting global structure to local structure. For example, Per corresponds to exactly periodic structure, whereas Per×SE corresponds to locally periodic structure, as shown in column 2 of **??**.

Functions with Growing Amplitude

Multiplying by a linear kernel means that the marginal standard deviation of the function being modeled will grow linearly away from the origin. Columns 3 and 4 of **??** show two examples.

More Combinations

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

2.3.3 Building Flexible Multidimensional Models

We can build flexible models of functions of more than one input simply by multiplying kernels between the different inputs. For example, products of SE kernels, having a different lengthscale parameter for each dimension, are called the SE-ARD kernel:

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

?? illustrates the SE-ARD kernel in two dimensions.

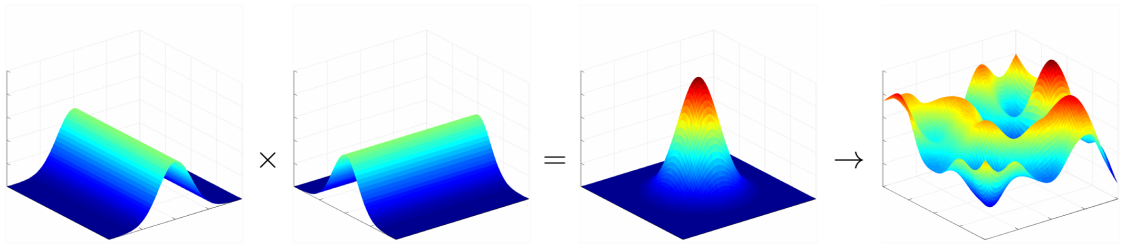


Fig. 2.2 A product of two one-dimensional kernels gives rise to a prior on functions which depend on both dimensions.

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

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

However, this flexibility means that they can sometimes be relatively slow to learn, due to the *curse of dimensionality* (**?**). In general, the more structure we account for in the data, the less data we'll need - the *blessing of abstraction* (**?**) counters the curse of dimensionality. Below, we'll investigate ways to encode more structure into our kernels.

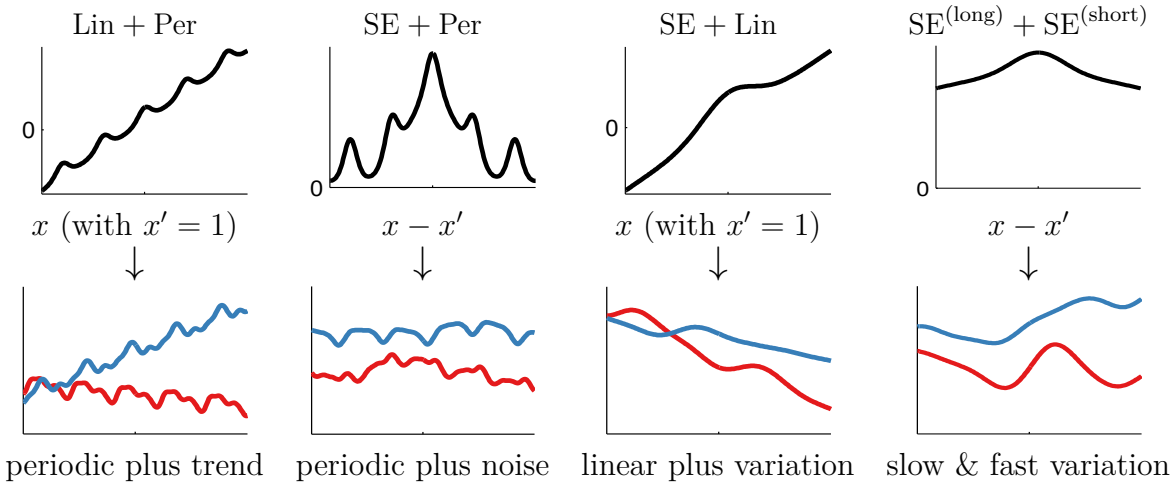


Table 2.2 Examples of one-dimensional structures expressible by adding kernels. Rows have the same meaning as in **??**. $\mathrm{SE}^{(\mathrm{long})}$ denotes a SE kernel whose lengthscale is long relative to that of $\mathrm{SE}^{(\mathrm{short})}$

2.4 Modeling Sums of Functions

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

Fortunately, it's easy to encode additivity into encode into GP models. Suppose functions f_1, f_2 are drawn independently from GP priors:

$$f_a \sim \mathcal{GP}(\mu_a, k_a) \tag{2.5}$$
$$f_b \sim \mathcal{GP}(\mu_b, k_b) \tag{2.6}$$

Then the sum of those functions is simply another GP:

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

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

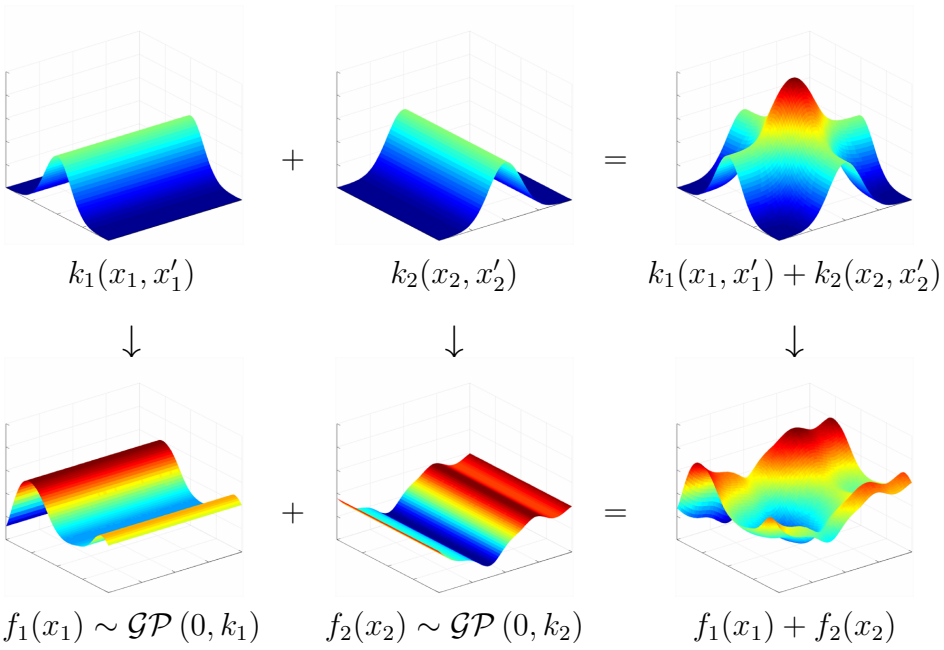


Fig. 2.3 A sum of two orthogonal one-dimensional kernels. *Top row:* An additive kernel is any sum of kernels. *Bottom row:* A draw from an additive kernel corresponds to a sum of draws from independent GP priors, each having the corresponding kernel.

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

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

Then this is equivalent to the model

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

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

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

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

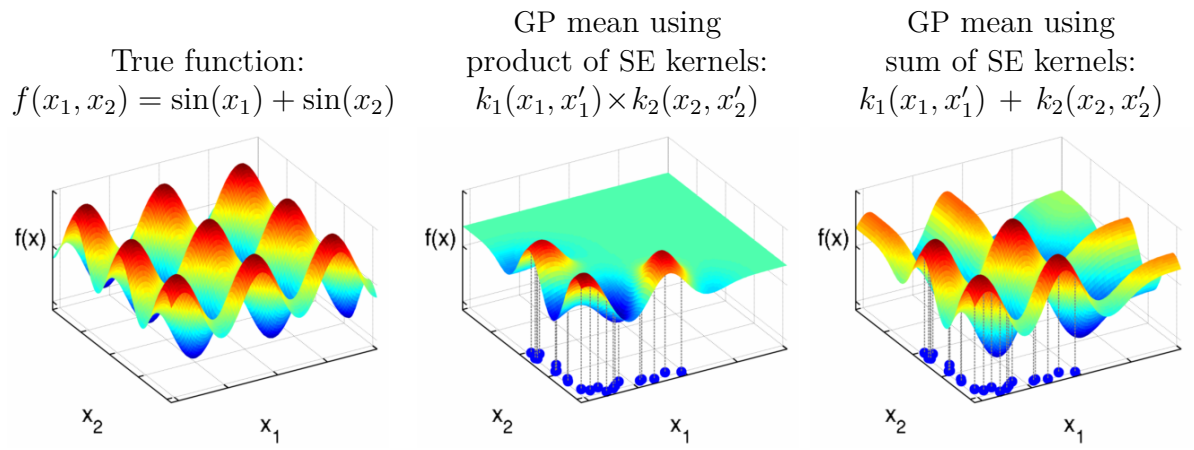


Fig. 2.4 Inference in functions with additive structure. When the function being modeled has additive structure, we can exploit this fact to extrapolate far from the training data. The product kernel allows a different function value for every combination of inputs, and so is uncertain about function values away from the training data.

2.4.2 Long-range Extrapolation through Additivity

Additive structure often allows us to make predictions far from the training data. **??** 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 height of the function at unseen combinations of inputs. The product-kernel model is more flexible, and so remains uncertain about the function away from the data.

These types of additive models have been well-explored in the statistics literature. For example, generalized additive models (**?**) have seen wide adoption. In high dimensions, we can also consider sums of functions of more than one dimension. **??** considers this model class in more detail.

2.4.3 Example: An Additive Model of Concrete Strength

To illustrate how additive kernels give rise to interpretable models, we'll build an additive model of the strength of concrete as a function of the amount of 7 different ingredients, plus the age of the concrete (**?**).

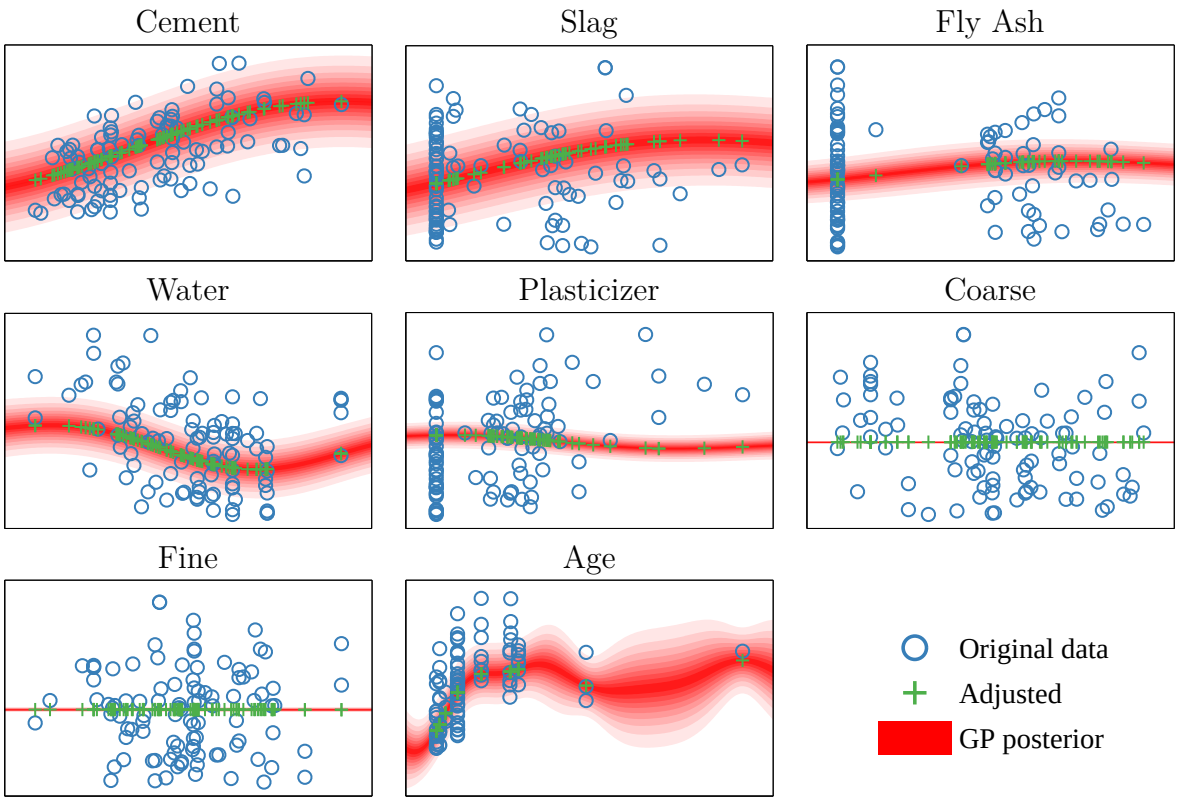


Fig. 2.5 By considering only one-dimensional terms of the additive kernel, we can plot the predictive distribution as a function of each dimension separately. Blue points indicate the original data, green points are data after the average contribution from all other terms has been subtracted. The vertical axis is the same for all plots.

Our simple additive model looks like

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

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

?? shows the marginal posterior distribution of each of the 8 components in **??**. 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, and

another example of automatic relevance determination (ARD) (**?**).

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

2.4.4 Posterior Variance of Additive Components

Here we derive the posterior variance and covariance of all of the additive components of a GP. These formulas allow us to make plots such as **??**.

First, we'll write down the joint prior over the sum of two functions drawn from GP priors. 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

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

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

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

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

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

By the formula for Gaussian conditionals, (given by **??**), we get that the conditional distribution of a GP-distributed function conditioned on its sum with another GP-distributed function is given by

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

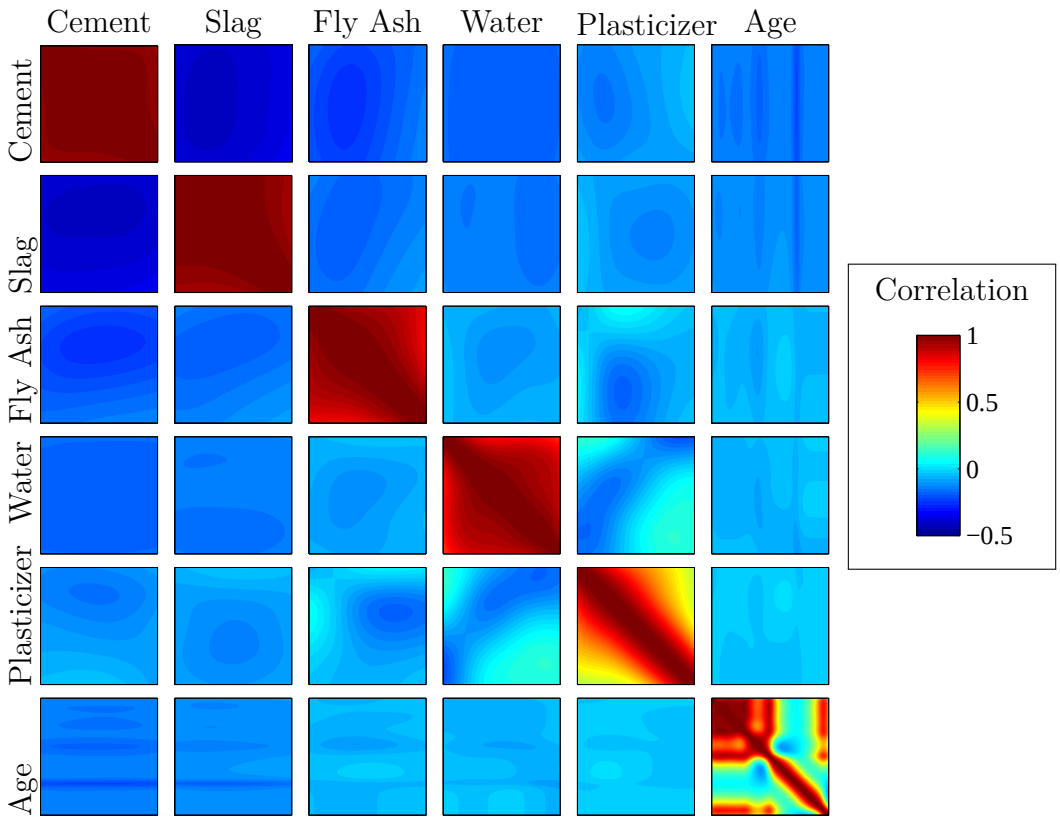


Fig. 2.6 Posterior correlations between the components explaining the concrete dataset. Each plot shows the additive model's posterior correlations between two components, plotted over the domain of the data ±5%. Red indicates high correlation, teal indicates no correlation, and blue indicates negative correlation. Plots on the diagonal show posterior correlations within each component.

These formulae express the model's posterior uncertainty about the 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 Additive Components

We can also compute the posterior covariance between any two components, conditioned on their sum: If this quantity is negative, it means that there is ambiguity about which of the two components explains the data at that location.

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

?? shows the posterior correlation between all non-zero components of the concrete model. Most of the correlation occurs within components, but there is also negative correlation between the "Cement" and "Slag" variables. This reflects an ambiguity in the model about which one of these functions is high and the other low. Dimensions 'Coarse' and 'Fine' are not shown, because their variance was zero.

2.5 Changepoints

A simple example of how combining kernels can give rise to more structured priors is given by changepoint kernels. Changepoints kernels can be defined through addition and multiplication with sigmoidal functions:

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

which can be written in shorthand as

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

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

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

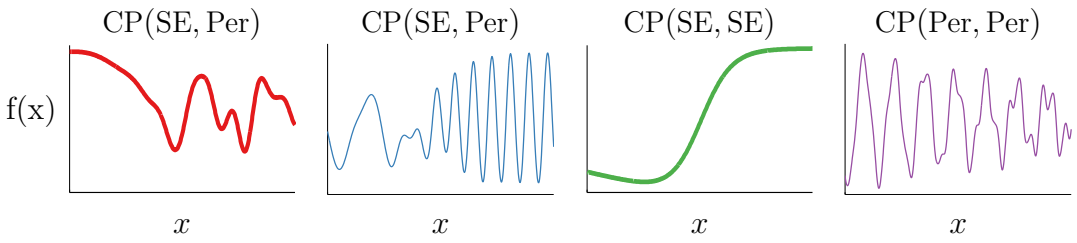


Fig. 2.7 Draws from different priors on using changepoint kernels, constructed by adding and multiplying together base kernels with sigmoidal functions.

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

2.5.1 Multiplication by a Known Function

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

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

2.6 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':

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

As a simple example, the squared-exponential kernel (SE) on the real line has a representation in terms of infinitely many radial-basis functions of the form $h_i(x) \propto \exp(-\frac{1}{2}\frac{(x-c_i)^2}{2\ell^2})$. More generally, any stationary kernel on the real line can be represented by a set of sines and cosines - a Fourier representation (**?**). In general, any particular 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 **??**.

2.6.1 Relation to Linear Regression

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

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

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

2.6.2 Feature-space view of Combining Kernels

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

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

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

and we consider their addition, then

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

$$= \begin{bmatrix} \mathbf{a}(\mathbf{x}) \\ \mathbf{b}(\mathbf{x}) \end{bmatrix}^\mathsf{T} \begin{bmatrix} \mathbf{a}(\mathbf{x}') \\ \mathbf{b}(\mathbf{x}') \end{bmatrix} \tag{2.27}$$

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:

$$k_a(\mathbf{x}, \mathbf{x}') \times k_b(\mathbf{x}, \mathbf{x}') = \left[\mathbf{a}(\mathbf{x})^\mathsf{T}\mathbf{a}(\mathbf{x}')\right] \times \left[\mathbf{b}(\mathbf{x})^\mathsf{T}\mathbf{b}(\mathbf{x}')\right] \tag{2.28}$$

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

$$= \sum_i \sum_j a_i(\mathbf{x})a_i(\mathbf{x}')b_j(\mathbf{x})b_j(\mathbf{x}') \tag{2.30}$$

$$= \sum_{i,j} \left[a_i(\mathbf{x})b_j(\mathbf{x})\right]\left[a_i(\mathbf{x}')b_j(\mathbf{x}')\right] \tag{2.31}$$

$$= \operatorname{vec}\left(\mathbf{a}(\mathbf{x}) \otimes \mathbf{b}(\mathbf{x}')\right)^\mathsf{T} \operatorname{vec}\left(\mathbf{a}(\mathbf{x}) \otimes \mathbf{b}(\mathbf{x}')\right) \tag{2.32}$$

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, the Cartesian product of the features of two one-dimensional SE kernels covers the plane with two-dimensional radial-basis functions of the form

$$h_{ij}(x_1, x_2) \propto \exp\left(-\frac{1}{2}\frac{(x_1 - c_i)^2}{2\ell_1^2}\right) \exp\left(-\frac{1}{2}\frac{(x_2 - c_j)^2}{2\ell_2^2}\right) \tag{2.33}$$

2.7 Expressing Symmetries and Invariants

When modeling functions, encoding known symmetries can improve predictive accuracy. In this section, we'll look at different ways in which we can encode symmetries into a

prior on functions. Many types of symmetry can be enforced through operations on the kernel.

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

? and **?** characterized the set of GP priors on functions which respect any given invariance. 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 with respect to 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 (up to a modification) if and only if $k(\cdot, \cdot)$ is argument-wise invariant:

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

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

$$g_1(f(x, y)) = f(x, y) \qquad \text{(identity)} \tag{2.35}$$

$$g_2(f(x, y)) = f(y, x) \qquad \text{(swap)} \tag{2.36}$$

It might not be clear how to find a kernel obeying these symmetries.

2.7.1 Three Recipes for Group Invariance

Fortunately, for finite groups, there are a few simple ways to transform any kernel into one which is argument-wise invariant to actions under any finite group:

1. **Sum over the Orbit.** **?** and **?** suggest enforcing invariants through a double sum over the orbits of $\mathbf{x}$ and $\mathbf{x}'$ with respect to G:

$$k_{\text{sum}}(\mathbf{x}, \mathbf{x}') = \sum_{g, \in G} \sum_{g' \in G} k(g(\mathbf{x}), g'(\mathbf{x}')) \tag{2.37}$$

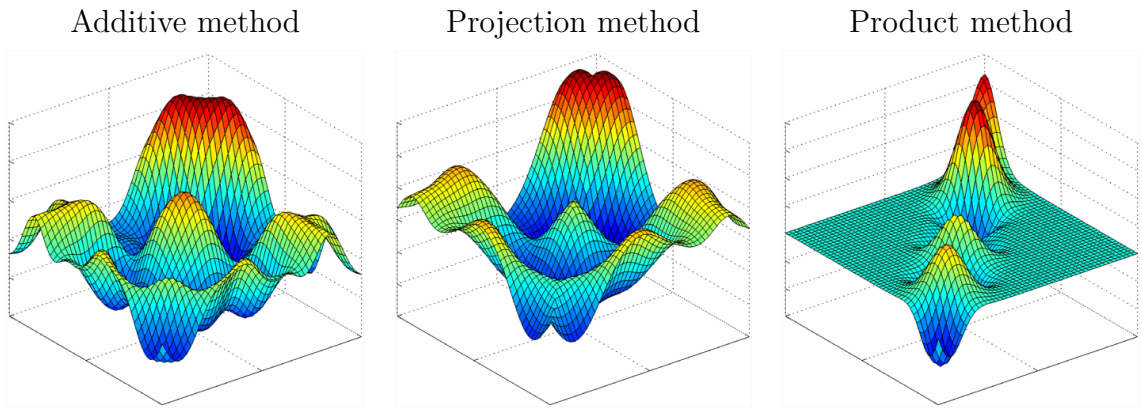


Fig. 2.8 Three methods of introducing symmetry, illustrated through draws from the corresponding priors. All three methods introduce a different type of nonstationarity.

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

$$k_{\text{switch}}(\mathbf{x}, \mathbf{x}') = \sum_{g, \in G_{\text{swap}}} \sum_{g' \in G_{\text{swap}}} k(g(\mathbf{x}), g'(\mathbf{x}')) \tag{2.38}$$

$$= k(x, y, x', y') + k(x, y, y', x') + k(y, x, x', y') + k(y, x, y', x') \tag{2.39}$$

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

2. **Project onto a Fundamental Domain.** **?** also explore the possibility of projecting each datapoint into a fundamental domain of the group, using a mapping A_G:

$$k_{\text{proj}}(\mathbf{x}, \mathbf{x}') = k(A_G(\mathbf{x}), A_G(\mathbf{x}')) \tag{2.40}$$

For the group G_{swap}, a fundamental domain is $\{x, y : x < y\}$, which can be mapped to using $A_{G_{\text{swap}}}(x, y) = \big[\min(x, y), \max(x, y)\big]$. Constructing a kernel using this method introduces a non-differentiable "seam" along $x = y$, as shown in **??**(b). The projection method also works for infinite groups, as we shall see below.

3. **Multiply over the Orbit.** Ryan P. Adams (personal communication) suggests

a construction enforcing invariants through products over the orbits:

$$k_{\text{sum}}(\mathbf{x}, \mathbf{x}') = \prod_{g, \in G} \prod_{g' \in G} k(g(\mathbf{x}), g'(\mathbf{x}')) \tag{2.41}$$

This method can sometimes produce GP priors with zero variance in some regions of the space, as in **??**(c).

There are many possible ways to achieve a given symmetry, but we must be careful to do so without compromising other qualities of the model we are constructing. For example, simply setting $k(\mathbf{x}, \mathbf{x}') = 0$ gives rise to a GP prior which obeys *all possible* symmetries, but this is presumably not a model we wish to use.

2.7.2 Periodicity

Periodicity in a one-dimensional function corresponds to the invariance

$$f(x) = f(x + \tau) \tag{2.42}$$

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

$$\text{Per}(x, x') = \text{SE}\left(\left[\sin(x), \cos(x)\right], \left[\sin(x'), \cos(x')\right]\right) \tag{2.43}$$

achieves the invariance in Equation (**??**). Simple algebra reduces this kernel to the form shown in Table **??**.

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

2.7.3 Symmetry About Zero

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

$$f(x) = f(-x) \tag{2.44}$$

using the sum over orbits method, by the transform

$$k_{\text{reflect}}(x, x') = k(x, x') + k(x, -x') + k(-x, x') + k(-x, -x') \tag{2.45}$$

2.7.4 Translation Invariance in Images

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

This sort of translation invariance is completely distinct from the stationarity of kernels such as SE or Per. A stationary kernel implies that the prior is invariant to translations of the entire training and test set. In contrast, we use translation invariance to refer to a the situation where the a signal has been discretized, and each pixel (or the audio equivalent) corresponds to a different input dimension. We are interested in creating priors on functions that are invariant to swapping pixels in a manner that corresponds to shifting a in some direction:

$$f\left(\quad\right) = f\left(\quad\right) \tag{2.46}$$


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

$$\text{shift}(\mathbf{x}, i) = \left[x_{\text{mod}(i+1,D)}, x_{\text{mod}(i+2,D)}, \ldots, x_{\text{mod}(i+D,D)}\right]^{\mathsf{T}} \tag{2.47}$$

As above, translation invariance in one dimension can be achieved by the transformation

$$k_{\text{invariant}}(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{D} \sum_{j=1}^{D} k(\text{shift}(\mathbf{x}, i), \text{shift}(\mathbf{x}, j)), \tag{2.48}$$

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

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

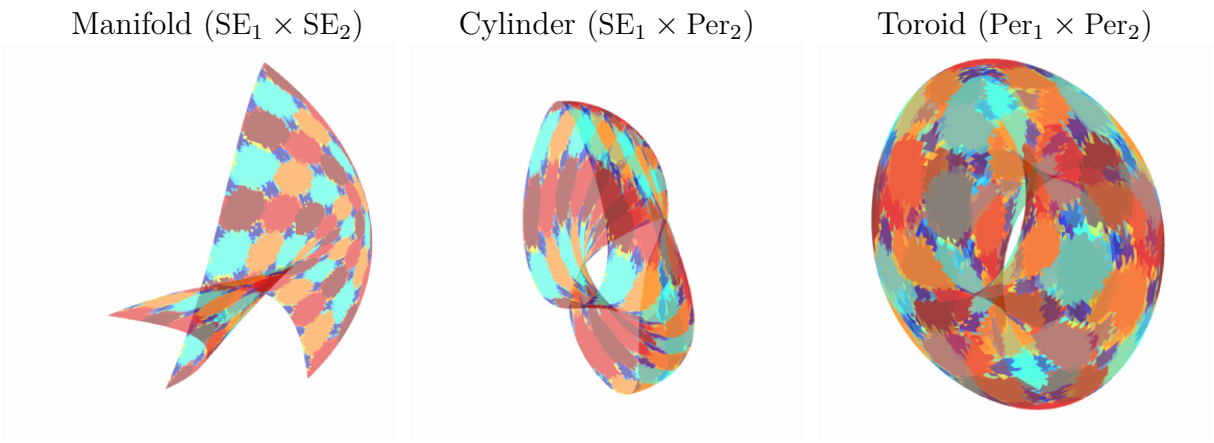


Fig. 2.9 Generating 2D manifolds with different topological structures. By enforcing that the functions mapping from $\mathbb{R}^2$ to $\mathbb{R}^3$ obey the appropriate symmetries, the surfaces created have the corresponding topologies, ignoring self-intersections.

2.8 Generating Topological Manifolds

In this section we give a geometric illustration of the symmetries encoded by different compositions 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 $f(\mathbf{x})$. To build a prior on 2-dimensional manifolds embedded in 3-dimensional space, we simply need a prior on mappings from $\mathbb{R}^2$ to $\mathbb{R}^3$, which we can construct using 3 independent functions $[f_1(\mathbf{x}), f_2(\mathbf{x}), f_3(\mathbf{x})]$ mapping from $\mathbb{R}^2$ to $\mathbb{R}$, specified by GP priors. Symmetries in $[f_1, f_2, f_3]$ will connect different parts of the manifolds, giving rise to non-trivial topologies on the sampled surfaces. **??** shows 2D meshes warped into 3D by functions drawn from GP priors with different kernels, giving rise to a variety of different topologies.

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

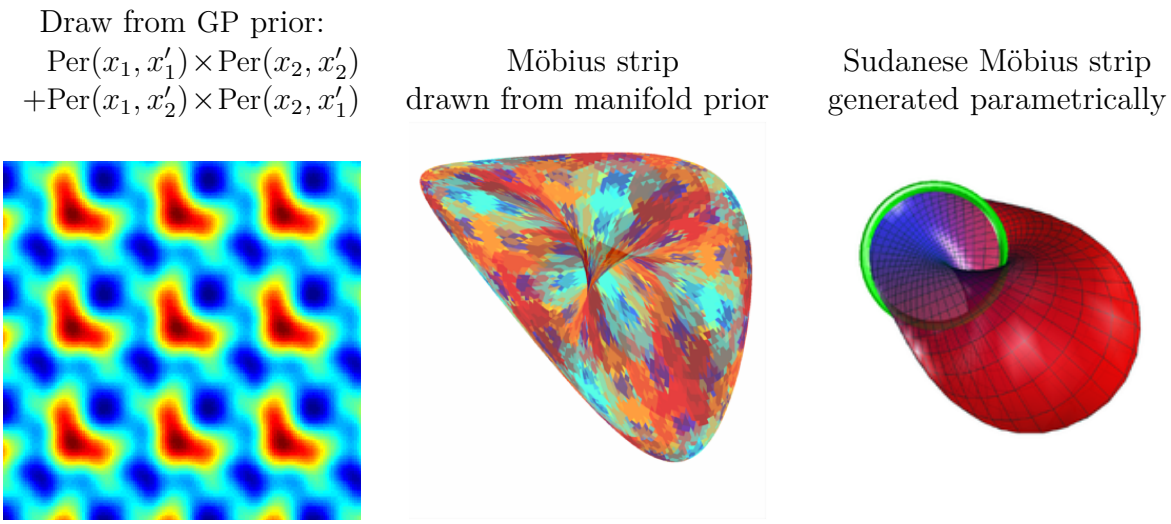


Fig. 2.10 Generating Möbius strips. *Left:* A function drawn from a GP prior obeying the same symmetries as a Möbius strip. *Center:* By enforcing that the functions mapping from $\mathbb{R}^2$ to $\mathbb{R}^3$ obey the appropriate symmetries, surfaces sampled from the prior have topology corresponding to a Möbius strip. Surfaces generated this way do not have the familiar shape of a flat surface connected to itself with a half-twist. Instead, they tend to look like *Sudanese* Möbius strips (**?**), whose edge has a circular shape. *Right:* A Sudanese projection of a Möbius strip. Image adapted from (**?**).

2.8.1 Möbius Strips

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

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

$$f(x, y) = f(x + \tau_x, y) \tag{2.50}$$

$$f(x, y) = f(y, x) \tag{2.51}$$

Moving along the diagonal $x = y$ of a function drawn from the corresponding GP prior is equivalent to moving along the edge of a notional Möbius strip which has had the function mapped on to its surface. **??**a shows an example of a function drawn from such a prior.

??(b) shows an example of a 2D mesh mapped to 3D by functions drawn from such a prior. This surface doesn't resemble the typical representation of a Möbius strip, but instead resembles an embedding known as the Sudanese Möbius strip (**?**), shown in **??**(c).

2.9 Kernels on Categorical Variables

Categorical variables are variables which can take values only from a discrete, unordered set, such as $\{\texttt{blue}, \texttt{green}, \texttt{red}\}$. A flexible way to construct a kernel over categorical variables is simply to represent that variable through a set of binary variables, using a one-of-k encoding. For example, if $\mathbf{x}$ can take one of 4 values, $x \in \{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}\}$, then a one-of-k encoding of x will correspond to 4 binary inputs, and one-of-k$(\texttt{C}) = [0, 0, 1, 0]$. Given a one-of-k encoding, we can place any multidimensional kernel on that space, such as the SE-ARD:

$$k_{\text{categorical}}(x, x') = \text{SE-ARD}(\text{one-of-k}(x), \text{one-of-k}(x')) \tag{2.52}$$

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

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

$$\text{SE-full}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{L}\mathbf{x}'\right) \tag{2.53}$$

where L is a symmetric matrix, individually parameterizing the covariance between each pair of function values of categories.

2.10 Building a Kernel in Practice

One difficulty in building GP models is choosing, or integrating over, the kernel parameters. If the kernel we construct has relatively few parameters, these parameters can be estimated by maximum marginal likelihood, using gradient-based optimizers. The kernel parameters estimated in the examples above were optimized using the GPML toolbox, available at gaussianprocess.org/gpml/code.

In the next chapter, we'll see how to perform such a search not just over the kernel paramaters, but also over the open-ended space of kernel expressions.

Source Code

Source code to produce all figures is available at github.com/duvenaud/pdh-thesis.

Chapter 3

Automatically Building Structured Regression Models

> "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 **??**, we saw that the choice of kernel determines the type of structure that can be learned by a GP model, and that a wide variety of models could be constructed through simply adding and multiplying a few base kernels together. However, we didn't answer the difficult question of how to tell which kernel to use for a given problem. Even for experts, choosing the kernel in nonparametric regression remains something of a black art.

In this chapter, we'll automate the process of building kernels for GP models. To do so, we'll need to define an open-ended space of kernels. We'll do this 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 main benefits. First, this procedure has very good predictive accuracy, since it tries out a large number of different regression models. Second, this procedure can discover interpretable structure in datasets. Because GP posteriors can be decomposed (as in **??**), we can also examine the resulting structures visually. In **??**, 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'll discuss the different parts we think are required to build an artificial intelligence that can do statistics.

1. An open-ended language of models Many learning algorithms consider all models in a class of fixed size. For example, graphical model learning algorithms (**??**) search over different connectivity graphs for a given set of nodes. While such methods can be powerful, human statisticians are capable of deriving novel model classes when required. An automatic search through an open-ended class of models can achieve some of this flexibility, growing the complexity of the model as needed, possibly combining existing structures in novel ways.

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

3. A model comparison procedure A search strategy requires an objective to optimize. In this work, we use approximate marginal likelihood to compare models, penalizing complexity using the Bayesian Information Criterion as a heuristic. More generally, an automatic statistician should be able to question the models it has constructed. **?** review the literature on model checking.

4. A model description procedure Part of the value of statistical models comes from helping humans to understand a dataset or a phenomenon. Furthermore, a clear description of the statistical structure found in a dataset helps a user to notice when the dataset has errors, the wrong question was asked, the model-building procedure failed to capture known structure, a relevant piece of data or constraint is missing, or when a novel statistical structure has been found.

In this chapter, we introduce a system containing all the above ingredients. We call this system the Automatic Bayesian Covariance Discovery (ABCD) system. The next four sections of this chapter describe the mechanisms we use to produce these four ingredients, for this particular example of an artificial intelligence which does statistics.

3.2 A Language of Regression Models

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

This language of models is made out of a set of base kernels which capture different properties of functions, and a set of composition rules which combine kernels to yield other valid kernels. In this chapter, we'll use such base kernels as white noise (WN), constant (C), linear (Lin), squared-exponential (SE), rational-quadratic (RQ), sigmoidal ($\boldsymbol{\sigma}$) and periodic (Per). We use a generalized form of Per due to **?** which has $\cos(x - x')$ as a special case. **??** shows the new kernels introduced in this chapter. For precise definitions of all kernels, see appendix **??**.

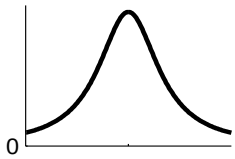
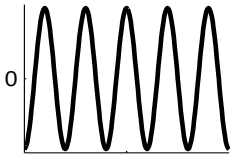
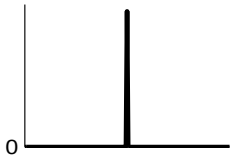
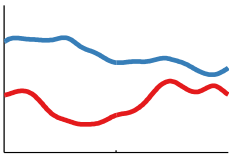
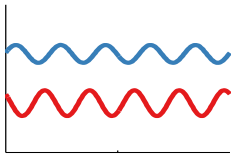
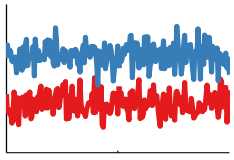
Kernel name:	Rational quadratic (RQ)	Cosine (cos)	White noise (Lin)
$k(x, x') =$	$\left(1 + \frac{(x-x')^2}{2\alpha\ell^2}\right)^{-\alpha}$	$\cos\left(2\pi\frac{(x-x')}{p}\right)$	$\delta(x - x')$
Plot of kernel:			
	$x - x'$	$x - x'$	x (with $x' = 1$)
Samples from prior:			
Type of structure:	multiscale variation	sine waves	uncorrelated noise

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

To specify an open-ended language of structured kernels, we'll consider the set of all

kernels that can be built by adding and multiplying these base kernels together:

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

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

Table **??** lists common regression models that can be expressed by this language.

Regression model	Kernel	Related work
Linear regression	C + Lin + WN	
Kernel ridge regression	SE + WN	
Linear Semi-parametric	Lin + SE + WN	(e.g. **?**)
Multiple kernel learning	$\sum$SE + WN	(e.g. **?**)
Trend, cyclical, irregular	$\sum$SE + $\sum$Per + WN	(**?**)
Fourier decomposition	C + $\sum$cos + WN	
Sparse spectrum GPs	$\sum$cos + WN	(**?**)
Spectral mixture	$\sum$SE $\times$ cos + WN	(**?**)
Changepoints	e.g. CP(SE, SE) + WN	(e.g. **?**)
Heteroscedasticity	e.g. SE + Lin$\times$WN	
Additive + Flexible	$\sum_d \mathrm{SE}_d + \prod_d \mathrm{SE}_d$	(**?**)

Table 3.2 Common regression models expressible by sums and products of base kernels. $\cos(\cdot, \cdot)$ is a special case of our reparametrised $\mathrm{Per}(\cdot, \cdot)$.

3.3 A Model Search Procedure

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

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

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

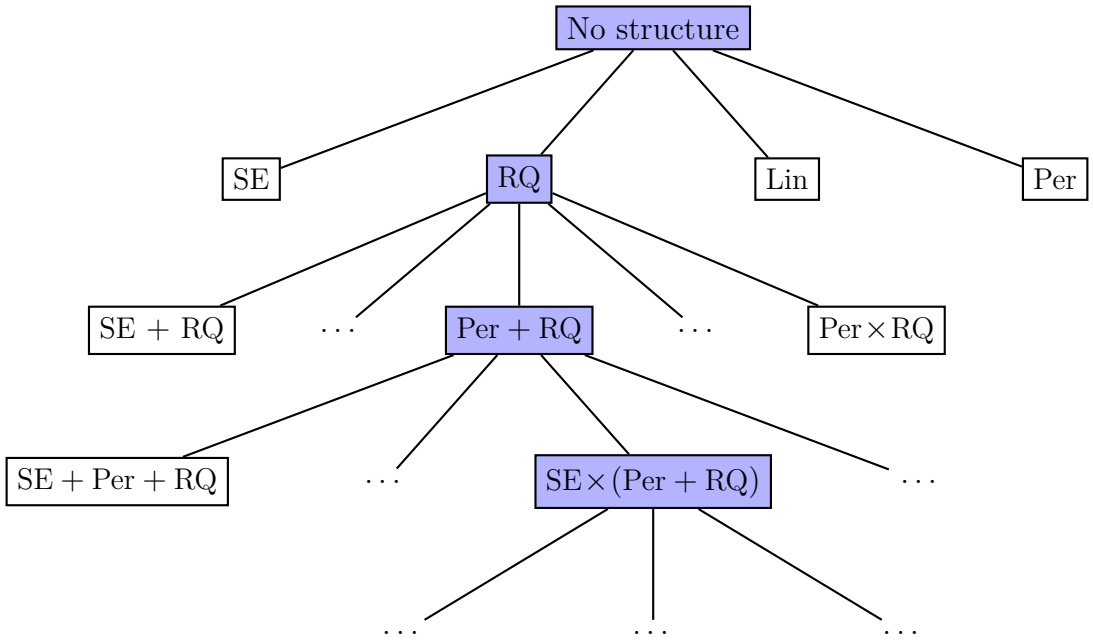


Fig. 3.1 An example of a search tree over kernel expressions. **??** shows the corresponding model increasing in sophistication as the kernel expression grows.

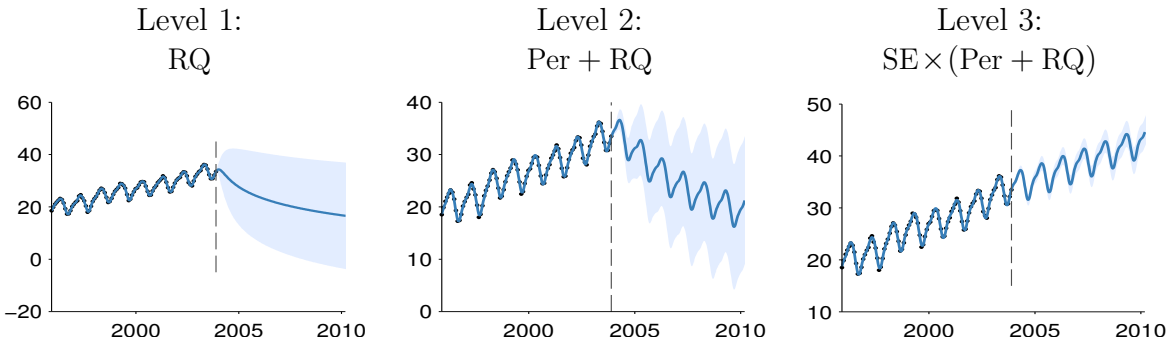


Fig. 3.2 Posterior mean and variance for different depths of kernel search on the Mauna Loa dataset. The dashed line marks the end of the dataset. *Left:* the function is only modeled as a locally smooth function, and the extrapolation is poor. *Middle:* a periodic component is added, and the extrapolation improves. *Right:* at depth 3, the kernel can capture most of the relevant structure, and is able to extrapolate reasonably.

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

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

- One can look for structure, such as periodicity, in the residuals of a model, and then extend the model to capture that structure. This corresponds to adding a new kernel to the existing structure.

- One can start with structure, such as linearity, which is assumed to hold globally, but find that it only holds locally. This corresponds to multiplying a kernel structure by a local kernel, such as SE.

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

Hyperparameter Initialization

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

3.4 A Model Comparison Procedure

Choosing a kernel requires a method for comparing models. We choose marginal likelihood as our criterion, since it balances the fit and complexity of a model (**?**). Conditioned on kernel parameters, the marginal likelihood of a GP can be computed analytically. Given a parametric form of a kernel, we can also choose its parameters using marginal likelihood. However, choosing kernel parameters by maximum likelihood (as opposed to integrating them out) raises the possibility of overfitting. In addition, if we compare two classes of kernels by the maximum likelihood found by optimizing over the kernel

parameters, then all else being equal, the kernel class having more free parameters will be chosen. This introduces a bias for more complex models.

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

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

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

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

3.5 A Model Description Procedure

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

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

This decomposition into additive components provides a method of visualizing the learned model, breaking down the different types of structure discovered in the data. In **??**, we'll extend this model visualization method to include automatically-generated English text explaining the meaning of each type of structure discovered.

3.6 Structure Discovery in Time Series

To investigate our method's ability to discover structure, we ran the kernel search on several time-series. In the following examples, the search was run to depth 10, using SE,

RQ, Lin, Per and WN as base kernels.

3.6.1 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.

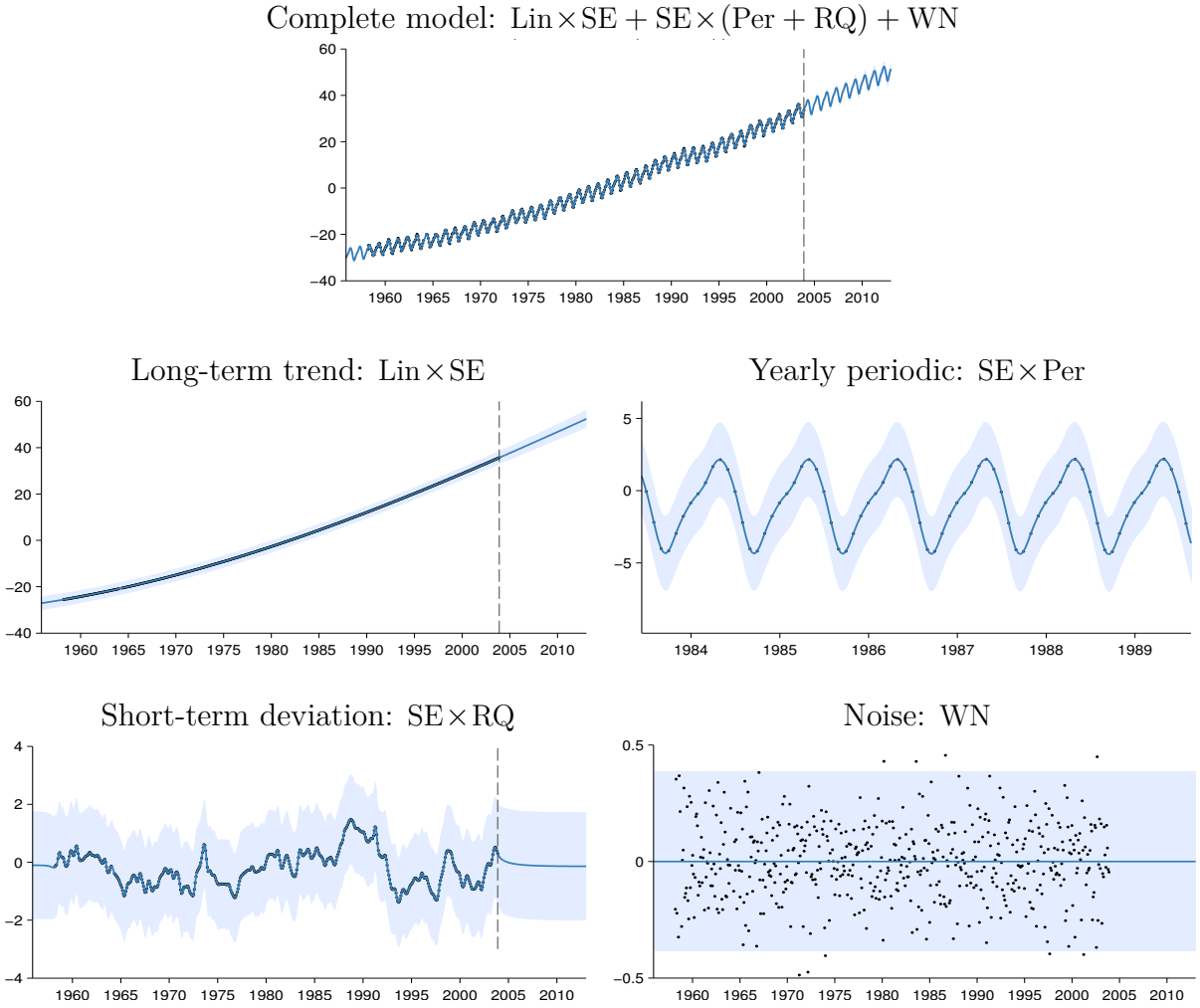


Fig. 3.3 *First row:* The full posterior on the Mauna Loa dataset, after a search of depth 10. *Subsequent rows:* The automatic decomposition of the time series. The decompositions is a sum of long-term, yearly periodic, medium-term components, and residual noise, respectively. The yearly periodic component has been rescaled for clarity.

?? shows the posterior mean and variance on this dataset as the search depth increases. While the data can be smoothly interpolated by a model with only a single

base kernel, the extrapolations improve dramatically as the increased search depth allows more structure to be included.

?? 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. These are the same components chosen in the kernel hand-constructed by **?**, Chapter 5. We also plot the residual noise, showing that there is little obvious structure left in the data.

3.6.2 Airline Passenger Counts

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

3.7 Related Work

Building Kernel Functions by Hand

?, Chapter 5 devote 4 pages to manually constructing a composite kernel to model the Mauna Loa dataset. Other examples of papers whose main contribution is to manually construct and fit a composite GP kernel are (**???**).

Nonparametric Regression in High Dimensions

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

Applying nonparametric methods in high-dimensional spaces can require imposing additional structure on the model. One such structure is additivity. Generalized additive models (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

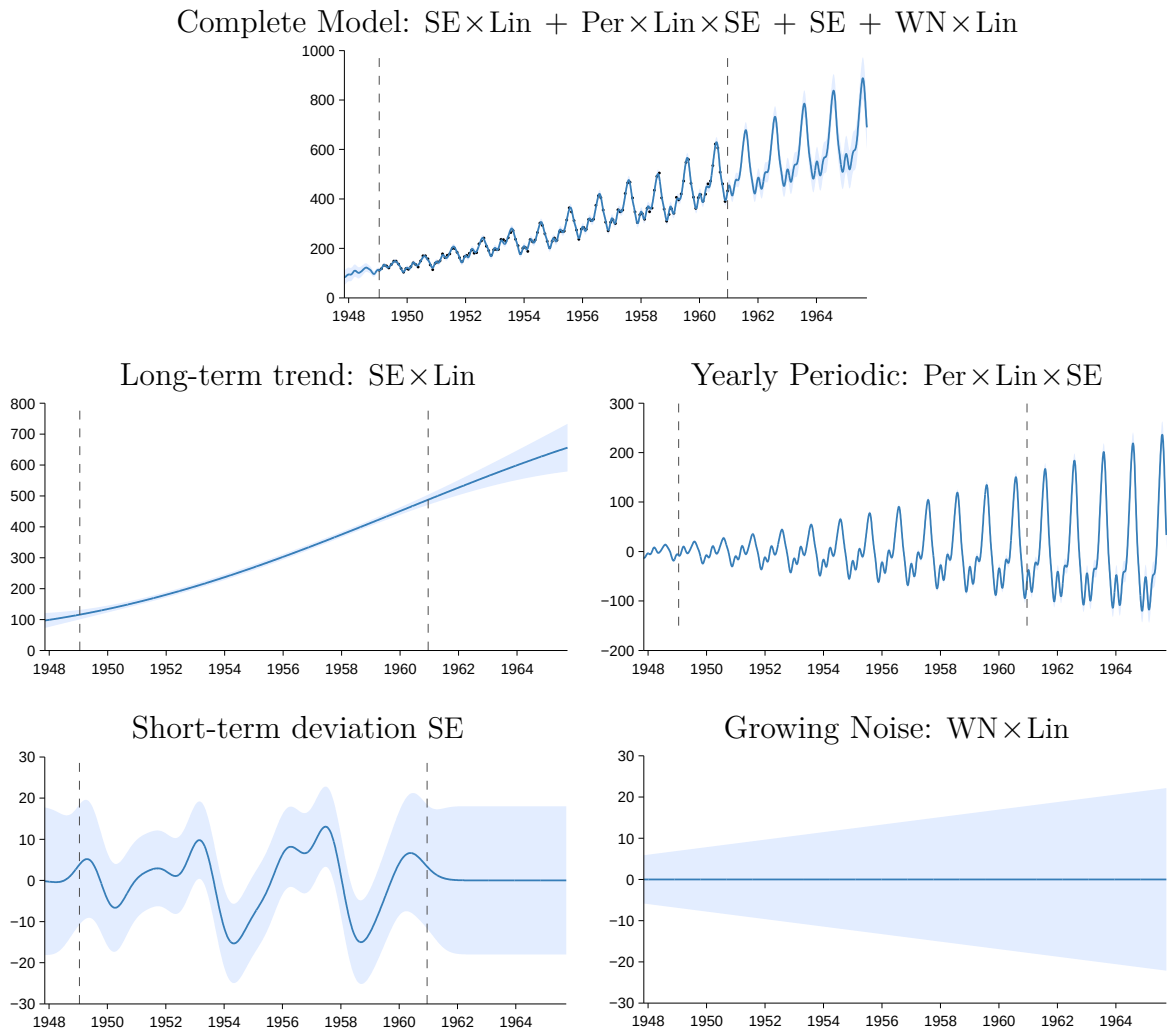


Fig. 3.4 *First row:* The airline dataset and posterior after a search of depth 10. *Subsequent rows:* Additive decomposition of posterior into long-term smooth trend, yearly variation, and short-term deviations. Due to the linear kernel, the marginal variance grows over time, making this a heteroskedastic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

grammar, we can capture such structure through sums of base kernels along different dimensions, although we have not yet tried incorporating a warping function $g(\cdot)$.

It is possible to add more flexibility to additive models by considering higher-order interactions between different dimensions. In **??**, we'll consider GP models whose kernel implicitly sums over all possible products of one-dimensional base kernels. **?** constructs a special case of this model class, summing an SE kernel along each dimension, with a single SE-ARD kernel (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 combinations of splines along each dimension, all pairs of dimensions, and possibly higher-order combinations. Because the number of terms to consider grows exponentially in the order, in practice, only terms of first and second order are usually considered.

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

Kernel Learning

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

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

Sparse spectrum GPs (**?**) 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 mixture of Gaussians – corresponding to kernels of the form $\sum \mathrm{SE} \times \cos$. Both groups 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 **??**).

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 **??**, 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 which don't have a simple parametric form but still have high-level structure. An examination of the structure discovered by the automatic equation-learning software Eureqa (**?**) on the airline and Mauna Loa datasets can be found in (**?**).

Structure Discovery through Grammars

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

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

? performed a greedy search over a compositional model class for unsupervised learning, using a grammar of matrix decomposition models, and a greedy search procedure based on held-out likelihood. This model class contains many existing unsupervised models as special cases, and was able to discover such structure automatically from data. Our framework takes a similar approach, but in a supervised setting.

Similarly, **?** showed to automatically perform inference in arbitrary compositions of discrete sequence models. More generally, **?** and **?** constructed grammars over programs, and automatically searched the resulting spaces.

3.8 Experiments

3.8.1 Interpretability versus Accuracy

BIC trades off model fit and complexity by penalizing the number of parameters in a kernel expression. This can result in ABCD favoring kernel expressions with nested products of sums, producing descriptions involving many additive components when expressions are expanded. While these models typically have good predictive performance, the large number of components can make them less interpretable. We experimented with not allowing parentheses during the search, discouraging nested expressions. We call this procedure ABCD-interpretability, in contrast to the unrestricted version of the search, ABCD-accuracy.

3.8.2 Predictive Accuracy on Time Series

We evaluate the performance of the algorithms listed below on 13 real time-series from various domains from the time series data library (**?**).

Algorithms

We compare ABCD to equation learning using Eureqa (**?**), as well as six other regression algorithms: linear regression, GP regression with a single SE kernel (squared exponential), a Bayesian variant of multiple kernel learning (MKL) (e.g. **?**), changepoint modeling (e.g. **???**), spectral mixture kernels (**?**) (spectral kernels), and trend-cyclical-irregular models (e.g. **?**).

We set Eureqa's search objective to the default mean-absolute-error. All algorithms besides Eureqa can be expressed as restrictions of our modeling language (see **??**), so we perform inference using the same search and objective function, 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 standard marginal likelihood optimization.

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

Extrapolation Experiments

To test extrapolation, we trained all algorithms on the first 90% of the data, predicted the remaining 10% and then computed the root mean squared error (RMSE). The RMSEs are then standardised by dividing by the smallest RMSE for each data set, so the best performance on each data set will have a value of 1.

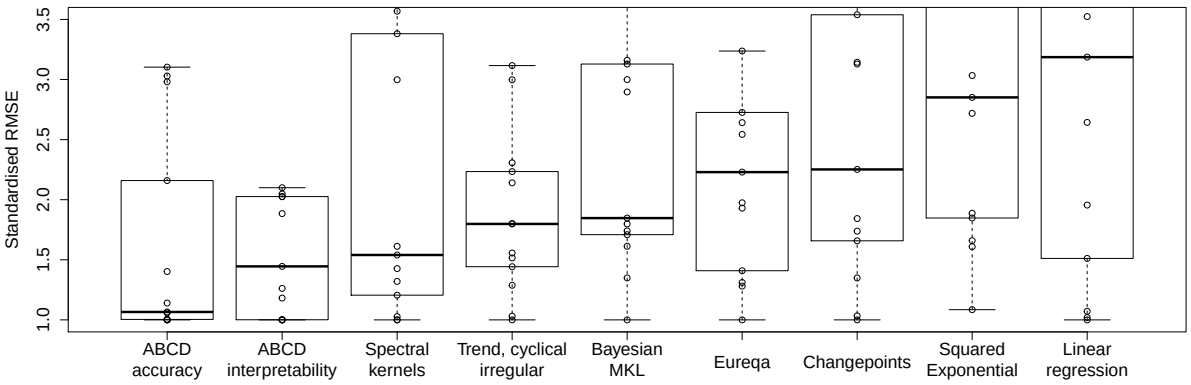


Fig. 3.5 Box plot (showing median and quartiles) of standardised extrapolation RMSE (best performance = 1) on 13 time-series. The methods are ordered by median.

?? shows the standardised RMSEs across algorithms. ABCD-accuracy outperforms ABCD-interpretability. Both algorithms have lower quartiles than all other methods.

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 normalized RMSEs of 11, 493, 22 and 29 respectively.

3.8.3 High-dimensional Prediction

ABCD can also be applied to multidimensional regression problems without modification. An experimental comparison with other methods can be found in **??**, where it outperforms a wide variety of multidimensional regression methods.

3.8.4 Structure Recovery on Synthetic Data

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

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

Table 3.3 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. Each kernel implicitly has a WN kernel included.

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	$(\mathrm{SE}_1 + \mathrm{SE}_2) \times \ldots$ $(\mathrm{SE}_3 \times \mathrm{Lin}_3 \times \mathrm{Lin}_1 + \mathrm{SE}_4)$	$(\mathrm{SE}_1 + \mathrm{SE}_2) \times \ldots$ $\mathrm{SE}_3 \times \mathrm{SE}_4$	-

?? shows the results. For the highest SNR, the method finds all relevant structure except in one case. The reported additional linear structure in the last row is 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 over-fitting.

Source Code

Source code to perform all experiments is available at `github.com/jamesrobertlloyd/gpss-research/`. All GP parameter optimisation was performed by automated calls to the GPML toolbox, available at `www.gaussianprocess.org/gpml/code/`.

3.9 Discussion

Towards the goal of automating statistical modeling, we developed a system which constructs an appropriate model from an open-ended language, and automatically generates plots decomposing the different types of structure present in the model.

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

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 hope that this procedure has the potential to make powerful statistical model-building techniques accessible to non-experts.

In the next chapter, we'll see how the model components found by this procedure can be automatically described in terms of English-language text.

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 can generate 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.

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 $\mathrm{SE} \times \mathrm{Per} \times \mathrm{Lin} \times \boldsymbol{\sigma}$ could be described as an

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

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:

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

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

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

This is converted to a sum of products:

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

which is simplified to

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

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 Summarizing 400 Years of Solar Activity

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

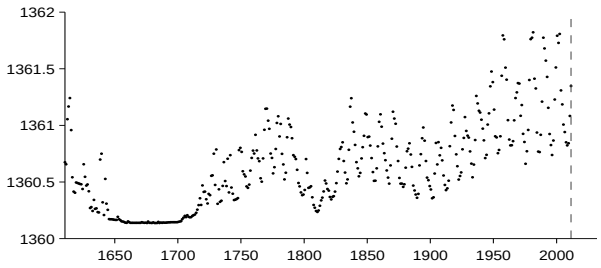


Fig. 4.1 Solar irradiance data.

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.

Figure **??** 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

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. 4.2 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.

Maunder minimum (second component) and 11-year solar cycle (fourth component). These components are visualized in figures **??** and **??**, respectively. The third component corresponds to long-term trends, as visualized in figure **??**.

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.

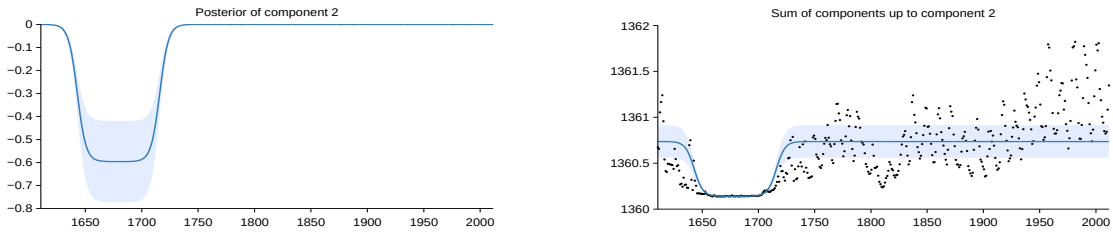


Figure 4: Pointwise posterior of component 2 (left) and the posterior of the cumulative sum of components with data (right)

Fig. 4.3 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.

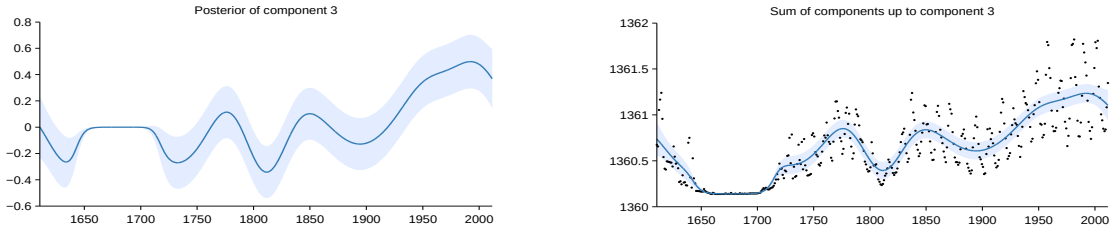


Figure 6: Pointwise posterior of component 3 (left) and the posterior of the cumulative sum of components with data (right)

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

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.

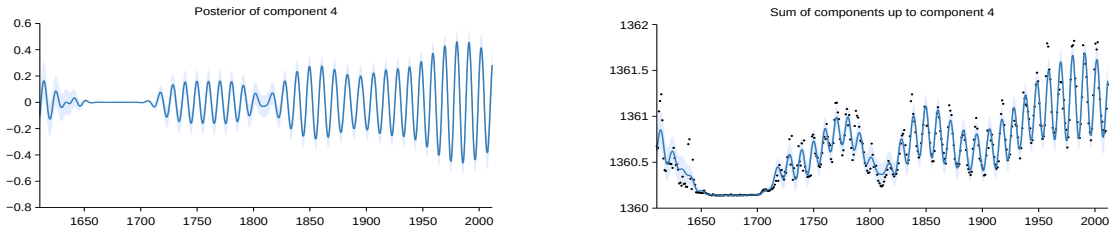


Figure 8: Pointwise posterior of component 4 (left) and the posterior of the cumulative sum of components with data (right)

Fig. 4.5 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 (**?**).

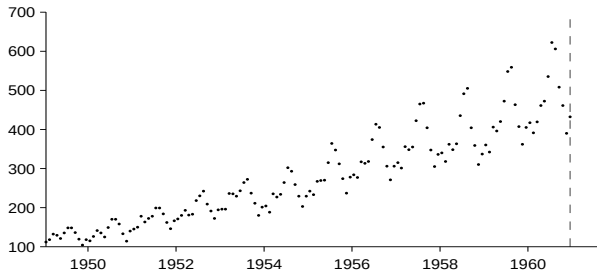


Fig. 4.6 International airline passenger monthly volume (e.g. **?**).

4.3 Describing Heteroscedasticity in Air Traffic Data

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

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

The second component (figure **??**) 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 **??**).

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.

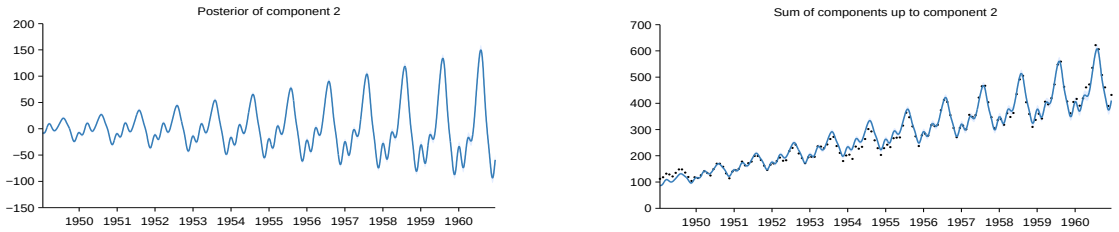


Figure 4: Pointwise posterior of component 2 (left) and the posterior of the cumulative sum of components with data (right)

Fig. 4.8 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.

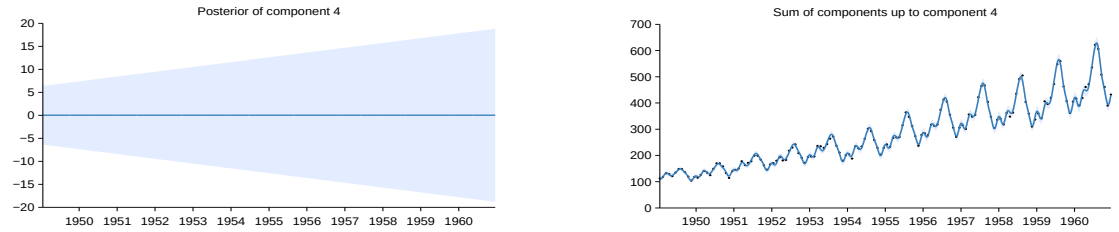


Figure 8: Pointwise posterior of component 4 (left) and the posterior of the cumulative sum of components with data (right)

Fig. 4.9 Modeling heteroscedasticity in the airline dataset.

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 (**?**).

Source Code

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

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 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:

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

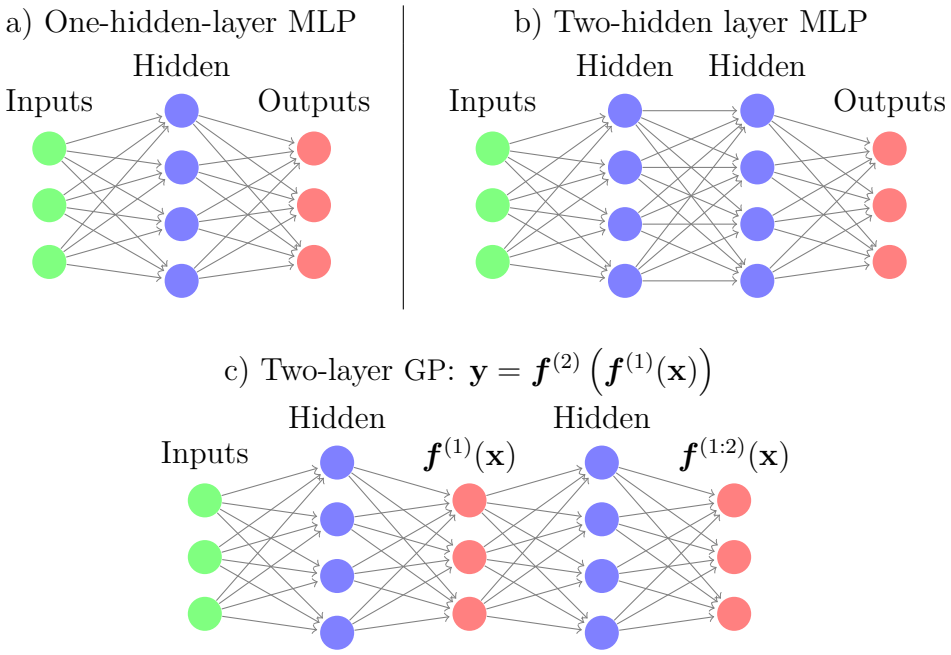


Fig. 5.1 a): GPs can be understood as a one-hidden-layer MLP with infinitely many hidden units. There are two interpretations of deep GPs as neural networks: b): As a neural network with a finite number of hidden units, each with a different non-parametric activation function. c) Alternatively, we can 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 **??**b. For example, if we extend the model given by (**??**) 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 **??**, 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 **??**c. 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-

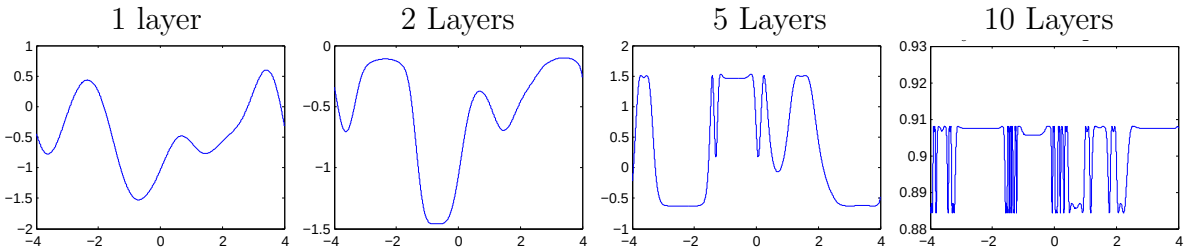


Fig. 5.2 One-dimensional draws from a deep gp prior. After a few layers, the functions begin to be either nearly flat, or highly varying, everywhere. This is a consequence of the distribution on derivatives becoming heavy-tailed.

mation bottlenecks. We can simply substitute (**??**) into (**??**) 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 **??** 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 parameter σ^2 controls the variance of functions drawn from the prior, and the parameter w controls the smoothness. The derivative of a GP with a squared-exp kernel is point-wise distributed as $\mathcal{N}(0, {}^{\sigma^2}/_{w^2})$. Intuitively, a 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:

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

Even if the expected magnitude of the derivative remains constant, the variance of the log-normal distribution grows without bound as the depth increases. Because the log-normal distribution is heavy-tailed and its domain is bounded below by zero, the derivative will become very small almost everywhere, with rare but very large jumps.

Figure **??** 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 (**??**), it follows that each row of J is independent. Lemma **??** 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.*

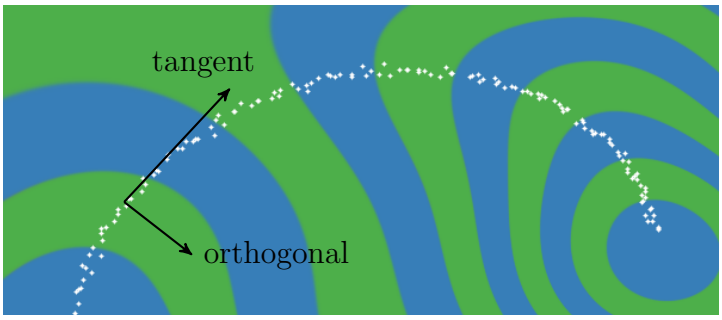


Fig. 5.3 Representing a 1-D data manifold. Colors correspond to the computed representation as a function of the input space. The representation (blue & green) varies in directions tangent to the data manifold (white), preserving information for later layers. The representation is also invariant to directions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

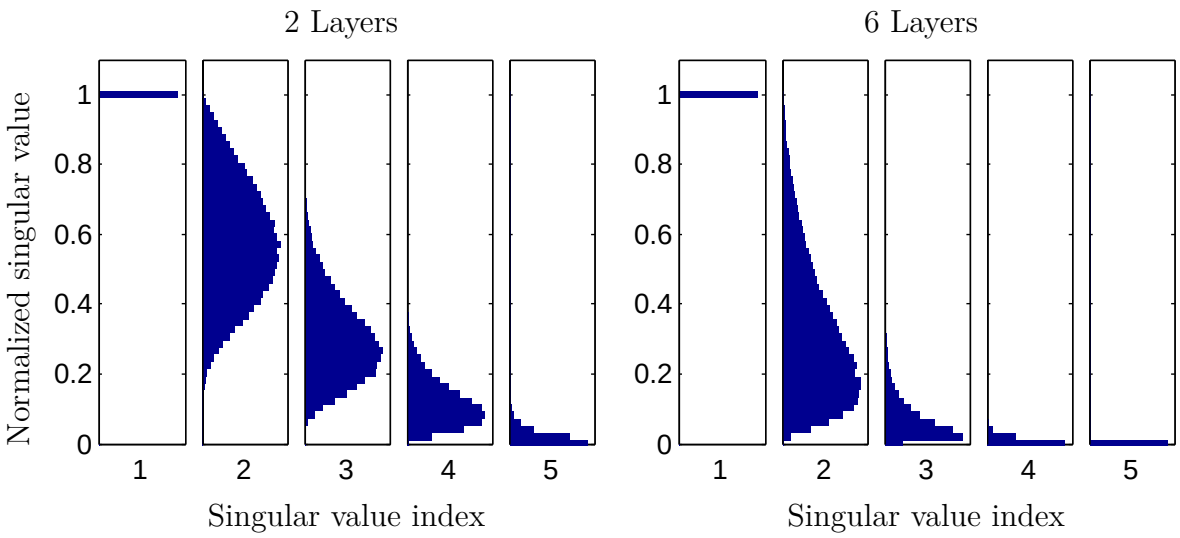


Fig. 5.4 The distribution of singular values relative to the largest, of the Jacobian of a function drawn from 5-dimensional deep GP prior, at 25 and 50 layers deep. As the net gets deeper, the largest singular value becomes much larger than the others. This implies that with high probability, there 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 **??** 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 **??** 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 unsuitable to model manifolds whose underlying dimensionality is greater than one.

To visualize this pathology in another way, figure **??** 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,

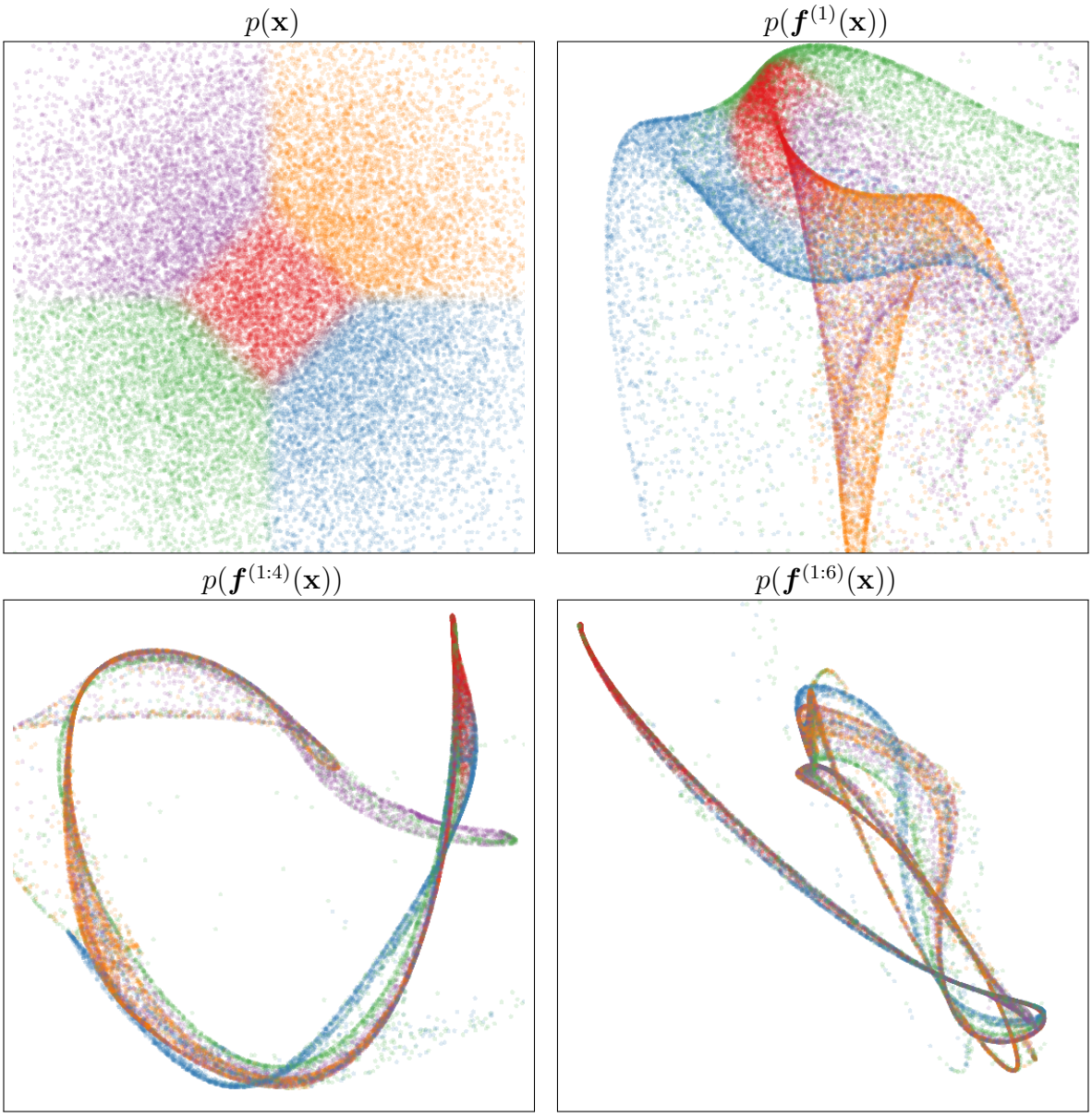


Fig. 5.5 Visualization of draws from a deep GP. A 2-dimensional Gaussian distribution (top left) is warped by successive functions drawn from a GP prior. As the number of layers increases, the density concentrates along one-dimensional filaments.

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

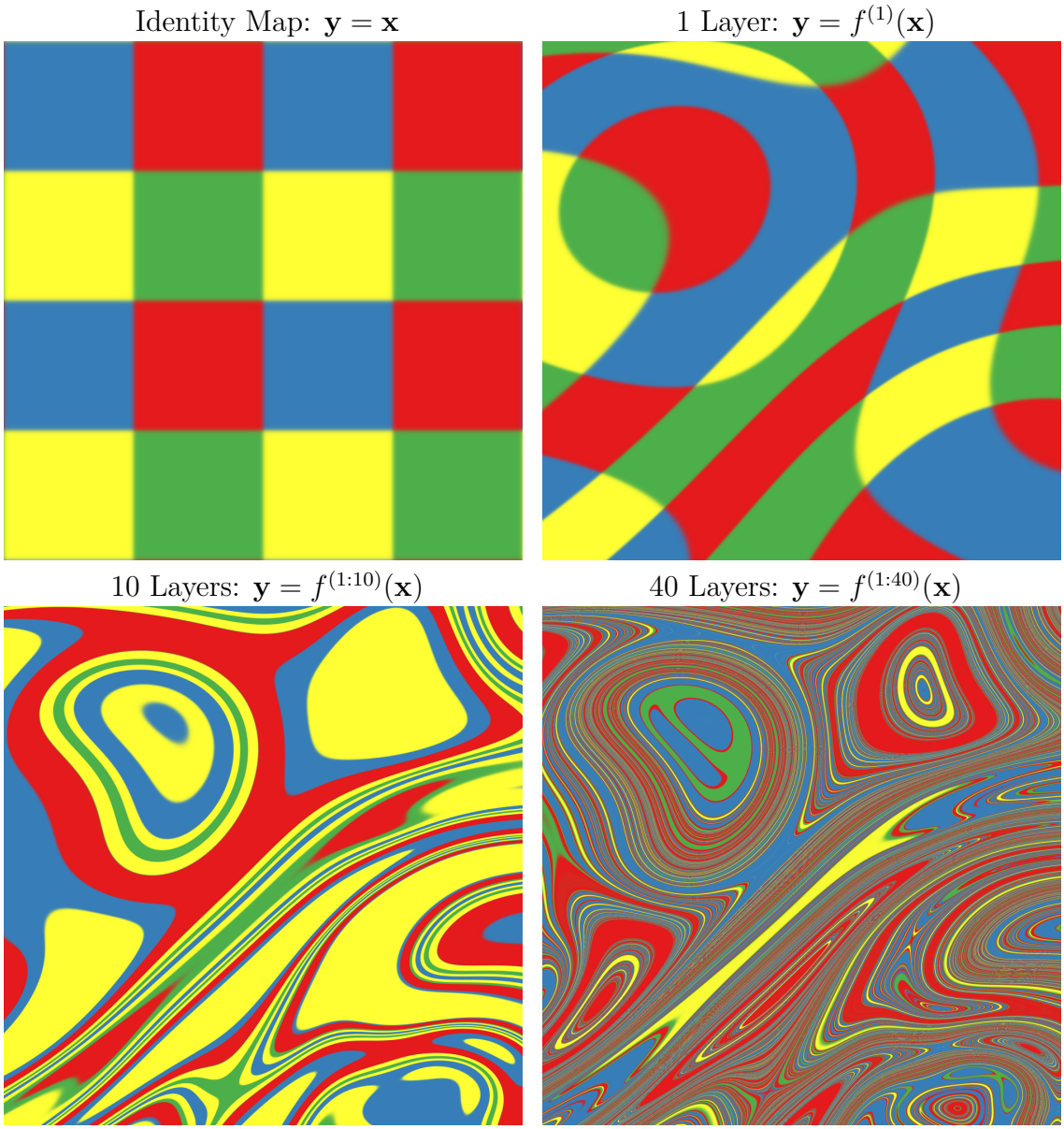


Fig. 5.6 Feature mapping of a deep GP. Colors correspond to the location $\mathbf{y} = \boldsymbol{f}(\mathbf{x})$ that each point is mapped to after being warped by a deep GP. The number of directions in which the color changes rapidly corresponds to the number of large singular values in the Jacobian.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

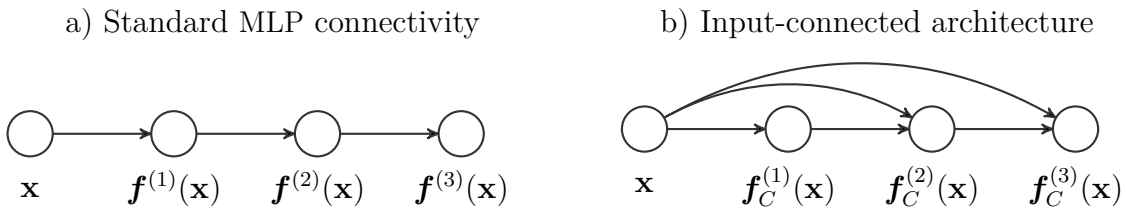


Fig. 5.7 Two different architectures for deep neural networks. The standard architecture connects each layer's outputs to the next layer's inputs. The input-connected architecture also connects the original input $\mathbf{x}$ to each layer.

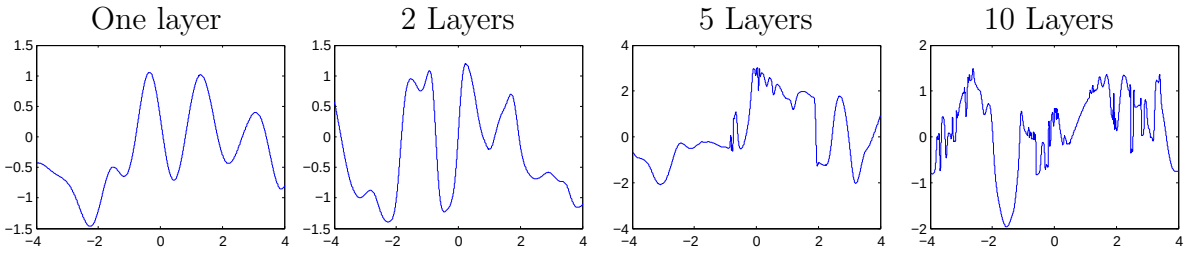


Fig. 5.8 Draws from a 1D deep GP prior with each layer connected to the input. Even after many layers, the functions remain smooth in some regions, while varying rapidly in other regions. Compare to standard-connectivity deep GP draws shown in figure **??**.

5.5 Fixing the Pathology

Following a suggestion from **?**, we can fix the pathologies exhibited in figures **??** and **??** by simply making each layer depend not only on the output of the previous layer, but also on the original input $\mathbf{x}$. We refer to these models as *input-connected* networks, and denote deep functions having this architecture with the subscript C, as in $f_C(\mathbf{x})$. Figure **??** 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

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

Draws from the resulting prior are shown in figures **??**, **??** and **??**. The Jacobian of an input-connected deep function is defined by the recurrence

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

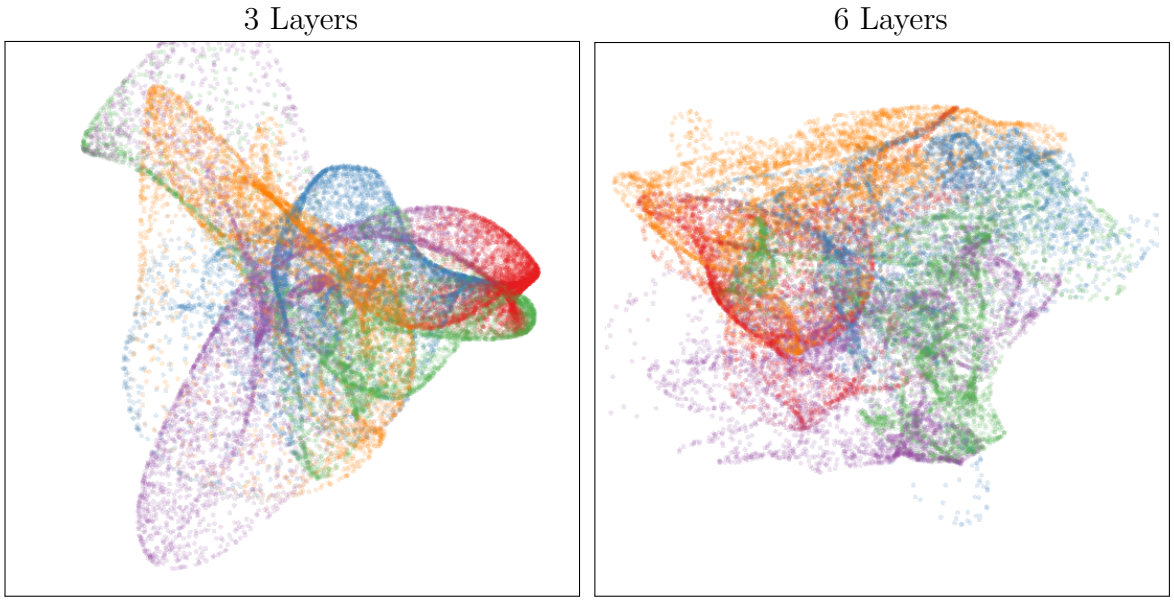


Fig. 5.9 Densities defined by a draw from a deep GP, with each layer connected to the input $\mathbf{x}$. As depth increases, the density becomes more complex without concentrating along filaments.

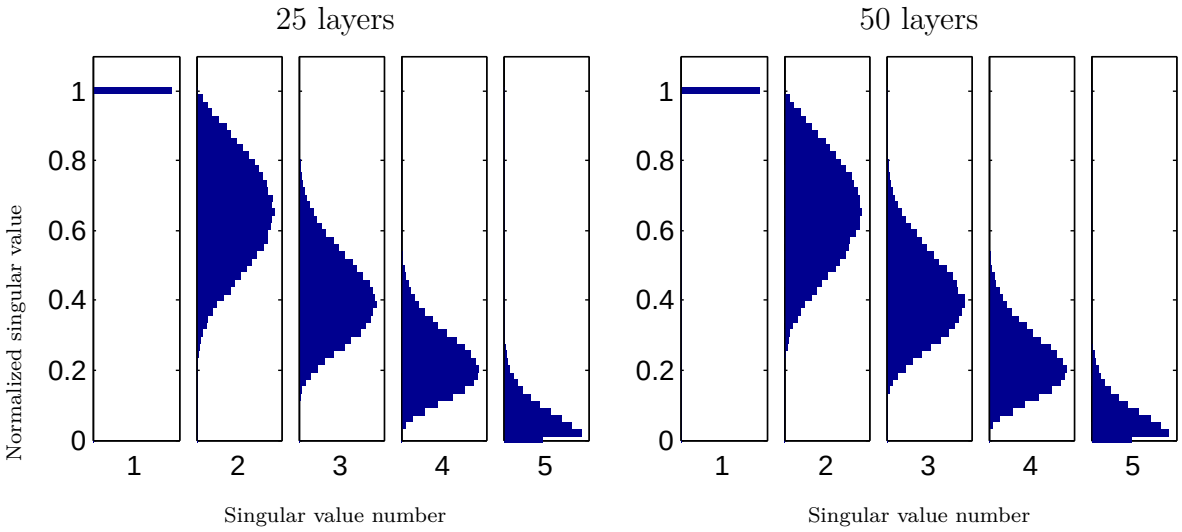


Fig. 5.10 The distribution of singular values drawn from 5-dimensional input-connected deep GP priors, 25 and 50 layers deep. The singular values remain roughly the same scale as one another.

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

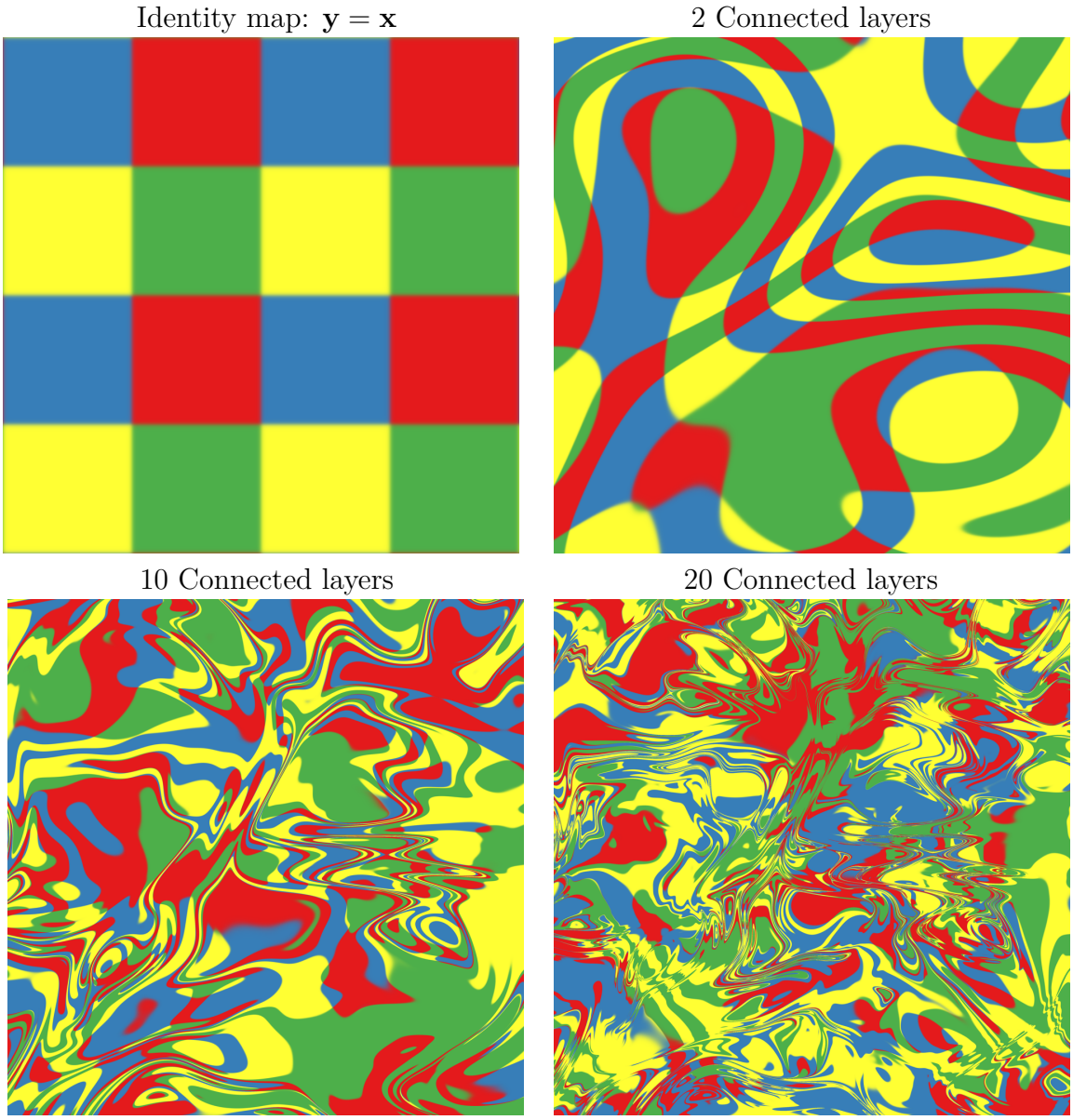


Fig. 5.11 Feature mapping of a deep GP with each layer connected to the input $\mathbf{x}$. Just as the densities in figure ?? remained locally two-dimensional even after many transformations, in this mapping there are often two directions that one can move locally in $\mathbf{x}$ to in order to change 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 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

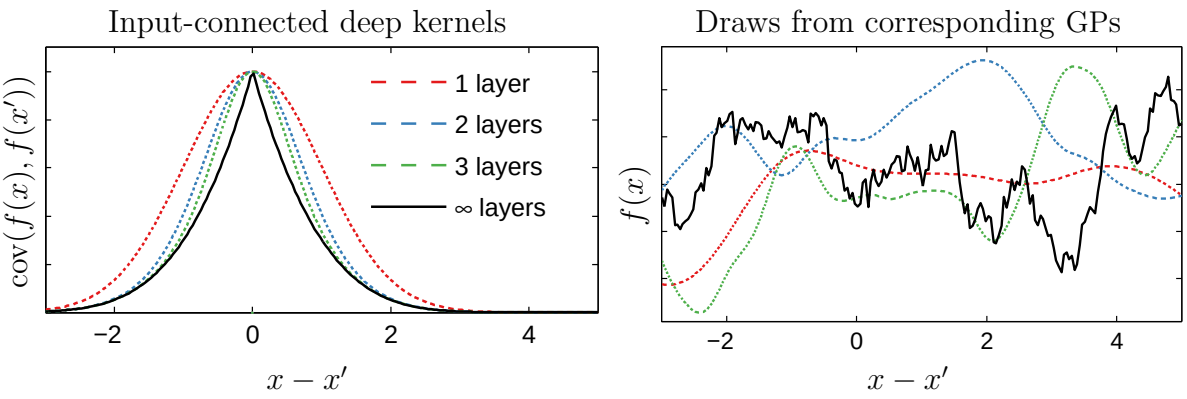


Fig. 5.12 *Left:* Input-connected deep kernels. By connecting the inputs $\mathbf{x}$ to each layer, the kernel can still depend on its input even after arbitrarily many layers of computation. *Right:* Draws from GP with deep input-connected kernels.

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

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

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 invariants 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 adds and multiplies functions drawn from GPs, in contrast to deep GPs, which compose functions drawn from GPs. This prior on functions might be amenable to a similar analysis to that of section **??**.

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 gradient-based training difficult.

Deep kernels **?** construct deep kernels in a time-series setting, constructing kernels corresponding to infinite-width recurrent neural networks.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 architecture 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.

Source Code

Source code to produce all figures is available at `github.com/duvenaud/deep-limits/`.

5.8 Conclusions

In this chapter, we attempted to gain insight into the properties of deep neural networks by characterizing the sorts of functions likely to be obtained under different choices of priors on compositions of functions.

First, we identified deep Gaussian processes as an easy-to-analyze model corresponding to multi-layer preceptrons with nonparametric activation functions. We then showed that representations based on repeated composition of independent functions exhibit a pathology where the representations becomes invariant to all 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 **??**, we saw how additive structure in a GP prior enabled long-range extrapolation in multivariate problems. In general, models of the form

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

are widely used in machine learning and statistics, because they are relatively easy to fit and interpret. Examples include logistic regression, linear regression, Generalized Linear Models (**?**) and Generalized Additive Models (GAM) (**?**).

At the other end of the spectrum are models which allow the response to depend on all input variables simultaneously, having the form

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

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

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

$$g(y) = f(x_1, x_2) + f(x_2, x_3) + f(x_1, x_3) \tag{6.3}$$

In this chapter, we'll consider a GP model consisting of a sum of functions of all possible combinations of input variables. We'll call this model class *Additive Gaussian processes*. This model can be expressed by specifying a kernel which is a sum of all possible products of one-dimensional kernels.

There are 2^D combinations of D kernels, so a naïve computation of such a kernel would be intractable. Furthermore, if each term has different kernel parameters, fitting or integrating over so many parameters would pose severe difficulty. To get around this problem, we introduce an expressive but tractable parameterization of the kernel function which also allows efficient evaluation of all interaction terms. Empirically, this kernel results in good predictive power in regression tasks, as well as increased interpretability.

In **??**, we showed how to learn the structure of a kernel by building it up piece-by-piece. This chapter presents an alternative approach, where we start with many different types of structure in the kernel from the beginning, and adjust the kernel parameters to discard whichever types *aren't* present in the current dataset. The advantage of this approach is that we don't need to run an expensive, mixed discrete-continuous optimization problem in order to build a structured model. Implementation is also much simpler.

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:

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

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

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

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

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:

$$k_{add_D}(\mathbf{x}, \mathbf{x}') = \sigma_D^2 \prod_{d=1}^{D} k_d(x_d, x'_d) \tag{6.8}$$

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:

$$k_{add_D}(\mathbf{x}, \mathbf{x}') = \sigma_D^2 \prod_{d=1}^{D} k_d(x_d, x'_d) = \sigma_D^2 \prod_{d=1}^{D} \exp\Big(-\frac{(x_d - x'_d)^2}{2l_d^2} \Big) = \sigma_D^2 \exp\Big(-\sum_{d=1}^{D} \frac{(x_d - x'_d)^2}{2l_d^2} \Big) \tag{6.9}$$

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 **??** 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 sum of one-dimensional functions; 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 discover this fact and exploit it. Discovering such structure allows long-range extrapolation, as we saw in **??**. If low-dimensional additive structure is not present, the kernel parameters can specify a suitably flexible model, with interactions between as many

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

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

$$k_{\text{add}_0}(\mathbf{x}, \mathbf{x}') = e_0(k_1, k_2, k_3, k_4) = 1 \tag{6.10}$$

$$k_{\text{add}_1}(\mathbf{x}, \mathbf{x}') = e_1(k_1, k_2, k_3, k_4) = k_1 + k_2 + k_3 + k_4 \tag{6.11}$$

$$k_{\text{add}_2}(\mathbf{x}, \mathbf{x}') = e_2(k_1, k_2, k_3, k_4) = k_1k_2 + k_1k_3 + k_1k_4 + k_2k_3 + k_2k_4 + k_3k_4 \tag{6.12}$$

$$k_{\text{add}_3}(\mathbf{x}, \mathbf{x}') = e_3(k_1, k_2, k_3, k_4) = k_1k_2k_3 + k_1k_2k_4 + k_1k_3k_4 + k_2k_3k_4 \tag{6.13}$$

$$k_{\text{add}_4}(\mathbf{x}, \mathbf{x}') = e_4(k_1, k_2, k_3, k_4) = k_1k_2k_3k_4 \tag{6.14}$$

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

nomials:

$$k_{\text{add}_n}(\mathbf{x}, \mathbf{x}') = e_n(k_1, \ldots, k_D) = \frac{1}{n} \sum_{a=1}^{n} (-1)^{(a-1)} e_{n-a}(k_1, \ldots, k_D) \sum_{i=1}^{D} k_i^a \tag{6.15}$$

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:

$$\frac{\partial k_{add_n}}{\partial k_j} = e_{n-1}(k_1, \ldots, k_{j-1}, k_{j+1}, \ldots k_D) \tag{6.16}$$

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

6.1.3 Computational Cost

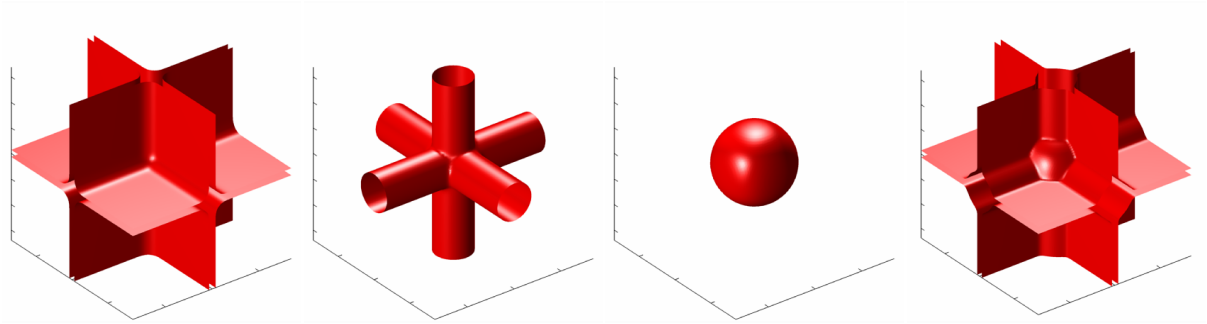
The computational cost of evaluating the Gram matrix of a product kernel (such as the SE kernel) is $\mathcal{O}(N^2 D)$, while the cost of evaluating the Gram matrix of the additive kernel is $\mathcal{O}(N^2 DR)$, where R is the maximum degree of interaction allowed (up to D). In higher dimensions, 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:

$$k(\mathbf{x}, \mathbf{x}') = g\left(\sum_{d=1}^{D} \left(\frac{x_d - x'_d}{l_d} \right)^2 \right) \tag{6.17}$$

? 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. **??** provides a geometric comparison between squared-exponential kernels and additive kernels in 3 dimensions.

In **??**, **??** gives an example of how additive kernels extrapolate differently than local kernels.

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.

Recall that GPs can be derived as an infinitely-wide one-layer neural network, with fixed activation functions $\mathbf{h}(\mathbf{x})$, and independent random weights $\boldsymbol{\alpha}$ with finite variance $\sigma_{\boldsymbol{\alpha}}^2$:

$$f(\mathbf{x}) = \frac{1}{K}\boldsymbol{\alpha}^\mathsf{T}\mathbf{h}(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K}\alpha_i h_i(\mathbf{x}) \tag{6.18}$$

$$\implies f(\mathbf{x}) \overset{K\to\infty}{\sim} \mathcal{GP}\Big(\mathbb{E}\left[\boldsymbol{\alpha}^\mathsf{T}\mathbf{h}(\mathbf{x})\right], \sigma_{\boldsymbol{\alpha}}^2\mathbf{h}(\mathbf{x})^\mathsf{T}\mathbf{h}(\mathbf{x}')\Big) \tag{6.19}$$

Mercer's theorem implies that we can write any GP prior equivalently in this way. Now that we've expressed a GP as a neural network, we can examine the prior we get from performing dropout in this network.

6.3.1 Dropout on Hidden Units

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

$$r_i \overset{\text{iid}}{\sim} \text{Ber}(p) \qquad \mathbb{V}[r_i\alpha_i] = p\sigma_\alpha^2 \,. \tag{6.20}$$

Because **??** is a result of the central limit theorem, it does not depend on the form of the distribution on $\boldsymbol{\alpha}$, only its mean and variance. Thus, dropping out features of an infinitely-wide MLP does not change the model at all, except to rescale the output variance, since no individual feature can have more than infinitesimal contribution to the network output. Indeed, multiplying all weights by $p^{-1/2}$, the initial variance is restored:

$$\mathbb{V}\left[\frac{1}{p^{\frac{1}{2}}}r_i\alpha_i\right] = \frac{p}{p}\sigma_\alpha^2 = \sigma_\alpha^2 \,. \tag{6.21}$$

In which case dropout on the hidden units has no effect at all.

6.3.2 Dropout on 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:

$$p\left(f(\mathbf{x})\right) = \frac{1}{2^D} \sum_{\mathbf{r} \in \{0,1\}^D} \mathrm{GP}\left(f(\mathbf{x}) \,|\, 0, \prod_{d=1}^{D} k_d(x_d, x'_d)^{r_d}\right) \tag{6.22}$$

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 (**??**) has the same covariance as

$$f(\mathbf{x}) \sim \mathrm{GP}\left(0, \frac{1}{2^{-2D}} \sum_{\mathbf{r} \in \{0,1\}^D} \prod_{d=1}^{D} k_d(x_d, x'_d)^{r_d}\right) \tag{6.23}$$

Therefore, dropout on the inputs to a GP can be approximated by the all-subsets additive kernel. This suggests an interpretation of additive kernels as an approximation to a mixture of models, each of which depends on only a subset of the input variables.

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 was 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 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:

$$k_a(\mathbf{x}, \mathbf{x}') = v^2 \prod_{d=1}^{D} (1 + \alpha k_d(x_d, x'_d)) \tag{6.24}$$

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

Figure **??** contrasts the HKL model class with the additive GP model. Neither method is strictly more flexible 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

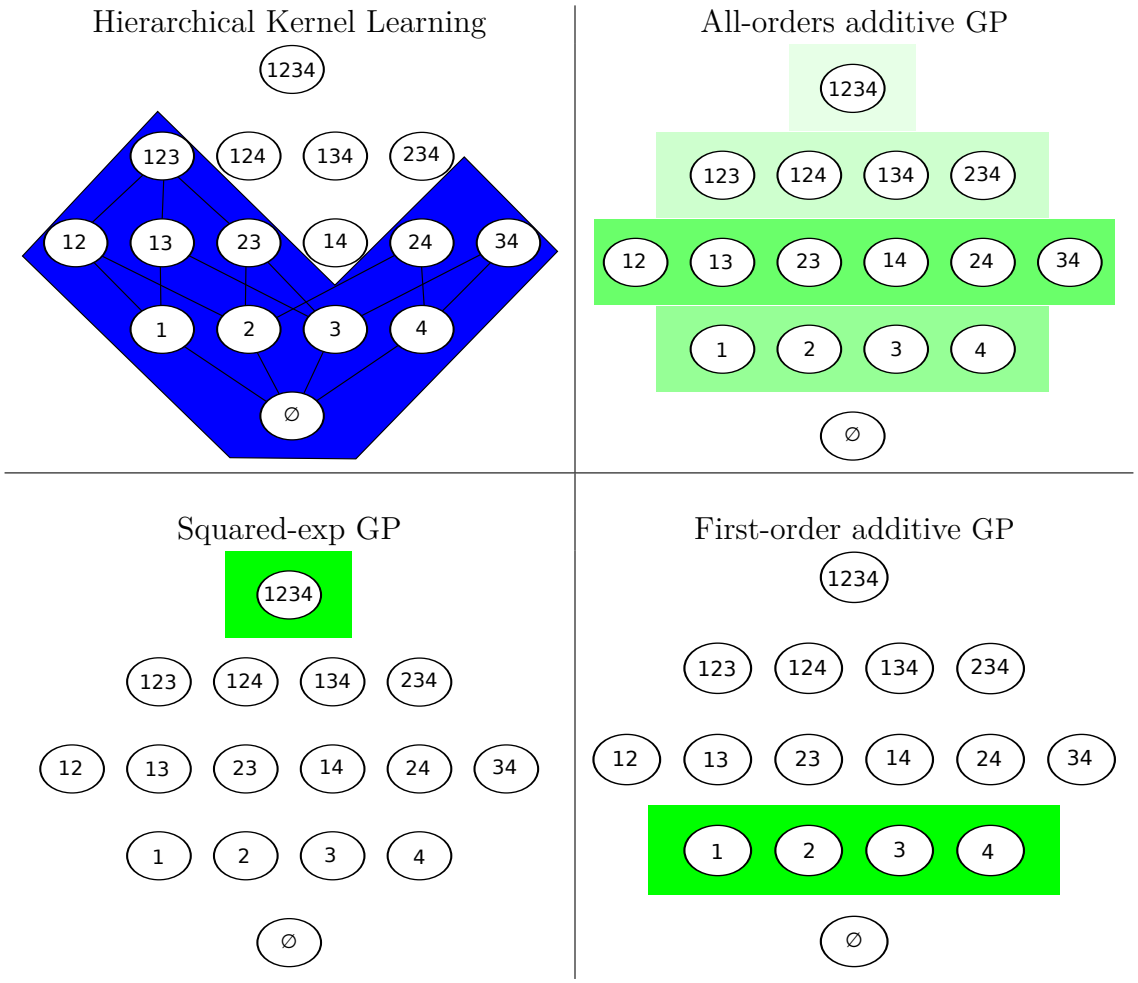


Fig. 6.2 A comparison of different additive model classes. Nodes represent different interaction terms, ranging from first-order to fourth-order interactions. Coloured boxes represent the weightings of different terms. *Top left:* HKL can select a hull of interaction terms, but must use a pre-determined 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.

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.

Other Related Models

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 class.

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

6.5 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 separate 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 six different methods. In the results tables below, GP Additive refers to a GP using the additive kernel with squared-exp base kernels. For speed, we limited the maximum order of interaction in the additive kernels to 10. GP-1st denotes an additive GP model with only first-order interactions - a sum of one-dimensional kernels. GP Squared-Exp is a GP 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 (**?**).

For the regression experiments, we also compared against the structure search method from **??**, run up to depth 10, using the SE and RQ base kernel families.

6.5.1 Datasets

We compared these methods on a diverse set of regression and classification datasets from the UCI repository (**?**). Their size and dimension are given in **????**:

Table 6.2 Regression Dataset Statistics

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

Table 6.3 Classification Dataset Statistics

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

Bach Synthetic Dataset

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

$$f(\mathbf{x}) = \sum_{i=1}^{4} \sum_{j=1+1}^{4} x_i x_j + \epsilon \qquad \epsilon \sim \mathcal{N}(0, \sigma_\epsilon) \tag{6.25}$$

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 the 16 cross-terms optimal for predicting the output. GP-SE, in contrast, can learn to ignore the noisy copy variables, but cannot learn to ignore the higher-order interactions between dimensions. 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.

6.5.2 Results

???????? 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 6.4 Regression Mean Squared Error

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

Table 6.5 Regression Negative Log Likelihood

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

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 and

Table 6.6 Classification percent error

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

Table 6.7 Classification negative log-likelihood

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

structure search methods usually outperformed the other methods, especially on regression problems. The structure search outperforms the additive GP, but at the cost of a slow search over kernels.

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 **??**). Because the additive GP is a superset of both the GP-1st 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 could be expected to benefit from approximately integrating over the kernel parameters.

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

6.5.3 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/additive-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 non-linearly separable clusters with complex shapes. The proposed model assumes that each observa-

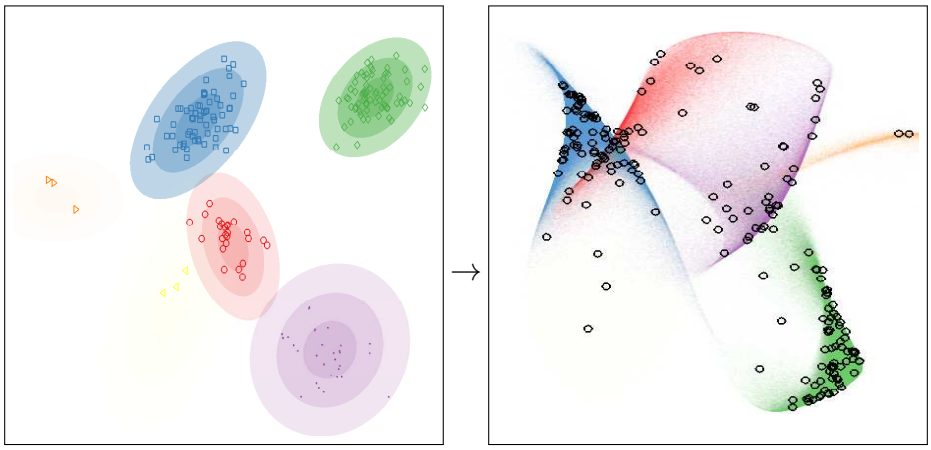


Fig. 7.1 A sample from the iWMM prior. *Left:* In the latent space, a mixture distribution is sampled from a Dirichlet process mixture of Gaussians. *Right:* The latent mixture is smoothly warped to produce non-Gaussian manifolds in the observed space.

tion 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 **??** 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.

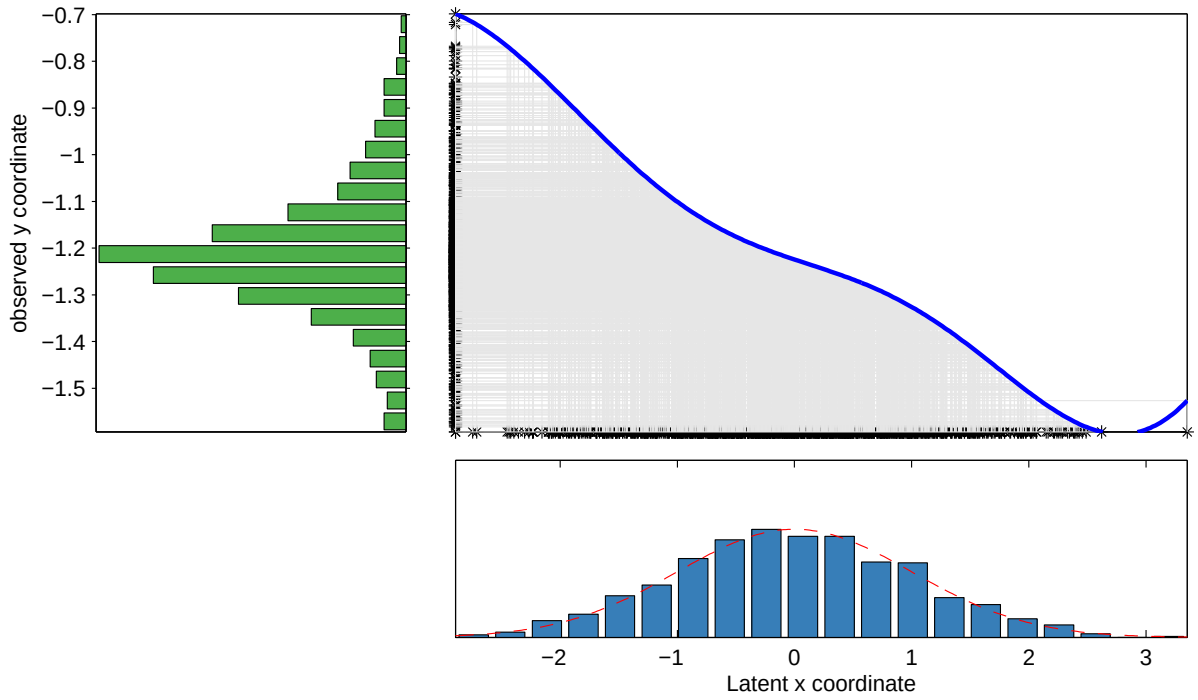


Fig. 7.2 A visual representation of the Gaussian process latent variable model. Bottom: density and samples from a 1D Gaussian, specifying the distribution $p(\mathbf{X})$ in the latent space. Top Right: A function drawn from a GP prior. Left: A nonparametric density defined by warping the latent density 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, at the cost of non-analytic inference. The GP-LVM can also be thought of as a method for modeling the covariance between rows of Y, using a number of parameters which grows only 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 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

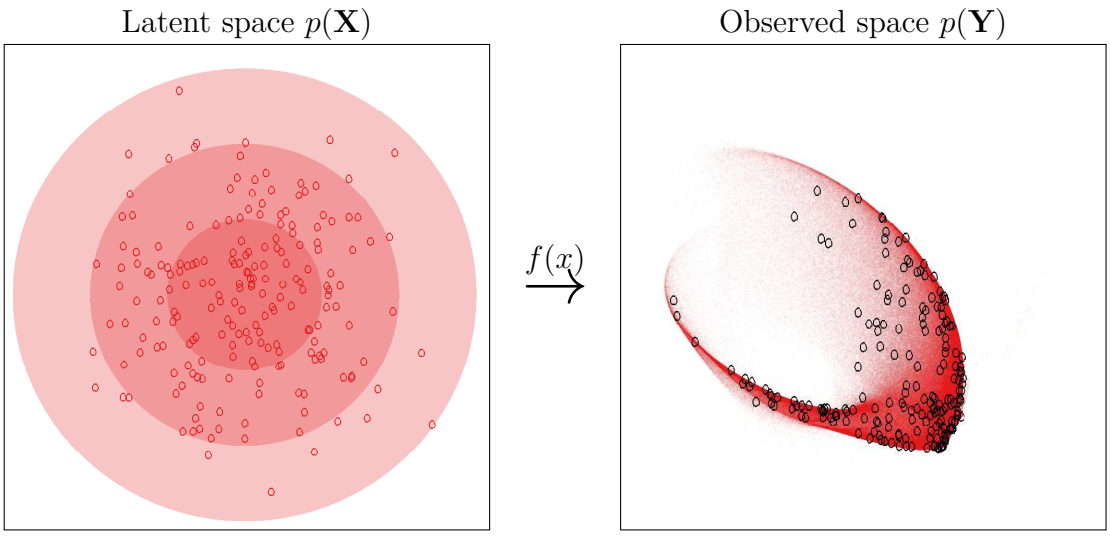


Fig. 7.3 A visual representation of the Gaussian process latent variable model. Left: Isocontours and samples from a 2D Gaussian, specifying the distribution $p(\mathbf{X})$ in the latent space. Right: Density and samples from a nonparametric density defined by warping the latent density through a function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

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 (**??**), (**??**) and (**??**), 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 $f_d(\mathbf{x}) \sim \text{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$
4. For each observation $n = 1, 2, \ldots, N$
   (a) Draw latent assignment $z_n \sim \text{Mult}(\boldsymbol{\lambda})$
   (b) Draw latent coordinates $\mathbf{x}_n \sim \mathcal{N}(\boldsymbol{\mu}_{z_n}, \mathbf{R}_{z_n}^{-1})$

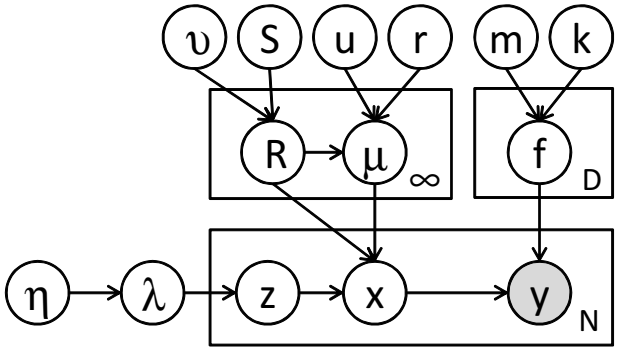


Fig. 7.4 A graphical model representation of the infinite warped mixture model, where the shaded and unshaded nodes indicate observed and latent variables, respectively, and plates indicate repetition.

(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 **??** 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 separate 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 un-normalized 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

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

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 (**??**).

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 (**??**) 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 **??**, **??**, and **??**.

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 **??**b.

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.

Figure **??** 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.

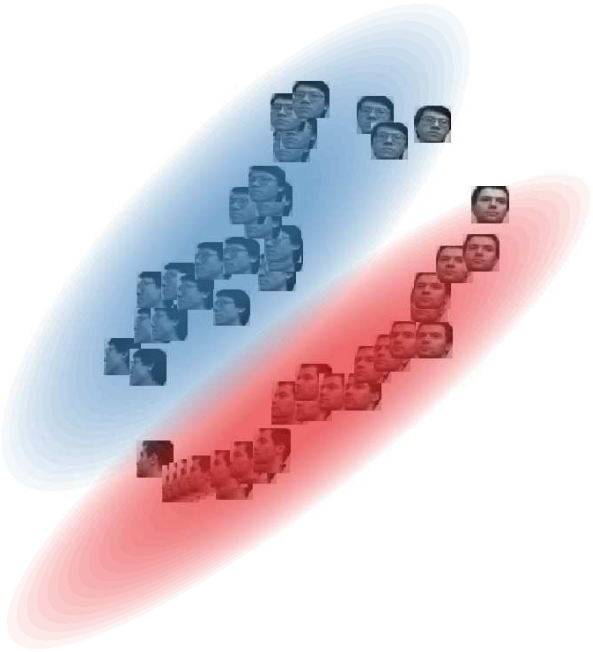
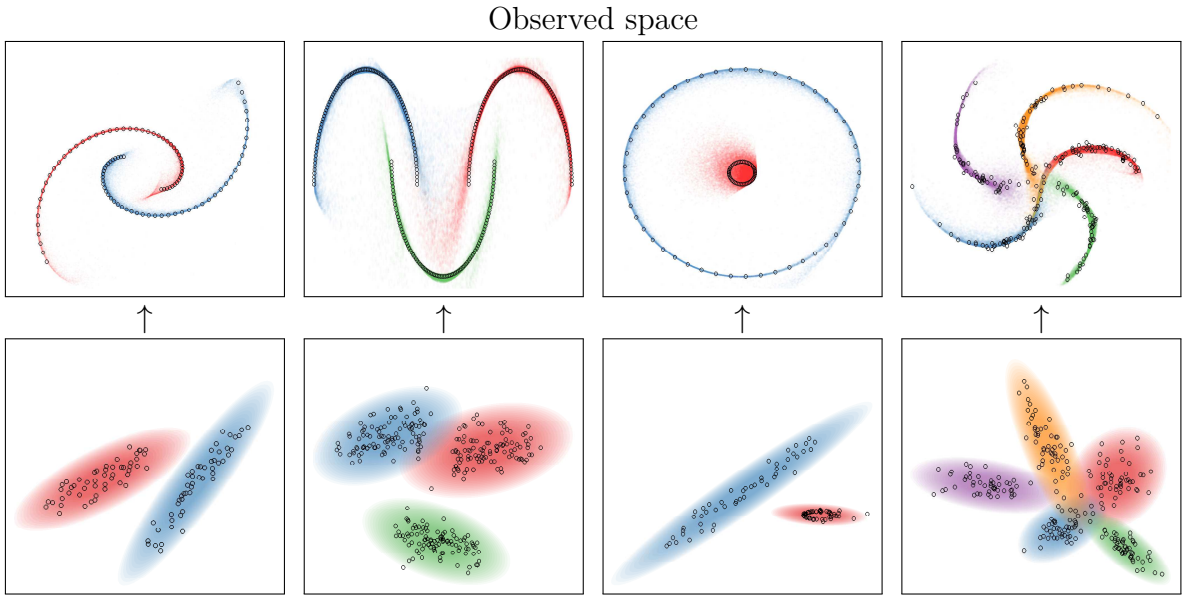


Fig. 7.5 A sample from the 2-dimensional latent space when modeling a series of 32x32 face images. Our model correctly discovers that the data consists of two separate manifolds, both approximately one-dimensional, which share the same head-turning structure.

7.5.2 Synthetic Datasets

Next, we demonstrate the proposed model on the four synthetic datasets shown in Figure **??**. None of these four datasets can be appropriately clustered by Gaussian mixture models (GMM). For example, consider the 2-curve data shown in Figure **??** (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 space, as shown at the bottom row of Figure **??** (a). The iWMM separated the two curved lines by nonlinearly warping two Gaussians from the latent space to the observed space.

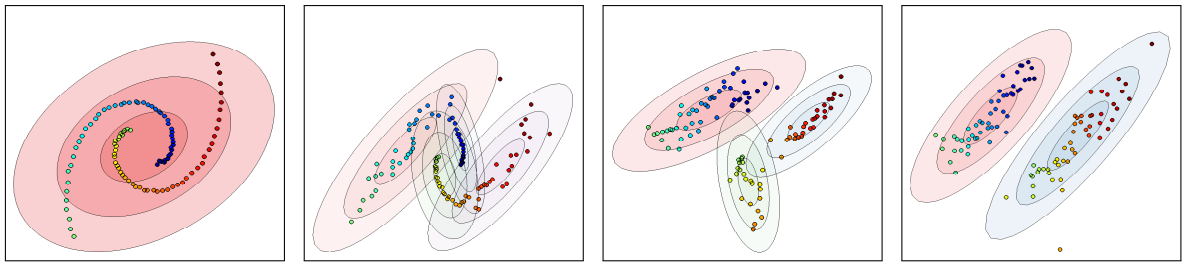
Figure **??** (c) shows an interesting manifold learning challenge: a dataset consisting of two circles. The outer circle is modeled in the latent space by a Gaussian with effectively one degree of freedom. This linear topology fits the outer circle in the observed space by bending the two ends until they overlap. In contrast, the sampler fails to discover the



(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.

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.



(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.

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 **??** shows the samples of the latent coordinates and clusters of the iWMM over time, when modeling the 2-curve data. **??**(a) shows the latent coordinates initialized at the observed coordinates, starting with one latent component. At the 500th iteration **??**(b), each curved line is modeled by two components. At the 1800th iteration **??**(c), the left curved line is modeled by a single component. At the 3000th iteration **??**(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

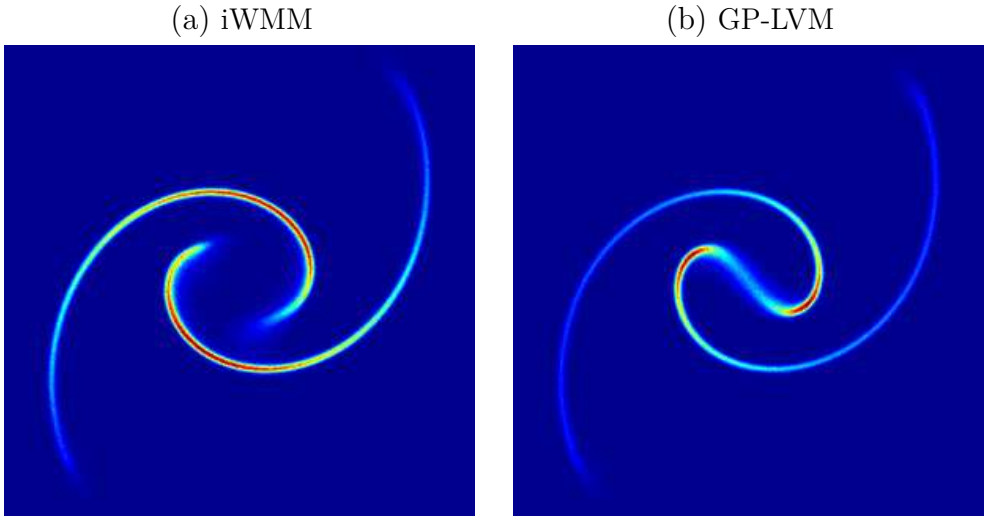


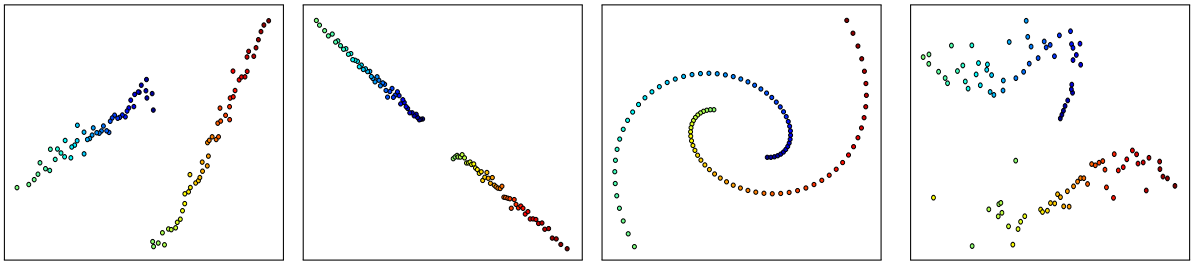
Fig. 7.8 *Left:* The posterior density in the observed space with the 2-curve data inferred by the iWMM. *Right:* The posterior inferred by the iWMM with one component, a model equivalent to the GP-LVM.

??(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 **??**(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. **??**(b) shows that the single-cluster variant of the iWMM posterior is forced to place significant density connecting the two clusters, since it has to reproduce the observed density manifold by warping a single Gaussian.

7.5.5 Visualization

Next, we briefly investigate the potential of the iWMM for low-dimensional visualization of data. Figure **??** (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 **??** (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 **??** (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 **??**. 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 **??** 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 **??** 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.

Appendix A

Gaussian Conditionals

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

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

then

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

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

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

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

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

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

and similarily for the mean vectors.

Appendix B

Kernel Definitions

Here we give the formulas for all one-dimensional base kernels mentioned in the thesis. Each of these formulas is multiplied by a scale factor σ_f^2, which we omit for clarity.

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

$$\mathrm{WN}(x, x') = \delta(x - x') \tag{B.2}$$

$$\mathrm{Lin}(x, x') = (x - c)(x' - c) \tag{B.3}$$

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

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

$$\mathrm{Per}(x, x') = \sigma_f^2 \frac{\exp\left(\frac{1}{\ell^2}\cos 2\pi\frac{(x-x')}{p}\right) - I_0\left(\frac{1}{\ell^2}\right)}{\exp\left(\frac{1}{\ell^2}\right) - I_0\left(\frac{1}{\ell^2}\right)} \tag{B.6}$$

$$\cos(x, x') = \cos\left(\frac{2\pi(x - x')}{p}\right) \tag{B.7}$$

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

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

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

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 kernel parameters. ?????? are plotted in ??, and ?????? are plotted in ??. Draws from GP priors with changepoint kernels are shown in ??.

The Generalized Periodic Kernel

? showed that the standard periodic kernel due to **?** can be decomposed into a periodic and a constant component. He derived the equivalent periodic kernel without any constant component, shown in **??**. He further showed that its limit as the lengthscale grows is the cosine kernel:

$$\lim_{\ell \to \infty} \mathrm{Per}(x, x') = \cos\left(\frac{2\pi(x - x')}{p}\right). \tag{B.11}$$

Separating out the constant component allows us to express negative prior covariance, as well as increasing the interpretability of the resulting models.

Appendix C

Search Operators

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

The search used in the multidimensional regression experiments in **????** used only the following search operators:

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

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

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

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

Afterwards, we added several more search operators in order to speed up the search. These new operators do not change the set of possible models.

To accommodate changepoints and changewindows, we introduced the following additional operators to our search:

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

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

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

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

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

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

$$\mathcal{S} \rightarrow \mathcal{B} \tag{C.8}$$

$$\mathcal{S} + \mathcal{S}' \rightarrow \mathcal{S} \tag{C.9}$$

$$\mathcal{S} \times \mathcal{S}' \rightarrow \mathcal{S} \tag{C.10}$$

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

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

Which allows a new base kernel to be added along with the constant kernel, for cases when multiplying by a base kernel by itself would restrict the model too much.

References

R.P. Adams and Z. Ghahramani. Archipelago: nonparametric Bayesian semi-supervised learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009.

Ryan P Adams, Hanna M Wallach, and Zoubin Ghahramani. Learning the structure of deep sparse graphical models. In *International Conference on Artificial Intelligence and Statistics*, pages 1–8, 2010.

F. Bach. Exploring large feature spaces with hierarchical multiple kernel learning. In *Advances in Neural Information Processing Systems*, pages 105–112. 2009a.

Francis Bach. High-dimensional non-linear variable selection through hierarchical kernel learning. *CoRR*, abs/0909.0844, 2009b.

Francis R Bach, Gert RG Lanckriet, and Michael I Jordan. Multiple kernel learning, conic duality, and the SMO algorithm. In *Proceedings of the twenty-first international conference on Machine learning*, page 6. ACM, 2004.

K. Bache and M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`.

Pierre Baldi and Peter J Sadowski. Understanding dropout. In *Advances in Neural Information Processing Systems*, pages 2814–2822, 2013.

A Barbu, A Bridge, Z Burchill, D Coroian, S Dickinson, S Fidler, A Michaux, S Mussman, S Narayanaswamy, D Salvi, L Schmidt, J Shangguan, JM Siskind, J Waggoner, S Wang, J Wei, Y Yin, and Z Zhang. Video in sentences out. In *Conference on Uncertainty in Artificial Intelligence*, 2012.

Richard Bellman. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences of the United States of America*, 42(10):767, 1956.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166, 1994.

Yoshua Bengio, Olivier Delalleau, and Nicolas Le Roux. The curse of highly variable functions for local kernel machines. *Advances in neural information processing systems*, 18:107–114, 2006. ISSN 1049-5258.

W. Bing, Z. Wen-qiong, C. Ling, and L. Jia-hong. A GP-based kernel construction and optimization method for RVM. In *International Conference on Computer and Automation Engineering (ICCAE)*, volume 4, pages 419–423, 2010.

Salomon Bochner. *Lectures on Fourier integrals*, volume 42. Princeton University Press, 1959.

George EP Box, Gwilym M Jenkins, and Gregory C Reinsel. *Time series analysis: forecasting and control*. Wiley. com, 2013.

G.E.P. Box, G.M. Jenkins, and G.C. Reinsel. *Time series analysis: forecasting and control*. 1976.

W. Cao and R. Haralick. Nonlinear manifold clustering by dimensionality. In *International Conference on Pattern Recognition (ICPR)*, volume 1, pages 920–924. IEEE, 2006.

Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, 2011.

Youngmin Cho. *Kernel methods for deep learning*. PhD thesis, University of California, San Diego, 2012.

M. Christoudias, R. Urtasun, and T. Darrell. Bayesian localized multiple kernel learning. *Technical report, EECS Department, University of California, Berkeley*, 2009.

Wikimedia Commons. Stereographic projection of a Sudanese Möbius band, 2005. URL http://commons.wikimedia.org/wiki/File:MobiusSnail2B.png.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20 (3):273–297, 1995.

Andreas Damianou and Neil Lawrence. Deep Gaussian processes. In *Artificial Intelligence and Statistics*, pages 207–215, 2013.

Eyal Dechter, Jon Malmaud, Ryan P Adams, and Joshua B Tenenbaum. Bootstrap learning via modular concept discovery. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 1302–1309. AAAI Press, 2013.

L. Diosan, A. Rogozan, and J.P. Pecuchet. Evolving kernel functions for SVMs by genetic programming. In *Machine Learning and Applications, 2007*, pages 19–24. IEEE, 2007.

Nicolas Durrande, David Ginsbourger, and Olivier Roustant. Additive kernels for gaussian process modeling. *arXiv preprint arXiv:1103.4023*, 2011.

David Duvenaud, Hannes Nickisch, and Carl Edward Rasmussen. Additive Gaussian processes. In *Advances in Neural Information Processing Systems 24*, pages 226–234, Granada, Spain, 2011.

David Duvenaud, James Robert Lloyd, Roger Grosse, Joshua B. Tenenbaum, and Zoubin Ghahramani. Structure discovery in nonparametric regression through compositional kernel search. In *Proceedings of the 30th International Conference on Machine Learning*, June 2013.

David Duvenaud, Oren Rippel, Ryan P. Adams, and Zoubin Ghahramani. Avoiding pathologies in very deep networks. Reykjavik, Iceland, April 2014. URL http://arxiv.org/pdf/1402.5836.pdf.

D. Eaton and K. Murphy. Bayesian structure learning using dynamic programming and MCMC. 2007. URL http://www.cs.ubc.ca/~murphyk/Papers/eaton-uai07.pdf.

Ehsan Elhamifar and René Vidal. Sparse manifold clustering and embedding. In *Advances in Neural Information Processing Systems*, pages 55–63, 2011.

E.B. Fox and D.B. Dunson. Multiresolution Gaussian Processes. In *Neural Information Processing Systems 25*. MIT Press, 2013.

N. Friedman and D. Koller. Being Bayesian about Network Structure: A Bayesian Approach to Structure Discovery in Bayesian Networks. *Machine Learning*, 50:95–126, 2003.

M. Ganesalingam and W. T. Gowers. A fully automatic problem solver with human-style output. *CoRR*, abs/1309.4501, 2013.

Roman Garnett, Michael A Osborne, Steven Reece, Alex Rogers, and Stephen J Roberts. Sequential bayesian prediction in the presence of changepoints and faults. *The Computer Journal*, 53(9):1430–1446, 2010.

A. Geiger, R. Urtasun, and T. Darrell. Rank priors for continuous non-linear dimensionality reduction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 880–887. IEEE, 2009.

Andrew Gelman. Why waste time philosophizing?, 2013. URL http://andrewgelman.com/2013/02/11/why-waste-time-philosophizing/.

Andrew Gelman and Cosma Rohilla Shalizi. Philosophy and the practice of bayesian statistics. *British Journal of Mathematical and Statistical Psychology*, 2012.

Robert Gens and Pedro Domingos. Learning the structure of sum-product networks. In *International Conference on Machine learning*, 2013.

Z. Ghahramani and M.J. Beal. Variational inference for Bayesian mixtures of factor analysers. *Advances in Neural Information Processing Systems*, 12:449–455, 2000.

Elad Gilboa, Yunus Saatçi, and J Cunningham. Scaling multidimensional inference for structured Gaussian processes. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.

D. Ginsbourger, O. Roustant, and N. Durrande. Invariances of random fields paths, with applications in Gaussian process regression. Technical Report arXiv:1308.1359 [math.ST], August 2013.

David Ginsbourger, Xavier Bay, Olivier Roustant, and Laurent Carraro. Argumentwise invariant kernels for the approximation of invariant functions. In *Annales de la Faculté de Sciences de Toulouse*, number 3, 2012.

Noah D Goodman, Tomer D Ullman, and Joshua B Tenenbaum. Learning a theory of causality. *Psychological review*, 118(1):110, 2011.

Daniel B Graham and Nigel M Allinson. Characterizing virtual eigensignatures for general purpose face recognition. *Face Recognition: From Theory to Applications*, 163:446–456, 1998.

Roger B. Grosse, Ruslan Salakhutdinov, William T. Freeman, and Joshua B. Tenenbaum. Exploiting compositionality to explore a large space of model structures. In *Uncertainty in Artificial Intelligence*, 2012.

C. Gu. *Smoothing spline ANOVA models*. Springer Verlag, 2002. ISBN 0387953531.

T.J. Hastie and R.J. Tibshirani. *Generalized additive models*. Chapman & Hall/CRC, 1990.

James Hensman, Nicolo Fusi, and Neil D Lawrence. Gaussian processes for big data. *arXiv preprint arXiv:1309.6835*, 2013.

Michiel Hermans and Benjamin Schrauwen. Recurrent kernel machines: Computing with infinite echo state networks. *Neural Computation*, 24(1):104–133, 2012.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Rob J. Hyndman. Time series data library, Accessed July 2013. URL `http://data.is/TSDLdemo`.

Tomoharu Iwata, David Duvenaud, and Zoubin Ghahramani. Warped mixtures for nonparametric cluster shapes. Bellevue, Washington, July 2013. URL `http://arxiv.org/pdf/1206.1846`.

E. T. Jaynes. Highly informative priors. In *Proceedings of the Second International Meeting on Bayesian Statistics*, 1985.

C.G. Kaufman and S.R. Sain. Bayesian functional anova modeling using Gaussian process prior distributions. *Bayesian Analysis*, 5(1):123–150, 2010.

C. Kemp and J.B. Tenenbaum. The discovery of structural form. *Proceedings of the National Academy of Sciences*, 105(31):10687–10692, 2008.

E.D. Klenske, M.N. Zeilinger, B. Scholkopf, and P. Hennig. Nonparametric dynamics estimation for time periodic systems. In *Communication, Control, and Computing (Allerton), 2013 51st Annual Allerton Conference on*, pages 486–493, Oct 2013.

Imre Risi Kondor. *Group theoretical methods in machine learning*. PhD thesis, Columbia University, 2008.

Gabriel Kronberger and Michael Kommenda. Evolution of covariance functions for gaussian process regression using genetic programming. *arXiv preprint arXiv:1305.3794*, 2013.

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

N.D. Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. *Advances in Neural Information Processing Systems*, 16:329–336, 2004.

N.D. Lawrence and R. Urtasun. Non-linear matrix factorization with Gaussian processes. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 601–608. ACM, 2009.

Neil D Lawrence and Andrew J Moore. Hierarchical Gaussian process latent variable models. In *Proceedings of the 24th international conference on Machine learning*, pages 481–488. ACM, 2007.

Miguel Lázaro-Gredilla, Joaquin Quiñonero-Candela, Carl Edward Rasmussen, and Aníbal R Figueiras-Vidal. Sparse spectrum gaussian process regression. *The Journal of Machine Learning Research*, 99:1865–1881, 2010.

J. Lean, J. Beer, and R. Bradley. Reconstruction of solar irradiance since 1610: Implications for climate change. *Geophysical Research Letters*, 22(23):3195–3198, 1995.

Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361, 1995.

Honglak Lee, Chaitanya Ekanadham, and Andrew Ng. Sparse deep belief net model for visual area v2. In *Advances in neural information processing systems*, pages 873–880, 2007.

D. Lerner and D. Asimov. The Sudanese Möbius band. In *SIGGRAPH Electronic Theatre*, 1984.

Percy Liang, Michael I Jordan, and Dan Klein. Learning programs: A hierarchical bayesian approach. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 639–646, 2010.

Douglas A Lind, William G Marchal, Samuel Adam Wathen, and Business Week Magazine. *Basic statistics for business and economics*. McGraw-Hill/Irwin Boston, 2006.

James Robert Lloyd. personal communication, 2013a.

James Robert Lloyd. GEFCom2012 hierarchical load forecasting: Gradient boosting machines and gaussian processes. *International Journal of Forecasting*, 2013b.

James Robert Lloyd, David Duvenaud, Roger Grosse, Joshua B. Tenenbaum, and Zoubin Ghahramani. Automatic construction and natural-language description of nonparametric regression models. Technical Report arXiv:1402.4304 [stat.ML], 2014.

I.G. Macdonald. *Symmetric functions and Hall polynomials*. Oxford University Press, USA, 1998. ISBN 0198504500.

S.N. MacEachern and P. Müller. Estimating mixture of Dirichlet process models. *Journal of Computational and Graphical Statistics*, pages 223–238, 1998.

David JC MacKay. *Bayesian methods for adaptive models*. PhD thesis, California Institute of Technology, 1992.

David J.C. MacKay. Introduction to gaussian processes. *NATO ASI Series F Computer and Systems Sciences*, 168:133–166, 1998.

David JC MacKay. *Information theory, inference, and learning algorithms*. Cambridge University press, 2003.

James Martens. Deep learning via Hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning*, pages 735–742, 2010.

JH Maunsell and David C van Essen. The connections of the middle temporal visual area (mt) and their relationship to a cortical hierarchy in the macaque monkey. *The Journal of neuroscience*, 3(12):2563–2586, 1983.

James Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, pages 415–446, 1909.

Charles A Micchelli, Yuesheng Xu, and Haizhang Zhang. Universal kernels. *The Journal of Machine Learning Research*, 7:2651–2667, 2006.

Ha Quang Minh, Partha Niyogi, and Yuan Yao. Mercer's theorem, feature maps, and smoothing. In *Learning theory*, pages 154–168. Springer, 2006.

T.P. Minka. Expectation propagation for approximate bayesian inference. In *Uncertainty in Artificial Intelligence*, volume 17, pages 362–369. Citeseer, 2001.

@ML_Hipster. "...essentially, all models are wrong, but yours are stupid too." – G.E.P. Box in a less than magnanimous mood., 2013. URL https://twitter.com/ML_Hipster/status/394577463990181888.

Grégoire Montavon, Dr Braun, and Klaus-Robert Müller. Layer-wise analysis of deep networks with Gaussian kernels. *Advances in Neural Information Processing Systems*, 23:1678–1686, 2010.

Radford M Neal. *Bayesian learning for neural networks*. PhD thesis, University of Toronto, 1995.

R.M. Neal. Density modeling and clustering using dirichlet diffusion trees. *Bayesian Statistics*, 7:619–629, 2003.

J.A. Nelder and R.W.M. Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society. Series A (General)*, 135(3):370–384, 1972.

A.Y. Ng, M.I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. *Advances in Neural Information Processing Systems*, 2:849–856, 2002.

H. Nickisch and C. Rasmussen. Gaussian mixture modeling with Gaussian process latent variable models. *Pattern Recognition*, pages 272–282, 2010.

J. Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980.

Nutonian. Eureqa, 2011. URL http://www.nutonian.com/.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *arXiv preprint arXiv:1211.5063*, 2012.

T.A. Plate. Accuracy versus interpretability in flexible modeling: Implementing a tradeoff using Gaussian process models. *Behaviormetrika*, 26:29–50, 1999. ISSN 0385-7417.

Daniel Preotiuc-Pietro and Trevor Cohn. A temporal model of text periodicities using Gaussian processes. In *EMNLP*, pages 977–988. ACL, 2013.

J. Quiñonero-Candela and C.E. Rasmussen. A unifying view of sparse approximate Gaussian process regression. *The Journal of Machine Learning Research*, 6:1939–1959, 2005.

Herschel Rabitz and Ömer F Aliş. General foundations of high-dimensional model representations. *Journal of Mathematical Chemistry*, 25(2-3):197–233, 1999.

W.M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, pages 846–850, 1971.

Carl Edward Rasmussen and Zoubin Ghahramani. Occam's razor. *Advances in neural information processing systems*, pages 294–300, 2001.

Carl Edward Rasmussen and Hannes Nickisch. Gaussian processes for machine learning (gpml) toolbox. *J. Mach. Learn. Res.*, 11:3011–3015, December 2010. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1756006.1953029.

C.E. Rasmussen. The infinite Gaussian mixture model. *Advances in Neural Information Processing Systems*, 12(5.2):2, 2000.

C.E. Rasmussen and C.K.I. Williams. *Gaussian Processes for Machine Learning*, volume 38. The MIT Press, Cambridge, MA, USA, 2006.

Salah Rifai, Grégoire Mesnil, Pascal Vincent, Xavier Muller, Yoshua Bengio, Yann Dauphin, and Xavier Glorot. Higher order contractive auto-encoder. In *Machine Learning and Knowledge Discovery in Databases*, pages 645–660. Springer, 2011a.

Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning*, pages 833–840, 2011b.

Oren Rippel and Ryan Prescott Adams. High-dimensional probability estimation with deep density models. *arXiv preprint arXiv:1302.5125*, 2013.

D. Ruppert, M.P. Wand, and R.J. Carroll. *Semiparametric regression*, volume 12. Cambridge University Press, 2003.

Yunus Saatçi, Ryan D Turner, and Carl E Rasmussen. Gaussian process change point models. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 927–934, 2010.

Ruslan Salakhutdinov and Geoffrey Hinton. Using deep belief nets to learn covariance kernels for Gaussian processes. *Advances in Neural information processing systems*, 20:1249–1256, 2008.

M. Salzmann, R. Urtasun, and P. Fua. Local deformation models for monocular 3D shape recovery. In *IEEE Conference on Computer Vision and Pattern Recognition*, CVPR, pages 1–8, 2008.

Andrew Saxe, Pang W Koh, Zhenghao Chen, Maneesh Bhand, Bipin Suresh, and Andrew Y Ng. On random weights and unsupervised feature learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1089–1096, 2011.

Andrew M Saxe, James L McClelland, and Surya Ganguli. Dynamics of learning in deep linear neural networks. 2013.

Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, April 2009. ISSN 1095-9203. doi: 10.1126/science.1165893.

G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2): 461–464, 1978.

Jayaram Sethuraman. A constructive definition of Dirichlet priors. *Statistica Sinica*, 4: 639–650, 1994.

E. Snelson and Z. Ghahramani. Sparse Gaussian processes using pseudo-inputs. *Advances in Neural Information Processing Systems*, 2006.

Ercan Solak, Roderick Murray-Smith, E. Solak, William Leithead, Carl Rasmussen, and Douglas Leith. Derivative observations in Gaussian process models of dynamic systems, 2003.

Nitish Srivastava. *Improving neural networks with dropout*. PhD thesis, University of Toronto, 2013.

Christian Steinruecken. *Bayesian Compression*. PhD thesis, University of Cambridge, 2014.

M. Stitson, A. Gammerman, V. Vapnik, V. Vovk, C. Watkins, and J. Weston. Support vector regression with ANOVA decomposition kernels. *Advances in kernel methods: Support vector learning*, pages 285–292, 1999.

Y.W. Teh, M.I. Jordan, M.J. Beal, and D.M. Blei. Hierarchical dirichlet processes. *Journal of the American Statistical Association*, 101(476):1566–1581, 2006.

M.E. Tipping and C.M. Bishop. Mixtures of probabilistic principal component analyzers. *Neural computation*, 11(2):443–482, 1999.

M. Titsias and N. Lawrence. Bayesian Gaussian process latent variable model. *AISTATS*, 2010.

L. Todorovski and S. Dzeroski. Declarative bias in equation discovery. In *International Conference on Machine Learning*, pages 376–384, 1997.

Jarno Vanhatalo, Jaakko Riihimaki, Jouni Hartikainen, Pasi Jylanki, Ville Tolvanen, and Aki Vehtari. Gpstuff, 2014. http://mloss.org/software/view/451/.

V.N. Vapnik. *Statistical learning theory*, volume 2. Wiley New York, 1998.

G. Wahba. *Spline models for observational data*. Society for Industrial Mathematics, 1990. ISBN 0898712440.

J. Wang, J. Lee, and C. Zhang. Kernel trick embedded Gaussian mixture model. In *Algorithmic Learning Theory*, pages 159–174. Springer, 2003.

Sida Wang and Christopher Manning. Fast dropout training. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 118–126, 2013.

T. Washio, H. Motoda, Y. Niwa, et al. Discovering admissible model equations from observed data based on scale-types and identity constraints. In *International Joint Conference On Artifical Intelligence*, volume 16, pages 772–779, 1999.

Andrew Wilson, David A Knowles, and Zoubin Ghahramani. Gaussian process regression networks. In *Proceedings of the 29th International Conference on Machine Learning*, pages 599–606, 2012.

Andrew Gordon Wilson and Ryan Prescott Adams. Gaussian process covariance kernels for pattern discovery and extrapolation. *arXiv: 1302.4245*, June 2013.

I-C Yeh. Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete research*, 28(12):1797–1808, 1998.

Y. Zhang and C. Sutton. Quasi-Newton Markov chain Monte Carlo. *Advances in Neural Information Processing Systems*, pages 2393–2401, 2011.