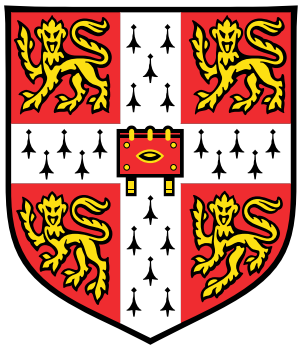
Structured Gaussian Process Models



David Kristjanson Duvenaud

Department of Engineering

University of Cambridge

This dissertation is submitted for the degree of

Doctor of Philosophy

Pembroke College

June 2014

I would like to dedicate this thesis to my loving parents ...

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

And I would like to acknowledge ...

Abstract

This is where you write your abstract ...

Contents

List of Figures

List of Tables

Chapter 1

Introduction

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

1.1 Regression

The general problem of regression consists of learning a function f mapping from some input space $\mathcal{X}$ to some output space $\mathcal{Y}$. We would like an expressive language which can represent both simple parametric forms of f such as linear, polynomial, etc. and also complex nonparametric functions specified in terms of properties such as smoothness, periodicity, etc. Fortunately, Gaussian processes (GPs) provide a very general and analytically tractable way of capturing both simple and complex functions.

1.2 Gaussian process models

Gaussian processes are distributions over functions such that any finite subset of function evaluations, $(f(x_1), f(x_2), \ldots f(x_N))$, have a joint Gaussian distribution (**?**). A GP is completely specified by its mean function, $\mu(x) = \mathbb{E}(f(x))$ and kernel (or covariance) function $k(x, x') = \mathrm{Cov}(f(x), f(x'))$. It is common practice to assume zero mean, since marginalizing over an unknown mean function can be equivalently expressed as a zero-mean GP with a new kernel. The structure of the kernel captures high-level properties of the unknown function, f, which in turn determines how the model generalizes or extrapolates to new data. We can therefore define a language of regression models by specifying a language of kernels.

1.2.1 Useful properties of Gaussian process models

- **Tractable inference** Given a kernel function, the 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 - for example, neural nets.

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

- **Closed-form posterior** The posterior predictive distribution of a GP is another GP. This means that GPs can easily be composed with other models or decision procedures. For example, (⋆) Carl's reinforcement learning work.

Figure 1.1 shows a Gaussian process posterior. Typically, it's rendered with the mean and +- 2SD, but there's nothing special about mean.

1.3 Latent Variable Models

Besides being useful for modeling functions, a simple extension allows GPs to be useful for general density modeling. Unfortunately, this extension causes many of the useful properties of the GP not to hold.

1.4 Covariance functions

Kernels specify similarity between function values of two objects, not between similarity of objects

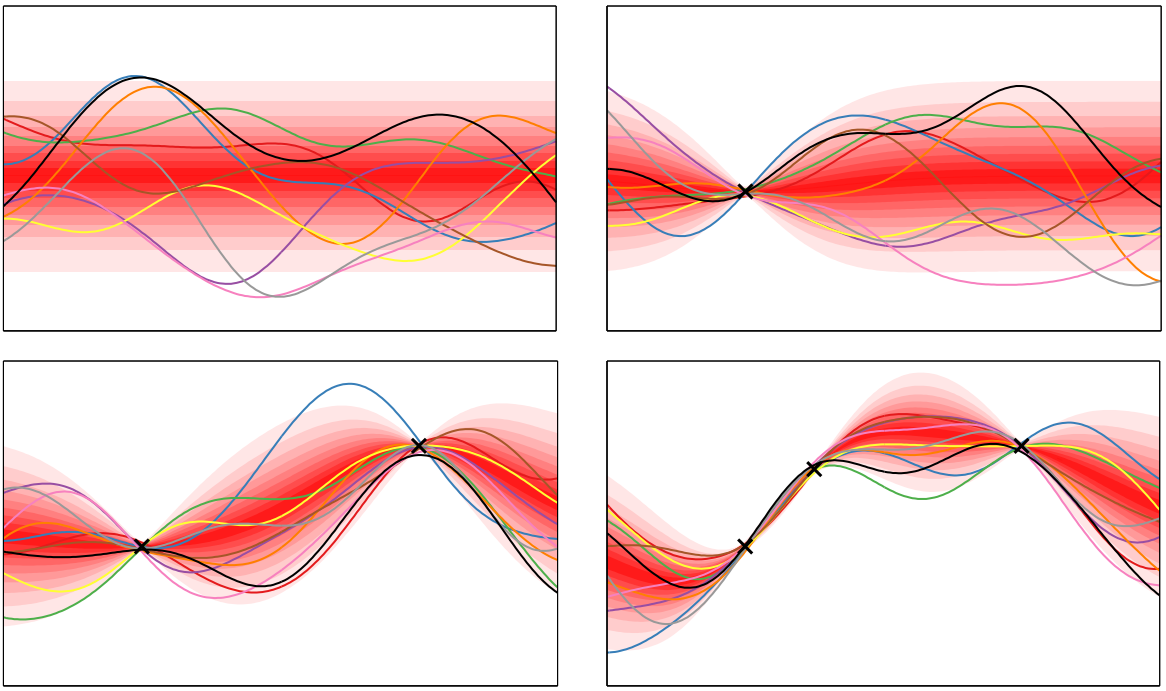


Fig. 1.1 A visual representation of a one-dimensional Gaussian process posterior. Red isocountours show the marginal density at each input location. Coloured lines are samples from the posterior.

1.5 Structure through additivity

A theme throughout this thesis is exploring the idea that a lot of the expressivity of GP models comes from the fact that these models can be combined and decomposed additively.

1.5.1 Derivation of Component Marginal Variance

In this section, we derive the posterior marginal variance and covariance of the additive components of a GP. These formulas let us plot the marginal variance of each component separately. These formulas can also be used to examine the posterior covariance between pairs of components.

Let us assume that our function $\mathbf{f}$ is a sum of two functions, $\mathbf{f}_1$ and $\mathbf{f}_2$, where $\mathbf{f} = \mathbf{f}_1 + \mathbf{f}_2$.

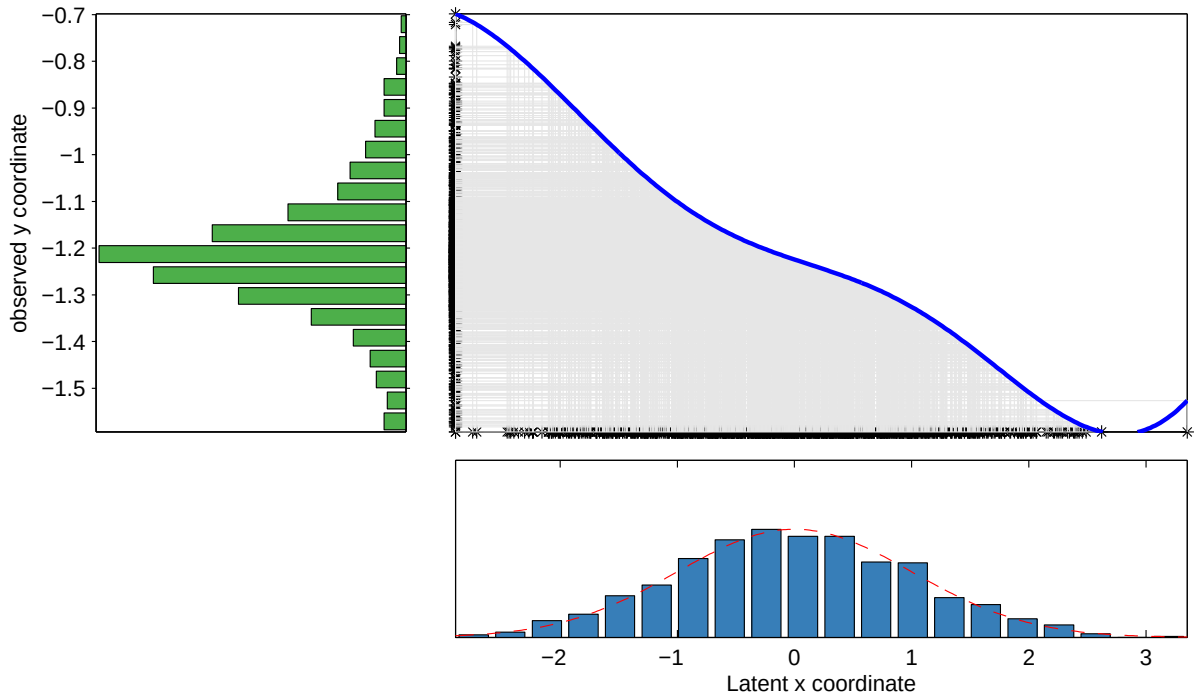


Fig. 1.2 A visual representation of the Gaussian process latent variable model. Bottom: density and samples from a 1D Gaussian, specifying the distribution $p(\mathbf{X})$ in the latent space. Top Right: A function drawn from a GP prior. Left: A nonparametric density defined by warping the latent density through the function drawn from a GP prior.

If $\mathbf{f}_1$ and $\mathbf{f}_2$ are a priori independent, and $\mathbf{f}_1 \sim \mathrm{GP}(\mu_1, k_1)$ and $\mathbf{f}_2 \sim \mathrm{GP}(\mu_2, k_2)$, then

$$\begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_1^\star \\ \mathbf{f}_2 \\ \mathbf{f}_2^\star \\ \mathbf{f} \\ \mathbf{f}^\star \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_1 \\ \mu_1^\star \\ \mu_2 \\ \mu_2^\star \\ \mu_1 + \mu_2 \\ \mu_1^\star + \mu_2^\star \end{bmatrix}, \begin{bmatrix} \mathbf{k}_1 & \mathbf{k}_1^\star & 0 & 0 & \mathbf{k}_1 & \mathbf{k}_1^\star \\ \mathbf{k}_1^\star & \mathbf{k}_1^{\star\star} & 0 & 0 & \mathbf{k}_1^\star & \mathbf{k}_1^{\star\star} \\ 0 & 0 & \mathbf{k}_2 & \mathbf{k}_2^\star & \mathbf{k}_2 & \mathbf{k}_2^\star \\ 0 & 0 & \mathbf{k}_2^\star & \mathbf{k}_2^{\star\star} & \mathbf{k}_2^\star & \mathbf{k}_2^{\star\star} \\ \mathbf{k}_1 & \mathbf{k}_1^\star & \mathbf{k}_2 & \mathbf{k}_2^\star & \mathbf{k}_1 + \mathbf{k}_2 & \mathbf{k}_1^\star + \mathbf{k}_2^\star \\ \mathbf{k}_1^\star & \mathbf{k}_1^{\star\star} & \mathbf{k}_2^\star & \mathbf{k}_2^{\star\star} & \mathbf{k}_1^\star + \mathbf{k}_2^\star & \mathbf{k}_1^{\star\star} + \mathbf{k}_2^{\star\star} \end{bmatrix} \right) \tag{1.1}$$

where $\mathbf{k}_1 = k_1(\mathbf{X}, \mathbf{X})$ and $\mathbf{k}_1^\star = k_1(\mathbf{X}^\star, \mathbf{X})$.

By the formula for Gaussian conditionals:

$$\mathbf{x}_A | \mathbf{x}_B \sim \mathcal{N}(\mu_A + \boldsymbol{\Sigma}_{AB}\boldsymbol{\Sigma}_{BB}^{-1}(\mathbf{x}_B - \mu_B), \boldsymbol{\Sigma}_{AA} - \boldsymbol{\Sigma}_{AB}\boldsymbol{\Sigma}_{BB}^{-1}\boldsymbol{\Sigma}_{BA}), \tag{1.2}$$

we get that the conditional variance of a Gaussian conditioned on its sum with another

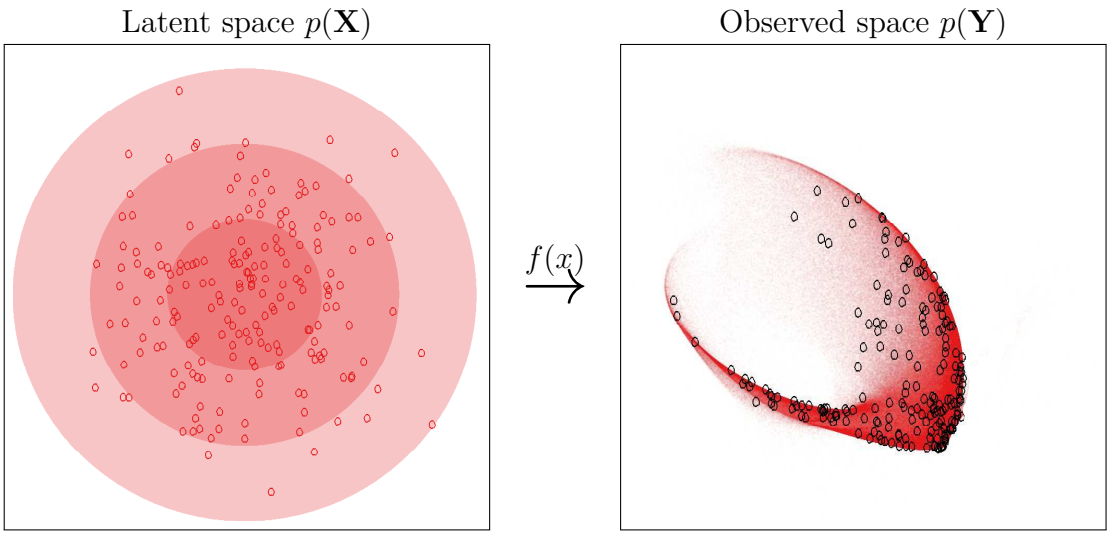


Fig. 1.3 A visual representation of the Gaussian process latent variable model. Left: Isocontours and samples from a 2D Gaussian, specifying the distribution $p(\mathbf{X})$ in the latent space. Right: Density and samples from a nonparametric density defined by warping the latent density through a function drawn from a GP prior.

Gaussian is given by

$$\mathbf{f}_1(\mathbf{x}^\star)|\mathbf{f}(\mathbf{x}) \sim \mathcal{N}\Big(\mu_1(\mathbf{x}^\star) + \mathbf{k}_1(\mathbf{x}^\star,\mathbf{x})\left[\mathbf{K}_1(\mathbf{x},\mathbf{x}) + \mathbf{K}_2(\mathbf{x},\mathbf{x})\right]^{-1}\left(\mathbf{f}(\mathbf{x}) - \mu_1(\mathbf{x}) - \mu_2(\mathbf{x})\right), \\ \mathbf{k}_1(\mathbf{x}^\star,\mathbf{x}^\star) - \mathbf{k}_1(\mathbf{x}^\star,\mathbf{x})\left[\mathbf{K}_1(\mathbf{x},\mathbf{x}) + \mathbf{K}_2(\mathbf{x},\mathbf{x})\right]^{-1}\mathbf{k}_1(\mathbf{x},\mathbf{x}^\star)\Big). \tag{1.3}$$

These formulae express the posterior model uncertainty about different components of the signal, integrating over the possible configurations of the other components.

Chapter 2

Expressing Structure through Kernels

2.1 Composing kernels

Gaussian process models use a kernel to define the covariance between any two function values: $\mathrm{Cov}(y, y') = k(x, x')$. The kernel specifies which structures are likely under the GP prior, which in turn determines the generalization properties of the model. In this section, we review the ways in which kernel families[1]can be composed to express diverse priors over functions.

There has been significant work on constructing GP kernels and analyzing their properties, summarized in Chapter 4 of **?**. Commonly used kernels families include the squared exponential (SE), periodic (Per), linear (Lin), and rational quadratic (RQ) (see Figure 2.1 and the appendix).

Composing Kernels Positive semidefinite kernels (i.e. those which define valid covariance functions) are closed under addition and multiplication. This allows one to create richly structured and interpretable kernels from well understood base components.

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

[1]When unclear from context, we use 'kernel family' to refer to the parametric forms of the functions given in the appendix. A kernel is a kernel family with all of the parameters specified.

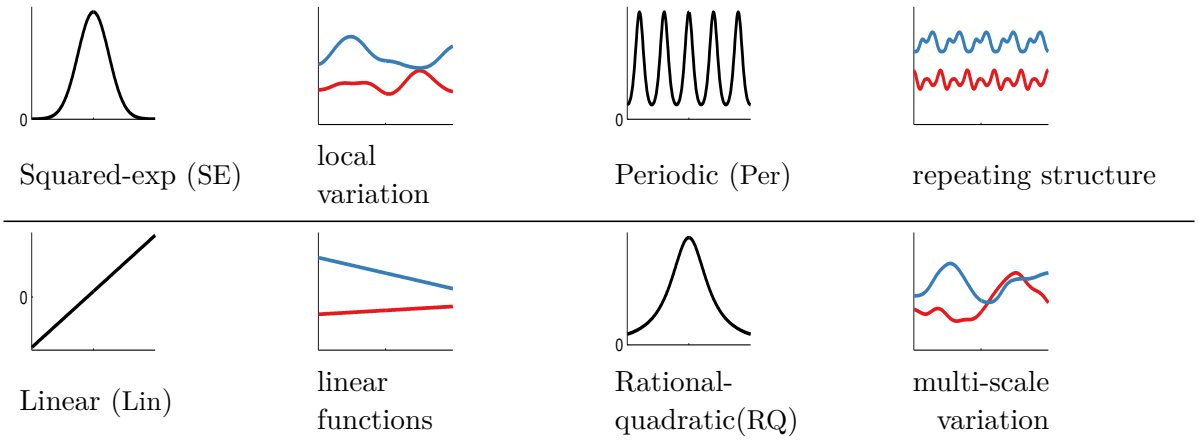


Fig. 2.1 Examples of structures expressible by base kernels. Left and third columns: base kernels $k(\cdot, 0)$. Second and fourth columns: draws from a GP with each repective kernel. The x-axis has the same range on all plots.

2.1.1 Summation

By summing kernels, we can model the data as a superposition of independent functions, possibly representing different structures. Suppose functions f_1, f_2 are draw from independent GP priors, $f_1 \sim \mathcal{GP}(\mu_1, k_1)$, $f_2 \sim \mathcal{GP}(\mu_2, k_2)$. Then $f := f_1 + f_2 \sim \mathcal{GP}(\mu_1 + \mu_2, k_1 + k_2)$.

In time series models, sums of kernels can express superposition of different processes, possibly operating at different scales. In multiple dimensions, summing kernels gives additive structure over different dimensions, similar to generalized additive models (**?**). These two kinds of structure are demonstrated in rows 2 and 4 of figure 4.1, respectively.

Synthetic Data

Because additive kernels can discover non-local structure in data, they are exceptionally well-suited to problems where local interpolation fails. Figure 2.3 shows a dataset which demonstrates this feature of additive GPs, consisting of data drawn from a sum of two axis-aligned sine functions. The training set is restricted to a small, L-shaped area; the test set contains a peak far from the training set locations. The additive GP recovered both of the original sine functions (shown in green), and inferred correctly that most of the variance in the function comes from first-order interactions. The ability of additive GPs to discover long-range structure suggests that this model may be well-suited to deal with covariate-shift problems.

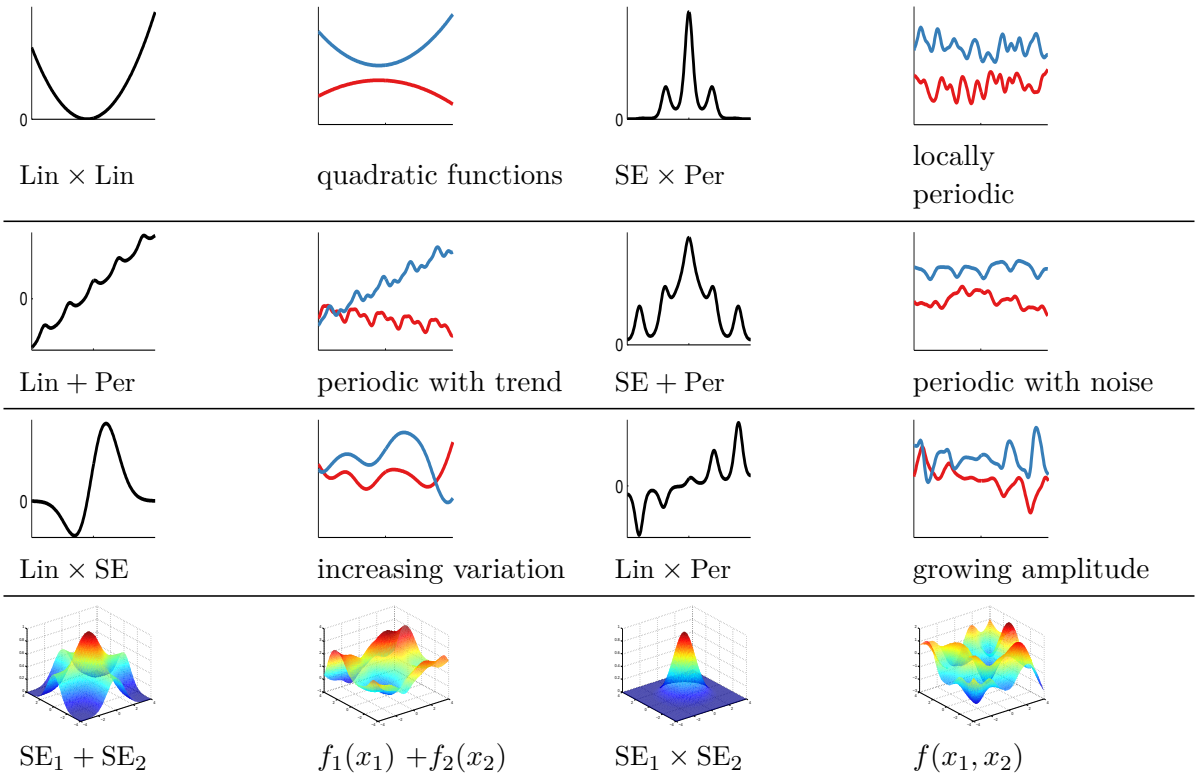
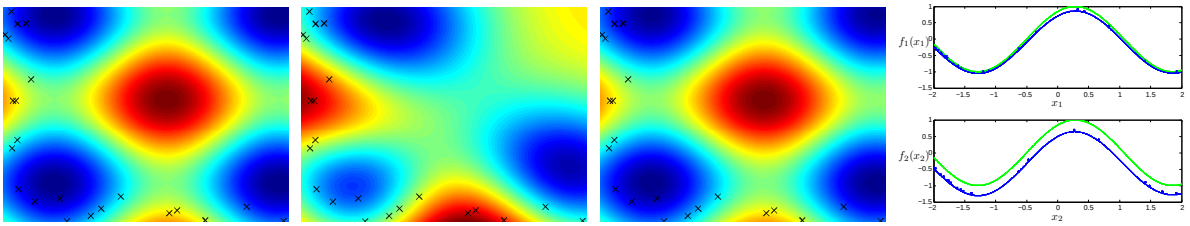


Fig. 2.2 Examples of structures expressible by composite kernels. Left column and third columns: composite kernels $k(\cdot, 0)$. Plots have same meaning as in figure 2.1.

2.1.2 Multiplication

Multiplying kernels allows us to account for interactions between different input dimensions or different notions of similarity. For instance, in multidimensional data, the multiplicative kernel $\mathrm{SE}_1 \times \mathrm{SE}_3$ represents a smoothly varying function of dimensions 1 and 3 which is not constrained to be additive. In univariate data, multiplying a kernel by SE gives a way of converting global structure to local structure. For example, Per corresponds to globally periodic structure, whereas Per × SE corresponds to locally periodic structure, as shown in row 1 of figure 4.1.

Many architectures for learning complex functions, such as convolutional networks **?** and sum-product networks **?**, include units which compute AND-like and OR-like operations. Composite kernels can be viewed in this way too. A sum of kernels can be understood as an OR-like operation: two points are considered similar if either kernel has a high value. Similarly, multiplying kernels is an AND-like operation, since two points are considered similar only if both kernels have high values. Since we are applying these operations to the similarity functions rather than the regression functions themselves,



True Function & data locations | Squared-exp GP posterior mean | Additive GP posterior mean | Additive GP 1st-order functions

Fig. 2.3 Long-range inference in functions with additive structure.

compositions of even a few base kernels are able to capture complex relationships in data which do not have a simple parametric form.

2.1.3 Kernels specify similarity between function values of two objects, not between similarity of objects.

2.1.4 Signal versus noise

When modeling functions, encoding known symmetries greatly aids learning and prediction. We demonstrate that in nonparametric regression, many types of symmetry can be enforced through operations on the covariance function. These symmetries can be composed to produce nonparametric priors on functions whose domains have interesting topological structure such as spheres, torii, and Möbius strips. We demonstrate that marginal likeihood an be used to automatically search over such structures.

Joint work with David Reshef, Roger Grosse, Joshua B. Tenenbaum

2.2 Introduction

It is well-known that the properties of the functions we wish to model can be expressed mainly through the covariance function **?**.

2.3 Expressing Symmetries

In this section, we give recipes for expressing several classes of symmetries. Later, we will show how these can be combined to produce more interesting structures.

Periodicity Given D dimensions, we can enforce rotational symmetry on any subset of the dimensions:

$$f(x) = f(x_i + k\tau_i) \quad \forall k \in \mathbb{Z} \tag{2.1}$$

by the applying a kernel between pairs transformed coordinates $\sin(x), \cos(x)$:

$$k_{\text{periodic}}(x, x') = k(\sin(x), \cos(x), \sin(x'), \cos(x')) \tag{2.2}$$

We can also apply rotational symmetry repeatedy to a single dimension.

Reflective Symmetry along an axis we can enforce the symmetry

$$f(x) = f(-x) \tag{2.3}$$

by the kernel transform

$$k_{\text{symm arg1}}(x, x') = k(x, x') + k(x, -x') + k(-x, x') + k(-x, -x') \tag{2.4}$$

Reflective Symmetry along a diagonal We can enforce symmetry between any two dimensions:

$$f(x, y) = f(y, x) \tag{2.5}$$

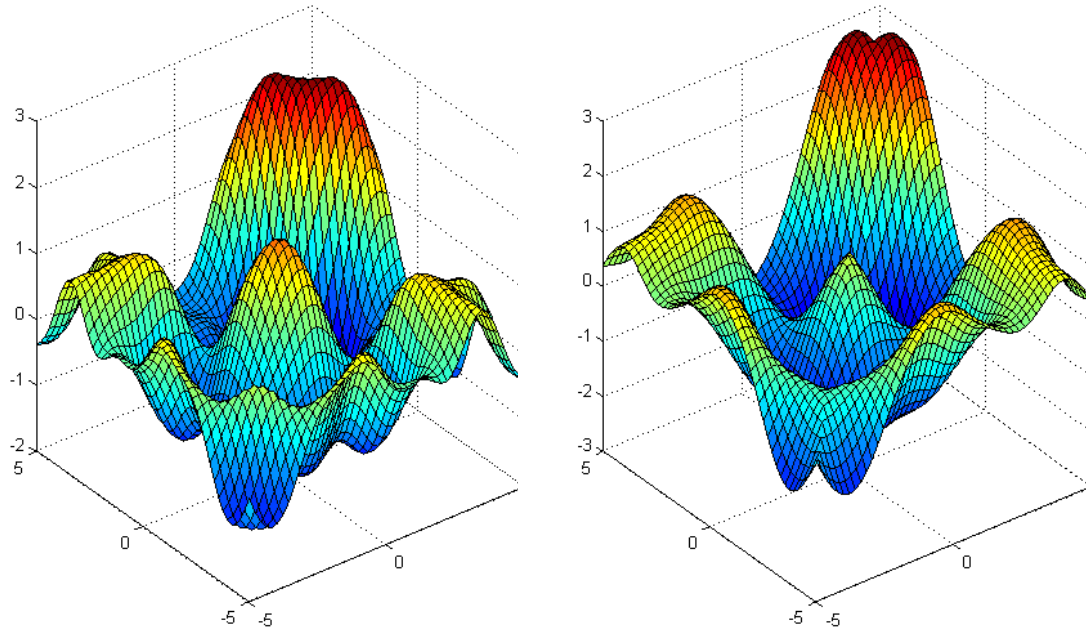
by two methods: In the additive method, we transform the kernel by:

$$k_{\text{reflect add}}(x, y, x', y') = k(x, y, x', y') + k(x, y, y', x') + k(y, x, x', y') + k(y, x, y', x') \tag{2.6}$$

or by

$$k_{\text{reflect min}}(x, y, x', y') = k(\min(x, y), \max(x, y), \min(x', y'), \max(x', y')) \tag{2.7}$$

however, the second method will in general lead to non-differentiability along $x = y$. Figure 2.4 shows the difference.



Additive method min method

Fig. 2.4 An illustration of two methods of introducing symmetry: The additive method or the min method. The additive method has half the marginal variance away from $y = x$, but the min method introduces a non-differentiable seam along $y = x$.

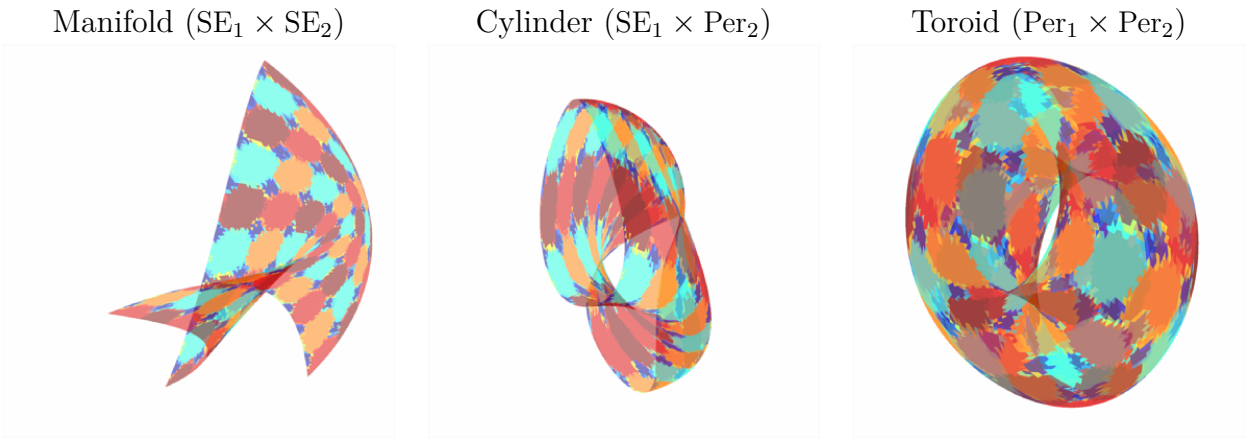


Fig. 2.5 Generating 2D manifolds with different topological structures. By enforcing that the functions mapping from $\mathbb{R}^2$ to $\mathbb{R}^3$ obey the appropriate symmetries, the surfaces created have the corresponding topologies, ignoring self-intersections.

2.3.1 Parametric embeddings

In general, we can always enforce the symmetries obeyed by a given surface by finding a parametric embedding to that surface. However, it is not clear how to do this in general without introducing unnecessary

2.4 How to generate 3D shapes with a given topology

First create a mesh in 2d. Then draw 3 independent functions from a GP prior with the relevant symmetries encoded in the kernel. Then, map the 2d points making up the mesh through those 3 functions to get the 3D coordinates of each point on the mesh.

This is similar in spirit to the GP-LVM model **?**, which learns an embedding of the data into a low-dimensional space, and constructs a fixed kernel structure over that space.

Function on Möbius strip Möbius manifold Sudanese Möbius strip

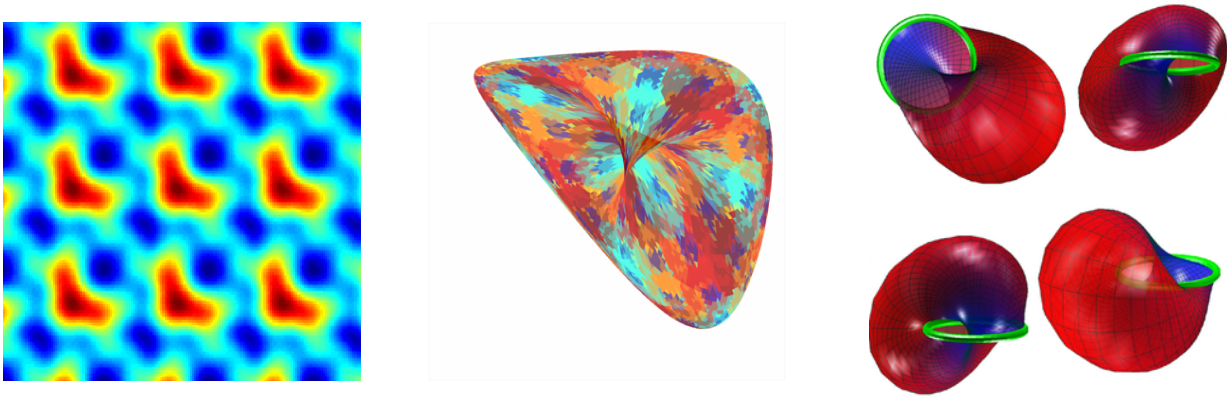


Fig. 2.6 Generating Möbius strips. By enforcing that the functions mapping from $\mathbb{R}^2$ to $\mathbb{R}^3$ obey the appropriate symmetries, the surfaces created have topology corresponding to a Möbius strip. TODO: Talk about Sudanese representation.

2.4.1 Möbius strips

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

$$f(x, y) = f(x, y + \tau_y) \tag{2.8}$$

$$f(x, y) = f(x + \tau_x, y) \tag{2.9}$$

$$f(x, y) = f(y, x) \tag{2.10}$$

If we imagine moving along the edge of a Möbius strip, that is equivalent to moving along a diagonal in the function generated. Figure 2.6 shows this. The second example is doesn't resemble a typical Möbius strip because the edge of the mobius strip is in a geometric circle. This kind of embedding is resembles the Sudanese Möbius strip [cite].

Another classic example of a function living on a Mobius strip is the auditory quality of 2-note intervals. The harmony of a pair of notes is periodic (over octaves) for each note, and the

2.5 Examples

2.5.1 Computing molecular energies

Figure 2.7 gives one example of a function which obeys the same symmetries as a Möbius strip, in some subsets of its arguments.

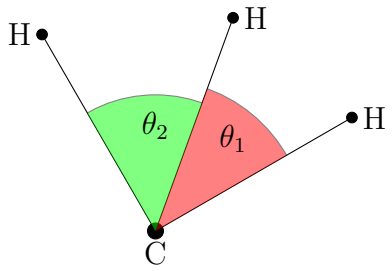


Fig. 2.7 An example of a function expressing the same symmetries as a Möbius strip in two of its arguments. The energy of a molecular configuration $f(\theta_1, \theta_2)$ depends only on the relative angles between atoms, and because each atom is indistiguishable, is invariant to permuting the atoms.

2.5.2 Translation invariance in images

Most models of images are invariant to spatial translations [cite convolution nets]. Similarily, most models of sounds are also invariant to translation through time.

Note that this sort of translational invariance is completely distinct from the stationarity properties of kernels used in Gaussian process priors. A stationary kernel implies that the prior is invariant to translations of the entire training and test set.

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

$$f\left(\quad\right) = f\left(\quad\right) \tag{2.11}$$

Translational invariance in this setting is equivalent to symmetries between dimensions in the input space.

This prior can be achieved in one dimension by using the following kernel transformation:

$$k\left((x_1, x_2, \ldots, x_D), (x'_1, x'_2, \ldots, x'_D)\right) = \sum_{i=1}^{D} \prod_{j=1}^{D} k(x_j, x'_{i+j \bmod D}) \tag{2.12}$$

Edge effects can be handled either by wrapping the image around, or by padding it with zeros.

Convolution The resulting kernel could be called a *discrete convolution kernel*. For an image with R, C rows and columns, it can also be written as:

$$k_{\text{conv}}\left((x_{11}, x_{12}, \ldots, x_{RC}), (x'_{11}, x'_{12}, \ldots, x'_{RC})\right) = \sum_{i=-L}^{L} \sum_{j=-L}^{L} k\left(\mathbf{x}, T_{ij}(\mathbf{x}')\right) \tag{2.13}$$

where $T_{ij}(\mathbf{x})$ is the operator which replaces each x_{mn} with $x_{m+i,n+j}$. Thus we are simply defining the covariance between two images to be the sum of all covariances between all relative translations of the two images. We can also normalize the kernel by pre-multiplying it with $\sqrt{k_{\text{conv}}(\mathbf{x}, \mathbf{x}) k_{\text{conv}}(\mathbf{x}', \mathbf{x}')}$.

Is there a pathology of the additive construction that appears in the limit?

2.5.3 Max-pooling

What we'd really like to do is a max-pooling operation. However, in general, a kernel which is the max of other kernels is not PSD [put counterexample here?]. Is the max over co-ordinate switching PSD?

2.6 Related Work

Invariances in Gaussian processes **?** show that, for Gaussian processes, with probability one, $f(\mathbf{x}) = f(T(\mathbf{x}))$ if and only if $k(x, x') = k(x, T(x'))$.

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

2.7 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:

$$k_1(\mathbf{x}, \mathbf{x}') = \mathbf{h}_1(\mathbf{x})^\mathsf{T}\mathbf{h}_1(\mathbf{x}') \tag{2.14}$$

$$k_2(\mathbf{x}, \mathbf{x}') = \mathbf{h}_2(\mathbf{x})^\mathsf{T}\mathbf{h}_2(\mathbf{x}') \tag{2.15}$$

$$(k_1 \circ k_2)(\mathbf{x}, \mathbf{x}') = k_2(\mathbf{h}_1(\mathbf{x}), \mathbf{h}_1(\mathbf{x}')) \tag{2.16}$$

$$= [\mathbf{h}_2(\mathbf{h}_1(\mathbf{x}))]^\mathsf{T}\mathbf{h}_2(\mathbf{h}_1(\mathbf{x}')) \tag{2.17}$$

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

$$\begin{aligned} k_{L+1}(\mathbf{x}, \mathbf{x}') &= k_{SE}(\mathbf{h}(\mathbf{x}), \mathbf{h}(\mathbf{x}')) = \\ &= \exp\left(-\frac{1}{2}||\mathbf{h}(\mathbf{x}) - \mathbf{h}(\mathbf{x}')||_2^2\right) \\ &= \exp\left(-\frac{1}{2}\left[\mathbf{h}(\mathbf{x})^\mathsf{T}\mathbf{h}(\mathbf{x}) - 2\mathbf{h}(\mathbf{x})^\mathsf{T}\mathbf{h}(\mathbf{x}') + \mathbf{h}(\mathbf{x}')^\mathsf{T}\mathbf{h}(\mathbf{x}')\right]\right) \\ &= \exp\left(-\frac{1}{2}\left[k_L(\mathbf{x}, \mathbf{x}) - 2k_L(\mathbf{x}, \mathbf{x}') + k_L(\mathbf{x}', \mathbf{x}')\right]\right) \end{aligned} \tag{2.18}$$

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

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

$$\begin{aligned} k_{L+1}(\mathbf{x}, \mathbf{x}') &= \exp\left(-\frac{1}{2}\left|\left|\begin{bmatrix}\mathbf{h}_L(\mathbf{x}) \\ \mathbf{x}\end{bmatrix} - \begin{bmatrix}\mathbf{h}_L(\mathbf{x}') \\ \mathbf{x}'\end{bmatrix}\right|\right|_2^2\right) \\ &= \exp\left(-\frac{1}{2}\left[k_L(\mathbf{x}, \mathbf{x}) - 2k_L(\mathbf{x}, \mathbf{x}') + k_L(\mathbf{x}', \mathbf{x}') - ||\mathbf{x} - \mathbf{x}'||_2^2\right]\right) \end{aligned} \tag{2.19}$$

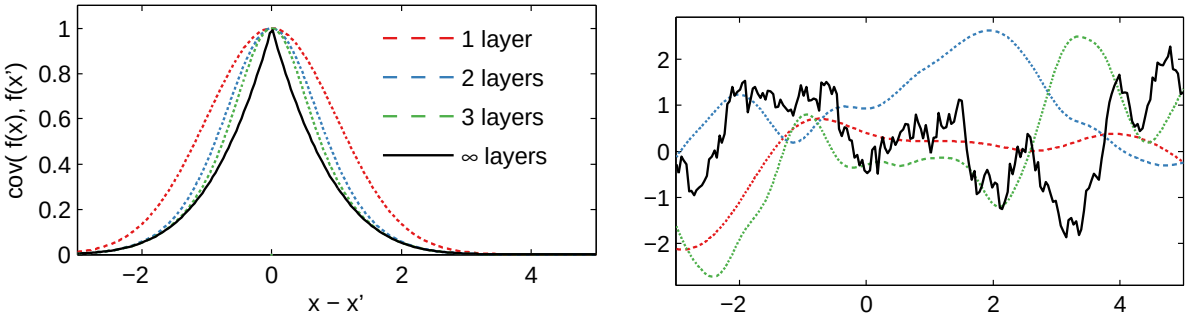


Fig. 2.8 Left: Input-connected deep kernels. By connecting the inputs $\mathbf{x}$ to each layer, the kernel can still depend on its input even after arbitrarily many layers of computation. Right: GP draws using deep input-connected kernels.

For the SE kernel, this repeated mapping satisfies

$$k_\infty(\mathbf{x}, \mathbf{x}') - \log\left(k_\infty(\mathbf{x}, \mathbf{x}')\right) = 1 + \frac{1}{2}||\mathbf{x} - \mathbf{x}'||_2^2 \tag{2.20}$$

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

2.7.1 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 translation and rotation invariance in images (**?**). **?** used a deep neural network to learn feature transforms for kernels, which learn invariances in an unsupervised manner. The relatively uninteresting properties of the kernels derived in this section simply reflect the fact that an arbitrary deep computation is not usually a useful representation, unless combined with learning.

Chapter 3

Automatically Building Structured Covariance Functions

> "It would be very nice to have a formal apparatus that gives us some 'optimal' way of recognizing unusual phenomena and inventing new classes of hypotheses that are most likely to contain the true one; but this remains an art for the creative human mind."
>
> ?

Joint work with James Robert Lloyd, Roger Grosse, Joshua B. Tenenbaum, Zoubin Ghahramani

Despite its importance, choosing the structural form of the kernel in nonparametric regression remains a black art. We define a space of kernel structures which are built compositionally by adding and multiplying a small number of base kernels. We present a method for searching over this space of structures which mirrors the scientific discovery process. The learned structures can often decompose functions into interpretable components and enable long-range extrapolation on time-series datasets. Our structure search method outperforms many widely used kernels and kernel combination methods on a variety of prediction tasks.

3.1 Introduction

Kernel-based nonparametric models, such as support vector machines and Gaussian processes (GPs), have been one of the dominant paradigms for supervised machine learning over the last 20 years. These methods depend on defining a kernel function, $k(x, x')$,

which specifies how similar or correlated outputs y and y' are expected to be at two inputs x and x'. By defining the measure of similarity between inputs, the kernel determines the pattern of inductive generalization.

Most existing techniques pose kernel learning as a (possibly high-dimensional) parameter estimation problem. Examples include learning hyperparameters (**?**), linear combinations of fixed kernels **?**, and mappings from the input space to an embedding space **?**.

However, to apply existing kernel learning algorithms, the user must specify the parametric form of the kernel, and this can require considerable expertise, as well as trial and error.

To make kernel learning more generally applicable, we reframe the kernel learning problem as one of structure discovery, and automate the choice of kernel form. In particular, we formulate a space of kernel structures defined compositionally in terms of sums and products of a small number of base kernel structures. This provides an expressive modeling language which concisely captures many widely used techniques for constructing kernels. We focus on Gaussian process regression, where the kernel specifies a covariance function, because the Bayesian framework is a convenient way to formalize structure discovery. Borrowing discrete search techniques which have proved successful in equation discovery **?** and unsupervised learning **?**, we automatically search over this space of kernel structures using marginal likelihood as the search criterion.

We found that our structure discovery algorithm is able to automatically recover known structures from synthetic data as well as plausible structures for a variety of real-world datasets. On a variety of time series datasets, the learned kernels yield decompositions of the unknown function into interpretable components that enable accurate extrapolation beyond the range of the observations. Furthermore, the automatically discovered kernels outperform a variety of widely used kernel classes and kernel combination methods on supervised prediction tasks.

While we focus on Gaussian process regression, we believe our kernel search method can be extended to other supervised learning frameworks such as classification or ordinal regression, or to other kinds of kernel architectures such as kernel SVMs. We hope that the algorithm developed in this paper will help replace the current and often opaque art of kernel engineering with a more transparent science of automated kernel construction.

Example expressions In addition to the examples given in Figure 4.1, many common motifs of supervised learning can be captured using sums and products of one-

dimensional base kernels:

Bayesian linear regression	Lin
Bayesian polynomial regression	Lin $\times$ Lin $\times \ldots$
Generalized Fourier decomposition	Per + Per + $\ldots$
Generalized additive models	$\sum_{d=1}^{D} \mathrm{SE}_d$
Automatic relevance determination	$\prod_{d=1}^{D} \mathrm{SE}_d$
Linear trend with local deviations	Lin + SE
Linearly growing amplitude	Lin $\times$ SE

We use the term 'generalized Fourier decomposition' to express that the periodic functions expressible by a GP with a periodic kernel are not limited to sinusoids.

3.2 Searching over structures

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

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

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

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

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

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

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

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

- One can look for structure, e.g. periodicity, in the residuals of a model, and then extend the model to capture that structure. This corresponds to applying rule (1).

- One can start with structure, e.g. linearity, which is assumed to hold globally, but find that it only holds locally. This corresponds to applying rule (2) to obtain the structure shown in rows 1 and 3 of figure 4.1.

- One can add features incrementally, analogous to algorithms like boosting, backfitting, or forward selection. This corresponds to applying rules (1) or (2) to dimensions not yet included in the model.

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

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

3.3 Related Work

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

methods in high-dimensional spaces can require imposing additional structure on the model.

One such structure is additivity. Generalized additive models (GAM) assume the regression function is a transformed sum of functions defined on the individual dimensions: $\mathbb{E}[f(\mathbf{x})] = g^{-1}(\sum_{d=1}^{D} f_d(x_d))$. These models have a limited compositional form, but one which is interpretable and often generalizes well. In our grammar, we can capture analogous structure through sums of base kernels along different dimensions.

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

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

Semiparametric regression (e.g. **?**) attempts to combine interpretability with flexibility by building a composite model out of an interpretable, parametric part (such as linear regression) and a 'catch-all' nonparametric part (such as a GP with an SE kernel). In our approach, this can be represented as a sum of SE and Lin.

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

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

? derive kernels of the form SE $\times \cos(x - x')$, forming a basis for stationary kernels.

These kernels share similarities with SE × Per but can express negative prior correlation, and could usefully be included in our grammar.

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

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

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

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

3.4 Structure discovery in time series

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

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

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

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

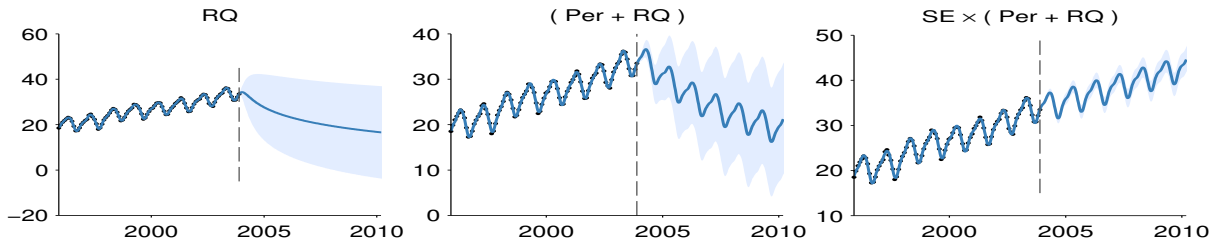


Fig. 3.1 Posterior mean and variance for different depths of kernel search. The dashed line marks the extent of the dataset. In the first column, the function is only modeled as a locally smooth function, and the extrapolation is poor. Next, a periodic component is added, and the extrapolation improves. At depth 3, the kernel can capture most of the relevant structure, and is able to extrapolate reasonably.

Mauna Loa atmospheric CO_2 Using our method, we analyzed records of carbon dioxide levels recorded at the Mauna Loa observatory. Since this dataset was analyzed in detail by **?**, we can compare the kernel chosen by our method to a kernel constructed by human experts.

Figure 3.4 shows the posterior mean and variance on this dataset as the search depth increases. While the data can be smoothly interpolated by a single base kernel model, the extrapolations improve dramatically as the increased search depth allows more structure to be included.

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

Airline passenger data Figure 3.6 shows the decomposition produced by applying our method to monthly totals of international airline passengers (**?**). We observe similar components to the previous dataset: a long term trend, annual periodicity and medium-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.

Solar irradiance Data Finally, we analyzed annual solar irradiation data from 1610 to 2011 (**?**). The posterior and residuals of the learned kernel are shown in figure 3.5. None of the models in our search space are capable of parsimoniously representing the lack of variation from 1645 to 1715. Despite this, our approach fails gracefully: the

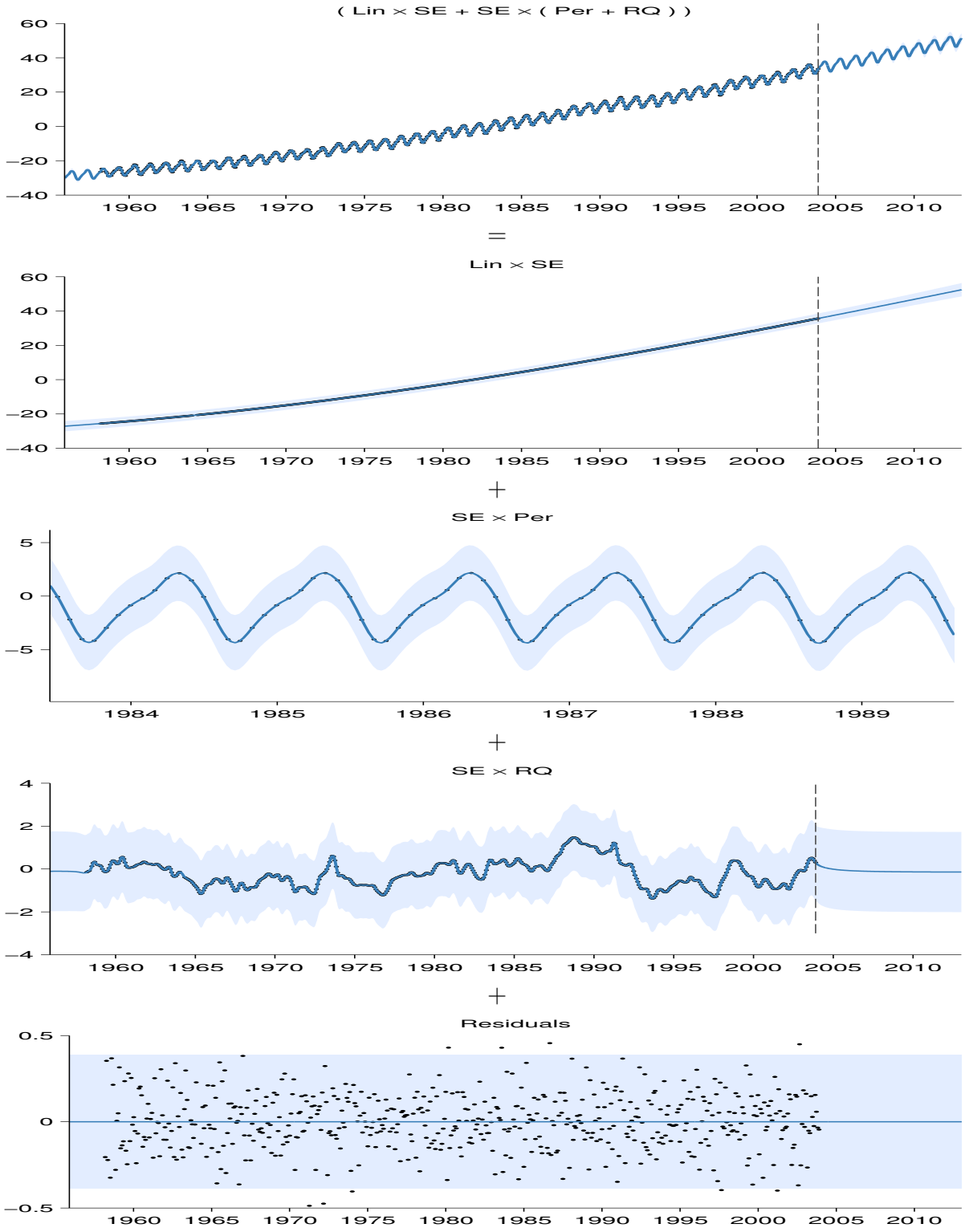


Fig. 3.2 First row: The posterior on the Mauna Loa dataset, after a search of depth 10. Subsequent rows show the automatic decomposition of the time series. The decompositions shows long-term, yearly periodic, medium-term anomaly components, and residuals, respectively. In the third row, the scale has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

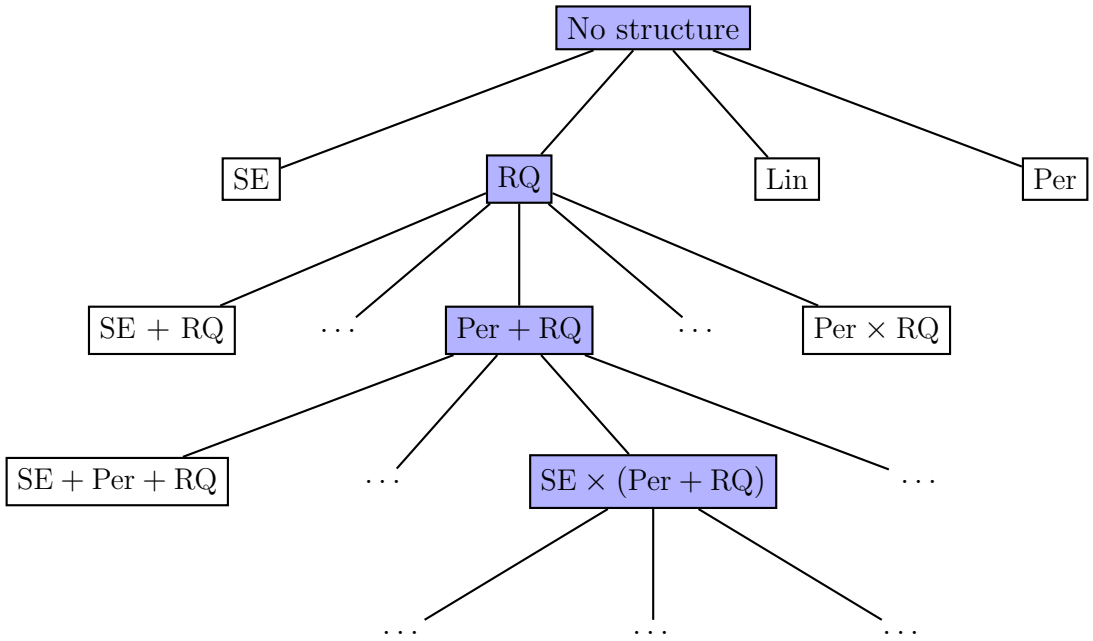


Fig. 3.3 An example of a search tree over kernel expressions. Figure 3.4 shows the model increasing in sophistication as the kernel expression grows.

learned kernel still captures the periodic structure, and the quickly growing posterior variance demonstrates that the model is uncertain about long term structure.

3.5 Validation on synthetic data

Table 3.1 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.

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

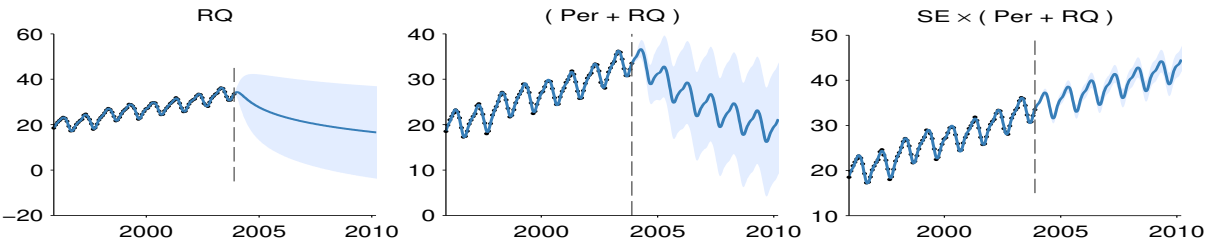


Fig. 3.4 Posterior mean and variance for different depths of kernel search. The dashed line marks the extent of the dataset. In the first column, the function is only modeled as a locally smooth function, and the extrapolation is poor. Next, a periodic component is added, and the extrapolation improves. At depth 3, the kernel can capture most of the relevant structure, and is able to extrapolate reasonably.

We validated our method's ability to recover known structure on a set of synthetic datasets. For several composite kernel expressions, we constructed synthetic data by first sampling 300 points uniformly at random, then sampling function values at those points from a GP prior. We then added i.i.d. Gaussian noise to the functions, at various signal-to-noise ratios (SNR).

Table 3.1 lists the true kernels we used to generate the data. Subscripts indicate which dimension each kernel was applied to. Subsequent columns show the dimensionality D of the input space, and the kernels chosen by our search for different SNRs. Dashes - indicate that no kernel had a higher marginal likelihood than modeling the data as i.i.d. Gaussian noise.

For the highest SNR, the method finds all relevant structure in all but one test. The reported additional linear structure is explainable by the fact that functions sampled from SE kernels with long length scales occasionally have near-linear trends. As the noise increases, our method generally backs off to simpler structures.

3.6 Quantitative evaluation

In addition to the qualitative evaluation in section 3.4, we investigated quantitatively how our method performs on both extrapolation and interpolation tasks.

3.6.1 Extrapolation

We compared the extrapolation capabilities of our model against standard baselines[1]. Dividing the airline dataset into contiguous training and test sets, we computed the

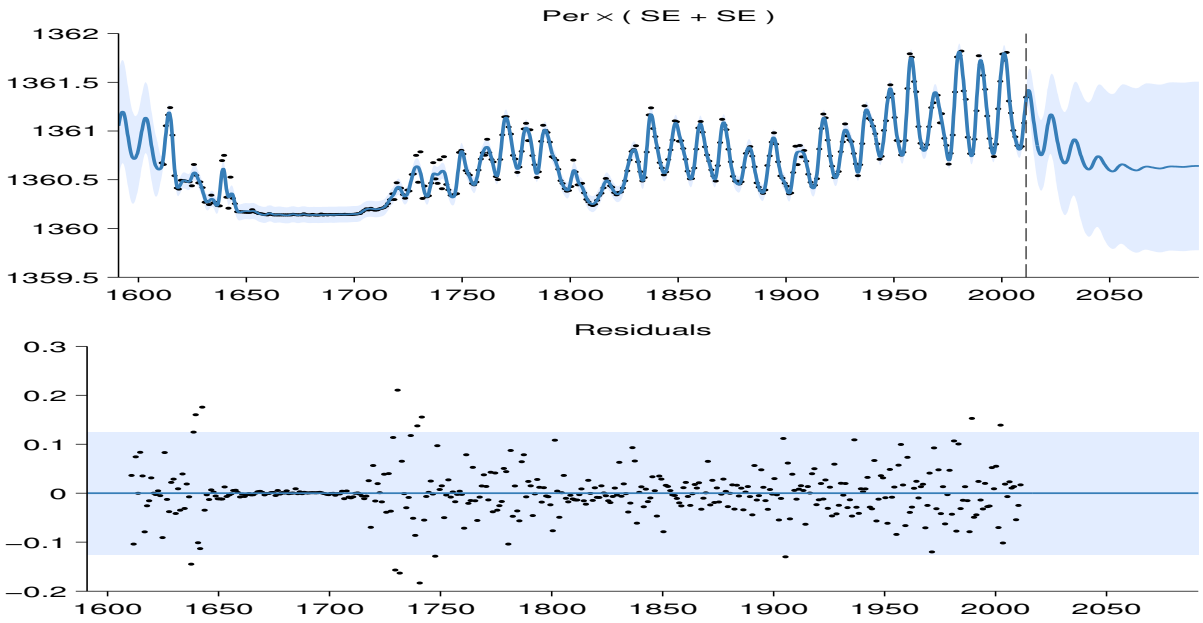


Fig. 3.5 Full posterior and residuals on the solar irradiance dataset.

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

	Mean Squared Error (MSE)					Negative Log-Likelihood				
Method	bach	concrete	puma	servo	housing	bach	concrete	puma	servo	housing
Linear Regression	1.031	0.404	0.641	0.523	0.289	2.430	1.403	1.881	1.678	1.052
GAM	1.259	0.149	0.598	0.281	0.161	1.708	0.467	1.195	0.800	0.457
HKL	**0.199**	0.147	0.346	0.199	0.151	-	-	-	-	-
GP SE-ARD	**0.045**	0.157	**0.317**	**0.126**	**0.092**	**−0.131**	0.398	**0.843**	0.429	0.207
Additive GP	**0.045**	**0.089**	**0.316**	**0.110**	0.102	**−0.131**	**0.114**	**0.841**	0.309	0.194
Search - SE, RQ	**0.044**	**0.087**	**0.315**	**0.102**	**0.082**	**−0.141**	**0.065**	**0.840**	0.265	**0.059**
Search - All kernels	**0.509**	**0.079**	**0.321**	**0.094**	**0.112**	**0.357**	**0.114**	**0.837**	**−0.427**	**0.151**

predictive mean-squared-error (MSE) of each method. We varied the size of the training set from the first 10% to the first 90% of the data.

Figure 3.7 shows the learning curves of linear regression, a variety of fixed kernel family GP models, and our method. GP models with only SE and Per kernels did not capture the long-term trends, since the best parameter values in terms of GP marginal likelihood only capture short term structure. Linear regression approximately captured the long-term trend, but quickly plateaued in predictive performance. The more richly

structured GP models (SE + Per and SE × Per) eventually captured more structure and performed better, but the full structures discovered by our search outperformed the other approaches in terms of predictive performance for all data amounts.

3.6.2 High-dimensional prediction

To evaluate the predictive accuracy of our method in a high-dimensional setting, we extended the comparison of **?** to include our method. We performed 10 fold cross validation on 5 datasets [2] comparing 5 methods in terms of MSE and predictive likelihood. Our structure search was run up to depth 10, using the SE and RQ base kernel families.

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

Results are presented in table 4.2. Our method outperformed the next-best method in each test, although not substantially.

All GP hyperparameter tuning was performed by automated calls to the GPML toolbox[3]; Python code to perform all experiments is available on github[4].

3.7 Discussion

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

We found that the learned structures are often capable of accurate extrapolation in complex time-series datasets, and are competitive with widely used kernel classes and kernel combination methods on a variety of prediction tasks. The learned kernels often yield decompositions of a signal into diverse and interpretable components, enabling

[1] In one dimension, the predictive means of all baseline methods in table 4.2 are identical to that of a GP with an SE kernel.

[2] The data sets had dimensionalities ranging from 4 to 13, and the number of data points ranged from 150 to 450.

[3] Available at `www.gaussianprocess.org/gpml/code/`

[4] `github.com/jamesrobertlloyd/gp-structure-search`

model-checking by humans. We believe that a data-driven approach to choosing kernel structures automatically can help make nonparametric regression and classification methods accessible to non-experts.

3.8 Appendix

Kernel definitions For scalar-valued inputs, the squared exponential (SE), periodic (Per), linear (Lin), and rational quadratic (RQ) kernels are defined as follows:

$$\begin{aligned} k_{\mathrm{SE}}(x,x') &= \sigma^2 \exp\left(-\tfrac{(x-x')^2}{2\ell^2}\right) \\ k_{\mathrm{Per}}(x,x') &= \sigma^2 \exp\left(-\tfrac{2\sin^2(\pi(x-x')/p)}{\ell^2}\right) \\ k_{\mathrm{Lin}}(x,x') &= \sigma_b^2 + \sigma_v^2(x-\ell)(x'-\ell) \\ k_{\mathrm{RQ}}(x,x') &= \sigma^2\left(1+\tfrac{(x-x')^2}{2\alpha\ell^2}\right)^{-\alpha} \end{aligned}$$

Posterior decomposition We can analytically decompose a GP posterior distribution over additive components using the following identity: The conditional distribution of a Gaussian vector $\mathbf{f}_1$ conditioned on its sum with another Gaussian vector $\mathbf{f} = \mathbf{f}_1 + \mathbf{f}_2$ where $\mathbf{f}_1 \sim \mathcal{N}(\mu_1, \mathbf{K}_1)$ and $\mathbf{f}_2 \sim \mathcal{N}(\mu_2, \mathbf{K}_2)$ is given by

$$\begin{aligned} \mathbf{f}_1|\mathbf{f} \sim \mathcal{N}\Big(&\mu_1 + {\mathbf{K}_1}^{\mathsf{T}}(\mathbf{K}_1+\mathbf{K}_2)^{-1}(\mathbf{f}-\mu_1-\mu_2), \\ &\mathbf{K}_1 - {\mathbf{K}_1}^{\mathsf{T}}(\mathbf{K}_1+\mathbf{K}_2)^{-1}\mathbf{K}_1\Big). \end{aligned}$$

3.9 abstract

This paper presents the beginnings of an automatic statistician, focusing on regression problems. Our system explores an open-ended space of possible statistical models to discover a good explanation of the data, and then produces a detailed report with figures and natural-language text.

Our approach treats unknown functions nonparametrically using Gaussian processes, which has two important consequences. First, Gaussian processes model functions in terms of high-level properties (e.g. smoothness, trends, periodicity, changepoints). Taken together with the compositional structure of our language of models, this allows us to automatically describe functions through a decomposition into additive parts. Second, the use of flexible nonparametric models and a rich language for composing them in an

open-ended manner also results in state-of-the-art extrapolation performance evaluated over 13 real time series data sets from various domains.

3.10 Introduction

Automating the process of statistical modeling would have a tremendous impact on fields that currently rely on expert statisticians, machine learning researchers, and data scientists. While fitting simple models (such as linear regression) is largely automated by standard software packages, there has been little work on the automatic construction of flexible but interpretable models. What are the ingredients required for an artificial intelligence system to be able to perform statistical modeling automatically? In this paper we conjecture that the following ingredients may be useful for building an AI system for statistics, and we develop a working system which incorporates them:

- **An open-ended language of models** expressive enough to capture many of the modeling assumptions and model composition techniques applied by human statisticians to capture real-world phenomena

- **A search procedure** to efficiently explore the space of models spanned by the language

- **A principled method for evaluating models** in terms of their complexity and their degree of fit to the data

- **A procedure for automatically generating reports** which explain and visualize different factors underlying the data, make the chosen modeling assumptions explicit, and quantify how each component improves the predictive power of the model

In this paper, we introduce a system for modeling time-series data containing the above ingredients which we call the Automatic Bayesian Covariance Discovery (ABCD) system. The system defines an open-ended language of Gaussian process models via a compositional grammar. The space is searched greedily, using marginal likelihood and the Bayesian Information Criterion (BIC) to evaluate models. 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.

We show examples of automatically generated reports which highlight interpretable features discovered in a variety of data sets (e.g. figure 3.8). The supplementary material to this paper includes 13 complete reports automatically generated by ABCD.

Good statistical modeling requires not only interpretability but predictive accuracy. We compare ABCD against existing model construction techniques in terms of predictive performance at extrapolation, and we find state-of-the-art performance on 13 time series. In the remainder of this paper we describe the components of ABCD in detail.

3.11 A language of regression models

The general problem of regression consists of learning a function f mapping from some input space $\mathcal{X}$ to some output space $\mathcal{Y}$. We would like an expressive language which can represent both simple parametric forms of f such as linear, polynomial, etc. and also complex nonparametric functions specified in terms of properties such as smoothness, periodicity, etc. Fortunately, Gaussian processes (GPs) provide a very general and analytically tractable way of capturing both simple and complex functions.

Gaussian processes are distributions over functions such that any finite subset of function evaluations, $(f(x_1), f(x_2), \ldots f(x_N))$, have a joint Gaussian distribution (**?**). A GP is completely specified by its mean function, $\mu(x) = \mathbb{E}(f(x))$ and kernel (or covariance) function $k(x, x') = \mathrm{Cov}(f(x), f(x'))$. It is common practice to assume zero mean, since marginalizing over an unknown mean function can be equivalently expressed as a zero-mean GP with a new kernel. The structure of the kernel captures high-level properties of the unknown function, f, which in turn determines how the model generalizes or extrapolates to new data. We can therefore define a language of regression models by specifying a language of kernels.

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

tively[5]. The composition rules are addition and multiplication:

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

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

Combining kernels using these operations can yield kernels encoding for richer structures such as approximate periodicity (SE $\times$ Per) or smooth functions with linear trends (SE + Lin).

This kernel composition framework (with different base kernels) was described by **?**. We extend and adapt this framework in several ways. In particular, we have found that incorporating changepoints into the language is essential for realistic models of time series (e.g. figure 3.8). Changepoints can be defined through addition and multiplication with sigmoidal functions:

$$\mathrm{CP}(k_1, k_2) = k_1 \times \boldsymbol{\sigma} + k_2 \times \bar{\boldsymbol{\sigma}} \tag{3.3}$$

where $\boldsymbol{\sigma} = \sigma(x)\sigma(x')$ and $\bar{\boldsymbol{\sigma}} = (1 - \sigma(x))(1 - \sigma(x'))$. Changewindows $\mathrm{CW}(\cdot, \cdot)$ can be defined similarly by replacing $\sigma(x)$ with a product of two sigmoids.

We also expanded and reparametrised the set of base kernels so that they were more amenable to automatic description and to extend the number of common regression models included in the language. Table 3.3 lists common regression models that can be expressed by our language.

Regression model	Kernel
GP smoothing	SE + WN
Linear regression	C + Lin + WN
Multiple kernel learning	$\sum$ SE + WN
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
Heteroscedasticity	e.g. SE + Lin $\times$ WN

Table 3.3 Common regression models expressible in our language. cos is a special case of our reparametrised Per.

[5]Definitions of kernels are in the supplementary material.

3.12 Model Search and Evaluation

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

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

$$\mathrm{BIC}(M) = -2 \log p(D \mid M) + p \log n \tag{3.4}$$

where p is the number of kernel parameters, $\log p(D|M)$ is the marginal likelihood of the data, D, and n is the number of data points. BIC trades off model fit and complexity and implements what is known as "Bayesian Occam's Razor" (e.g. **??**).

3.13 Automatic description of regression models

Overview In this section, we describe how ABCD generates natural-language descriptions of the models found by the search procedure. 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.

Sum of products normal form 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:

$$K \prod_m \mathrm{Lin}^{(m)} \prod_n \boldsymbol{\sigma}^{(n)}, \tag{3.5}$$

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.

Sums of kernels are sums of functions Formally, if $f_1(x) \sim \mathrm{GP}(0, k_1)$ and independently $f_2(x) \sim \mathrm{GP}(0, k_2)$ then $f_1(x) + f_2(x) \sim \mathrm{GP}(0, k_1 + k_2)$. This lets us describe each product of kernels separately.

Each kernel in a product modifies a model in a consistent way This allows us to describe the contribution of each kernel in a product as an adjective, or more generally as a modifier of a noun. We now describe how each kernel modifies a model and how this can be described in natural language:

- **Multiplication by SE** removes long range correlations from a model since $\mathrm{SE}(x, x')$ decreases monotonically to 0 as $|x - x'|$ increases. This can be described as making an existing model's correlation structure 'local' or 'approximate'.
- **Multiplication by Lin** is equivalent to multiplying the function being modeled by a linear function. If $f(x) \sim \mathrm{GP}(0, k)$, then $xf(x) \sim \mathrm{GP}(0, k \times \mathrm{Lin})$. This causes the standard deviation of the model to vary linearly without affecting the correlation and can be described as e.g. 'with linearly increasing standard deviation'.
- **Multiplication by $\boldsymbol{\sigma}$** is equivalent to multiplying the function being modeled by a sigmoid which means that the function goes to zero before or after some point. This can be described as e.g. 'from [time]' or 'until [time]'.
- **Multiplication by Per** modifies the correlation structure in the same way as multiplying the function by an independent periodic function. Formally, if $f_1(x) \sim \mathrm{GP}(0, k_1)$ and $f_2(x) \sim \mathrm{GP}(0, k_2)$ then

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

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

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

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

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

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

where Per has been selected as the head noun.

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

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

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

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

3.13.1 Worked example

Suppose we start with a kernel of the form

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

This is converted to a sum of products:

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

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]'.

3.14 Example descriptions of time series

We demonstrate the ability of our procedure to discover and describe a variety of patterns on two time series. Full automatically-generated reports for 13 data sets are provided as supplementary material.

3.14.1 Summarizing 400 Years of Solar Activity

We show excerpts from the report automatically generated on annual solar irradiation data from 1610 to 2011 (figure 3.9). This time series has two pertinent features: a 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 3.10 shows the natural-language summaries of the top four components chosen by ABCD. From these short summaries, we can see that our system has identified the Maunder minimum (second component) and 11-year solar cycle (fourth component). These components are visualized in figures 3.11 and 3.8, respectively. The third component corresponds to long-term trends, as visualized in figure 3.12.

3.14.2 Finding heteroscedasticity in air traffic data

Next, we present the analysis generated by our procedure on international airline passenger data (figure 3.13). The model constructed by ABCD has four components: $\text{Lin} + \text{SE} \times \text{Per} \times \text{Lin} + \text{SE} + \text{WN} \times \text{Lin}$, with descriptions given in figure 3.14.

The second component (figure 3.15) 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 3.16).

3.14.3 Comparison to equation learning

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

The learned function for the solar irradiance data is

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

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

The learned function for the airline passenger data is

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

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

3.15 Related work

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

?? and **?** search over a similar space of models as ABCD using genetic algorithms but do not interpret the resulting models. Our procedure is based on the model construction method of **?** which automatically decomposed models but components were interpreted manually and the space of models searched over was smaller than that in this work.

Kernel Learning Sparse spectrum GPs (**?**) approximate the spectral density of a stationary kernel function using delta functions which corresponds to kernels of the form $\sum \cos$. Similarly, **?** introduce spectral mixture kernels which approximate the spectral density using a scale-location mixture of Gaussian distributions corresponding to kernels of the form $\sum \mathrm{SE} \times \cos$. Both demonstrate, using Bochner's theorem (**?**), that these kernels can approximate any stationary covariance function. Our language of kernels includes both of these kernel classes (see table 3.3).

There is a large body of work attempting to construct rich kernels through a weighted sum of base kernels called multiple kernel learning (MKL) (e.g. **?**). These approaches find the optimal solution in polynomial time but only if the component kernels and parameters are pre-specified. We compare to a Bayesian variant of MKL in section 3.16 which is expressed as a restriction of our language of kernels.

Equation learning **?**, **?** and **?** learn parametric forms of functions specifying time series, or relations between quantities. In contrast, ABCD learns a parametric form for the covariance, allowing it to model functions without a simple parametric form.

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

Natural-language output To the best of our knowledge, our procedure is the first example of automatic description of nonparametric statistical models. However, systems with natural language output have been built in the areas of video interpretation (**?**) and automated theorem proving (**?**).

3.16 Predictive Accuracy

In addition to our demonstration of the interpretability of ABCD, we compared the predictive accuracy of various model-building algorithms at interpolating and extrapolating time-series. ABCD outperforms the other methods on average.

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

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

We use the default mean absolute error criterion when using Eureqa. All other algorithms can be expressed as restrictions of our modeling language (see table 3.3) so we perform inference using the same search methodology and selection criterion[6] with appropriate restrictions to the language. For MKL, trend-cyclical-irregular and spectral kernels, the greedy search procedure of ABCD corresponds to a forward-selection algorithm. For squared exponential and linear regression the procedure corresponds to marginal likelihood optimisation. More advanced inference methods are typically used for changepoint modeling but we use the same inference method for all algorithms for comparability.

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

Interpretability versus accuracy BIC trades off model fit and complexity by penalizing the number of parameters in a kernel expression. This can result in ABCD favoring kernel expressions with nested products of sums, producing descriptions involving many additive components. While these models have good predictive performance the large number of components can make them less interpretable. We experimented with distributing all products over addition during the search, causing models with many additive components to be more heavily penalized by BIC. We call this procedure ABCD-interpretability, in contrast to the unrestricted version of the search, ABCD-accuracy.

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

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

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

Figure 3.17 shows the standardised RMSEs across algorithms. ABCD-accuracy outperforms ABCD-interpretability but both versions 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 values of 11, 493, 22 and 29 respectively. All of these outliers occurred on a data set with a large discontinuity (see the call centre data in the supplementary material).

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

3.17 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. Our procedure's extrapolation and interpolation performance on time-series are state-of-the-art compared to existing model construction techniques. We believe this procedure has the potential to make powerful statistical model-building techniques accessible to non-experts.

Source Code Source code to perform all experiments is available on github[7].

[7] http://www.github.com/jamesrobertlloyd/gpss-research. All GP parameter optimisation was performed by automated calls to the GPML toolbox available at http://www.gaussianprocess.org/gpml/code/.

3.18 Kernels

3.18.1 Base kernels

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

$$\mathrm{WN}(x, x') = \sigma^2 \delta_{x,x'} \tag{3.6}$$

$$\mathrm{C}(x, x') = \sigma^2 \tag{3.7}$$

$$\mathrm{Lin}(x, x') = \sigma^2 (x - \ell)(x' - \ell) \tag{3.8}$$

$$\mathrm{SE}(x, x') = \sigma^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right) \tag{3.9}$$

$$\mathrm{Per}(x, x') = \sigma^2 \frac{\exp\left(\frac{\cos\frac{2\pi(x-x')}{p}}{\ell^2}\right) - I_0\left(\frac{1}{\ell^2}\right)}{\exp\left(\frac{1}{\ell^2}\right) - I_0\left(\frac{1}{\ell^2}\right)} \tag{3.10}$$

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 parameters of the kernel functions.

3.18.2 Changepoints and changewindows

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

$$\begin{aligned} \mathrm{CP}(k_1, k_2)(x, x') = {} & \sigma(x) k_1(x, x') \sigma(x') \\ & + (1 - \sigma(x)) k_2(x, x') (1 - \sigma(x')) \end{aligned} \tag{3.11}$$

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

$$\mathrm{CP}(k_1, k_2) = \boldsymbol{\sigma} k_1 + \bar{\boldsymbol{\sigma}} k_2 \tag{3.12}$$

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

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

3.18.3 Properties of the periodic kernel

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

$$\mathrm{Per}(x, x') \to \sigma^2 \cos\left(\frac{2\pi(x - x')}{p}\right) \quad \text{as} \quad \ell \to \infty. \tag{3.13}$$

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

3.19 Model construction / search

3.19.1 Overview

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

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

3.19.2 Search operators

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

$$\mathcal{S} \quad \to \quad \mathcal{S} + \mathcal{B} \tag{3.14}$$

$$\mathcal{S} \quad \to \quad \mathcal{S} \times \mathcal{B} \tag{3.15}$$

$$\mathcal{B} \quad \to \quad \mathcal{B}' \tag{3.16}$$

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

To accommodate changepoint/window operators we introduce the following addi-

tional operators

$$\mathcal{S} \rightarrow \mathrm{CP}(\mathcal{S}, \mathcal{S}) \tag{3.17}$$
$$\mathcal{S} \rightarrow \mathrm{CW}(\mathcal{S}, \mathcal{S}) \tag{3.18}$$
$$\mathcal{S} \rightarrow \mathrm{CW}(\mathcal{S}, \mathrm{C}) \tag{3.19}$$
$$\mathcal{S} \rightarrow \mathrm{CW}(\mathrm{C}, \mathcal{S}) \tag{3.20}$$

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

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

$$\mathcal{S} \rightarrow \mathcal{S} \times (\mathcal{B} + \mathrm{C}) \tag{3.21}$$
$$\mathcal{S} \rightarrow \mathcal{B} \tag{3.22}$$
$$\mathcal{S} + \mathcal{S}' \rightarrow \mathcal{S} \tag{3.23}$$
$$\mathcal{S} \times \mathcal{S}' \rightarrow \mathcal{S} \tag{3.24}$$

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

3.20 Predictive accuracy

Interpolation To test the ability of the methods to interpolate, we randomly divided each data set into equal amounts of training data and testing data. We trained each algorithm on the training half of the data, produced predictions for the remaining half and then computed the root mean squared error (RMSE). The values of the RMSEs are then standardised by dividing by the smallest RMSE for each data set i.e. the best performance on each data set will have a value of 1.

Figure 3.18 shows the standardised RMSEs for the different algorithms. The box plots show that all quartiles of the distribution of standardised RMSEs are lower for both versions of ABCD. The median for ABCD-accuracy is 1; it is the best performing algorithm on 7 datasets. The largest outliers of ABCD and spectral kernels are similar in value.

Changepoints performs slightly worse than MKL despite being strictly more general than Changepoints. The introduction of changepoints allows for more structured

models, but it introduces parametric forms into the regression models (i.e. the sigmoids expressing the changepoints). This results in worse interpolations at the locations of the change points, suggesting that a more robust modeling language would require a more flexible class of changepoint shapes or improved inference (e.g. fully Bayesian inference over the location and shape of the changepoint).

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

3.20.1 Tabels of standardised RMSEs

See table 3.4 for raw interpolation results and table 3.5 for raw extrapolation results. The rows follow the order of the datasets in the rest of the supplementary material. The following abbreviations are used: ABCD-accuracy (ABCD-acc), ABCD-interpretability ((ABCD-int), Spectral kernels (SP), Trend-cyclical-irregular (TCI), Bayesian MKL (MKL), Eureqa (EL), Changepoints (CP), Squared exponential (SE) and Linear regression (Lin).

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

Table 3.4 Interpolation standardised RMSEs

3.21 Guide to the automatically generated reports

Additional supplementary material to this paper is 13 reports automatically generated by ABCD. A link to these reports will be maintained at http://mlg.eng.cam.ac.uk/lloyd/.

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

Table 3.5 Extrapolation standardised RMSEs

We recommend that you read the report for '01-airline' first and review the reports that follow afterwards more briefly. '02-solar' is discussed in the main text. '03-mauna' analyses a dataset mentioned in the related work. '04-wheat' demonstrates changepoints being used to capture heteroscedasticity. '05-temperature' extracts an exactly periodic pattern from noisy data. '07-call-centre' demonstrates a large discontinuity being modeled by a changepoint. '10-sulphuric' combines many changepoints to create a highly structured model of the data. '12-births' discovers multiple periodic components.

3.22 Ingredients of an automatic statistician

? asks "How can an artificial intelligence do statistics? ... It needs not just an inference engine, but also a way to construct new models and a way to check models. Currently, those steps are performed by humans, but the AI would have to do it itself."

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

1. An open-ended language of models Many statistical procedures consider all models in a class of fixed size - for example, graphical model construction algorithms [(1)] search over connectivity graphs for a given set of nodes. While these methods can be powerful, human statisticians are capable of deriving novel model classes when required by the modelling task. An automatic search through an open-ended class of models can

achieve some of this flexibility, growing the complexity of the model to fit the task at hand, and possibly combining existing structures in novel ways.

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

3. Model comparison and checking model fit (2) An automatic statistician should be able to question the models it has constructed, and formal procedures from model checking provide a way for it to do this. **?** review the literature on model checking. (1) In this work, we use approximate marginal likelihood to compare models, penalizing complexity using the Bayesian Information Criterion as a heuristic.

(2) JL: Two section haps - or are they pletely intertwined?

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

In this work, we demonstrate that the properties of Gaussian processes allow for a modular description generation procedure. Whether or not such modularity and interpretability is present in other open-ended model classes is an open question.

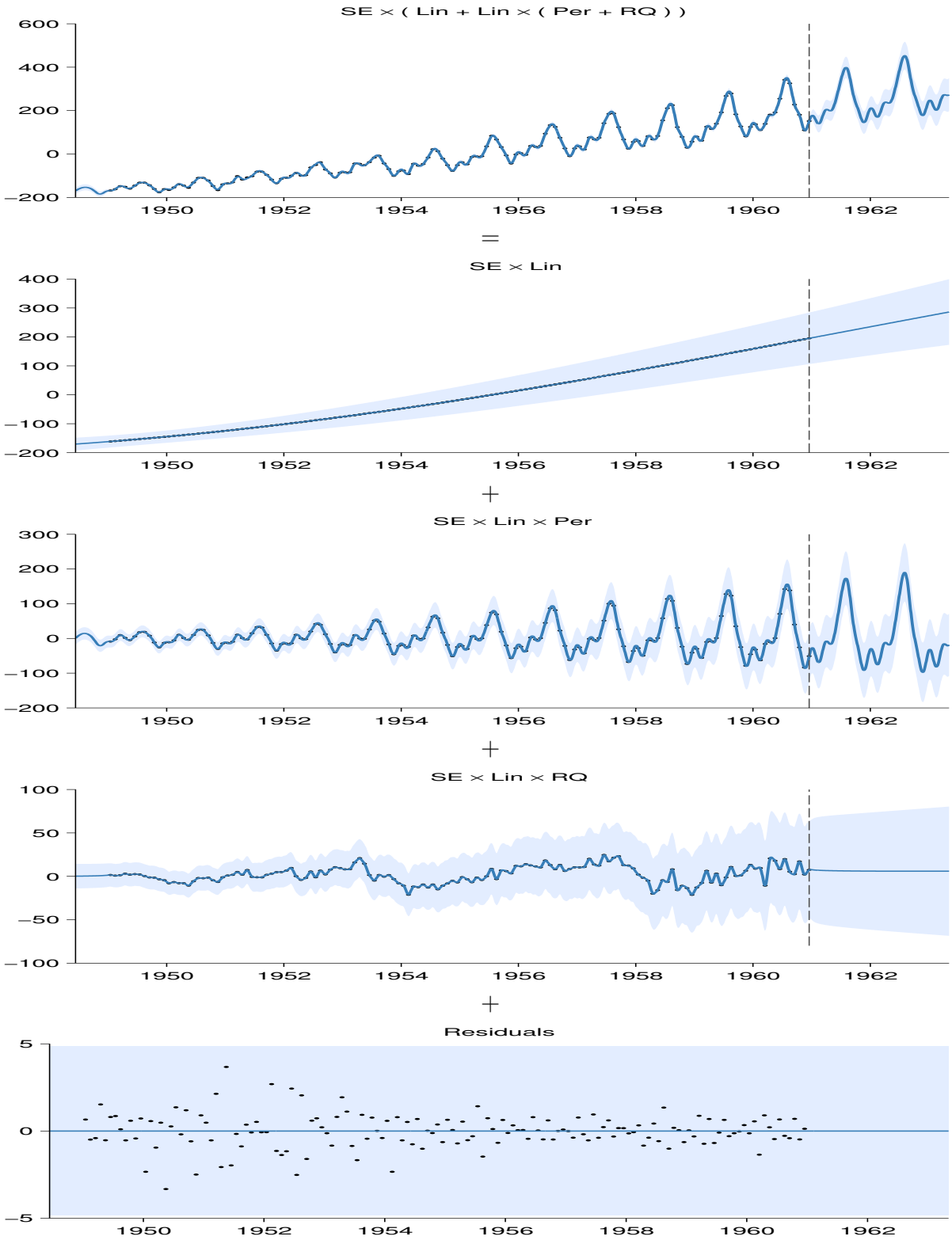


Fig. 3.6 First row: The airline dataset and posterior after a search of depth 10. Subsequent rows: Additive decomposition of posterior into long-term smooth trend, yearly variation, and short-term deviations. Due to the linear kernel, the marginal variance grows over time, making this a heteroskedastic model.

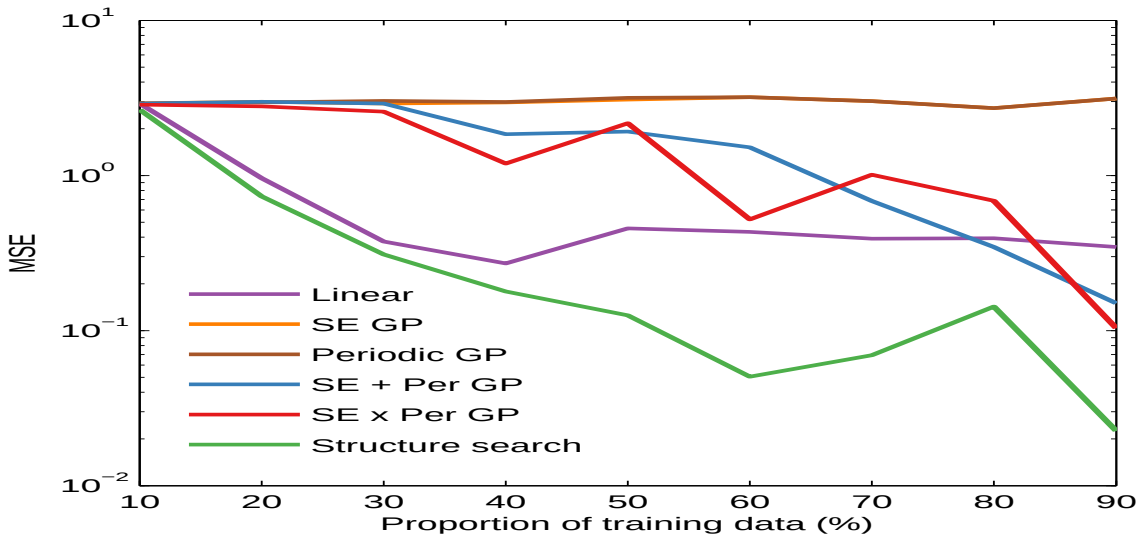


Fig. 3.7 Extrapolation performance on the airline dataset. We plot test-set MSE as a function of the fraction of the dataset used for training.

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.

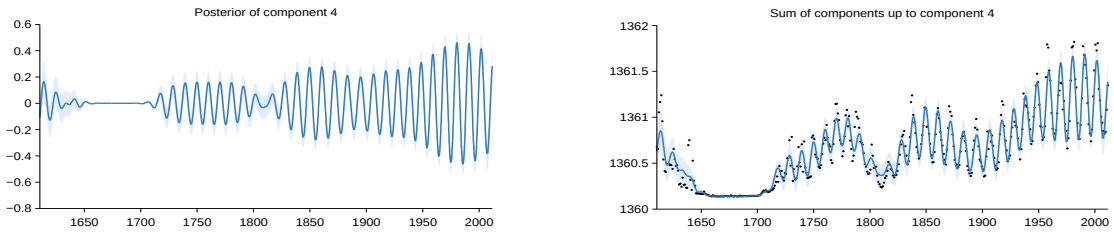


Figure 8: Pointwise posterior of component 4 (left) and the posterior of the cumulative sum of components with data (right)

Fig. 3.8 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 (**?**).

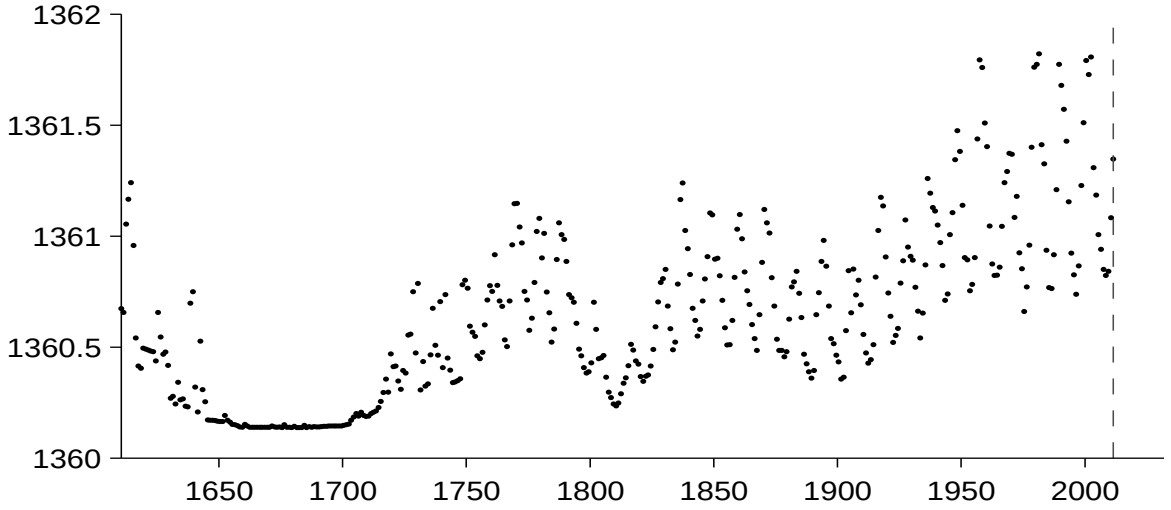


Fig. 3.9 Solar irradiance data.

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

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

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

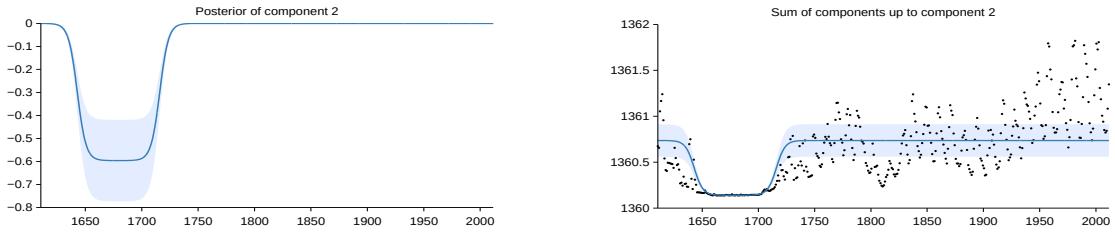


Figure 4: Pointwise posterior of component 2 (left) and the posterior of the cumulative sum of components with data (right)

Fig. 3.11 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.

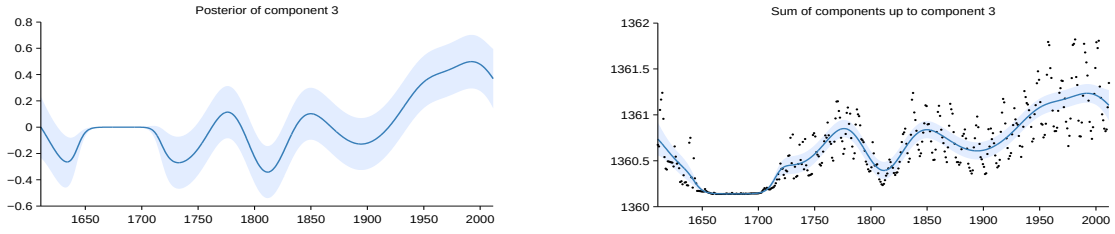


Figure 6: Pointwise posterior of component 3 (left) and the posterior of the cumulative sum of components with data (right)

Fig. 3.12 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.

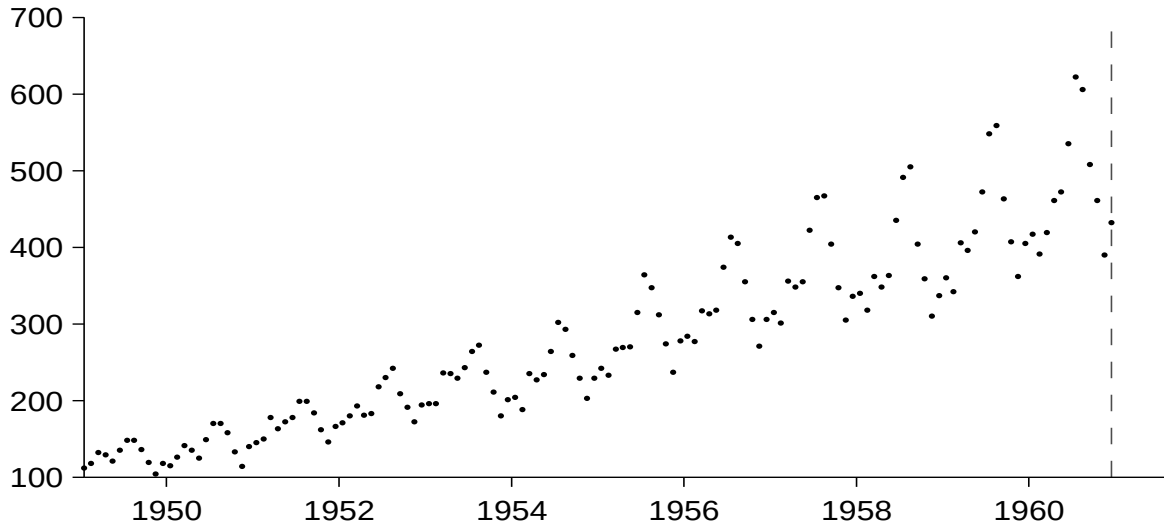


Fig. 3.13 International airline passenger monthly volume (e.g. **?**).

The structure search algorithm has identified four additive components in the data. The first 2 additive components explain 98.5% of the variation in the data as shown by the coefficient of determination (R^2) values in table 1. The first 3 additive components explain 99.8% of the variation in the data. After the first 3 components the cross validated mean absolute error (MAE) does not decrease by more than 0.1%. This suggests that subsequent terms are modelling very short term trends, uncorrelated noise or are artefacts of the model or search procedure. Short summaries of the additive components are as follows:

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

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

Fig. 3.14 Short descriptions and summary statistics for the four components of the airline model.

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.

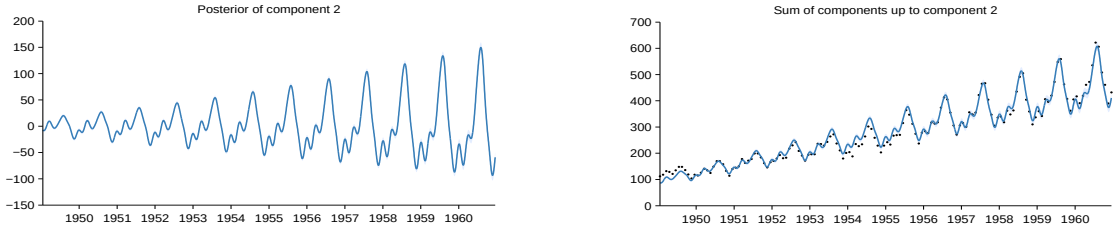


Figure 4: Pointwise posterior of component 2 (left) and the posterior of the cumulative sum of components with data (right)

Fig. 3.15 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.

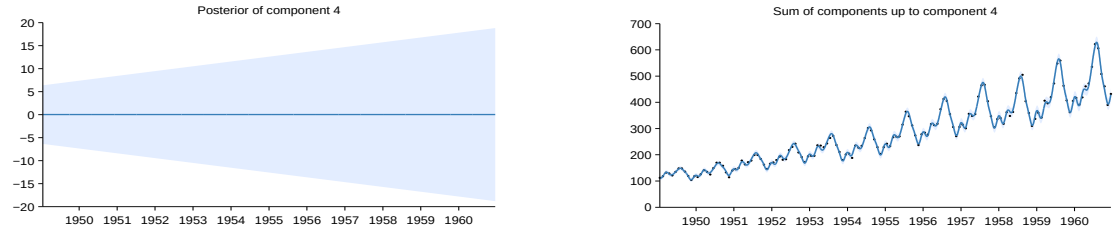


Figure 8: Pointwise posterior of component 4 (left) and the posterior of the cumulative sum of components with data (right)

Fig. 3.16 Modeling heteroscedasticity in the airline dataset.

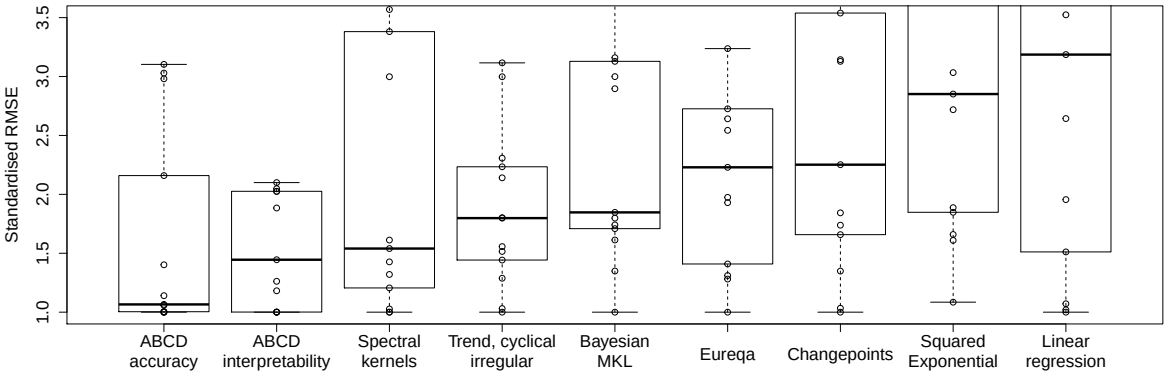


Fig. 3.17 Raw data, and box plot (showing median and quartiles) of standardised extrapolation RMSE (best performance = 1) on 13 time-series. The methods are ordered by median.

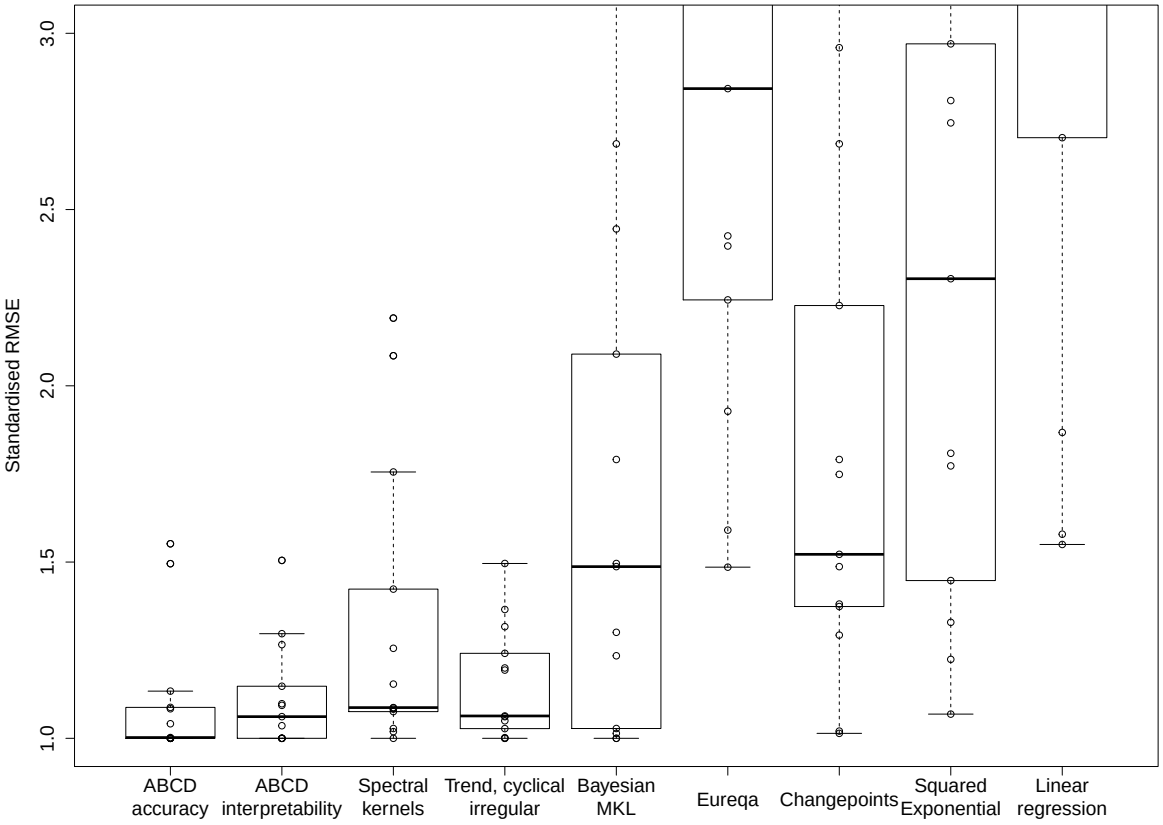


Fig. 3.18 Box plot of standardised RMSE (best performance = 1) on 13 interpolation tasks.

Chapter 4

Dropout in Gaussian processes

4.1 Dropout in Gaussian processes

Dropout is a method for regularizing neural networks (**??**). Training with dropout entails randomly and independently "dropping" (setting to zero) some proportion p of features or inputs, in order to improve the robustness of the resulting network by reducing co-dependence between neurons. To maintain similar overall activation levels, weights are multiplied by $1/p$ at test time. Alternatively, feature activations are multiplied by $1/p$ during training. At test time, the set of models defined by all possible ways of dropping-out neurons is averaged over, usually in an approximate way.

? and **?** analyzed dropout in terms of the effective prior induced by this procedure in several models, such as linear and logistic regression. In this section, we examine the priors on functions that result from performing dropout in the one-hidden-layer neural network implicitly defined by a GP (equation (7.4)).

4.1.1 Dropout on feature activations

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

$$r_i \overset{\text{iid}}{\sim} \text{Ber}(p) \qquad \mathbb{E}[r_i\alpha_i] = p\mu, \mathbb{V}[r_i\alpha_i] = p^2\sigma^2 \; . \tag{4.1}$$

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

$$\mathbb{E}\left[\frac{1}{p}r_i\alpha_i\right] = \frac{p}{p}\mu = \mu, \quad \mathbb{V}\left[\frac{1}{p}r_i\alpha_i\right] = \frac{p^2}{p^2}\sigma^2 = \sigma^2 \,. \tag{4.2}$$

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

4.1.2 Dropping out inputs

In a GP with kernel $k(\mathbf{x}, \mathbf{x}') = \prod_{d=1}^{D} k_d(\mathbf{x}_d, \mathbf{x}'_d)$, 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:

$$p\left(f(\mathbf{x})\right) = \frac{1}{2^D} \sum_{\mathbf{r}\in\{0,1\}^D} \mathrm{GP}\left(0, \prod_{d=1}^{D} k_d(\mathbf{x}_d, \mathbf{x}'_d)^{r_d}\right) \tag{4.3}$$

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

First, if the kernel on each dimension has the form $k_d(\mathbf{x}_d, \mathbf{x}'_d) = g\left(\frac{\mathbf{x}_d - \mathbf{x}'_d}{w_d}\right)$, as does the SE kernel (7.10), 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 (4.3) has the same covariance as

$$f(\mathbf{x}) \sim \mathrm{GP}\left(0, \frac{1}{2^{-2D}} \sum_{\mathbf{r}\in\{0,1\}^D} \prod_{d=1}^{D} k_d(\mathbf{x}_d, \mathbf{x}'_d)^{r_d}\right) \tag{4.4}$$

A GP whose covariance is a sum of kernels corresponds to a sum of functions, each distributed according to a GP. Therefore, (4.4) describes a sum of 2^D functions, each depending on a different subset of the inputs. This model class was studied by **?**, who showed that exact inference in these models can be performed in $\mathcal{O}(N^2D^2 + N^3)$.

In this chapter, we introduce a Gaussian process model of functions which are *additive*. An additive function is one which decomposes into a sum of low-dimensional functions, each depending on only a subset of the input variables. Additive GPs generalize both Generalized Additive Models, and the standard GP models which use squared-

exponential kernels. Hyperparameter learning in this model can be seen as Bayesian Hierarchical Kernel Learning (HKL). We introduce an expressive but tractable parameterization of the kernel function, which allows efficient evaluation of all input interaction terms, whose number is exponential in the input dimension. The additional structure discoverable by this model results in state-of-the-art predictive power in regression tasks, as well as increased interpretability.

4.2 Introduction

Most statistical regression models in use today are of the form: $g(y) = f(x_1) + f(x_2) + \cdots + f(x_D)$. Popular examples include logistic regression, linear regression, and Generalized Linear Models**?**. This family of functions, known as Generalized Additive Models (GAM)**?**, are typically easy to fit and interpret. Some extensions of this family, such as smoothing-splines ANOVA **?**, add terms depending on more than one variable. However, such models generally become intractable and difficult to fit as the number of terms increases.

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

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

We note that a similar breakthrough has recently been made, called Hierarchical Kernel Learning (HKL)**?**. HKL explores a similar class of models, and sidesteps the

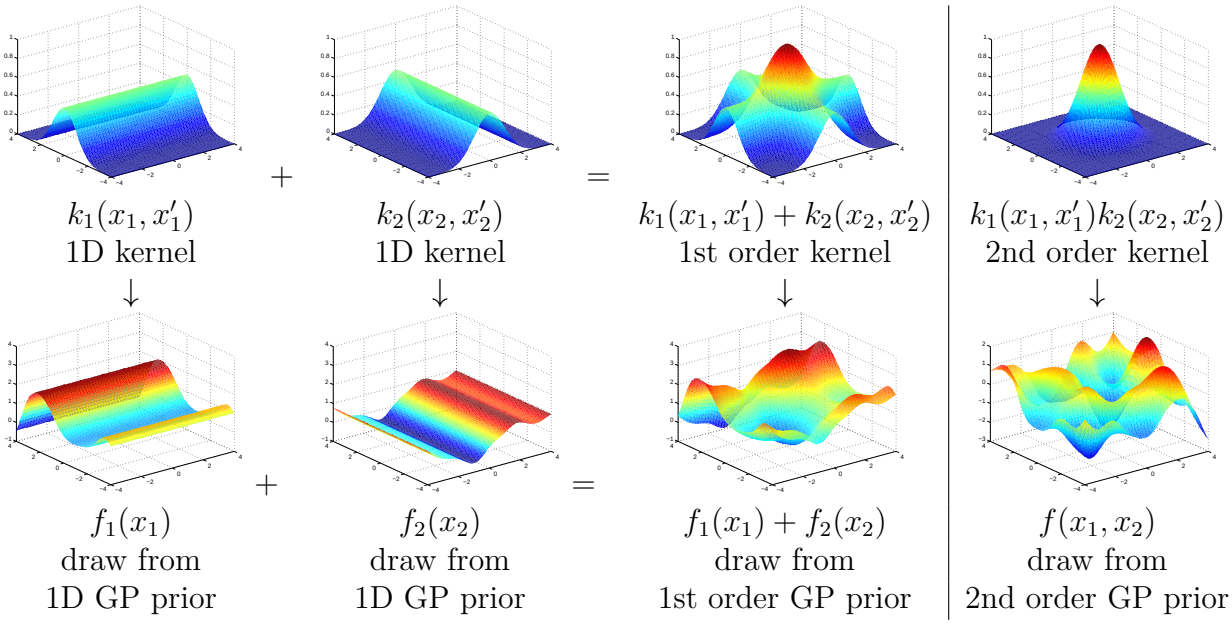


Fig. 4.1 A first-order additive kernel, and a product kernel. Left: a draw from a first-order additive kernel corresponds to a sum of draws from one-dimensional kernels. Right: functions drawn from a product kernel prior have weaker long-range dependencies, and less long-range structure.

possibly exponential number of interaction terms by cleverly selecting only a tractable subset. However, this method suffers considerably from the fact that cross-validation must be used to set hyperparameters. In addition, the machinery necessary to train these models is immense. Finally, on real datasets, HKL is outperformed by the standard SE-GP **?**.

4.3 Gaussian Process Models

Gaussian processes are a flexible and tractable prior over functions, useful for solving regression and classification tasks**?**. The kind of structure which can be captured by a GP model is mainly determined by its *kernel*: the covariance function. One of the main difficulties in specifying a Gaussian process model is in choosing a kernel which can represent the structure present in the data. For small to medium-sized datasets, the kernel has a large impact on modeling efficacy.

Figure 4.1 compares, for two-dimensional functions, a first-order additive kernel with a second-order kernel. We can see that a GP with a first-order additive kernel is an example of a GAM: Each function drawn from this model is a sum of orthogonal one-

dimensional functions. Compared to functions drawn from the higher-order GP, draws from the first-order GP have more long-range structure.

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

4.4 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:

$$k_{add_1}(\mathbf{x}, \mathbf{x}') = \sigma_1^2 \sum_{i=1}^{D} k_i(x_i, x_i') \tag{4.5}$$

$$k_{add_2}(\mathbf{x}, \mathbf{x}') = \sigma_2^2 \sum_{i=1}^{D} \sum_{j=i+1}^{D} k_i(x_i, x_i') k_j(x_j, x_j') \tag{4.6}$$

$$k_{add_n}(\mathbf{x}, \mathbf{x}') = \sigma_n^2 \sum_{1 \le i_1 < i_2 < \ldots < i_n \le D} \left[\prod_{d=1}^{n} k_{i_d}(x_{i_d}, x_{i_d}') \right] \tag{4.7}$$

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:

$$k_{add_D}(\mathbf{x}, \mathbf{x}') = \sigma_D^2 \prod_{d=1}^{D} k_d(x_d, x_d') \tag{4.8}$$

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:

$$k_{add_D}(\mathbf{x}, \mathbf{x}') = \sigma_D^2 \prod_{d=1}^{D} k_d(x_d, x_d') = \sigma_D^2 \prod_{d=1}^{D} \exp\left(-\frac{(x_d - x_d')^2}{2l_d^2} \right) = \sigma_D^2 \exp\left(-\sum_{d=1}^{D} \frac{(x_d - x_d')^2}{2l_d^2} \right) \tag{4.9}$$

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

4.4.1 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 hyperparameters of each dimension-wise kernel, additive kernels are equipped with a set of D hyperparameters $\sigma_1^2 \dots \sigma_D^2$ controlling how much variance we assign to each order of interaction. These "order variance" hyperparameters 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 4.1 shows examples of normalized order variance hyperparameters learned on real datasets.

Table 4.1 Relative variance contribution of each order in the additive model, on different datasets. Here, the maximum order of interaction is set to 10, or smaller if the input dimension less than 10. Values are normalized to sum to 100.

Order of interaction	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

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

Because the hyperparameters can specify which degrees of interaction are important, the additive GP is an extremely general model. If the function we are modeling is decomposable into a sum of low-dimensional functions, our model can discover this fact and exploit it (see Figure 2.3) . If this is not the case, the hyperparameters can specify a suitably flexible model.

4.4.2 Interpretability

As noted by **?**, one of the chief advantages of additive models such as GAM is their interpretability. Plate also notes that by allowing high-order interactions as well as low-order interactions, one can trade off interpretability with predictive accuracy. In the case where the hyperparameters indicate that most of the variance in a function can be explained by low-order interactions, it is useful and easy to plot the corresponding low-order functions, as in Figure 4.2.

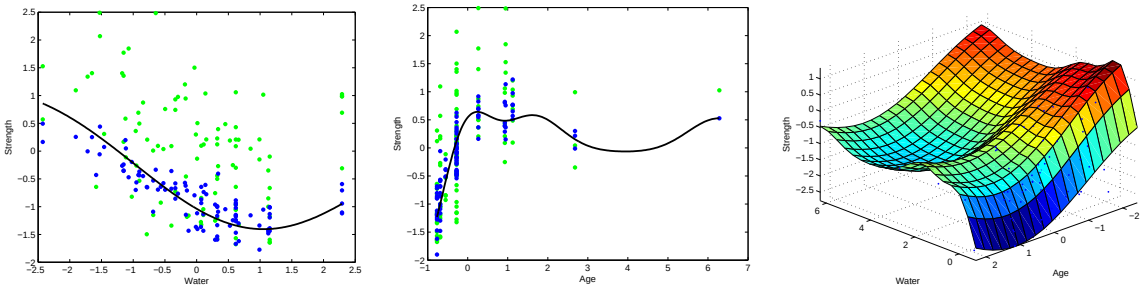


Fig. 4.2 Low-order functions on the concrete dataset. Left, Centre: By considering only first-order terms of the additive kernel, we recover a form of Generalized Additive Model, and can plot the corresponding 1-dimensional functions. Green points indicate the original data, blue points are data after the mean contribution from the other dimensions' first-order terms has been subtracted. The black line is the posterior mean of a GP with only one term in its kernel. Right: The posterior mean of a GP with only one second-order term in its kernel.

4.4.3 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 $O(D^2)$.

The nth order additive kernel corresponds to the nth *elementary symmetric polynomial***?** **?**, which we denote e_n. For example: if $\mathbf{x}$ has 4 input dimensions ($D = 4$), and if we let $z_i = k_i(x_i, x_i')$, then

$$
\begin{aligned}
k_{add_1}(\mathbf{x}, \mathbf{x}') &= e_1(z_1, z_2, z_3, z_4) = z_1 + z_2 + z_3 + z_4 \\
k_{add_2}(\mathbf{x}, \mathbf{x}') &= e_2(z_1, z_2, z_3, z_4) = z_1 z_2 + z_1 z_3 + z_1 z_4 + z_2 z_3 + z_2 z_4 + z_3 z_4 \\
k_{add_3}(\mathbf{x}, \mathbf{x}') &= e_3(z_1, z_2, z_3, z_4) = z_1 z_2 z_3 + z_1 z_2 z_4 + z_1 z_3 z_4 + z_2 z_3 z_4 \\
k_{add_4}(\mathbf{x}, \mathbf{x}') &= e_4(z_1, z_2, z_3, z_4) = z_1 z_2 z_3 z_4
\end{aligned}
$$

The Newton-Girard formulae give an efficient recursive form for computing these polynomials. If we define s_k to be the kth power sum: $s_k(z_1, z_2, \ldots, z_D) = \sum_{i=1}^{D} z_i^k$, then

$$k_{add_n}(\mathbf{x}, \mathbf{x}') = e_n(z_1, \ldots, z_D) = \frac{1}{n} \sum_{k=1}^{n} (-1)^{(k-1)} e_{n-k}(z_1, \ldots, z_D) s_k(z_1, \ldots, z_D) \tag{4.10}$$

Where $e_0 \triangleq 1$. The Newton-Girard formulae have time complexity $O(D^2)$, while computing a sum over an exponential number of terms.

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:

$$\frac{\partial k_{add_n}}{\partial z_j} = e_{n-1}(z_1, \ldots, z_{j-1}, z_{j+1}, \ldots z_D) \tag{4.11}$$

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

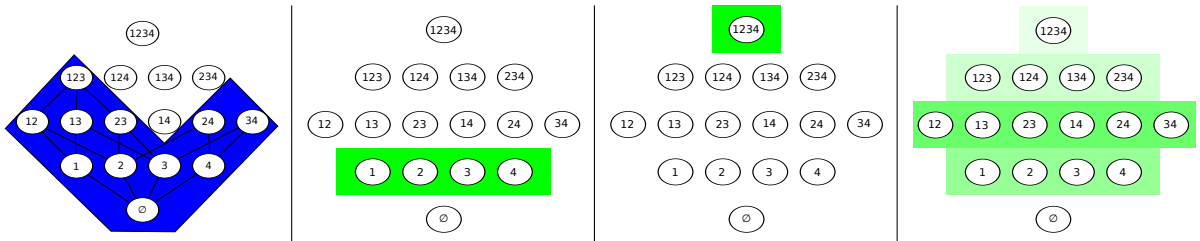
4.4.4 Computation

The computational cost of evaluating the Gram matrix of a product kernel (such as the SE kernel) is $O(N^2 D)$, while the cost of evaluating the Gram matrix of the additive kernel is $O(N^2 DR)$, 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 $O(N^3)$ 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.

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. All of the experiments in this paper were performed using the standard GPML toolbox[1]; code to perform all experiments is available at the author's website.[2]

[1] Available at `http://www.gaussianprocess.org/gpml/code/`

[2] Example code available at: `http://mlg.eng.cam.ac.uk/duvenaud/`



HKL kernel | GP-GAM kernel | Squared-exp GP kernel | Additive GP kernel

Fig. 4.3 A comparison of different additive model classes. Nodes represent different interaction terms, ranging from first-order to fourth-order interactions. Far left: HKL can select a hull of interaction terms, but must use a pre-determined weighting over those terms. Far right: the additive GP model can weight each order of interaction seperately. Neither the HKL nor the additive model dominate one another in terms of flexibility, however the GP-GAM and the SE-GP are special cases of additive GPs.

4.5 Related Work

Plate**?** constructs a form of additive GP, but 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.

Christoudias et al.**?** 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.

4.5.1 Hierarchical Kernel Learning

Bach**?** uses a regularized optimization framework to learn a weighted sum over an exponential number of kernels which can be computed in polynomial time. The subsets of kernels considered by this method are restricted to be a *hull* of kernels.[3] Given each

[3]In the setting we are considering in this paper, a hull can be defined as a subset of all terms such that if term $\prod_{j \in J} k_j(\mathbf{x}, \mathbf{x}')$ is included in the subset, then so are all terms $\prod_{j \in J/i} k_j(\mathbf{x}, \mathbf{x}')$, for all $i \in J$.

dimension's kernel, and a pre-defined weighting over all terms, HKL performs model selection by searching over hulls of interaction terms. In **?**, Bach also fixes the relative weighting between orders of interaction with a single term α, computing the sum over all orders by:

$$k_a(\mathbf{x}, \mathbf{x}') = v_D^2 \prod_{d=1}^{D} (1 + \alpha k_d(x_d, x'_d)) \tag{4.12}$$

which has computational complexity $O(D)$. However, this formulation forces the weight of all nth order terms to be weighted by α^n.

Figure 4.3 contrasts the HKL hull-selection method with the Additive GP hyperparameter-learning method. Neither method dominates the other in flexibility. The main difficulty with the approach of **?** is that hyperparameters are hard to set other than by cross-validation. In contrast, our method optimizes the hyperparameters of each dimension's base kernel, as well as the relative weighting of each order of interaction.

4.5.2 ANOVA Procedures

Vapnik **?** introduces the support vector ANOVA decomposition, which has the same form as our additive kernel. However, they recommend approximating the sum over all D orders with only one term "of appropriate order", presumably because of the difficulty of setting the hyperparameters of an SVM. Stitson et al.**?** performed experiments which favourably compared the support vector ANOVA decomposition to polynomial and spline kernels. They too allowed only one order to be active, and set hyperparameters by cross-validation.

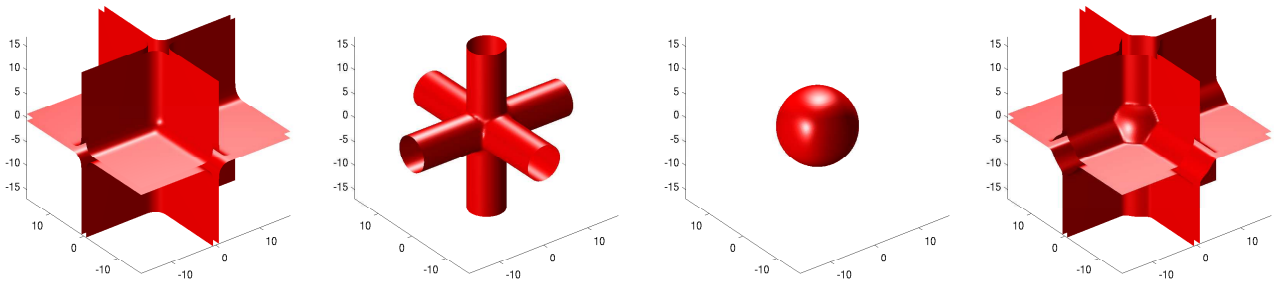
A closely related procedure from the statistics literature is smoothing-splines ANOVA (SS-ANOVA)**?**. An SS-ANOVA model is estimated as a weighted sum of splines along each dimension, plus a sum of splines over all pairs of dimensions, all triplets, etc, with each individual interaction term having 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. Learning in SS-ANOVA is usually done via penalized-maximum likelihood with a fixed sparsity hyperparameter.

In contrast to these procedures, our method can easily include all D orders of interaction, each weighted by a separate hyperparameter. As well, we can learn kernel hyperparameters individually per input dimension, allowing automatic relevance determination to operate.

For details, see **?**.

4.5.3 Non-local Interactions

By far the most popular kernels for regression and classification tasks on continuous data are the squared exponential (Gaussian) kernel, and the Matérn kernels. These kernels depend only on the scaled Euclidean distance between two points, both having the form: $k(\mathbf{x}, \mathbf{x}') = f(\sum_{d=1}^{D} (x_d - x'_d)^2 / l_d^2)$. Bengio et al.**?** argue that models based on squared-exponential kernels are particularily 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 interactions $k_1 + k_2 + k_3$ | 2nd order interactions $k_1k_2 + k_2k_3 + k_1k_3$ | 3rd order interactions $k_1k_2k_3$ (Squared-exp kernel) | All interactions (Additive kernel)

Fig. 4.4 Isocontours of additive kernels in 3 dimensions. The third-order kernel only considers nearby points relevant, while the lower-order kernels allow the output to depend on distant points, as long as they share one or more input value.

Additive kernels have a much more complex structure, and allow extrapolation based on distant parts of the input space, without spreading the mass of the kernel over the whole space. For example, additive kernels of the second order allow strong non-local interactions between any points which are similar in any two input dimensions. Figure 4.4 provides a geometric comparison between squared-exponential kernels and additive kernels in 3 dimensions.

4.6 Experiments

4.6.1 Experimental Setup

On a diverse collection of datasets, we compared five different models. In the results tables below, GP Additive refers to a GP using the additive kernel with squared-exp base kernels. For speed, we limited the maximum order of interaction in the additive kernels to 10. GP-GAM denotes an additive GP model with only first-order interactions. GP Squared-Exp is a GP model with a squared-exponential ARD kernel. HKL[4] 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 hyperparameters 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 hyperparameters, we fit a constant mean function to the data. In the classification experiments, GP inference was done using Expectation Propagation **?**.

4.6.2 Bach Synthetic Dataset

In addition to standard UCI repository datasets, we generated a synthetic dataset following the same recipe as **?**: From a covariance matrix drawn from a Wishart distribution with 1024 degrees of freedom, we select 8 variables. We then construct the non-linear function $f(X) = \sum_{i=1}^{4} \sum_{j=1+1}^{4} X_i X_j + \epsilon$, which sums all 2-way products of the first 4 variables, and adds Gaussian noise ϵ. This dataset is one which can be predicted well by a kernel which is a sum of two-way interactions over the first 4 variables, ignoring the extra 4 noisy copies.

This dataset was designed by **?** to demonstrate the advantages of HKL over GP-ARD.

If the dataset is large enough, HKL can construct a hull around only those subsets of cross terms that are optimal for predicting the output. GP-ARD, in contrast, can only learn to ignore the noisy copy variables, but cannot learn to ignore the higher-term interactions between the predictive variables. However, a GP with an additive kernel can learn both to ignore irrelevant variables, and to ignore certain orders of interaction. In this example, the additive GP is able to recover the relevant structure.

[4] Code for HKL available at `http://www.di.ens.fr/~fbach/hkl/`

4.6.3 Results

Tables 4.2, 4.3, 4.4 and 4.5 show mean performance across 10 train-test splits. Because HKL does not specify a noise model, it could not be included in the likelihood comparisons.

Table 4.2 Regression Mean Squared Error

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

Table 4.3 Regression Negative Log Likelihood

Method	bach	concrete	pumadyn-8nh	servo	housing
Linear Regression	2.430	1.403	1.881	1.678	1.052
GP GAM	1.708	0.467	1.195	0.800	0.457
GP Squared-exp	**−0.131**	0.398	**0.843**	0.429	**0.207**
GP Additive	**−0.131**	**0.114**	**0.841**	**0.309**	**0.194**

Table 4.4 Classification percent error

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

The model with best performance on each dataset is in bold, along with all other models that were not significantly different under a paired t-test. The additive model never performs significantly worse than any other model, and sometimes performs significantly better than all other models. The difference between all methods is larger in the

Table 4.5 Classification negative log-likelihood

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

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

The additive GP performed best on datasets well-explained by low orders of interaction, and approximately as well as the SE-GP model on datasets which were well explained by high orders of interaction (see table 4.1). Because the additive GP is a superset of both the GP-GAM model and the SE-GP model, instances where the additive GP performs slightly worse are presumably due to over-fitting, or due to the hyperparameter optimization becoming stuck in a local maximum. Additive GP performance can be expected to benefit from integrating out the kernel hyperparameters.

4.6.4 Future Work

Since the non-local structure capturable by additive kernels is necessarily axis-aligned, we can naturally consider that combining the hyperparameter optimization with an initial rotation of the input space might allow us to recover non-axis aligned additivity in functions.

Note that we are free to choose a different covariance function along each dimension.

4.7 Conclusion

We present additive Gaussian processes: a simple family of models which generalizes two widely-used classes of models. Additive GPs also introduce a tractable new type of structure into the GP framework. Our experiments indicate that such additive structure is present in real datasets, allowing our model to perform better than standard GP models. In the case where no such structure exists, our model can recover arbitrarily flexible models, as well.

In addition to improving modeling efficacy, the additive GP also improves model

interpretability: the order variance hyperparameters indicate which sorts of structure are present in our model.

Compared to HKL, which is the only other tractable procedure able to capture the same types of structure, our method benefits from being able to learn individual kernel hyperparameters, as well as the weightings of different orders of interaction. Our experiments show that additive GPs are a state-of-the-art regression model.

Chapter 5

Warped Mixture Models

5.1 abstract

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.

Joint work with Tomoharu Iwata

5.2 Introduction

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,

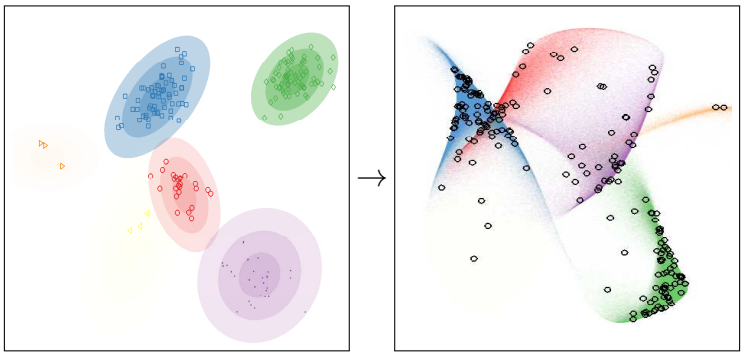


Fig. 5.1 A sample from the iWMM prior. Left: In the latent space, a mixture distribution is sampled from a Dirichlet process mixture of Gaussians. Right: The latent mixture is smoothly warped to produce non-Gaussian manifolds in the observed space.

but this does not address the problem that real clusters may not be well matched by any parametric density.

In this paper, we propose a nonparametric Bayesian model that can find nonlinearly separable clusters with complex shapes. The proposed model assumes that each observation has coordinates in a latent space, and is generated by warping the latent coordinates via a nonlinear function from the latent space to the observed space. By this warping, complex shapes in the observed space can be modeled by simpler shapes in the latent space. In the latent space, we assume an infinite Gaussian mixture model **?**, which allows us to automatically infer the number of clusters. For the prior on the nonlinear mapping function, we use Gaussian processes **?**, which enable us to flexibly infer the nonlinear warping function from the data. We call the proposed model the *infinite warped mixture model* (iWMM). Figure 5.1 shows a set of manifolds and datapoints sampled from the prior defined by this model.

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

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

5.3 Gaussian Process Latent Variable Model

In this section, we give a brief introduction to the GPLVM, which can be viewed as a special case of the iWMM. The GPLVM is a probabilistic model of nonlinear manifolds. While not typically thought of as a density model, the GPLVM 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 $\mathbf{Y} = (\mathbf{y}_1, \cdots, \mathbf{y}_N)^\top$, where $\mathbf{y}_n \in \mathbb{R}^D$, and they are associated with a set of latent coordinates $\mathbf{X} = (\mathbf{x}_1, \cdots, \mathbf{x}_N)^\top$, where $\mathbf{x}_n \in \mathbb{R}^Q$. The GPLVM assumes that observations are generated by mapping the latent coordinates through a set of smooth functions, over which Gaussian process priors are placed. Under the GPLVM, the probability of observations given the latent coordinates, integrating out the mapping functions, is

$$p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta}) = (2\pi)^{-\frac{DN}{2}} |\mathbf{K}|^{-\frac{D}{2}} \exp\left(-\frac{1}{2}\mathrm{tr}(\mathbf{Y}^\top \mathbf{K}^{-1}\mathbf{Y})\right), \tag{5.1}$$

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:

$$\begin{aligned} k(\mathbf{x}_n, \mathbf{x}_m) = \alpha \exp\left(-\frac{1}{2\ell^2}(\mathbf{x}_n - \mathbf{x}_m)^\top(\mathbf{x}_n - \mathbf{x}_m)\right) \\ + \delta_{nm}\beta^{-1}. \end{aligned} \tag{5.2}$$

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

Typically, the GPLVM 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 GPLVM can be viewed as a manifold learning algorithm, the iWMM can be viewed as learning a set of manifolds, one for each cluster.

5.4 Infinite Warped Mixture Model

In this section, we define in detail the infinite warped mixture model (iWMM). In the same way as the GPLVM, 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,

$$p(\mathbf{x}|\{\lambda_c, \boldsymbol{\mu}_c, \mathbf{R}_c\}) = \sum_{c=1}^{\infty} \lambda_c \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_c, \mathbf{R}_c^{-1}), \tag{5.3}$$

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 $\{\boldsymbol{\mu}_c, \mathbf{R}_c\}$,

$$p(\boldsymbol{\mu}_c, \mathbf{R}_c) = \mathcal{N}(\boldsymbol{\mu}_c|\mathbf{u}, (r\mathbf{R}_c)^{-1})\mathcal{W}(\mathbf{R}_c|\mathbf{S}^{-1}, \nu), \tag{5.4}$$

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:

$$\mathcal{W}(\mathbf{R}|\mathbf{S}^{-1}, \nu) = \frac{1}{G}|\mathbf{R}|^{\frac{\nu-Q-1}{2}} \exp\left(-\frac{1}{2}\text{tr}(\mathbf{SR})\right), \tag{5.5}$$

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 $\mathbf{Z} = (z_1, \cdots, z_N)$ is obtained by integrating out the Gaussian parameters $\{\boldsymbol{\mu}_c, \mathbf{R}_c\}$ as follows:

$$\begin{aligned} p(\mathbf{X}|\mathbf{Z}, \mathbf{S}, \nu, r) = &\prod_{c=1}^{\infty} \pi^{-\frac{N_c Q}{2}} \frac{r^{Q/2}|\mathbf{S}|^{\nu/2}}{r_c^{Q/2}|\mathbf{S}_c|^{\nu_c/2}} \\ &\times \prod_{q=1}^{Q} \frac{\Gamma(\frac{\nu_c+1-q}{2})}{\Gamma(\frac{\nu+1-q}{2})}, \end{aligned} \tag{5.6}$$

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

$$r_c = r + N_c, \qquad \nu_c = \nu + N_c,$$

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

$$\mathbf{S}_c = \mathbf{S} + \sum_{n:z_n=c} \mathbf{x}_n \mathbf{x}_n^\top + r\mathbf{u}\mathbf{u}^\top - r_c \mathbf{u}_c \mathbf{u}_c^\top, \tag{5.7}$$

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:

$$p(\mathbf{Z}|\eta) = \frac{\eta^C \prod_{c=1}^{C} (N_c - 1)!}{\eta(\eta+1)\cdots(\eta+N-1)}, \tag{5.8}$$

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

$$\begin{aligned} &p(\mathbf{Y}, \mathbf{X}, \mathbf{Z}|\boldsymbol{\theta}, \boldsymbol{S}, \nu, \mathbf{u}, r, \eta) \\ &= p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta}) p(\mathbf{X}|\mathbf{Z}, \boldsymbol{S}, \nu, \mathbf{u}, r) p(\mathbf{Z}|\eta), \end{aligned} \tag{5.9}$$

where factors in the right hand side can be calculated by (5.1), (5.6) and (5.8), respectively.

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

1. Draw mixture weights $\boldsymbol{\lambda} \sim \text{GEM}(\eta)$
2. For each component $c = 1, \cdots, \infty$
   (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 $d = 1, \cdots, D$
   (a) Draw function $f_d(\mathbf{x}) \sim \text{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$
4. For each observation $n = 1, \cdots, N$
   (a) Draw latent assignment $z_n \sim \text{Mult}(\boldsymbol{\lambda})$
   (b) Draw latent coordinates $\mathbf{x}_n \sim \mathcal{N}(\boldsymbol{\mu}_{z_n}, \mathbf{R}_{z_n}^{-1})$
   (c) For each observed dimension $d = 1, \cdots, D$

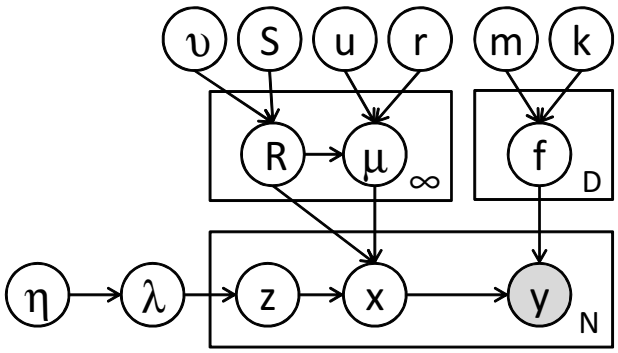


Fig. 5.2 A graphical model representation of the infinite warped mixture model, where the shaded and unshaded nodes indicate observed and latent variables, respectively, and plates indicate repetition.

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 5.2 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 GPLVM 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 GPLVM, while the iWMM with fixed direct mapping function $f_d(\mathbf{x}) = x_d$ and $Q = D$ corresponds to the iGMM.

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

5.5 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 hybrid Monte Carlo sampling of $\mathbf{X}$. Given $\mathbf{X}$, we can efficiently sample $\mathbf{Z}$ using collapsed Gibbs sampling, integrating out the mixture parameters. Given $\mathbf{Z}$, we can calculate the gradient of the unnormalized posterior distribution of $\mathbf{X}$, integrat-

ing over warping functions. This gradient allows us to sample $\mathbf{X}$ using hybrid 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 resample cluster assignments, integrating out the iGMM likelihood in close form. Given the current state of all but one latent component z_n, a new value for z_n is sampled from the following probability:

$$\begin{aligned} &p(z_n = c|\mathbf{X}, \mathbf{Z}_{\backslash n}, \boldsymbol{S}, \nu, \mathbf{u}, r, \eta) \\ &\propto \begin{cases} N_{c\backslash n} \cdot p(\mathbf{x}_n|\mathbf{X}_{c\backslash n}, \boldsymbol{S}, \nu, \mathbf{u}, r) & \text{existing components} \\ \eta \cdot p(\mathbf{x}_n|\boldsymbol{S}, \nu, \mathbf{u}, r) & \text{a new component} \end{cases} \end{aligned} \tag{5.10}$$

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:

$$\begin{aligned} &p(\mathbf{x}_n|\mathbf{X}_{c\backslash n}, \boldsymbol{S}, \nu, \mathbf{u}, r) \\ &= \pi^{-\frac{N_{c\backslash n}Q}{2}} \frac{r_{c\backslash n}^{Q/2}|\mathbf{S}_{c\backslash n}|^{\nu_{c\backslash n}/2}}{r_{c\backslash n}'^{Q/2}|\mathbf{S}'_{c\backslash n}|^{\nu'_{c\backslash n}/2}} \prod_{d=1}^{Q} \frac{\Gamma(\frac{\nu'_{c\backslash n}+1-d}{2})}{\Gamma(\frac{\nu_{c\backslash n}+1-d}{2})}, \end{aligned} \tag{5.11}$$

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 $p(\mathbf{x}_n|\boldsymbol{S}, \nu, \mathbf{u}, r)$.

Hybrid 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 $\log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta}) + \log p(\mathbf{X}|\mathbf{Z}, \boldsymbol{S}, \nu, \mathbf{u}$ The first term of the gradient can be calculated by

$$\frac{\partial \log p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta})}{\partial \mathbf{K}} = -\frac{1}{2}D\mathbf{K}^{-1} + \frac{1}{2}\mathbf{K}^{-1}\mathbf{Y}\mathbf{Y}^T\mathbf{K}^{-1}, \tag{5.12}$$

and

$$\begin{aligned} &\frac{\partial k(\mathbf{x}_n, \mathbf{x}_m)}{\partial \mathbf{x}_n} \\ &= -\frac{\alpha}{\ell^2} \exp\left(-\frac{1}{2\ell^2}(\mathbf{x}_n - \mathbf{x}_m)^\top(\mathbf{x}_n - \mathbf{x}_m)\right)(\mathbf{x}_n - \mathbf{x}_m), \end{aligned} \tag{5.13}$$

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

$$\frac{\partial \log p(\mathbf{X}|\mathbf{Z}, \boldsymbol{S}, \nu, \mathbf{u}, r)}{\partial \mathbf{x}_n} = -\nu_{z_n} \boldsymbol{S}_{z_n}^{-1}(\mathbf{x}_n - \mathbf{u}_{z_n}). \tag{5.14}$$

We also infer kernel hyperparameters $\boldsymbol{\theta} = \{\alpha, \beta, \ell\}$ via HMC, using the gradient of the log unnormalized posterior with respect to the kernel hyperparameters. The complexity of each iteration of HMC is dominated by the $\mathcal{O}(N^3)$ computation of $\mathbf{K}^{-1}$ [1].

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 procedures:

1. For each observation $n = 1, \cdots, N$, sample the component assignment z_n by collapsed Gibbs sampling (5.10).

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

5.5.1 Posterior Predictive Density

In the GP-LVM, the predictive density of at test point $y^\star$ is usually computed by finding the point $x^\star$ which which is most likely to be mapped to $y^\star$, then using the density of $p(x^\star)$ and the Jacobian of the warping at that point to approximately compute the density at $y^\star$. When inference is done by simply optimizing the location of the latent points, this estimation method simply requires solving a single optimization for each $y^\star$.

For our model, we use approximate integration to estimate $p(y^\star)$. 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 $p(y^\star)$. Second, because we do not optimize the latent coordinates but rather sample them, we would need to perform optimizations for each $p(y^\star)$ 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:

$$\begin{aligned} & p(\mathbf{y}_*|\mathbf{Y}) \\ &= \iint p(\mathbf{y}_*, \mathbf{x}_*, \mathbf{X}|\mathbf{Y}) d\mathbf{x}_* d\mathbf{X} \\ &= \iint p(\mathbf{y}_*|\mathbf{x}_*, \mathbf{X}, \mathbf{Y}) p(\mathbf{x}_*|\mathbf{X}, \mathbf{Y}) p(\mathbf{X}|\mathbf{Y}) d\mathbf{x}_* d\mathbf{X}. \end{aligned} \tag{5.15}$$

We approximate $p(\mathbf{X}|\mathbf{Y})$ using the samples from the Gibbs and hybrid Monte Carlo

[1] This complexity could be improved by making use of an inducing point approximation such as **??**

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

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

When a new component $C+1$ is assigned to z_*, the prior Gaussian-Wishart distribution is used for sampling in steps 2 and 3. The first factor of (5.15) can be calculated by

$$\begin{aligned} &p(\mathbf{y}_*|\mathbf{x}_*, \mathbf{X}, \mathbf{Y}) \\ &= \mathcal{N}(\mathbf{k}_*^\top \mathbf{K}^{-1}\mathbf{Y}, k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^\top \mathbf{K}^{-1}\mathbf{k}_*), \end{aligned} \tag{5.16}$$

where $\mathbf{k}_* = (k(\mathbf{x}_*, \mathbf{x}_1), \cdots, k(\mathbf{x}_*, \mathbf{x}_N))^\top$. Each step of this procedure is exact, and since the observations $\mathbf{y}_*$ 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 5.1, 5.4, and 5.6.

5.6 Related work

The GPLVM is effective as a nonlinear latent variable model in a wide variety of applications **???**. The latent positions $\mathbf{X}$ in the GPLVM 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 is 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 different dimension in the observed space. This ability is demonstrated in figure 5.4b.

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.

5.7 Experimental results

5.7.1 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 5.3 shows a sample from the posterior over the latent coordinates and 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.

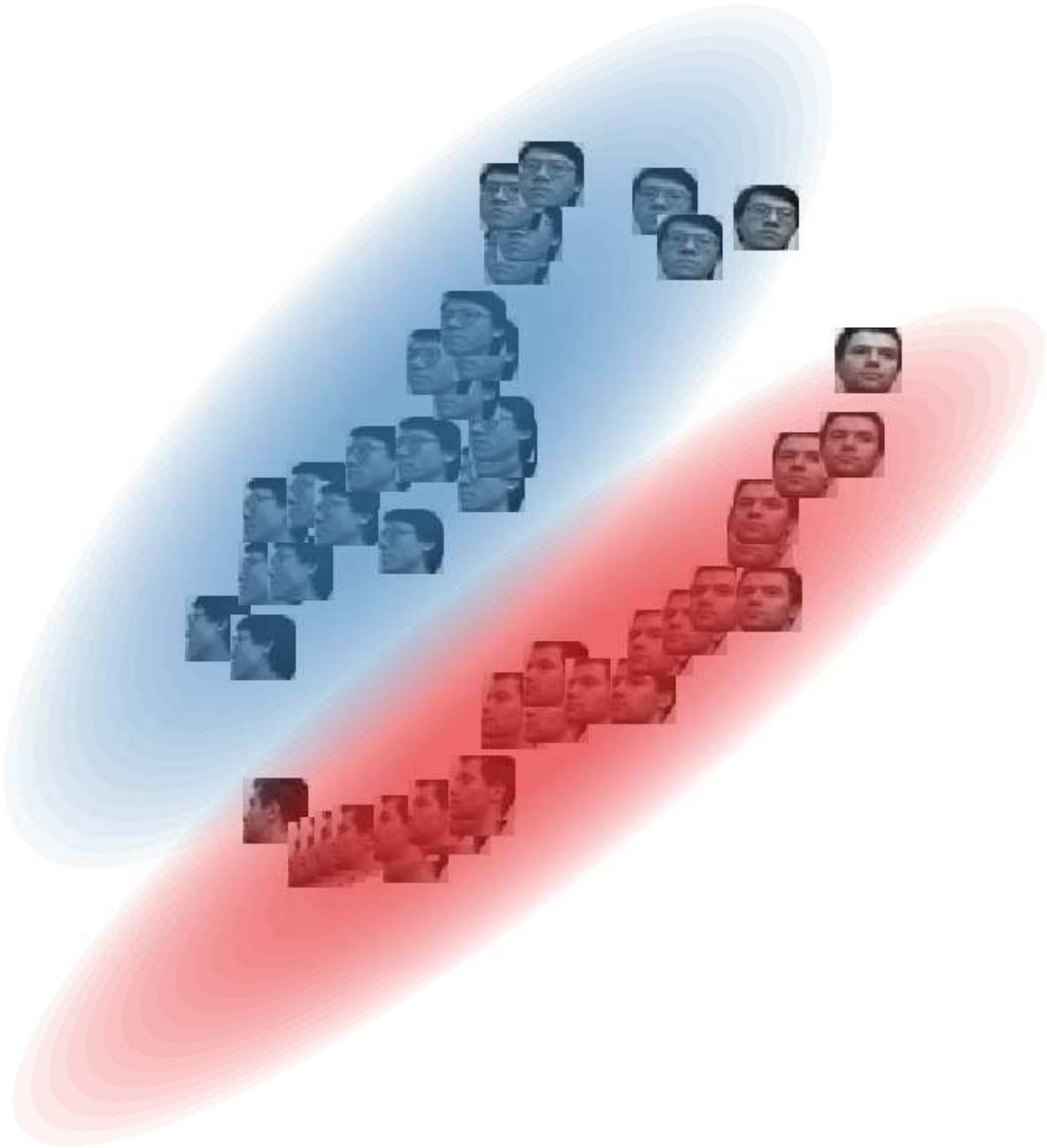
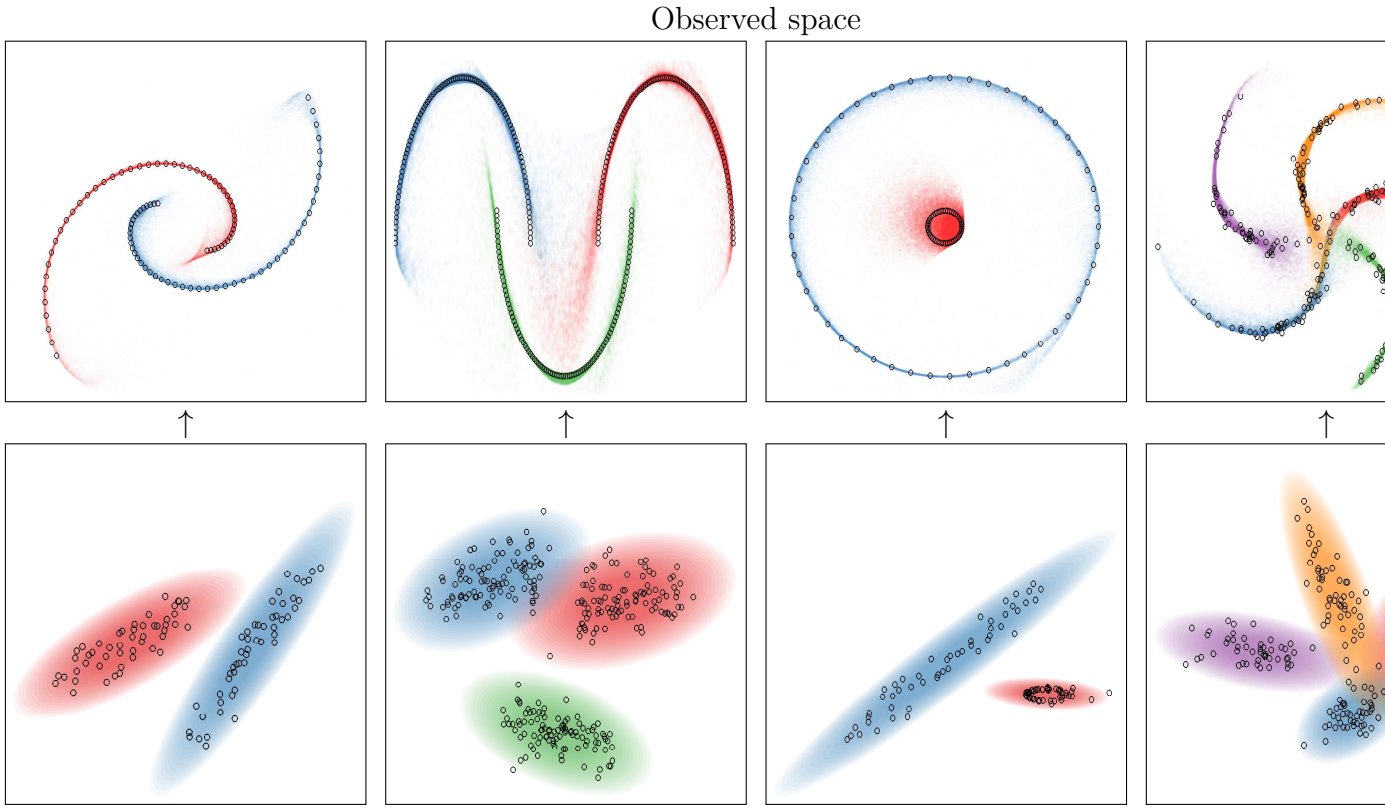
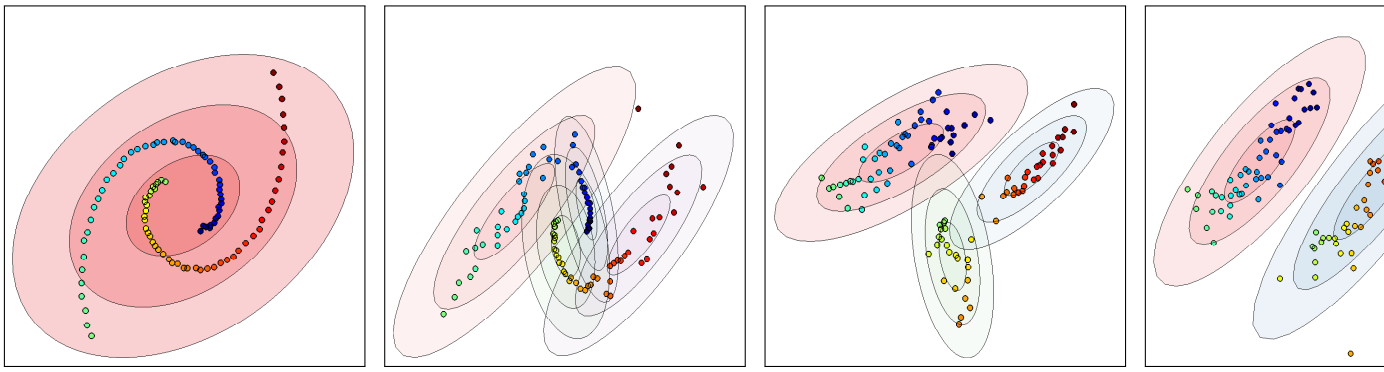


Fig. 5.3 A sample from the 2-dimensional latent space when modeling a series of 32x32 face images. Our model correctly discovers that the data consists of two seperate manifolds, both approximately one-dimensional, which share the same head-turning structure.



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

Fig. 5.4 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.



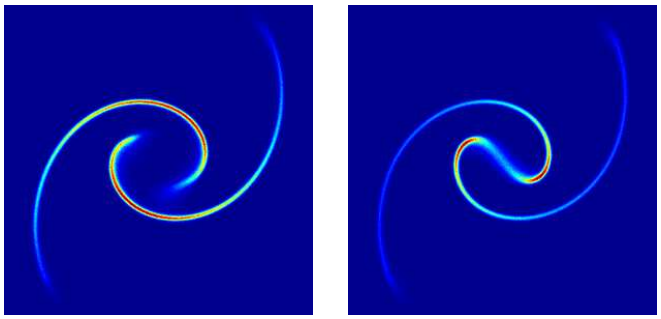
(a) 1 (b) 500 (c) 1800 (d) 3000

Fig. 5.5 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.

5.7.2 Synthetic Datasets

Next, we demonstrate the proposed model on the four synthetic datasets shown in Figure 5.4. None of these four datasets can be appropriately clustered by Gaussian mixture models (GMM). For example, consider the 2-curve data shown in Figure 5.4 (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 5.4 (a). The iWMM separated the two curved lines by nonlinearly warping two Gaussians from the latent space to the observed space.

Figure 5.4 (c) shows an interesting manifold learning challenge: a dataset consisting of two circles. The outer circle is modeled in the latent space by a Gaussian with effectively one degree of freedom. This linear topology fits the outer circle in the observed space by bending the two ends until they overlap. In contrast, the sampler fails to discover the 1D topology of the inner circle, modeling it with a 2D manifold instead. This example demonstrates that each cluster in the iWMM manifold can have a different effective dimension.



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

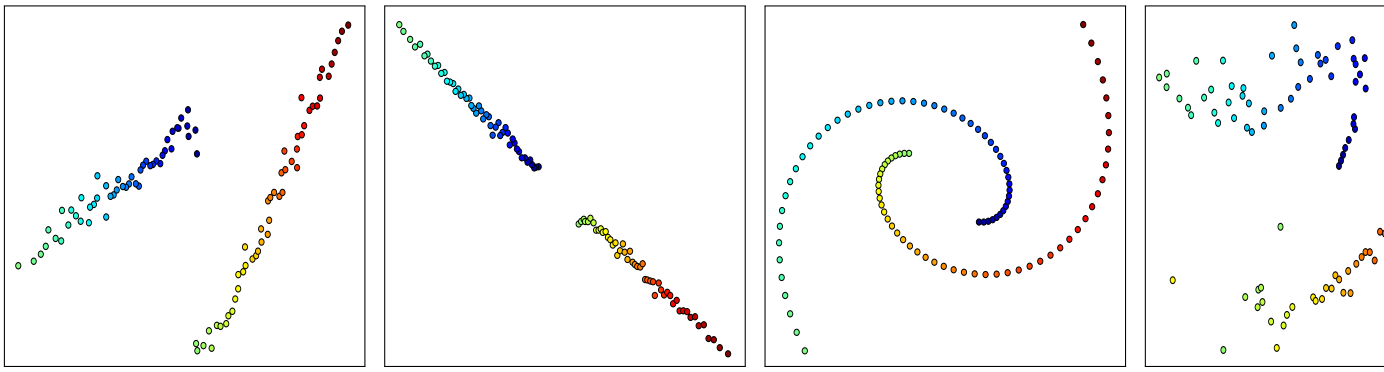
Fig. 5.6 The posterior density in the observed space with the 2-curve data inferred by the iWMM (a), and that inferred by the iWMM with one component (b).

5.7.3 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 5.5 shows the samples of the latent coordinates and clusters of the iWMM over time, when modeling the 2-curve data. 5.5(a) shows the latent coordinates initialized at the observed coordinates, starting with one latent component. At the 500th iteration 5.5(b), each curved line is modeled by two components. At the 1800th iteration 5.5(c), the left curved line is modeled by a single component. At the 3000th iteration 5.5(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.

5.7.4 Density Estimation

Figure 5.6 (a) shows the posterior density in the observed space inferred by the iWMM on the 2-curve data, computed using 1000 samples from the Markov chain. The two separate manifolds of high density implied by the two curved lines was recovered by the iWMM. Note also that the density along the manifold varies with the density of data shown in Figure 5.4 (a). This result can be compared to a special case of our model, which uses only a single Gaussian to model the latent coordinates instead of an infinite GMM. Figure 5.6 (b) shows that the result of the iWMM with $C = 1$, where posterior is forced to place significant density connecting the two clusters. Figure 5.6 (b) shows that the single-cluster variant of the iWMM posterior is forced to place significant density connecting the two clusters.



(a) iWMM (b) iWMM ($C = 1$) (c) GPLVM (d) BGPLV

Fig. 5.7 The estimated latent coordinates of the 2-curve data by (a) iWMM, (b) iWMM ($C = 1$), (c) GPLVM, and (d) Bayesian GPLVM.

5.7.5 Visualization

Next, we briefly investigate the potential of the iWMM for visualization. Figure 5.7 (a) shows the latent coordinates obtained by averaging over 1000 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 5.7 (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 5.7 (c) and (d) show the latent coordinates estimated by the GPLVM when optimizing or integrating out the latent coordinates, respectively. Recall that the iWMM ($C = 1$) is a more flexible model than the GPLVM, since the GPLVM 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.

Table 5.1 The statistics of datasets used for evaluation.

	2-curve	3-semi	2-circle	Pinwheel	Iris	Glass	Wine	Vowel
number of samples: N	100	300	100	250	150	214	178	528
observed dimensionality: D	2	2	2	2	4	9	13	10
number of clusters: C	2	3	2	5	3	7	3	11

5.7.6 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 5.1. In each experiment, we show the results of ten-fold cross-validation. Results in bold are not significantly different from the best performing method in each column according to a paired t-test.

Table 5.2 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 5.2 compares the clustering performance of the iWMM with the iGMM, quantified by the Rand index **?**, which measures the correspondence between inferred clusters and true clusters. The iGMM is another probabilistic generative model commonly used for clustering, which can be seen as a special case of the iWMM in which the Gaussian clusters are not warped. These experiments demonstrate the extent to which nonparametric cluster shapes allow a mixture model to recover more meaningful clusters.

Table 5.3 lists average test log likelihood, comparing the proposed models with kernel density estimation (KDE), and the infinite Gaussian mixture model (iGMM). In KDE, the kernel width is estimated by maximizing the leave-one-out log densities. Since the manifold on which the observed data lies can be at most D-dimensional, we set the latent dimension Q equal to the observed dimension D in iWMMs. We also include the $Q = 2$ case in an attempt to characterize how much modeling power is lost by forcing the latent representation to be visualizable. The proposed models achieved high test log likelihoods compared with the KDE and iGMM.

Table 5.3 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

5.7.7 Source code

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

5.8 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 hybrid 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.

5.9 Conclusion

In this paper, 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 cluster-

ing, manifold learning, density estimation and visualization. We demonstrated that a simple but flexible probabilistic generative model can perform well at all these tasks.

Acknowledgements

The authors would like to thank Dominique Perrault-Joncas, Carl Edward Rasmussen, and Ryan Prescott Adams for helpful discussions.

Chapter 6

Bayesian Estimation of Integrals

6.1 Log-Gaussian Process Models

When integrating

$$Z = \int f(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \tag{6.1}$$

$$= \mathbb{E}_{p(\mathbf{x})} \left[f(\mathbf{x}) \right] \tag{6.2}$$

We typically model the function $f(\mathbf{x})$ with a Gaussian process prior. However, for many problems in Bayesian inference, $f(\mathbf{x})$ is a likelihood function, typically of the form $f(\mathbf{x}) = \exp(\log \ell(\mathbf{x}))$. This implies that $f(\mathbf{x})$ is both positive and that it has a high dynamic range, making a stationary GP a poor model for this function.

A much more believable model would be a GP prior on $\log \ell(\mathbf{x})$. A GP prior on $\log \ell$ induces a $\log \mathrm{GP}$ prior on $\ell(\mathbf{x})$. However, this model makes the expectation of the integral over f seemingly intractable:

$$\mathbb{E}[Z] = \mathbb{E}_{\mathrm{GP}(\log \ell)} \left[\int \exp(\log \ell(\mathbf{x})) p(\mathbf{x}) d\mathbf{x} \right] \tag{6.3}$$

$$= \int \int \exp(\log \ell(\mathbf{x})) p(\mathbf{x}) p(\log \ell) d\mathbf{x} d\log \ell \tag{6.4}$$

$$= \int \int \exp(\log \ell(\mathbf{x})) p(\log \ell(\mathbf{x})) d\log \ell(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \tag{6.5}$$

$$= \int \mathbb{E}_{\mathrm{GP}(\log \ell)} \left[\exp(\log \ell(\mathbf{x})) \right] p(\mathbf{x}) d\mathbf{x} \tag{6.6}$$

Where the posterior mean function of $\exp(\log \ell)$ is:

$$\mathbb{E}_{\mathrm{GP}(\log \ell)}\left[\exp(\log \ell(\mathbf{x}))\right] = \exp\left[\mathbb{E}_{\mathrm{GP}(\log \ell)}\left[\log \ell(\mathbf{x})\right] + \frac{1}{2}\mathbb{V}_{\mathrm{GP}(\log \ell)}\left[\log \ell(\mathbf{x})\right]\right] \tag{6.7}$$

$$= \exp\left[k(\mathbf{x}, \mathbf{X})K^{-1}\mathbf{y} + \frac{1}{2}\left(k(x, x) - k(\mathbf{x}, \mathbf{X})K^{-1}k(\mathbf{X}, \mathbf{x})\right)\right] \tag{6.8}$$

$$= m_{\exp}(\mathbf{x}) \tag{6.9}$$

Which is not known to have a tractable form for common choices of covariance functions.

However, we can approximate this integral by fitting a GP to points $\mathbf{x}_c$ along the function $m_{\exp}(\mathbf{x})$, and exactly integrating under that integral. We will call the mean of this GP $\bar{\ell}(\mathbf{x})$. We can then compute a mean estimate, Z_0, by exactly integrating under this GP:

$$Z_0 = \int \bar{\ell}(\mathbf{x})p(\mathbf{x})d\mathbf{x} \tag{6.10}$$

$$= z^T K^{-1}\left[\exp\left(m_{\log \ell}(\mathbf{x}_c) + \frac{1}{2}V_{\log \ell}(\mathbf{x}_c)\right)\right] \tag{6.11}$$

Where z is defined below.

The approximation Z_0 exactly approaches $\mathbb{E}_{\mathrm{GP}(\log \ell)}[Z]$ in the limit of infinite candidate points $\mathbf{x}_c$. This is important, since we believe that a GP is a much better model of $\log \ell(\mathbf{x})$ than of $\ell(\mathbf{x})$.

The mean function $\bar{\ell}$ will be inflated in regions of high marginal variance in $\log \ell$. If we were simply to fit a GP to $\exp(m_{\log \ell}(\mathbf{x}_c))$, we would underestimate $\mathbb{E}_{\mathrm{GP}(\log \ell)}[Z]$ even in the limit of infinite samples.

6.2 Linearization

Note that Z_0 is a functional of $\log \ell$:

$$Z[\log \ell] = \int \bar{\ell}(\mathbf{x})p(\mathbf{x})d\mathbf{x} \tag{6.12}$$

Here we approximately compute $\mathbb{E}[Z]$ when we have only a sparse set of samples. We will first linearize Z as a functional of $\log \ell$ around the posterior mean, $\log \ell_0$.

$$Z[\log \ell] = \int \bar{\ell}(\mathbf{x})p(\mathbf{x})d\mathbf{x} \tag{6.13}$$

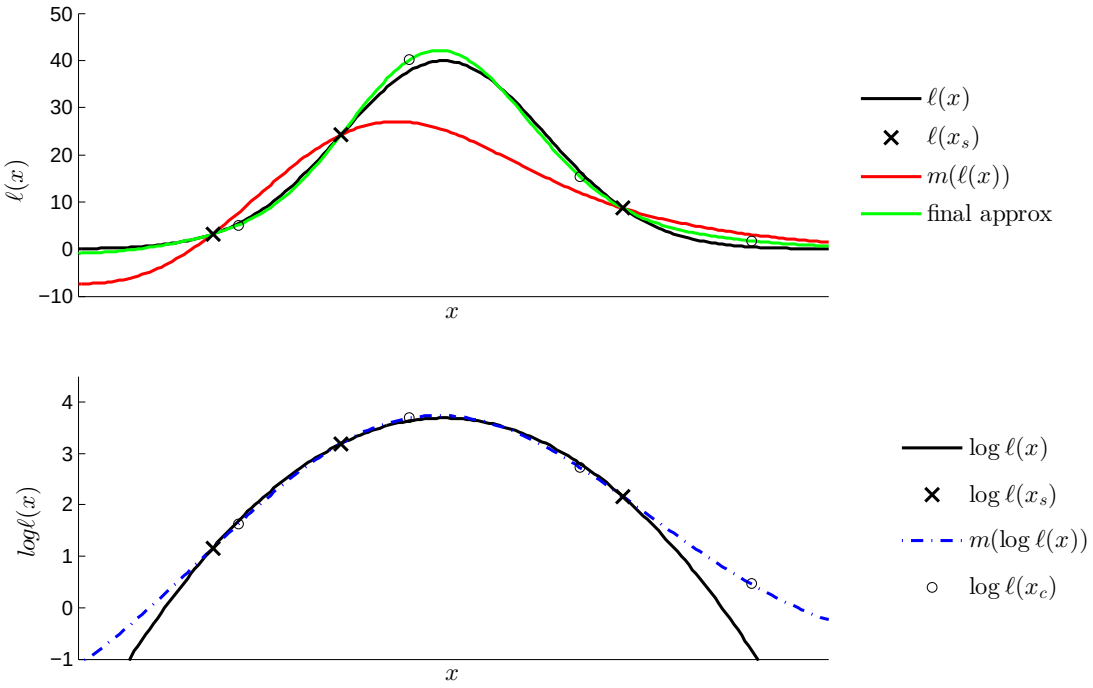


Fig. 6.1 A GP versus a log-GP

$$\log \ell_0(\mathbf{x}) = \mathbb{E}_{\mathrm{GP}(\log \ell)}\left[\log \ell(\mathbf{x})\right] \tag{6.14}$$

$$\hat{Z}[\log \ell] := Z_0 + \varepsilon[\log \ell] \tag{6.15}$$

$$Z_0 = \int \bar{\ell}(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \tag{6.16}$$

$$\varepsilon[\log \ell] = \int \left(\left. \frac{\partial \hat{Z}}{\partial \log \ell(\mathbf{x})} \right|_{\log \ell_0(\mathbf{x})} \right) \left[\log \ell(\mathbf{x}) - \log \ell_0(\mathbf{x})\right] d\mathbf{x} \tag{6.17}$$

We do not have a closed-form expression for $\left. \frac{\partial \hat{Z}}{\partial \log \ell(\mathbf{x})} \right|_{\log \ell_0(\mathbf{x})}$ everywhere, but at the points $\mathbf{x}_c$, it will be

$$\left. \frac{\partial \hat{Z}}{\partial \log \ell(\mathbf{x}_c)} \right|_{\log \ell_0(\mathbf{x}_c)} = \left. \frac{\partial \hat{Z}}{\partial \log \ell(\mathbf{x}_c)} \right|_{\log \ell_0(\mathbf{x}_c)} \tag{6.18}$$

$$= \bar{\ell}(\mathbf{x}) p(\mathbf{x}) \tag{6.19}$$

We hope that $\bar{\ell}(\mathbf{x})$ is a good approximation everywhere else, and make the approximation:

$$\left. \frac{\partial Z}{\partial \log \ell(\mathbf{x})} \right|_{\log \ell_0(\mathbf{x})} \approxeq \bar{\ell}(\mathbf{x}) p(\mathbf{x}) \tag{6.20}$$

so

$$\varepsilon[\log \ell] \approxeq \int \bar{\ell}(\mathbf{x}) p(\mathbf{x}) \left[\log \ell(x) - \log \ell_0(\mathbf{x})\right] d\mathbf{x} \tag{6.21}$$

Which is the same as in the SBQ paper, except that Z_0 will be higher than it would be in the SBQ setup, because $\bar{\ell}$ will be inflated due to uncertainty in $\log \ell$.

6.2.1 Mean of $\hat{Z}$

Under this linearization, the mean of $\hat{Z}$ is simply

$$\mathbb{E}_{\mathrm{GP}(\log \ell)}[\hat{Z}] = \mathbb{E}_{\mathrm{GP}(\log \ell)}\left[Z_0 + \varepsilon[\log \ell]\right] \tag{6.22}$$

$$= Z_0 \tag{6.23}$$

Again, Z_0 will be higher than it would be in the SBQ setup, because $\bar{\ell}$ will be inflated due to uncertainty in $\log \ell$.

Now we derive a closed-form expression for the case where the prior is $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|b, B)$, the kernel function $k_\ell(x_\star, \mathbf{x}') = h_\ell \mathcal{N}(\mathbf{x}|\mathbf{x}', A_\ell)$, and the mean function $\mu_\ell(\mathbf{x}) = \mu_\ell$.

$$Z_0 = \int \bar{\ell}(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \tag{6.24}$$

$$= \mathbb{E}_{\mathcal{GP}(\ell(\mathbf{x}))}\left[\int \ell(\mathbf{x}) p(\mathbf{x}) d\mathbf{x}\right] \tag{6.25}$$

$$= \int \bar{\ell}(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \tag{6.26}$$

where

$$\bar{\ell}(\mathbf{x}_\star) = \mu(\mathbf{x}_\star) + k_\ell(\mathbf{x}_\star, \mathbf{x}_c) K_\ell(\mathbf{x}_c, \mathbf{x}_c)^{-1} \left(m_{\exp}(\mathbf{x}_c) - \mu(\mathbf{x}_c)\right) \tag{6.27}$$

$$= \mu(\mathbf{x}_\star) + \sum_{i \in c} k_\ell(\mathbf{x}_\star, \mathbf{x}_i) \beta_i \tag{6.28}$$

$$= \mu(\mathbf{x}_\star) + \sum_{i \in c} h_\ell \mathcal{N}(\mathbf{x}_\star|\mathbf{x}_i, A_\ell)\, \beta_i \tag{6.29}$$

where

$$\beta_i = K_\ell(\mathbf{x}_c, \mathbf{x}_c)^{-1} \left(m_{\exp}(\mathbf{x}_i) - \mu(\mathbf{x}_i)\right) \tag{6.30}$$

$$\tag{6.31}$$

are simply constants. So

$$Z_0 = \int \bar{\ell}(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} \tag{6.32}$$

$$Z_0 = \int \left[\mu_\ell + \sum_{i \in c} h_\ell \mathcal{N}(\mathbf{x}|\mathbf{x}_i, \mathbf{A}_\ell)\, \beta_i \right] \mathcal{N}(\mathbf{x}|\mathbf{b}, \mathbf{B})\, d\mathbf{x} \tag{6.33}$$

$$Z_0 = \mu_\ell + h_\ell \int \left[\sum_{i \in c} \mathcal{N}(\mathbf{x}|\mathbf{x}_i, \mathbf{A}_\ell)\, \beta_i \right] \mathcal{N}(\mathbf{x}|\mathbf{b}, \mathbf{B})\, d\mathbf{x} \tag{6.34}$$

$$Z_0 = \mu_\ell + h_\ell \sum_{i \in c} \beta_i \int \mathcal{N}(\mathbf{x}|\mathbf{x}_i, \mathbf{A}_\ell)\, \mathcal{N}(\mathbf{x}|\mathbf{b}, \mathbf{B})\, d\mathbf{x} \tag{6.35}$$

and the integral

$$z_i = \int \mathcal{N}(\mathbf{x}|\mathbf{x}_i, \mathbf{A}_\ell)\, \mathcal{N}(\mathbf{x}|\mathbf{b}, \mathbf{B})\, d\mathbf{x} \tag{6.36}$$

$$= \text{ProdNormZ}(\mathbf{x}_i, \mathbf{A}_\ell, \mathbf{b}, \mathbf{B}) \tag{6.37}$$

$$\tag{6.38}$$

where

$$\text{ProdNormZ}(a, A, b, B) = \mathcal{N}(\mathbf{x}|B(A+B)^{-1}a + A(A+B)^{-1}b, A(A+B)^{-1}B)\, \mathcal{N}(a|b, A+B) \tag{6.39}$$

6.2.2 Variance of $\hat{Z}$

$$\mathbb{V}[\hat{Z}] = \mathbb{E}[\hat{Z}^2] - \left(\mathbb{E}[\hat{Z}]\right)^2 \tag{6.40}$$

$$\mathbb{V}[\hat{Z}] = \mathbb{E}[\hat{Z}^2] - Z_0^2 \tag{6.41}$$

$$= \int \left(\hat{Z}[\log \ell]\right)^2 p(\log \ell) d\log \ell - Z_0^2 \tag{6.42}$$

$$= \int \left[Z_0 + \varepsilon(\log \ell)\right]^2 p(\log \ell) d\log \ell - Z_0^2 \tag{6.43}$$

$$= \int \left[Z_0^2 + 2Z_0\varepsilon(\log \ell) + (\varepsilon(\log \ell))^2\right] p(\log \ell) d\log \ell - Z_0^2 \tag{6.44}$$

$$= \int \left[2Z_0\varepsilon(\log \ell) + (\varepsilon(\log \ell))^2\right] p(\log \ell) d\log \ell \tag{6.45}$$

$$= \int 2Z_0\varepsilon(\log \ell) p(\log \ell) d\log \ell + \int (\varepsilon(\log \ell))^2 p(\log \ell) d\log \ell \tag{6.46}$$

$$= 0 + \int (\varepsilon(\log \ell))^2 p(\log \ell) d\log \ell \tag{6.47}$$

$$= \int \left(\int \bar{\ell}(\mathbf{x}) p(\mathbf{x}) \left[\log \ell(\mathbf{x}) - \log \ell_0(\mathbf{x})\right] d\mathbf{x}\right)^2 p(\log \ell) d\log \ell \tag{6.48}$$

$$= \int \left[\int \bar{\ell}(\mathbf{x}) p(\mathbf{x}) \left[\log \ell(\mathbf{x}) - \log \ell_0(\mathbf{x})\right] d\mathbf{x}\right] \left[\int \bar{\ell}(\mathbf{x}') p(\mathbf{x}') \left[\log \ell(\mathbf{x}') - \log \ell_0(\mathbf{x}')\right] d\mathbf{x}'\right] p(\log \ell) d\log \ell$$

$$= \iiint \Big(\left[\log \ell(\mathbf{x}) - \log \ell_0(\mathbf{x})\right] \left[\log \ell(\mathbf{x}') - \log \ell_0(\mathbf{x}')\right] p(\log \ell) d\log \ell \Big) \bar{\ell}(\mathbf{x}) \bar{\ell}(\mathbf{x}') p(\mathbf{x}) p(\mathbf{x}') d\mathbf{x} d\mathbf{x}'$$

$$= \iint \mathbb{E}_{p(\log \ell)}\Big[\log \ell(\mathbf{x}) - \mathbb{E}\left[\log \ell(\mathbf{x})\right]\Big] \Big[\log \ell(\mathbf{x}') - \mathbb{E}\left[\log \ell(\mathbf{x}')\right]\Big] \bar{\ell}(\mathbf{x}) \bar{\ell}(\mathbf{x}') p(\mathbf{x}) p(\mathbf{x}') d\mathbf{x} d\mathbf{x}'$$

$$= \iint \mathrm{Cov}_{\log \ell}(\mathbf{x}, \mathbf{x}') \bar{\ell}(\mathbf{x}) \bar{\ell}(\mathbf{x}') \mathrm{p}(\mathbf{x}) \mathrm{p}(\mathbf{x}') \mathrm{d}\mathbf{x} \mathrm{d}\mathbf{x}' \tag{6.49}$$

Which has a nice form: It is simply the standard BMC variance estimate, but scaled to be proportional to the height of $\bar{\ell}(\mathbf{x})$.

If we assume that $k_{\log \ell}(x_\star, \mathbf{x}') = h_{\log \ell} \mathcal{N}(\mathbf{x}_\star | \mathbf{x}', A_{\log \ell})$, we can continue the derivation to get a closed-form solution:

$$\mathrm{Cov}_{\log \ell}(\mathbf{x}_\star, \mathbf{x}'_\star) = k(\mathbf{x}_\star, \mathbf{x}') - k(\mathbf{x}_\star, \mathbf{x}_s) K(\mathbf{x}_s, \mathbf{x}_s)^{-1} k(\mathbf{x}_s, \mathbf{x}'_\star) \tag{6.50}$$

$$= k(\mathbf{x}_\star, \mathbf{x}') - k(\mathbf{x}_\star, \mathbf{x}_s) K_{\log \ell}^{-1} k(\mathbf{x}_s, \mathbf{x}'_\star) \tag{6.51}$$

so

$$\mathbb{V}[\hat{Z}] = \iint \mathrm{Cov}_{\log \ell}(\mathbf{x}, \mathbf{x}')\bar{\ell}(\mathbf{x})\bar{\ell}(\mathbf{x}')p(\mathbf{x})p(\mathbf{x}')d\mathbf{x}d\mathbf{x}' \tag{6.52}$$

$$= \iint \left[k(\mathbf{x}, \mathbf{x}') - k(\mathbf{x}, \mathbf{x}_s)K^{-1}_{\log \ell} k(\mathbf{x}_s, \mathbf{x}')\right] \tag{6.53}$$

$$\times \left[\mu(\mathbf{x}) + \sum_{i \in c} k_\ell(\mathbf{x}, \mathbf{x}_i)\beta_i\right] \left[\mu(\mathbf{x}') + \sum_{i' \in c} k_\ell(\mathbf{x}', \mathbf{x}_{i'})\beta_{i'}\right] p(\mathbf{x})p(\mathbf{x}')d\mathbf{x}d\mathbf{x}' \tag{6.54}$$

$$= \iint \left[h_{\log \ell}\mathcal{N}(\mathbf{x}|\mathbf{x}', A_{\log \ell}) - h^2_{\log \ell}\mathcal{N}(\mathbf{x}|\mathbf{x}_s, A_{\log \ell})\, K^{-1}_{\log \ell}\mathcal{N}(\mathbf{x}'|\mathbf{x}_s, A_{\log \ell})\right] \tag{6.55}$$

$$\times \left[\mu_\ell + h_\ell \sum_{i \in c} \mathcal{N}(\mathbf{x}|\mathbf{x}_i, A_\ell)\, \beta_i\right] \left[\mu_\ell + h_\ell \sum_{i' \in c} \mathcal{N}(\mathbf{x}'|\mathbf{x}_{i'}, A_\ell)\, \beta_{i'}\right] \tag{6.56}$$

$$\times \mathcal{N}(\mathbf{x}|\mathbf{b}, \mathbf{B})\, \mathcal{N}(\mathbf{x}'|\mathbf{b}, \mathbf{B})\, d\mathbf{x}d\mathbf{x}' \tag{6.57}$$

6.2.3 Marginal Variance

The marginal variance of $\bar{\ell}(\mathbf{x})$ when integrating over $\log \ell(\mathbf{x})$ can be found by evaluating (6.58) at a single point:

$$\mathbb{V}[\bar{\ell}(\mathbf{x})] = \mathbb{V}_{\log \ell}(\mathbf{x})\bar{\ell}(\mathbf{x})^2 \tag{6.58}$$

which is again proportional to the height of $\bar{\ell}(\mathbf{x})$, a desirable property that can now by handled in closed form.

6.3 Nonparametric Inference via Bayesian Quadrature

We extend the approximate integration method of Bayesian Quadrature to infinite structured domains, introducing a new inference method for nonparametric models.This method admits more flexible sampling schemes than Markov chains, for example active learning, and provides natural convergence diagnostics. We give conditions necessary for consistency, and show how to construct kernels between structures which take advantage of symmetries in the likelihood function. We demonstrate our inference method on both the Dirichlet process mixture model and the Indian buffet process.

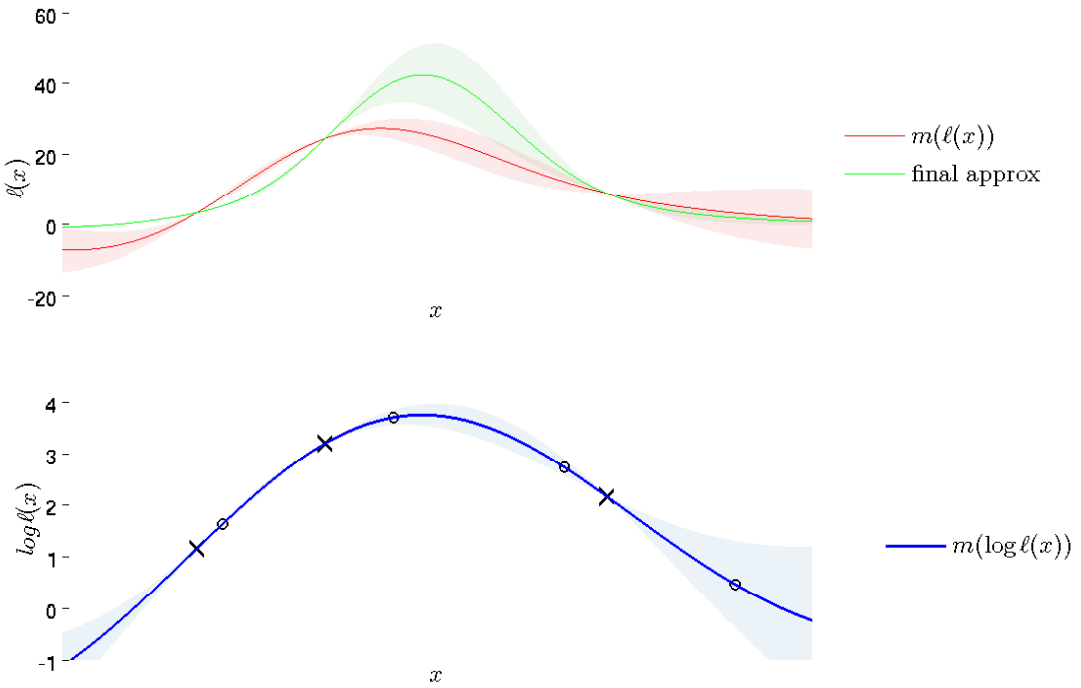


Fig. 6.2 A GP versus a log-GP

6.4 Introduction

The central problem of probabilistic inference is to compute integrals over probability distributions of the form

$$Z_{f,p} = \int f(\theta)p(\theta)d\theta \tag{6.59}$$

Examples include computing marginal distributions, making predictions while marginalizing over parameters, or computing the Bayes risk in a decision problem. Machine learning has produced a rich set of methods for computing these integrals, such as Expectation Propagation[cite], Variational Bayes[cite], and many variations of Markov chain Monte Carlo (MCMC) [cite Iain Murray].

In non-parametric models, estimating (6.59) is especially challending, as the domain of integration in is infinite-dimensional. A variety of MCMC methods have been developed to tackle this problem. However, MCMC has known weaknesses, such as difficulty diagnosing convergence, the requirement of a burn-in period, and difficulty obtaining samples from a given subset of possible θ.

Bayesian quadrature (BQ) **?**, also known as Bayesian Monte Carlo **?**, is a model-based method of approximate integration. BQ infers a posterior distribution over f conditioned on a set of samples $f(\theta_s)$, and gives a posterior distribution on $Z_{f,p}$. BQ remains a relatively unexplored integration method, and has so far only been used in low-dimensional, continuous spaces **?**.

Summary of contributions In this paper, we extend the BQ method to infinite, structured domains, introducing a new family of inference methods for non-parametric models. We give conditions necessary for consistency. We introduce kernels for inference problems using the Indian Buffet process and Dirichlet Proces mixture model which encode the many symmetries of the likelihood functions.

We then demonstrate the advantages of model-based inference on synthetic datasets, such as uncertainty estimates, flexibility in sampling methods, and post-hoc sensitivity analysis of prior and likelihood parameters. We then discuss limitations of the framework as it stands.

6.5 Bayesian Quadrature

In contrast to MCMC, a procedure which computes (6.59) in the limit of infinite samples, Bayesian quadrature is a *model-based* integration method. This means that BQ puts a

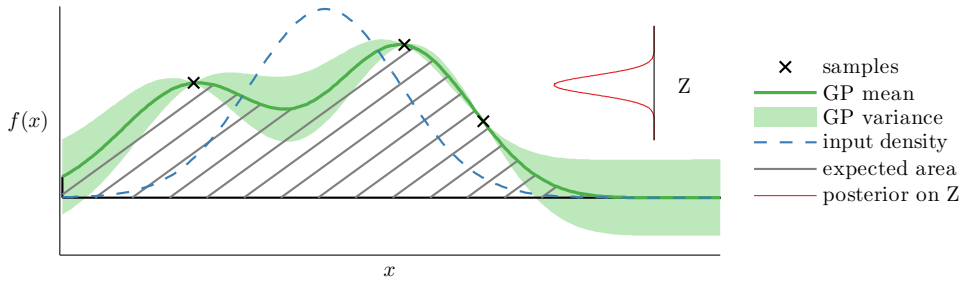


Fig. 6.3 An illustration of Bayesian quadrature. The function $f(x)$ is sampled at a set of input locations. This induces a Gaussian process posterior distribution on f, which is integrated in closed form against the target density, $p(\mathbf{x})$. Since the amount of volume under f is uncertain, this gives rise to a (Gaussian) posterior distribution over $Z_{f,p}$.

prior distribution on f, then conditions on some observations $f(\theta_s)$ at some query points θ_s. The posterior distribution over f then implies a distribution over $Z_{f,p}$, which can be obtained by integrating over all possible f. See Figure 6.3 for an illustration of Bayesian Quadrature.

It may seem circular to introduce an integral over an uncountable number of functions in order to solve what was originally an integral over a single function. However, the GPposterior has a simple form which makes integration possible in closed form in many cases. If f is assigned a Gaussian process prior with kernel function k and mean 0, then after conditioning on function evaluations $\mathbf{y} = f(\theta_1) \dots f(\theta_N)$, we obtain:

$$p(\mathbf{f}(\theta_\star)|\mathbf{y}) = \mathcal{N}\Big(\mathbf{f}(\theta_\star)\Big|\bar{\mathbf{f}}(\theta_\star), \mathrm{cov}(\theta_\star, \theta'_\star)\Big) \tag{6.60}$$

where

$$\bar{\mathbf{f}}(\mathbf{x}_\star) = k(\theta_\star, \theta_s)K^{-1}\mathbf{y} \tag{6.61}$$

$$\mathrm{cov}(\mathbf{x}_\star, \mathbf{x}'_\star) = k(\theta_\star, \theta_\star) - k(\theta_\star, \theta_s)\mathbf{K}^{-1}k(\theta_s, \theta_\star) \tag{6.62}$$

and $\mathbf{K}_{mn} = k(\theta_m, \theta_n)$. Conveniently, the GP posterior implies a closed form for the expectation and variance of (6.59):

$$\mathbb{E}\left[Z_{f,p}|\mathbf{y}\right] = \mathbb{E}_{\mathrm{GP}(f|\mathbf{y})}\left[\int f(\theta)p(\theta)d\theta\right] = \left[\int k(\theta, \theta_s)p(\theta)d\theta\right]\mathbf{K}^{-1}\mathbf{y} = \mathbf{z}^\mathsf{T}\mathbf{K}^{-1}\mathbf{y} \tag{6.63}$$

$$\mathbb{V}\left[Z_{f,p}|\mathbf{y}\right] = \mathbb{V}_{\mathrm{prior}}\left[Z_{f,p}\right] - \mathbf{z}^\mathsf{T}\mathbf{K}^{-1}\mathbf{z} \tag{6.64}$$

where

$$z_n = \int k(\theta, \theta_n) p(\theta) d\theta \tag{6.65}$$

$$\mathbb{V}_{\text{prior}}\left[Z_{f,p}\right] = \iint k(\theta, \theta') p(\theta) p(\theta') d\theta d\theta'. \tag{6.66}$$

For longer derivations, see the supplementary material. This brings us to the one of the main technical constraints of this method: Choosing a form for the kernel function $k(\theta, \theta')$ such that we can compute (6.65) and (6.66) efficiently. We must also set or integrate out any parameters of k.

6.5.1 BQ as an inference method

If we assume that in (6.59), the form of $p(\theta)$ is such that we can compute z_n, then applying BQ is straighforward. For example, in **?**, BQ was used to solve problems where $p(\theta)$ was a Gaussian prior over parameters, and $f(\theta) = p(x|\theta)$ was a likelihood function, making Z the *model evidence*, a useful quantity when comparing models.

Typically, however, we are also interested in other quantities besides the model evidence, such as marginal distributions of latent parameter. In that case, we must solve a more difficult integral, where $p(\theta) = p^*(\theta|x)$ is a possibly unnormalized distribution of unknown form.

$$\mathbb{E}_{p(\theta|\mathbf{x})}\left[f(\theta)\right] = \int f(\theta) p(\theta|x) d\theta \quad = \frac{1}{Z} \int f(\theta) p(x|\theta) p(\theta) d\theta \quad \text{where} \quad Z = \int p(x|\theta) p(\theta) d\theta \tag{6.67}$$

Integrals of this form can also be solved using Bayesian quadrature. For a thorough treatment of this problem, see [Mike's BQR paper].

This will show that BQ can be applied in several ways: For instance, if we wish to perform inference about an unknown parameter τ, we can include it as a variable to be integrated over by the GP. However, if for some technical reason this is difficult, we can also simply vary our extra parameter over some range, recomputing $\mathbf{y}(\tau)$ and obtaining a marginal distribution over $Z_{f,p}$ for each value of τ. [Reference an experiment?] This approach is useful for post-hoc sensitivity analysis.

6.6 Guidelines for Constructing a Kernel

As pointed out above, one of the most important design decisions in BQ is the choice of kernel function $k(\theta, \theta')$, which specifies the prior covariance between the values of the likelihood function $p(\mathbf{x}|\theta)$ and $p(\mathbf{x}|\theta')$. This function is somewhat analogous to the proposal distribution required to construct a Metropolis-Hastings sampler [cite]. In this section, we give guidance on how to construct an appropriate kernel.

The kernel typically should encode as much prior knowledge about the function being modeled as possible. In regression problems, this usually amounts to specifying the smoothness properties of the function being modeled. When doing inference, however, we typically know the likelihood function in closed form. The more properties of the likelihood function we can encode in the kernel, the fewer samples we will need in order to learn about the value of its integral. In particular, we should encode any known symmetries:

Symmetry Encoding: The prior correlation $\frac{k(\theta,\theta')}{\sqrt{k(\theta,\theta)}\sqrt{k(\theta',\theta')}}$ should equal 1 when $f(\theta) = f(\theta')$. That is to say, if two parameter settings are indistinguishable under the likelihood, our model can converge more quickly if it enforces that those likelihood values are identical. This is another way of saying that the covariance function should encode all known symmetries.

The ability to encode symmetries in the kernel is one of the major advantages of BQ over Monte Carlo methods. In unidentifiable models such as mixture models, often it is known that there exist many symmeteric modes in the posterior, which represents a major difficulty when computing model evidence. By encoding these symmetries in the prior over likelihood functions, BQ neatly solves the problem of identifiability when estimating $Z_{f,p}$.

6.6.1 Convergence

In the existing BQ literature [cite only 4 papers], the integration problems considered have been low-dimensional, and the kernel function used has always been, to the best of the authors' knowledge, the squared-exponential kernel. In that case, existing results on the consistency of GP regression [cite consistency] imply that, under some conditions, the BQ estimator (a linear transform of the GP posterior) is also consistent.

For infinite-dimnensional spaces with complex kernels, it is more difficult to prove consistency, although the known structure of the likelihood functions may help. In this

section, we give conditions necessary for convergence.

First, the kernel must be positive-definite **?**. In addition, in order to ensure convergence, we must have that the following condition holds:

Identifiability: The prior correlation $\frac{k(\theta,\theta')}{\sqrt{k(\theta,\theta)}\sqrt{k(\theta',\theta')}}$ must be less than 1 if it is possible that $f(\theta) \neq f(\theta')$. That is to say, if two values of the likelihood function can be different, our model must not enforce that those two function values are identical.

Proposition 1. *The above condition is necessary to guarantee convergence of the BQ posterior to the true value of Z.*

Proof sketch. Consider a prior $p(\theta) = \frac{1}{2}\delta_{\theta_1}(\theta) + \frac{1}{2}\delta_{\theta_2}(\theta)$ where $\theta_1 \neq \theta_2$. If the prior correlation between $f(\theta_1)$ and $f(\theta_2)$ is one, then after observing one of those two values, say $f(\theta_1)$ we have that $\mathbb{E}_{\mathrm{GP}}[Z] = f(\theta_1)$ and $\mathbb{V}_{\mathrm{GP}}[Z] = 0$. However if $f(\theta_1) \neq f(\theta_2)$, then $Z_0 = \frac{1}{2}f(\theta_1) + \frac{1}{2}f(\theta_2) \neq f(\theta_1)$. Thus the estimator will have converged to the wrong hypothesis. □

For a more detailed proof, see the supplementary materical. The statements in this section also hold for the problem of GP regression in general, but are specially releveant for the problem of inferring likelihood functions over latent structures. This is for two reasons: First, likelihood functions over structures can typically by shown to have many symmetries. Secondly, in the quadrature setting, we must learn about the function everywhere under the prior, not just in a small region or manifold, as is typical for regression problems. Exploiting the symmetries of the likelihood function is both possible, and necessary for fast convergence.

6.7 Inference in the Indian Buffet Process

In this section, we construct a kernel and demonstrate the use of BQ for inference in an infinite latent model, the Indian buffet process. The Indian buffet process (IBP) **?** is a distribution over binary matrices, usually used as a prior over latent features of a set of items. The IBP is nonparametric in the sense that the number of latent features is unbounded. For example, in **?**, a model of images is constructed where the entries of a binary matrix specify which objects appear in which images. Although the number of objects is unknown beforehand, given a set of images, the flexible IBP prior allows inference on both the number of objects, and their presence over the dataset.

Under an IBP prior, the probability of seeing a matrix $\mathbf{Z}$ with K columns is

$$P(\mathbf{Z}|\alpha) = \prod_{k=1}^{K} \frac{\frac{\alpha}{K}\Gamma(m_k + \frac{\alpha}{K})\Gamma(N - m_k + 1)}{\Gamma(N + 1 + \frac{\alpha}{K})} \tag{6.68}$$

where m_k is the number of ones in column k, and α is the concentration parameter. There is an unfortunate clash of notation here, where $Z_{f,p}$ denotes the model evidence, and $\mathbf{Z}$ is used to denote a binary matrix.

To fully specify a model, we must also specify the likelihood of a set of observations $\mathbf{X}$ given the latent structure, $p(\mathbf{X}|\mathbf{Z})$. For simplicity, we will use as a simple example the linear-Gaussian model from **?**. This model assumes the data is generated as $\mathbf{X} = \mathbf{A}\mathbf{Z}+\epsilon$, with $A_{ij} \sim \mathcal{N}(0, \sigma_A)$ and $\epsilon_{ij} \sim \mathcal{N}(0, \sigma_X)$. The features in $\mathbf{A}$ are turned on and off for each row of $\mathbf{X}$ by the entries of $\mathbf{Z}$. Conveniently, we can integrate out the matrix $\mathbf{A}$ to obtain a collapsed likelihood:

$$p(\mathbf{X}|\mathbf{Z}, \sigma_X, \sigma_A) = \frac{\exp\left\{-\frac{1}{2\sigma_X^2}\operatorname{tr}(\mathbf{X}^T(\mathbf{I} - \mathbf{Z}(\mathbf{Z}^T\mathbf{Z} + \frac{\sigma_X^2}{\sigma_A^2}\mathbf{E}_{)}^{-1}\mathbf{Z}^T)\mathbf{X})\right\}}{(2\pi)^{ND/2}\sigma_X^{(N-K)D}\sigma_A^{KD}|\mathbf{Z}^T\mathbf{Z} + \frac{\sigma_X^2}{\sigma_A^2}\mathbf{I}|^{\frac{D}{2}}} \tag{6.69}$$

The goal of inference in the IBP is usually do discover statistics about the matrices $\mathbf{Z}$ and $\mathbf{A}$, or to produce a predictive distribution over new rows of $\mathbf{X}$. In this paper, we will show how to [...]

6.7.1 A kernel between binary matrices

Here we give a kernel which satisfies the guidelines given in section 6.6. Since the likelihood in (6.69) does not depend on the order of columns of $\mathbf{Z}$, we will construct a kernel function $k(\mathbf{Z}, \mathbf{Z}')$ which is also invariant to permutations over columns of both $\mathbf{Z}$ and $\mathbf{Z}'$:

$$k(\mathbf{Z}, \mathbf{Z}') = \sum_{k=1}^{K}\sum_{k'=1}^{K'}\sum_{n=1}^{N} \mathbf{Z}_{n,k}\mathbf{Z}'_{n,k'} \tag{6.70}$$

This kernel has the property that, for a given number of ones in $\mathbf{Z}$ and $\mathbf{Z}'$, it attains its maximum value when every element of some permutation of columns of $\mathbf{Z}$ is equal to $\mathbf{Z}'$ (i.e., they are in the same equivalence class). [TODO: show that it achieves identifiability condition]

6.7.2 Computing z_n and the prior variance

Now that we have defined our prior and kernel, we can compute the "mini-normalization constants", $z_1, \ldots, z_n$, given by (6.65). These quantities represent the expected covariance of the likelihood of a latent structure θ', with the likelihood of another structure drawn from the prior.

If matrices $\mathbf{Z}$ and $\mathbf{Z}'$ have K and K' columns respectively, then combining (6.68) and (6.70), we have:

$$z_n(\mathbf{Z}) = \sum_{\mathbf{Z}'} k(\mathbf{Z}, \mathbf{Z}') p(\mathbf{Z}'|\alpha) = \sum_{\mathbf{Z}'} \left[\sum_{n=1}^{N} \sum_{k=1}^{K} \sum_{k'=1}^{K'} \mathbf{Z}_{n,k} \mathbf{Z}'_{n,k'} \right] \left[\prod_{k^*=1}^{K} P(\mathbf{Z}'_{:,k^*}|\alpha) \right] \tag{6.71}$$

$$= \sum_{k=1}^{K} \sum_{n=1}^{N} \mathbf{Z}_{n,k} \frac{\alpha}{1 + \frac{\alpha}{K'}} = \alpha \sum_{k=1}^{K} \sum_{n=1}^{N} \mathbf{Z}_{n,k} \qquad \text{as } K' \to \infty \tag{6.72}$$

See the supplementary material for longer derivations. We can interpret (6.72) as saying that the prior covariance of the likelihood $p(\mathbf{X}|\mathbf{Z})$ with a randomly drawn likelihood $p(\mathbf{X}|\mathbf{Z}')$ is proportional to the number of ones in $\mathbf{Z}$.

Next we compute the prior variance (6.66), which follows a similar derivation. This is the expected variance of Z before we have seen any data.

$$V_{\text{prior}} = \sum_{\mathbf{Z}'} \sum_{\mathbf{Z}} p(\mathbf{Z}|\alpha) k(\mathbf{Z}, \mathbf{Z}') p(\mathbf{Z}'|\alpha) = \frac{\alpha^2 N}{1 + \frac{\alpha}{K}} = \alpha^2 N \qquad \text{as } K \to \infty \tag{6.73}$$

This form is intuitive: it scales with the expected number of ones per row, α. As α or N approach zero, the likelihood surface can be expected to become less varied, since there will be fewer ways that individual samples of $\mathbf{Z}$ can differ.

6.8 Infinite Mixture Models

In this section we develop a more general example, placing a kernel between mixture distributions. This will allow us to perform model-based inference in Dirichlet process mixture models.

A finite mixture model with k components gives a distribution over the observations x as follows: $p(x|\pi, \theta) = \sum_{i=1}^{k} \pi_i p(x|\theta_i)$ where θ_i represents a single set of mixture parameters. By specifying the form of θ, we can perform inference in a wide variety of infinite mixture models, such as an latent Dirichlet allocation-type model, infinite mixture of regressors, or a mixture of Gaussians.

We will divide our derivation into two parts. First, we will define a kernel between multinomial distributions, and derive (6.65) and (6.66) for this kernel, leaving unspecified the kernel between individual mixture elements. Then, we will continue the derivation for an infinite mixture of Gaussians.

6.8.1 A Kernel Between Multinomial Distributions

Here we define a kernel between multinomial distributions, possibly of different dimension. Here, π represents weights which sum to one, and θ the atoms of the multinomial distribution.

$$k(\pi, \theta, \pi', \theta') = \sum_{i}^{n_\theta} \sum_{j}^{n_{\theta'}} \pi_i \pi'_j k_a(\theta_i, \theta'_j) \tag{6.74}$$

where $k_a(\theta, \theta')$ specifies the covariance between individual atoms of the distribution. Changing the order of mixture components, or splitting a given mixture component among two identical atoms, will not affect the value of this covariance function. If $k_a(\theta_i, \theta'_j)$ is a Mercer kernel, then so is (6.74).

If our mixture θ has n_θ components, then

$$\begin{aligned} z(\pi, \theta) &= \iint k(\pi, \theta, \pi', \theta') p(\theta') p(\pi') d\theta' d\pi' = \iint \left[\sum_{i=1}^{n_\theta} \sum_{j=1}^{n_{\theta'}} \pi_i \pi'_j k(\theta_i, \theta'_j) \right] \left[\prod_a^{n_{\theta'}} p(\theta'_a) \right] p(\pi') d\theta' d\pi' \\ &= \int \sum_{i=1}^{n_\theta} \sum_{j=1}^{n_{\theta'}} \pi_i \pi'_j \underbrace{\int k(\theta_i, \theta'_j) p(\theta'_j) d\theta'_j}_{z_a(\theta_i)} p(\pi') d\pi' = \sum_{i=1}^{n_\theta} \pi_i z_a(\theta_i) \end{aligned} \tag{6.75}$$

$$\mathbb{V}_{\text{prior}}\left[Z_{f,p}\right] = \iint z(\pi, \theta) p(\theta) p(\pi) d\theta d\pi = \underbrace{\int z(\theta_i) p(\theta_i) d\theta_i}_{V_a} \underbrace{\int p(\pi) \sum_{i=1}^{n_\theta} \pi_i d\pi}_{\text{sums to one}} = V_a \tag{6.76}$$

where $V_a = \iint p(\theta) k_a(\theta, \theta') p(\theta') d\theta' d\theta$ is the prior variance of k_a, which will be defined below.

Perhaps surprisingly, neither z_n nor V_{prior} depend on the number of proposed mixture components. This means that we are free to take the infinite limit $n_\theta \to \infty$. Perhaps this makes sense: The likelihood does not change if we divide up our clusters to give equivalent mixtures but having more components. These quantities also do not depend on the form of the prior over mixture components $p(\pi)$, nor the prior over individual components $p(\theta)$, as long as it factorizes over components.

6.8.2 Infinite Mixture of Gaussians

The above kernel between mixtures can be used for inference in a wide variety of infinite mixture models. For simplicity, in this paper we will use as an example the infinite mixture of Gaussians **?**:

$$p(x|\pi,\theta) = \sum_{i=1}^{k} \pi_i p(x|\theta_i) = \sum_{i=1}^{k} \pi_i \mathcal{N}(y|\mu_i, \Sigma_i) \tag{6.77}$$

where $\theta_i = \{\mu_i, \Sigma_i\}$ represent the parameters of a single Gaussian. To complete our example, all that remains is to specify a kernel k_a between individual mixture components, and to compute z_k and V_a. For simplicity, we will specify a Gaussian kernel between densities, and assume that the variance of each mixture component is identical:

$$k(\mu, \Sigma, \mu', \Sigma') = \mathcal{N}(\mu|\mu', \Sigma_k) \tag{6.78}$$

where the entries of Σ_k are kernel parameters. Next, we give (6.65) and (6.66) for this kernel:

$$z_k(\mu_i, \Sigma_i) = \iint k(\mu_i, \Sigma_i, \mu'_j, \Sigma'_j) p(\mu'_j, \Sigma'_j) d\mu'_j d\Sigma'_j = \mathcal{N}(\mu_i|\lambda, \Sigma_k + \Sigma_p) \tag{6.79}$$

$$V_a = \int z_k(\mu_i, \Sigma_i) p(\mu_i, \Sigma_i) d\mu_i d\Sigma_i = \mathcal{N}(0|0, \Sigma_k + 2\Sigma_p) \tag{6.80}$$

Note that the prior variance V_k depends only on the shape of the prior Σ_p, and not its location.

6.9 Related Work

String kernels **?**. Graph kernels. Tree-structured kernels. Sequence kernels. A review of kernels in structured domains can be found in **?**.

6.10 Experiments

Integrating Kernel Hyperparameters One complicating issue of inference using BQ is how to set kernel hyperparameters. In **?**, these were set by maximum likelihood. In our experiments, hyperparameters were integrated out numerically, except for the *output variance* (a scale factor in front of the kernel) which is possible to integrate out

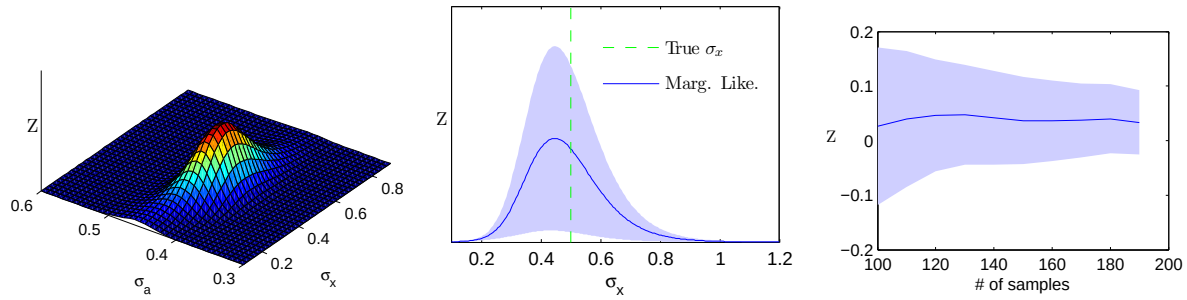


Fig. 6.4 Computing marginal likelihoods with BQ. Left: The marginal likelihood as a function of two nuisance parameters. Center: A slice of the 2-D marginal likelihood function, at $\sigma_A = 0.4$. The true value of σ_X lies roughly in the center of the likelihood function. Right: The marginal likelihood as a function of the number of evaluations of the likelihood function. The shaded error represents uncertainty about the value of the marginal likelihood.

in closed form, giving a final posterior variance of $\sigma_2 = \frac{1}{N}\left[\mathbf{y}^t\mathbf{K}^{-1}\mathbf{y}\right]\left[V_k - \mathbf{z}_t\mathbf{K}^{-1}\mathbf{z}\right]$. For a derivation, see the supplementary material.

6.10.1 IBP Experiments

We used collapsed IBP sampling code from **?** to obtain samples and likelihood values. We used data from **?**, where the true value of $\sigma_x = 0.5$. We set the number of datapoints to be small (N = 25), since this is a regime where VB is known to perform poorly **?**.

Figure 6.4 demonstrates the use of BQ, showing that a set of samples gathered under one parameter setting can be used to make inferences the likelihood of other parameter settings. This allows us to use incorrect samplers, use samples from the burn-in period, etc. For example, the likelihood function (6.69) has two "nuisance parameters", σ_X and σ_A. In [cite Finale], these parameters are simply set in an ad-hoc way. This may be acceptable, but using MCMC, it is not clear how to tell whether the answer is sensitive to these nuisance parameters without re-running the chain under several different settings. Figure 6.4 shows that BQ allows one to run a post-hoc sensitivity analysis to check whether extra parameters are important to the analysis. In addition, we recover an estimate of the certainty of our analysis, indicating whether or not the existing samples are sufficient to draw strong conclusions.

6.10.2 DP Mixture Experiments

Figure 6.4 demonstrates the use of BQ to examine sensitivity to prior parameters.

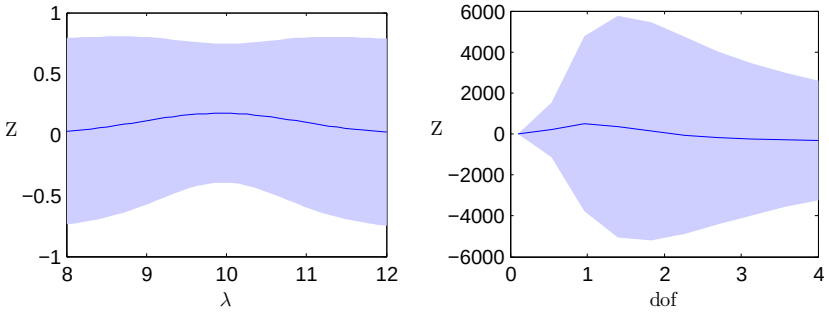


Fig. 6.5 A demonstration of computing marginal likelihoods. Left: The marginal likelihood versus the prior mean. Right: Marginal likelihood versus the degrees of freedom of the mixture components. dof = 1 is Cauchy, dof = 2 is Student's t, dof = ∞ is Gaussian. The shaded error represents uncertainty about the value of the marginal likelihood.

Figures 6.4 and 6.5 show not only the marginal likelihood of different parameter settings, but also the marginal variance of the likelihood functions. We must distinguish between two types of uncertainty represented here. First, the shape of the likelihood function indicates our posterior uncertainty over its parameter, given the data. Second, the GP posterior uncertainty in the likelihood function(represented by the shaded areas) represents our uncertainty about this likelihood function. Our uncertainty about the likelihood function can be made arbitrarily small by continuing to sample the likelihood function. However, we may still remain uncertain about the value of the latent parameter.

Code to produce all experiments will be made available at the authors' website.

6.11 Discussion

Nuisance parameters such as σ_X could also be dealt with in the BQ by adding them to the kernel and the domain of integration if desired.

6.11.1 Appropriateness of the GP prior

Placing a GP prior on the likelihood function allows us to take advantage of our knowledge of the smoothness of this function. However, there is ample reason to be believe that the GP prior is inappropriate for modeling likelihood functions. In **?** it is suggested to place the GP on the log-likelihood function, which would presumably make the additive form of the kernels given in the paper much more appropriate. This was done

by [cite BQR paper], and resulted in better performance, at the cost of a much more complicated inference procedure.

As is generally true for Bayesian methods, there exist many cases where BQ will underestimate its uncertainty. Our response to this objection is that any uncertainty estimate is better than none at all. In our experiments, we observed cases in which the model's posterior uncertainty is significant, alerting us to the fact that we have not yet observed enough about the likelihood function to be certain about its shape. In the case of MCMC, this sort of uncertainty estimate is typically unavailable. That is to say, our model cannot account for 'unknown unknowns', but it can at least account for 'known unknowns', which is a step up from the point estimates provided by MCMC.

6.12 Conclusions

We have extended Bayesian quadrature to infinite, structured domains, and demonstrated that this method can be used for inference in nonparametric models. We have given necessary conditions for convergence and examples of how to construct kernels which take advantage of the many symmetries of typical likelihood functions. We demonstrated some properties of this method, which include uncertainty estimates, flexibility in sampling methods, and the ability to re-use samples from on setting to perform post-hoc sensitivity analysis of nuisance parameters.

6.13 Gaussian Process Posteriors

Imagine we have a GP posterior distribution over functions, after having conditioned on data points $\mathbf{X}, \mathbf{f}(\mathbf{X})$.

Our posterior distribution is given by:

$$p(\mathbf{f}(\mathbf{x}_\star)|\mathbf{X}, \mathbf{f}(\mathbf{X})) = \mathcal{N}\Big(\mathbf{f}(\mathbf{x}_\star)\Big|\bar{\mathbf{f}}(\mathbf{x}_\star), \mathrm{cov}(\mathbf{x}_\star, \mathbf{x}'_\star)\Big) \tag{6.81}$$

where

$$\bar{\mathbf{f}}(\mathbf{x}_\star) = \mathbb{E}_{\mathrm{GP}}\left[f(\mathbf{x}_\star))\right] = k(\mathbf{x}_\star, \mathbf{X})K^{-1}\mathbf{f}(\mathbf{X}) \tag{6.82}$$

$$\mathrm{cov}(\mathbf{x}_\star, \mathbf{x}'_\star) = \mathbb{V}_{\mathrm{GP}}\left[f(\mathbf{x}_\star))\right] = k(\mathbf{x}_\star, \mathbf{x}_\star) - k(\mathbf{x}_\star, \mathbf{X})K^{-1}k(\mathbf{X}, \mathbf{x}_\star) \tag{6.83}$$

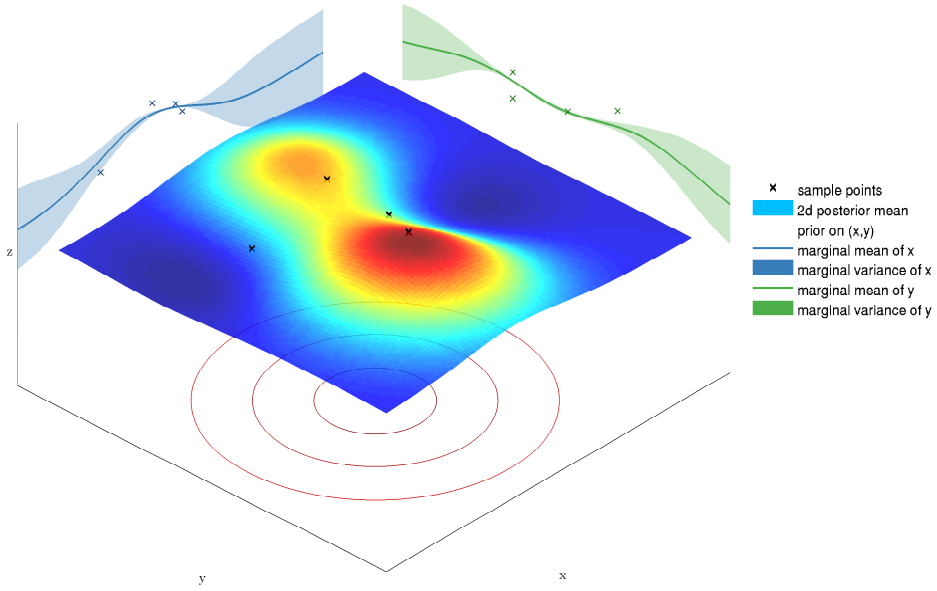


Fig. 6.6 A 2D GP posterior, and its 1D marginals.

Often, we are interested in the posterior distribution over this function, integrating over several input dimensions. Let us redefine our function to be over a set of variables $\mathbf{x}$, which we are interested in, and nuisance variables $\mathbf{y}$, which we wish to integrate out.

$$p(\mathbf{f}(\mathbf{x}_\star, \mathbf{y}_\star)|\mathbf{X}, \mathbf{Y}, \mathbf{f}(\mathbf{X}, \mathbf{Y})) = \mathcal{N}\Big(\mathbf{f}(\mathbf{x}_\star, \mathbf{y}_\star)\Big|\bar{\mathbf{f}}(\mathbf{x}_\star, \mathbf{y}_\star), \mathrm{cov}(\mathbf{x}_\star, \mathbf{y}_\star, \mathbf{x}'_\star, \mathbf{y}'_\star)\Big) \tag{6.84}$$

where

$$\bar{\mathbf{f}}(\mathbf{x}_\star, \mathbf{y}_\star) = \mathbb{E}_{\mathrm{GP}}\left[f(\mathbf{x}_\star, \mathbf{y}_\star))\right] = k(\mathbf{x}_\star, \mathbf{y}_\star, \mathbf{X}, \mathbf{Y})K^{-1}\mathbf{f}(\mathbf{X}, \mathbf{Y}) \tag{6.85}$$

$$\mathrm{cov}(\mathbf{x}_\star, \mathbf{x}_\star) = \mathbb{V}_{\mathrm{GP}}\left[f(\mathbf{x}_\star, \mathbf{y}_\star))\right] = k(\mathbf{x}_\star, \mathbf{y}_\star, \mathbf{x}_\star, \mathbf{y}_\star) - k(\mathbf{x}_\star, \mathbf{y}_\star, \mathbf{X}, \mathbf{Y})K^{-1}k(\mathbf{X}, \mathbf{Y}, \mathbf{x}_\star, \mathbf{y}_\star) \tag{6.86}$$

We are interested in the marginal

$$p(\mathbf{f}(\mathbf{x}_\star)|\mathbf{X},\mathbf{Y},\mathbf{f}(\mathbf{X},\mathbf{Y})) = \int p(\mathbf{f}(\mathbf{x}_\star,\mathbf{y})|\mathbf{X},\mathbf{Y},\mathbf{f}(\mathbf{X},\mathbf{Y}))d\mathbf{y} \tag{6.87}$$

$$= \int \mathcal{N}\Big(\mathbf{f}(\mathbf{x}_\star,\mathbf{y}_\star)\Big|\bar{\mathbf{f}}(\mathbf{x}_\star,\mathbf{y}_\star),\mathrm{cov}(\mathbf{x}_\star,\mathbf{y}_\star,\mathbf{x}'_\star,\mathbf{y}'_\star)\Big)\, d\mathbf{y} \tag{6.88}$$

6.13.1 Marginal Mean Function

Here we work out the posterior marginal mean function $\mathbb{E}\left[\mathbf{f}(\mathbf{x}_\star)\right]$:

$$\mathbb{E}_{p(f(\mathbf{x}_\star)|\mathbf{X},\mathbf{Y},\mathbf{f}(\mathbf{X},\mathbf{Y}))}\left[\mathbf{f}(\mathbf{x}_\star)\right] = \iint f(\mathbf{x}_\star,\mathbf{y})p(f(\mathbf{x}_\star,\mathbf{y})|\mathbf{X},\mathbf{Y},\mathbf{f}(\mathbf{X},\mathbf{Y}))p(\mathbf{y}|\mathbf{x}_\star)d\mathbf{y}df \tag{6.89}$$

$$= \int \mathbb{E}_{p(f(\mathbf{x}_\star,\mathbf{y})|\mathbf{X},\mathbf{Y},\mathbf{f}(\mathbf{X},\mathbf{Y}))}\left[f(\mathbf{x}_\star,\mathbf{y})\right] p(\mathbf{y}|\mathbf{x}_\star)d\mathbf{y} \tag{6.90}$$

$$= \int \bar{\mathbf{f}}(\mathbf{x}_\star,\mathbf{y})p(\mathbf{y})d\mathbf{y} \tag{6.91}$$

$$= \int k(\mathbf{x}_\star,\mathbf{y},\mathbf{X},\mathbf{Y}) \underbrace{K^{-1}\mathbf{f}(\mathbf{X},\mathbf{Y})}_{\beta} p(\mathbf{y}|\mathbf{x}_\star)d\mathbf{y} \tag{6.92}$$

$$= \sum_i \beta_i \int k(\mathbf{x}_\star,\mathbf{y},\mathbf{X}_i,\mathbf{Y}_i)p(\mathbf{y}|\mathbf{x}_\star)d\mathbf{y} \tag{6.93}$$

If we choose the kernel function and prior such that

$$k(\mathbf{x},\mathbf{y},\mathbf{x}',\mathbf{y}') = h\mathcal{N}\left(\left[\begin{array}{c}\mathbf{x}\\ \mathbf{y}\end{array}\right]\middle|\left[\begin{array}{c}\mathbf{x}'\\ \mathbf{y}'\end{array}\right], \left[\begin{array}{cc}\mathrm{I}\ell_{\mathbf{x}} & 0\\ 0 & \mathrm{I}\ell_{\mathbf{y}}\end{array}\right]\right) \tag{6.94}$$

$$p(\mathbf{x},\mathbf{y}) = \mathcal{N}\left(\left[\begin{array}{c}\mathbf{x}\\ \mathbf{y}\end{array}\right]\middle|\left[\begin{array}{c}\mu_{\mathbf{x}}\\ \mu_{\mathbf{y}}\end{array}\right], \left[\begin{array}{cc}\Sigma_{\mathbf{xx}} & \Sigma_{\mathbf{xy}}\\ \Sigma_{\mathbf{yx}} & \Sigma_{\mathbf{yy}}\end{array}\right]\right) \tag{6.95}$$

then the conditional prior on $p(\mathbf{y}|\mathbf{x}_\star)$ is:

$$p(\mathbf{y}|\mathbf{x}_\star) = \mathcal{N}\Big(\mathbf{y}\Big|\mu_{\mathbf{y}|\mathbf{x}_\star},\Sigma_{\mathbf{y}|\mathbf{x}_\star}\Big) \tag{6.96}$$

$$\mu_{\mathbf{y}|\mathbf{x}_\star} = \mu_{\mathbf{y}} + \Sigma_{\mathbf{xy}}\Sigma_{\mathbf{xx}}^{-1}(\mathbf{x}_\star - \mu_{\mathbf{x}}) \tag{6.97}$$

$$\Sigma_{\mathbf{y}|\mathbf{x}_\star} = \mu_{\mathbf{y}} + \Sigma_{\mathbf{yx}}\Sigma_{\mathbf{xx}}^{-1}\Sigma_{\mathbf{xy}} \tag{6.98}$$

then we can obtain a closed-form solution for Equation (6.93):

$$\int k(\mathbf{x}_\star, \mathbf{y}, \mathbf{X}_i, \mathbf{Y}_i) p(\mathbf{y}) d\mathbf{y} = \int h\mathcal{N}\left(\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \middle| \begin{bmatrix} \mathbf{x}' \\ \mathbf{y}' \end{bmatrix}, \begin{bmatrix} \mathrm{I}\ell_{\mathbf{x}} & 0 \\ 0 & \mathrm{I}\ell_{\mathbf{y}} \end{bmatrix} \right) \mathcal{N}(\mathbf{y}|\mu_{\mathbf{y}}, \Sigma_{\mathbf{y}})\, d\mathbf{y} \quad (6.99)$$

$$= \int h\mathcal{N}(\mathbf{x}|\mathbf{X}_i, \mathrm{I}\ell_{\mathbf{x}})\, \mathcal{N}(\mathbf{y}|\mathbf{Y}_i, \mathrm{I}\ell_{\mathbf{y}})\, \mathcal{N}(\mathbf{y}|\mu_{\mathbf{y}}, \Sigma_{\mathbf{y}})\, d\mathbf{y} \quad (6.100)$$

$$= h\mathcal{N}(\mathbf{x}|\mathbf{X}_i, \mathrm{I}\ell_{\mathbf{x}}) \int \mathcal{N}(\mathbf{y}|\mathbf{Y}_i, \mathrm{I}\ell_{\mathbf{y}})\, \mathcal{N}(\mathbf{y}|\mu_{\mathbf{y}}, \Sigma_{\mathbf{y}})\, d\mathbf{y} \quad (6.101)$$

$$= h\mathcal{N}(\mathbf{x}|\mathbf{X}_i, \mathrm{I}\ell_{\mathbf{x}})\, \mathcal{N}(\mathbf{Y}_i|\mu_{\mathbf{y}}, \mathrm{I}\ell_{\mathbf{y}} + \Sigma_{\mathbf{y}}) \quad (6.102)$$

Thus our final form for $\mathbb{E}\left[\mathbf{f}(\mathbf{x}_\star)\right]$ is:

$$\mathbb{E}_{p(\mathbf{f}(\mathbf{x}_\star)|\mathbf{X},\mathbf{Y},\mathbf{f}(\mathbf{X},\mathbf{Y}))}\left[\mathbf{f}(\mathbf{x}_\star)\right] = \sum_i \beta_i \int k(\mathbf{x}_\star, \mathbf{y}, \mathbf{X}_i, \mathbf{Y}_i) p(\mathbf{y}) d\mathbf{y} \quad (6.103)$$

$$= \sum_i \beta_i h\mathcal{N}(\mathbf{x}_\star|\mathbf{X}_i, \mathrm{I}\ell_{\mathbf{x}})\, \mathcal{N}(\mathbf{Y}_i|\mu_{\mathbf{y}}, \mathrm{I}\ell_{\mathbf{y}} + \Sigma_{\mathbf{y}}) \quad (6.104)$$

6.13.2 Marginal Variance Function

One major advantage of model-based approaches to integration is that they provide a value for the uncertainty in the given estimate. Here, we derive the marginal covariance function $\mathbb{V}\left[f(\mathbf{x}_\star), f(\mathbf{x}'_\star)\right]$:

$$\begin{aligned}
\mathrm{Cov}\left[f(\mathbf{x}_\star), f(\mathbf{x}'_\star)\right] &= \mathbb{E}_{p(f)}\left[\Big(f(\mathbf{x}_\star) - \bar{\mathbf{f}}(\mathbf{x}_\star)\Big)\Big(f(\mathbf{x}'_\star) - \bar{\mathbf{f}}(\mathbf{x}'_\star)\Big)\right] && (6.105)\\
&= \mathbb{E}_{p(f)}\Bigg[\left(\int f(\mathbf{x}_\star, \mathbf{y})p(\mathbf{y})d\mathbf{y} - \int \bar{\mathbf{f}}(\mathbf{x}_\star, \mathbf{y}')p(\mathbf{y}')d\mathbf{y}'\right) && (6.106)\\
&\qquad \times \left(\int f(\mathbf{x}'_\star, \mathbf{y})p(\mathbf{y})d\mathbf{y} - \int \bar{\mathbf{f}}(\mathbf{x}'_\star, \mathbf{y}')p(\mathbf{y}')d\mathbf{y}'\right)\Bigg] && (6.107)\\
&= \int \left(\int f(\mathbf{x}_\star, \mathbf{y})p(\mathbf{y})d\mathbf{y} - \int \bar{\mathbf{f}}(\mathbf{x}_\star, \mathbf{y}')p(\mathbf{y}')d\mathbf{y}'\right) && (6.108)\\
&\quad \times \left(\int f(\mathbf{x}'_\star, \mathbf{y})p(\mathbf{y})d\mathbf{y} - \int \bar{\mathbf{f}}(\mathbf{x}'_\star, \mathbf{y}')p(\mathbf{y}')d\mathbf{y}'\right)p(f) && (6.109)\\
&= \iiint\Big[f((\mathbf{x}_\star, \mathbf{y}) - \bar{\mathbf{f}}(\mathbf{x}_\star, \mathbf{y}')\Big]\Big[f((\mathbf{x}'_\star, \mathbf{y}) - \bar{\mathbf{f}}(\mathbf{x}'_\star, \mathbf{y}')\Big]p(\mathbf{y})p(\mathbf{y}')d\mathbf{y}d\mathbf{y}'p(f)df && (6.110)\\
&= \iiint\Big[f((\mathbf{x}_\star, \mathbf{y}) - \bar{\mathbf{f}}(\mathbf{x}_\star, \mathbf{y})\Big]\Big[f((\mathbf{x}'_\star, \mathbf{y}') - \bar{\mathbf{f}}(\mathbf{x}'_\star, \mathbf{y}')\Big]p(f)dfp(\mathbf{y})p(\mathbf{y}')d\mathbf{y}d\mathbf{y}' && (6.111)\\
&= \iint \mathrm{Cov}\left[f((\mathbf{x}_\star, \mathbf{y}), f((\mathbf{x}'_\star, \mathbf{y}')\right]p(\mathbf{y})p(\mathbf{y}')d\mathbf{y}d\mathbf{y}' && (6.112)\\
&= \iint\Big[k(\mathbf{x}_\star, \mathbf{y}, \mathbf{x}_\star, \mathbf{y}') - k(\mathbf{x}_\star, \mathbf{y}, \mathbf{X}, \mathbf{Y})K^{-1}k(\mathbf{X}, \mathbf{Y}, \mathbf{x}_\star, \mathbf{y}')\Big]p(\mathbf{y})p(\mathbf{y}')d\mathbf{y}d\mathbf{y}' && (6.113)\\
&= \iint k(\mathbf{x}_\star, \mathbf{y}, \mathbf{x}_\star, \mathbf{y}')p(\mathbf{y})p(\mathbf{y}')d\mathbf{y}d\mathbf{y}' - \iint k(\mathbf{x}_\star, \mathbf{y}, \mathbf{X}, \mathbf{Y})K^{-1}k(\mathbf{X}, \mathbf{Y}, \mathbf{x}_\star, \mathbf{y}')p(\mathbf{y})p(\mathbf{y}')d\mathbf{y} && (6.114)\\
&= t_1(\mathbf{x}_\star, \mathbf{x}'_\star) - t_2(\mathbf{x}_\star, \mathbf{x}'_\star) && (6.115)
\end{aligned}$$

Again, if we choose the kernel function and prior that consist of scaled Normal distributions, then we can obtain a closed-form solution for Equation (6.115). Here we consider

the first term, t_1:

$$t_1(\mathbf{x}_\star, \mathbf{x}'_\star) = \iint k(\mathbf{x}_\star, \mathbf{y}, \mathbf{x}_\star, \mathbf{y}') p(\mathbf{y}) p(\mathbf{y}') d\mathbf{y} d\mathbf{y}' \tag{6.116}$$

$$= \iint h\mathcal{N}\left(\begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} \middle| \begin{bmatrix} \mathbf{x}' \\ \mathbf{y}' \end{bmatrix}, \begin{bmatrix} \mathrm{I}\ell_{\mathbf{x}} & 0 \\ 0 & \mathrm{I}\ell_{\mathbf{y}} \end{bmatrix} \right) \mathcal{N}(\mathbf{y}|\mu_{\mathbf{y}}, \Sigma_{\mathbf{y}}) \mathcal{N}(\mathbf{y}'|\mu_{\mathbf{y}}, \Sigma_{\mathbf{y}}) \, d\mathbf{y} d\mathbf{y}' \tag{6.117}$$

$$= \iint h\mathcal{N}(\mathbf{x}_\star|\mathbf{x}'_\star, \mathrm{I}\ell_{\mathbf{x}}) \mathcal{N}(\mathbf{y}|\mathbf{y}', \mathrm{I}\ell_{\mathbf{y}}) \mathcal{N}(\mathbf{y}|\mu_{\mathbf{y}}, \Sigma_{\mathbf{y}}) \mathcal{N}(\mathbf{y}'|\mu_{\mathbf{y}}, \Sigma_{\mathbf{y}}) \, d\mathbf{y} d\mathbf{y}' \tag{6.118}$$

$$= h\mathcal{N}(\mathbf{x}_\star|\mathbf{x}'_\star, \mathrm{I}\ell_{\mathbf{x}}) \iint \mathcal{N}(\mathbf{y}|\mathbf{y}', \mathrm{I}\ell_{\mathbf{y}}) \mathcal{N}(\mathbf{y}|\mu_{\mathbf{y}}, \Sigma_{\mathbf{y}}) \mathcal{N}(\mathbf{y}'|\mu_{\mathbf{y}}, \Sigma_{\mathbf{y}}) \, d\mathbf{y} d\mathbf{y}' \tag{6.119}$$

$$= h\mathcal{N}(\mathbf{x}_\star|\mathbf{x}'_\star, \mathrm{I}\ell_{\mathbf{x}}) \int \mathcal{N}(\mathbf{y}'|\mu_{\mathbf{y}}, \Sigma_{\mathbf{y}}) \int \mathcal{N}(\mathbf{y}|\mathbf{y}', \mathrm{I}\ell_{\mathbf{y}}) \mathcal{N}(\mathbf{y}|\mu_{\mathbf{y}}, \Sigma_{\mathbf{y}}) \, d\mathbf{y} d\mathbf{y}' \tag{6.120}$$

$$= h\mathcal{N}(\mathbf{x}_\star|\mathbf{x}'_\star, \mathrm{I}\ell_{\mathbf{x}}) \int \mathcal{N}(\mathbf{y}'|\mu_{\mathbf{y}}, \Sigma_{\mathbf{y}}) \mathcal{N}(\mathbf{y}'|\mu_{\mathbf{y}}, \mathrm{I}\ell_{\mathbf{y}} + \Sigma_{\mathbf{y}}) \, d\mathbf{y}' \tag{6.121}$$

$$= h\mathcal{N}(\mathbf{x}_\star|\mathbf{x}'_\star, \mathrm{I}\ell_{\mathbf{x}}) \mathcal{N}(\mu_{\mathbf{y}}|\mu_{\mathbf{y}}, \mathrm{I}\ell_{\mathbf{y}} + 2\Sigma_{\mathbf{y}}) \tag{6.122}$$

$$= h\mathcal{N}(\mathbf{x}_\star|\mathbf{x}'_\star, \mathrm{I}\ell_{\mathbf{x}}) \mathcal{N}(0|0, \mathrm{I}\ell_{\mathbf{y}} + 2\Sigma_{\mathbf{y}}) \tag{6.123}$$

$t_1(\mathbf{x}_\star, \mathbf{x}'_\star)$ is called the *prior covariance*, and does not depend on the observed function values.

Next, we consider the second term, t_2:

$$t_2(\mathbf{x}_\star, \mathbf{x}'_\star) = \tag{6.124}$$

$$= \iint k(\mathbf{x}_\star, \mathbf{y}, \mathbf{X}, \mathbf{Y}) K^{-1} k(\mathbf{X}, \mathbf{Y}, \mathbf{x}'_\star, \mathbf{y}') p(\mathbf{y}) p(\mathbf{y}') d\mathbf{y} d\mathbf{y}' \tag{6.125}$$

$$= \sum_i \sum_j K^{-1}_{ij} \iint k(\mathbf{x}_\star, \mathbf{y}, \mathbf{X}_i, \mathbf{Y}_i) k(\mathbf{X}_j, \mathbf{Y}_j, \mathbf{x}'_\star, \mathbf{y}') p(\mathbf{y}) p(\mathbf{y}') d\mathbf{y} d\mathbf{y}' \tag{6.126}$$

$$= \sum_i \sum_j K^{-1}_{ij} \iint h\mathcal{N}(\mathbf{x}_\star | \mathbf{X}_i, \mathrm{I}\ell_\mathbf{x}) \mathcal{N}(\mathbf{y} | \mathbf{Y}_i, \mathrm{I}\ell_\mathbf{y}) h\mathcal{N}(\mathbf{x}'_\star | \mathbf{X}_j, \mathrm{I}\ell_\mathbf{x}) \mathcal{N}(\mathbf{y}' | \mathbf{Y}_j, \mathrm{I}\ell_\mathbf{y}) \mathcal{N}(\mathbf{y} | \mu_\mathbf{y}, \Sigma_\mathbf{y}) \mathcal{N}(\mathbf{y}' | \mu_\mathbf{y}, \Sigma_\mathbf{y}) d \tag{6.127}$$

$$= \sum_i \sum_j K^{-1}_{ij} h\mathcal{N}(\mathbf{x}_\star | \mathbf{X}_i, \mathrm{I}\ell_\mathbf{x}) h\mathcal{N}(\mathbf{x}'_\star | \mathbf{X}_j, \mathrm{I}\ell_\mathbf{x}) \iint \mathcal{N}(\mathbf{y} | \mathbf{Y}_i, \mathrm{I}\ell_\mathbf{y}) \mathcal{N}(\mathbf{y}' | \mathbf{Y}_j, \mathrm{I}\ell_\mathbf{y}) \mathcal{N}(\mathbf{y} | \mu_\mathbf{y}, \Sigma_\mathbf{y}) \mathcal{N}(\mathbf{y}' | \mu_\mathbf{y}, \Sigma_\mathbf{y}) d\mathbf{y} \tag{6.128}$$

$$= \sum_i \sum_j K^{-1}_{ij} h\mathcal{N}(\mathbf{x}_\star | \mathbf{X}_i, \mathrm{I}\ell_\mathbf{x}) h\mathcal{N}(\mathbf{x}'_\star | \mathbf{X}_j, \mathrm{I}\ell_\mathbf{x}) \int \mathcal{N}(\mathbf{y} | \mathbf{Y}_i, \mathrm{I}\ell_\mathbf{y}) \mathcal{N}(\mathbf{y} | \mu_\mathbf{y}, \Sigma_\mathbf{y}) d\mathbf{y} \int \mathcal{N}(\mathbf{y}' | \mu_\mathbf{y}, \Sigma_\mathbf{y}) \mathcal{N}(\mathbf{y} | \mathbf{Y}_j, \mathrm{I}\ell_\mathbf{y} \tag{6.129}$$

$$= \sum_i \sum_j K^{-1}_{ij} h\mathcal{N}(\mathbf{x}_\star | \mathbf{X}_i, \mathrm{I}\ell_\mathbf{x}) h\mathcal{N}(\mathbf{x}'_\star | \mathbf{X}_j, \mathrm{I}\ell_\mathbf{x}) \mathcal{N}(\mathbf{Y}_i | \mu_\mathbf{y}, \mathrm{I}\ell_\mathbf{y} + \Sigma_\mathbf{y}) \mathcal{N}(\mathbf{Y}_j | \mu_\mathbf{y}, \mathrm{I}\ell_\mathbf{y} + \Sigma_\mathbf{y}) \tag{6.130}$$

$$= \sum_i \sum_j K^{-1}_{ij} h\mathcal{N}(\mathbf{x}_\star | \mathbf{X}_i, \mathrm{I}\ell_\mathbf{x}) \mathcal{N}(\mathbf{Y}_i | \mu_\mathbf{y}, \mathrm{I}\ell_\mathbf{y} + \Sigma_\mathbf{y}) h\mathcal{N}(\mathbf{x}'_\star | \mathbf{X}_j, \mathrm{I}\ell_\mathbf{x}) \mathcal{N}(\mathbf{Y}_j | \mu_\mathbf{y}, \mathrm{I}\ell_\mathbf{y} + \Sigma_\mathbf{y}) \tag{6.131}$$

$$= h\mathcal{N}(\mathbf{x}_\star | \mathbf{X}, \mathrm{I}\ell_\mathbf{x}) \mathcal{N}(\mathbf{Y} | \mu_\mathbf{y}, \mathrm{I}\ell_\mathbf{y} + \Sigma_\mathbf{y}) K^{-1} h\mathcal{N}(\mathbf{x}'_\star | \mathbf{X}, \mathrm{I}\ell_\mathbf{x}) \mathcal{N}(\mathbf{Y} | \mu_\mathbf{y}, \mathrm{I}\ell_\mathbf{y} + \Sigma_\mathbf{y}) \tag{6.132}$$

$$= \mathbf{z}(\mathbf{x}_\star) K^{-1} \mathbf{z}(\mathbf{x}'_\star) \tag{6.133}$$

where

$$z_i(\mathbf{x}) = h\mathcal{N}(\mathbf{x} | \mathbf{X}_i, \mathrm{I}\ell_\mathbf{x}) \mathcal{N}(\mathbf{Y}_i | \mu_\mathbf{y}, \mathrm{I}\ell_\mathbf{y} + \Sigma_\mathbf{y}) \tag{6.134}$$

Thus our final form for $\mathrm{Cov}\left[f(\mathbf{x}_\star), f(\mathbf{x}'_\star)\right]$ is:

$$\mathrm{Cov}_{p(\mathbf{f}(\mathbf{x}_\star)|\mathbf{X},\mathbf{Y},\mathbf{f}(\mathbf{X},\mathbf{Y}))}\left[f(\mathbf{x}_\star), f(\mathbf{x}'_\star)\right] = t_1(\mathbf{x}_\star, \mathbf{x}'_\star) - t_2(\mathbf{x}_\star, \mathbf{x}'_\star) \tag{6.135}$$

$$= h\mathcal{N}(\mathbf{x}_\star | \mathbf{x}'_\star, \mathrm{I}\ell_\mathbf{x}) \mathcal{N}(0 | 0, \mathrm{I}\ell_\mathbf{y} + 2\Sigma_\mathbf{y}) - \mathbf{z}(\mathbf{x}_\star) K^{-1} \mathbf{z}(\mathbf{x}'_\star) \tag{6.136}$$

6.14 Objective Function

When choosing points at which to evaluate $\mathbf{f}(\mathbf{x})$ in order to learn more about the integrand, there is a question about which objective function to minimize. Standard sampling for BQ attempts to minimize the expected variance in our integral, Z. However, if we are only interested in the marginal distribution over some variable, then we might be afraid that minimizing the variance in Z is only tangentially related to what we care about.

Here, we show that minimizing the variance in Z is equivalent to minimizing the variance of the marginal function, weighted by the prior $p(\mathbf{x})$.

. . .

Note that since our posterior over Z is a Gaussian, minimizing the variance in Z is equivalent to minimizing the entropy of our distribution over Z.

Chapter 7

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.

This chapter was joint work with Oren Rippel, Ryan P. Adams and Zoubin Ghahramani.

7.1 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:

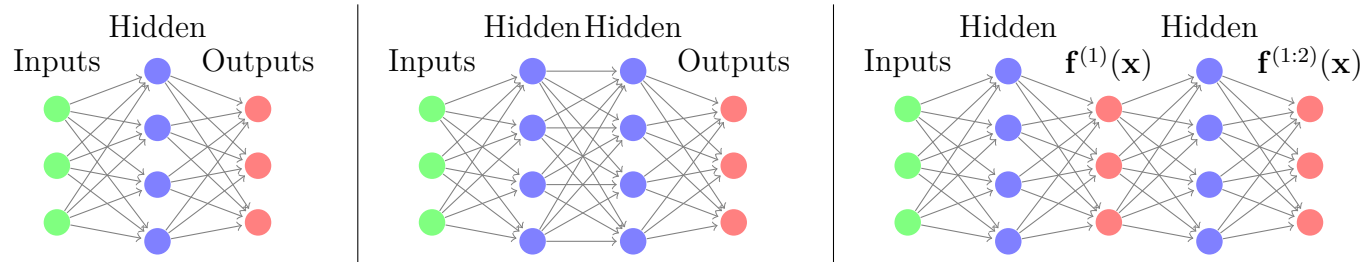
$$\mathbf{f}^{(1:L)}(\mathbf{x}) = \mathbf{f}^{(L)}(\mathbf{f}^{(L-1)}(\dots \mathbf{f}^{(2)}(\mathbf{f}^{(1)}(\mathbf{x}))\dots)) \tag{7.1}$$

$$\mathbf{f}_d^{(\ell)} \overset{\text{ind}}{\sim} \text{GP}\left(0, k_d^{\ell}(\mathbf{x}, \mathbf{x}')\right) \tag{7.2}$$

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.



a) One-hidden-layer MLP b) Two-hidden layer MLP c) Two-layer GP: $\mathbf{y} = \mathbf{f}^{(2)}\left(\mathbf{f}^{(1)}(\mathbf{x})\right)$

Fig. 7.1 GPs can be understood as one-hidden-layer MLP with infinitely many hidden units (a). There are two possible interpretations of deep GPs: We can consider the deep GP to be a neural network with a finite number of hidden units, each with a different non-parametric activation function (b). Alternatively, we can consider every second layer to be a random linear combination of an infinite number of fixed parametric hidden units (c).

7.2 Relating deep neural nets and deep Gaussian processes

7.2.1 Single-layer models

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

$$\mathbf{h}^{(1)}(\mathbf{x}) = \sigma\left(\mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}\right) \tag{7.3}$$

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:

$$f(\mathbf{x}) = \mathbf{V}^{(1)}\sigma\left(\mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}\right) = \mathbf{V}^{(1)}\mathbf{h}^{(1)}(\mathbf{x}) \tag{7.4}$$

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

$$f(\mathbf{x}) = \frac{1}{K}\alpha^{\mathsf{T}}\mathbf{h}(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K}\alpha_i h_i(\mathbf{x}), \tag{7.5}$$

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 $\mathcal{N}\Big(0, \frac{\sigma^2}{K}\sum_{i=1}^{K} h_i(\mathbf{x})h_i(\mathbf{x}')\Big)$. 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.

7.2.2 Multiple hidden layers

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

$$\mathbf{h}^{(\ell)}(\mathbf{x}) = \sigma\left(\mathbf{b}^{(\ell)} + \mathbf{W}^{(\ell)}\mathbf{h}^{(\ell-1)}(\mathbf{x})\right) . \tag{7.6}$$

This architecture is shown in figure 7.1b. For example, if we extend the model given by (7.4) to have two layers of feature mappings, the resulting model is

$$f(\mathbf{x}) = \frac{1}{K}\alpha^\mathsf{T}\mathbf{h}^{(2)}\left(\mathbf{h}^{(1)}(\mathbf{x})\right) . \tag{7.7}$$

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

$$k(\mathbf{x}, \mathbf{x}') = \left(\mathbf{h}^{(2)}(\mathbf{h}^{(1)}(\mathbf{x}))\right)^\mathsf{T}\mathbf{h}^{(2)}(\mathbf{h}^{(1)}(\mathbf{x}')) \tag{7.8}$$

These models, examined in section 2.7, 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 7.1c. The D_ℓ outputs $\mathbf{f}^{(\ell)}(\mathbf{x})$ 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 information bottlenecks. We can simply substitute (7.4) into (7.6) to get

$$\mathbf{h}^{(\ell)}(\mathbf{x}) = \sigma\left(\mathbf{b}^{(\ell)} + \mathbf{W}^{(\ell)}\mathbf{V}^{(\ell-1)}\mathbf{h}^{(\ell-1)}(\mathbf{x})\right) . \tag{7.9}$$

Thus, ignoring the intermediate outputs $\mathbf{f}^{(\ell)}(\mathbf{x})$, 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 $\mathbf{f}^{(1:\ell)}(\mathbf{x})$ 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.

7.3 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 7.4 to formally identify a pathology that emerges in very deep networks.

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

7.3.1 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:

$$k_{\text{SE}}(x, x') = \sigma^2 \exp\left(\frac{-(x-x')^2}{2w^2}\right) . \tag{7.10}$$

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

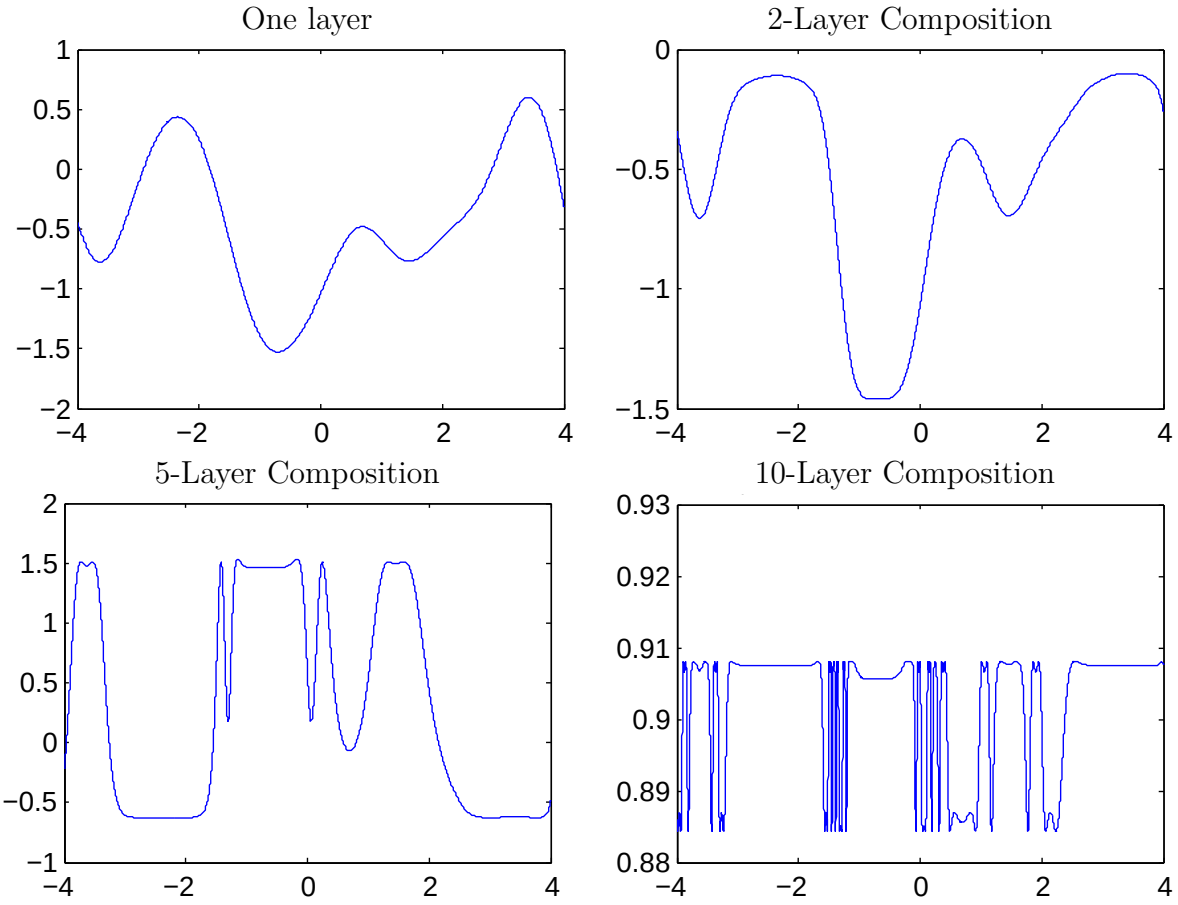


Fig. 7.2 One-dimensional draws from a deep gp prior. After a few layers, the functions begin to be either nearly flat, or highly varying, everywhere. This is a consequence of the distribution on derivatives becoming heavy-tailed.

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

$$\begin{aligned} m_{\log} &= \mathbb{E}\left[\log\left|\frac{\partial f(x)}{\partial x}\right|\right] = 2\log\left(\frac{\sigma}{w}\right) - \log 2 - \gamma \\ v_{\log} &= \mathbb{V}\left[\log\left|\frac{\partial f(x)}{\partial x}\right|\right] = \frac{\pi^2}{4} + \frac{\log^2 2}{2} - \gamma^2 - \gamma\log 4 \\ &\quad + 2\log\left(\frac{\sigma}{w}\right)\left[\gamma + \log 2 - \log\left(\frac{\sigma}{w}\right)\right] \end{aligned} \tag{7.11}$$

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

$$\begin{aligned} \log\left|\frac{\partial f^{(1:L)}(x)}{\partial x}\right| &= \sum_{\ell=1}^{L} \log\left|\frac{\partial f^{(\ell)}(x)}{\partial x}\right| \\ \implies \log\left|\frac{\partial f^{(1:L)}(x)}{\partial x}\right| &\overset{L\to\infty}{\sim} \mathcal{N}\Big(Lm_{\log}, L^2 v_{\log}\Big) \end{aligned} \tag{7.12}$$

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 7.2 shows this behavior in a draw from a 1D deep GP prior, at varying depths. This figure also shows that once the derivative in one region of the input space becomes very large or very small, it is likely to remain that way in subsequent layers.

7.3.2 Distribution of the Jacobian

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

Lemma 7.3.1. *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 **?**:

$$\operatorname{cov}\left(\frac{\partial f(\mathbf{x})}{\partial x_{d_1}}, \frac{\partial f(\mathbf{x})}{\partial x_{d_2}}\right) = \left.\frac{\partial^2 k(\mathbf{x}, \mathbf{x}')}{\partial x_{d_1} \partial x'_{d_2}}\right|_{\mathbf{x}=\mathbf{x}'} \tag{7.13}$$

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:

$$\begin{aligned} f(\mathbf{x}) &\sim \mathrm{GP}\left(0, \sigma^2 \prod_{d=1}^{D} \exp\left(-\frac{1}{2}\frac{(x_d - x'_d)^2}{w_d^2}\right)\right) \\ \implies \operatorname{cov}\left(\frac{\partial f(\mathbf{x})}{\partial x_{d_1}}, \frac{\partial f(\mathbf{x})}{\partial x_{d_2}}\right) &= \begin{cases} \frac{\sigma^2}{w_{d_1}^2} & \text{if } d_1 = d_2 \\ 0 & \text{if } d_1 \neq d_2 \end{cases} \end{aligned} \tag{7.14}$$

Lemma 7.3.2. *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 $\mathbf{f}(\mathbf{x})$ is a matrix J with elements $J_{ij} = \frac{\partial \mathbf{f}_i(\mathbf{x})}{\partial x_j}$. Because we've assumed that the GPs on each output dimension $\mathbf{f}_d(\mathbf{x})$ are independent (7.2), it follows that each row of J is independent. Lemma 7.3.1 shows that the elements of each row are independent Gaussian. Thus all entries in the Jacobian of a GP-distributed transform are independent Gaussian R.V.'s. □

Theorem 7.3.3. *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.*

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 $\mathbf{f}^{(L)}(\mathbf{f}^{(L-1)}(\ldots \mathbf{f}^{(3)}(\mathbf{f}^{(2)}(\mathbf{f}^{(1)}(\mathbf{x})))\ldots))$ is

$$J^{1:L}(\mathbf{x}) = J^L J^{(L-1)} \ldots J^3 J^2 J^1. \tag{7.15}$$

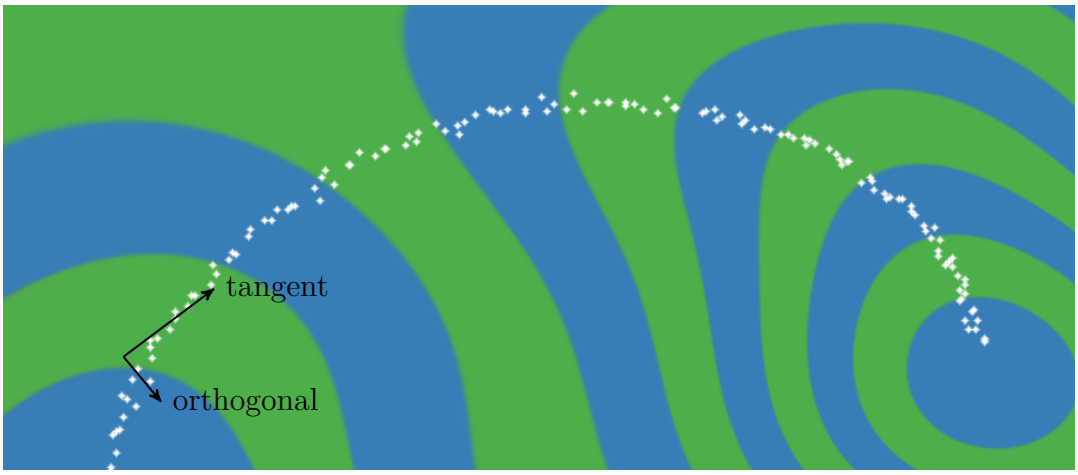


Fig. 7.3 Representing a 1-D manifold. Colors show the output of the computed representation as a function of the input space. The representation (blue & green) is invariant in directions orthogonal to the data manifold (white), making it robust to noise in those directions, and reducing the number of parameters needed to represent a datapoint. The representation also changes in directions tangent to the manifold, preserving information for later layers.

By lemma 7.3.2, each $J_{i,j}^{\ell} \overset{\text{ind}}{\sim} \mathcal{N}$, so the complete Jacobian is a product of independent Gaussian matrices, with each entry of each matrix drawn independently. □

Theorem 7.3.3 allows us to analyze the representational properties of a deep Gaussian process by simply examining the properties of products of independent Gaussian matrices, a well-studied object.

7.4 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 7.3 visualizes this idea. We follow **?** in characterizing 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 are examining are stationary, the distribution of the Jacobian is identical everywhere.

Figure 7.4 shows the singular value spectrum for 5-dimensional deep GPs of different depths. As the net gets deeper, the largest singular value dominates, implying there is usually only one effective degree of freedom in representation being computed.

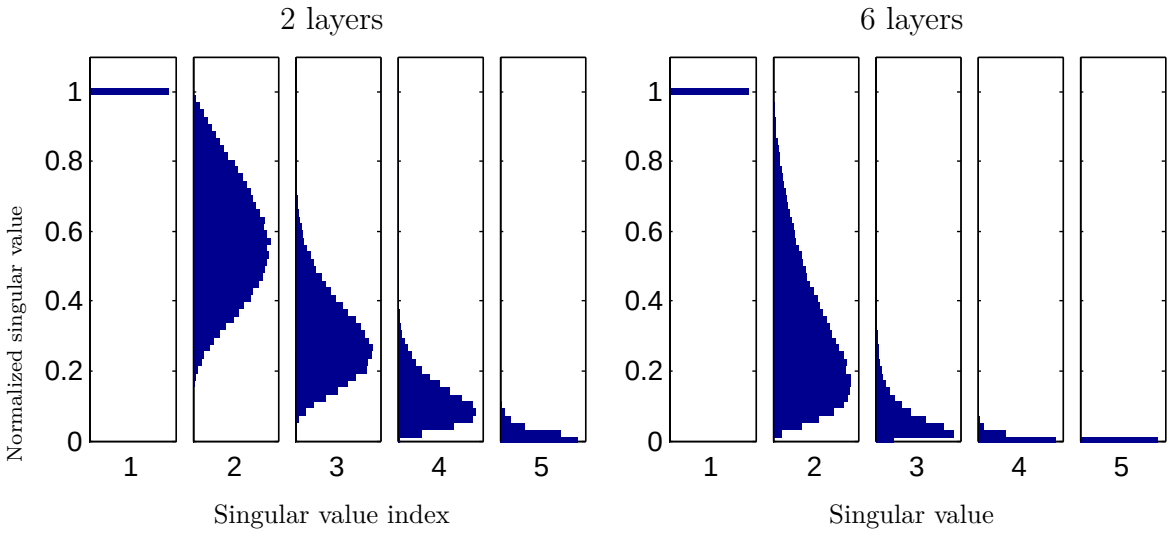


Fig. 7.4 The normalized singular value spectrum of the Jacobian of a deep GP. As the net gets deeper, the largest singular value dominates. This implies that with high probability, there is only one effective degree of freedom in the representation being computed.

Figure 7.5 demonstrates a related pathology that arises when composing functions to produce a deep density model. The density in the observed space eventually becomes locally concentrated onto one-dimensional manifolds, or *filaments*, implying that such models are insuitable to model manifolds whose underlying dimensionality is greater than one.

To visualize this pathology in another way, figure 7.6 illustrates a colour-coding of the representation computed by a deep GP, evaluated at each point in the input space. After 10 layers, we can see that locally, there is usually only one direction that one can move in $\mathbf{x}$-space in order to change the value of the computed representation. This means that such representations are likely to be unsuitable for decision tasks that depend on more than one property of the input.

To what extent are these pathologies present in nets being used today? In simulations, we found that for deep functions with a fixed latent dimension D, the singular value spectrum remained relatively flat for hundreds of layers as long as $D > 100$. Thus, these pathologies are unlikely to severely affect relatively shallow, wide networks.

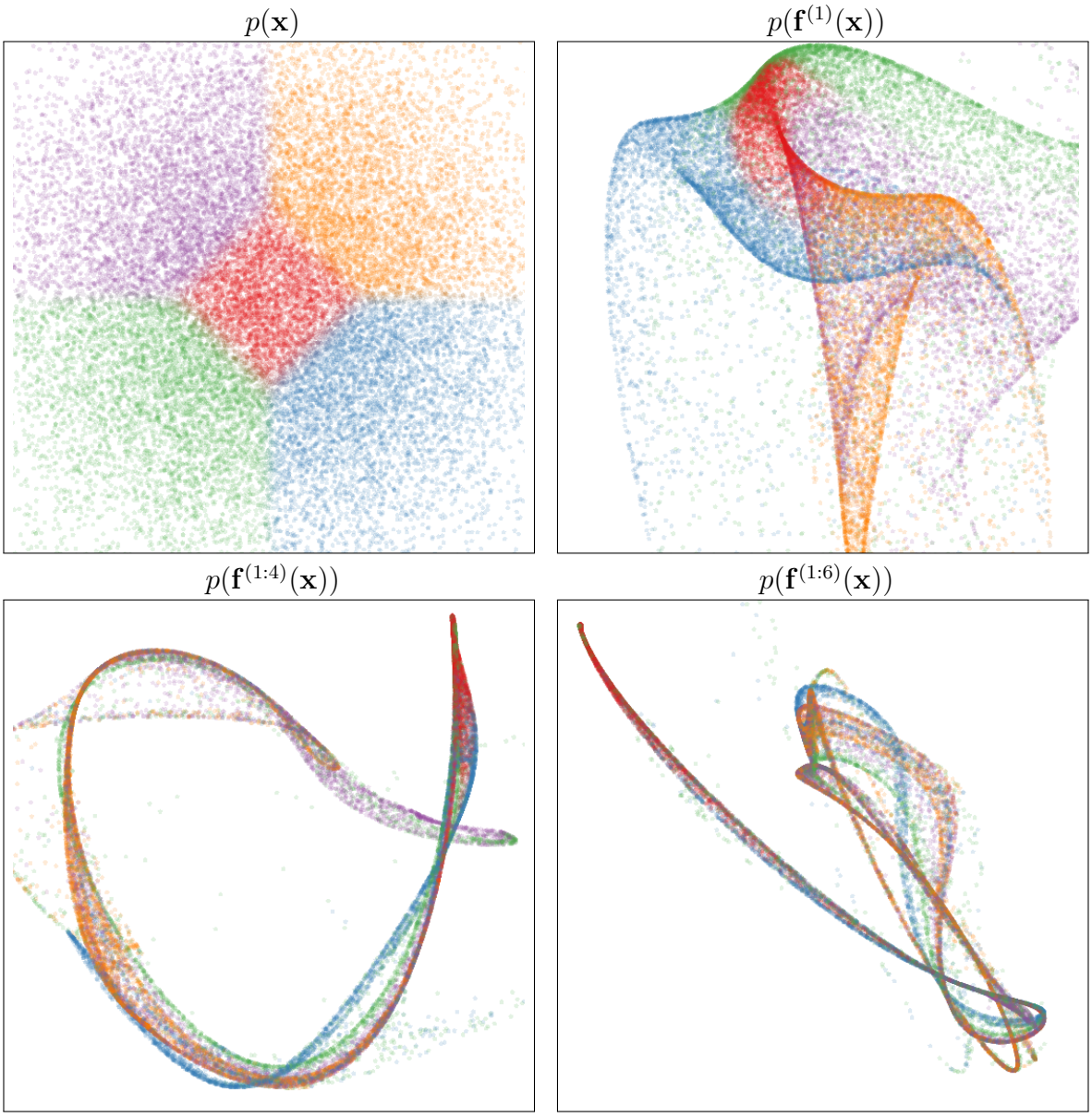


Fig. 7.5 Visualization of draws from a deep GP. A 2-dimensional Gaussian distribution (top left) is warped by successive functions drawn from a GP prior. As the number of layers increases, the density concentrates along one-dimensional filaments.

7.5 Fixing the pathology

Following a suggestion from **?**, we can fix the pathologies exhibited in figures 7.5 and 7.6 by simply making each layer depend not only on the output of the previous layer, but also on the original input $\mathbf{x}$. We refer to these models as *input-connected* networks. Figure 7.7 shows a graphical representation of the two connectivity architectures. Similar

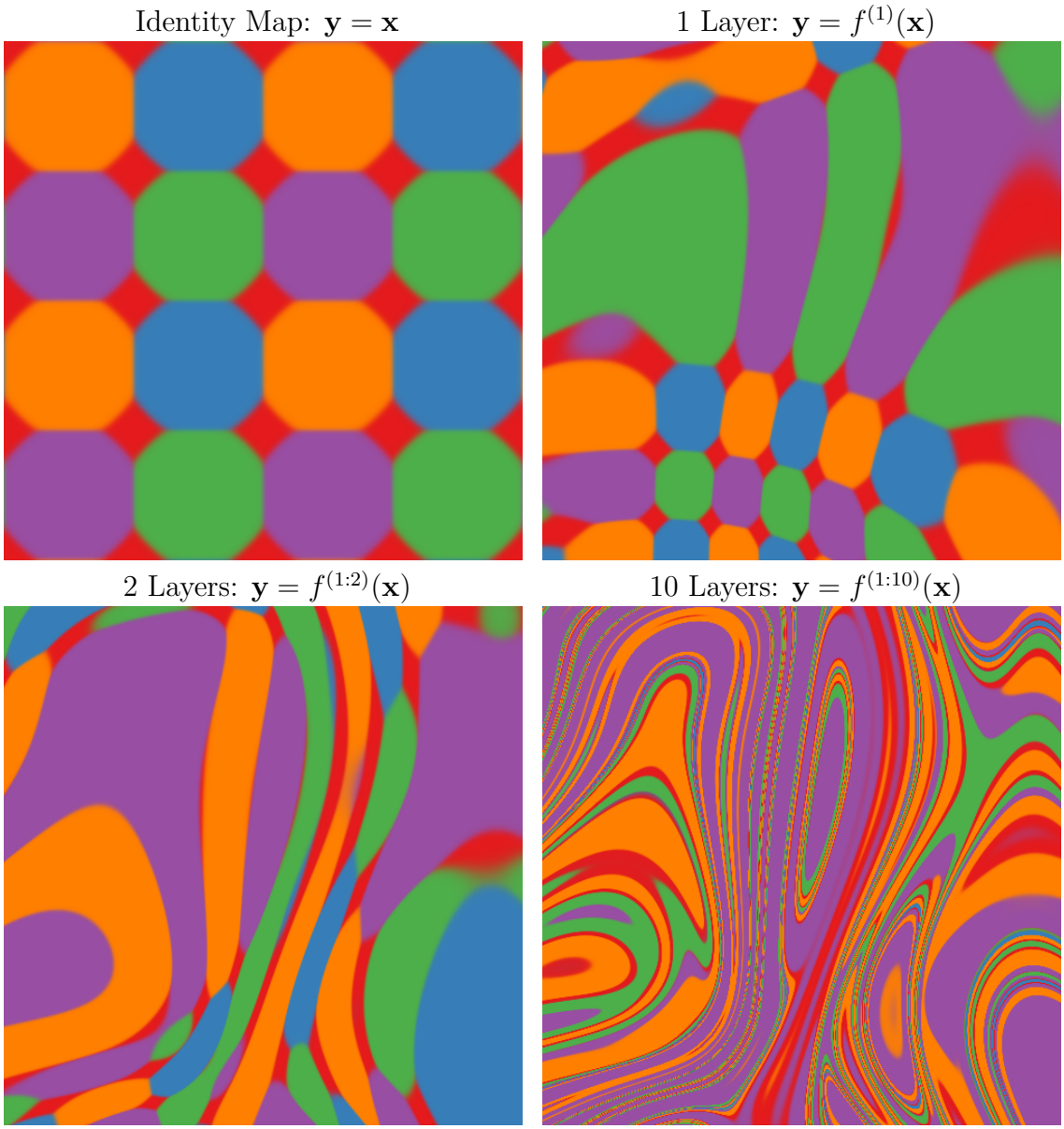
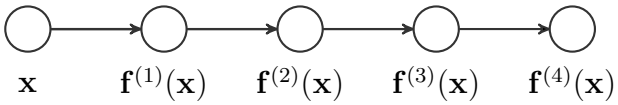
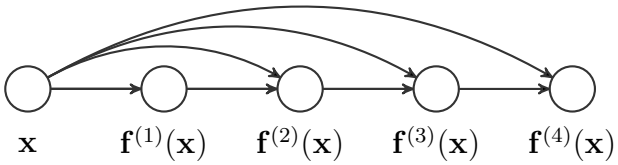


Fig. 7.6 Feature mapping of a deep GP. Colors correspond to the location $\mathbf{y} = \mathbf{f}(\mathbf{x})$ that each point is mapped to after being warped by a deep GP. The number of directions in which the color changes rapidly corresponds to the number of large singular values in the Jacobian. Just as the densities in figure 7.5 became locally one-dimensional, there is usually only one direction that one can move $\mathbf{x}$ in locally to change $\mathbf{y}$. This means that $\mathbf{f}$ is unlikely to be a suitable representation for decision tasks that depend on more than one aspect of $\mathbf{x}$.

connections between non-adjacent layers can also be found the primate visual cortex (**?**).



a) The standard MLP connectivity architecture.



b) Input-connected architecture.

Fig. 7.7 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.

Formally, this functional dependence can be written as

$$\mathbf{f}^{(1:L)}(\mathbf{x}) = \mathbf{f}^{(L)}\left(\mathbf{f}^{(1:L-1)}(\mathbf{x}), \mathbf{x}\right), \quad \forall L \tag{7.16}$$

Draws from the resulting prior are shown in figures 7.8, 7.9 and 7.11. The Jacobian of the composed, input-connected deep function is defined by the recurrence

$$J^{1:L}(\mathbf{x}) = J^{L}\begin{bmatrix} J^{1:L-1} \\ I_D \end{bmatrix}. \tag{7.17}$$

Figure 7.10 shows that with this architecture, even 50-layer deep GPs have well-behaved singular value spectra.

7.6 Related work

Deep GPs were first proposed by **?**. Variational inference in deep GPs was developed by **?**, who also analyzed the effect of automatic relevance determination in that model.

? proposed a prior on deep Bayesian networks. Their architecture has no connections except between adjacent layers, and may also be expected to have similar pathologies as 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. Such priors are a promising approach to alleviating the pathology discussed in this paper.

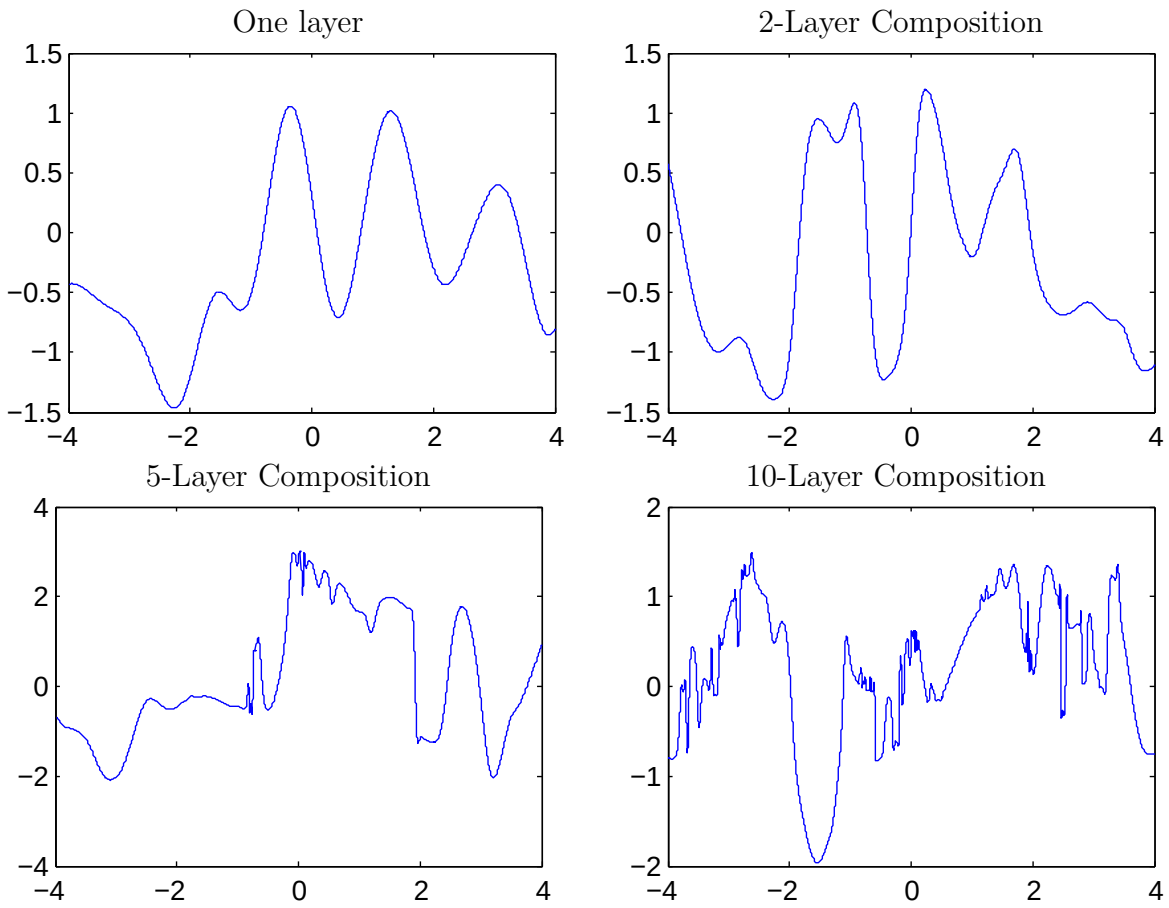


Fig. 7.8 Draws from a 1D deep GP prior with each layer connected to the input. Even after many layers, the functions remain smooth in some regions, while varying rapidly in other regions. Compare to standard-connectivity deep GP draws shown in figure 7.2.

Recurrent networks **?** and **?** analyze a related problem with gradient-based learning in recurrent nets, the "exploding-gradients" problem. **?** analyze deep kernels corresponding to recurrent neural networks.

Analysis 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.

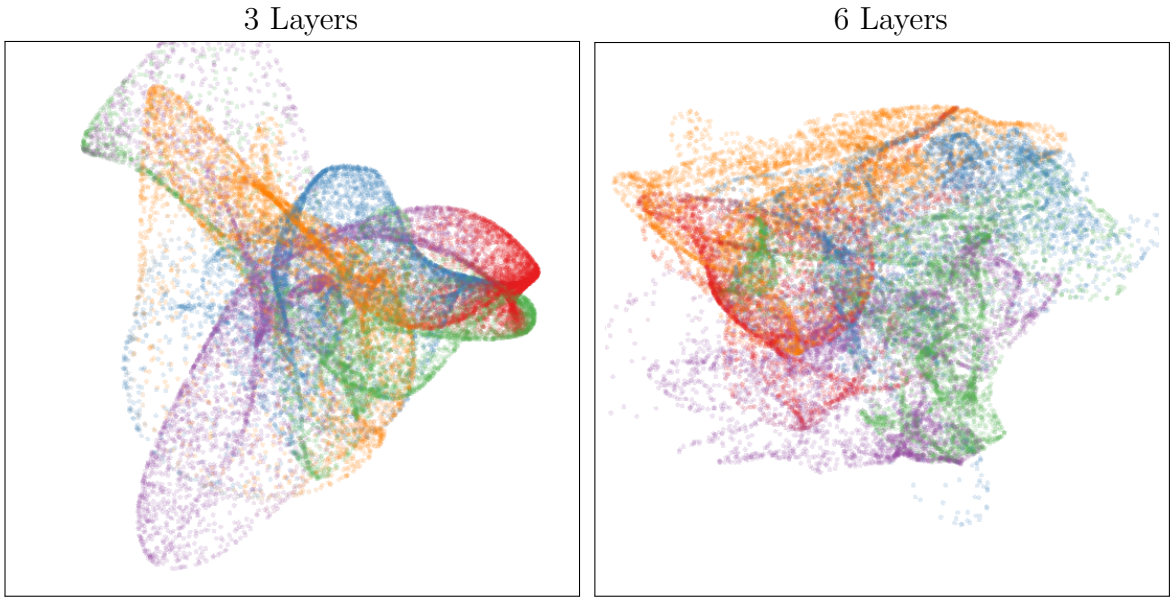


Fig. 7.9 Left: Densities defined by a draw from a deep GP, with each layer connected to the input $\mathbf{x}$. As depth increases, the density becomes more complex without concentrating along filaments.

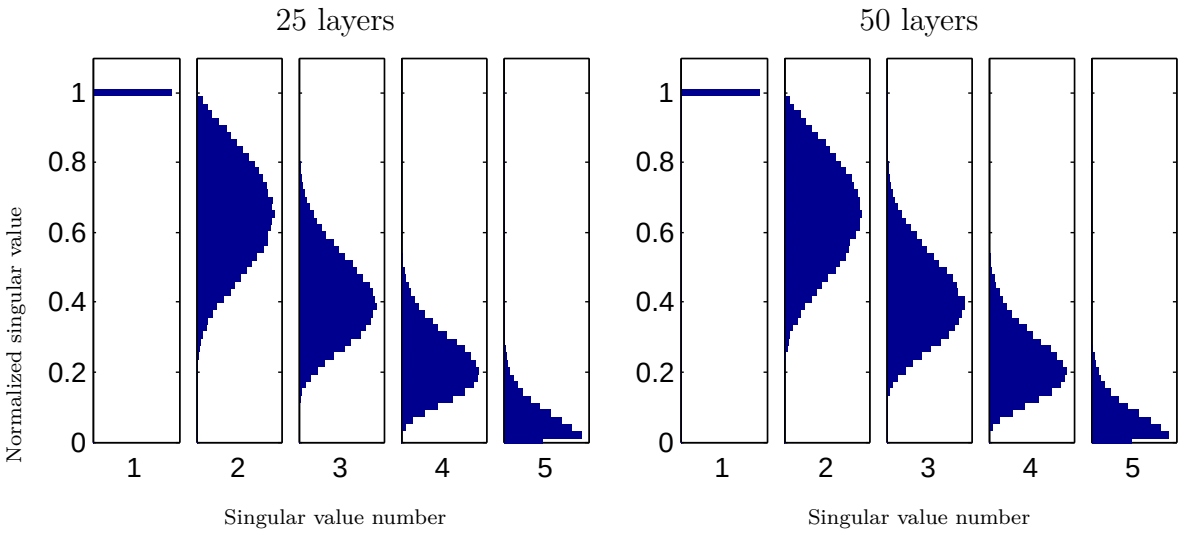


Fig. 7.10 The distribution of singular values drawn from 5-dimensional input-connected deep GP priors 25 and 50 layers deep. The singular values remain roughly the same scale as one another.

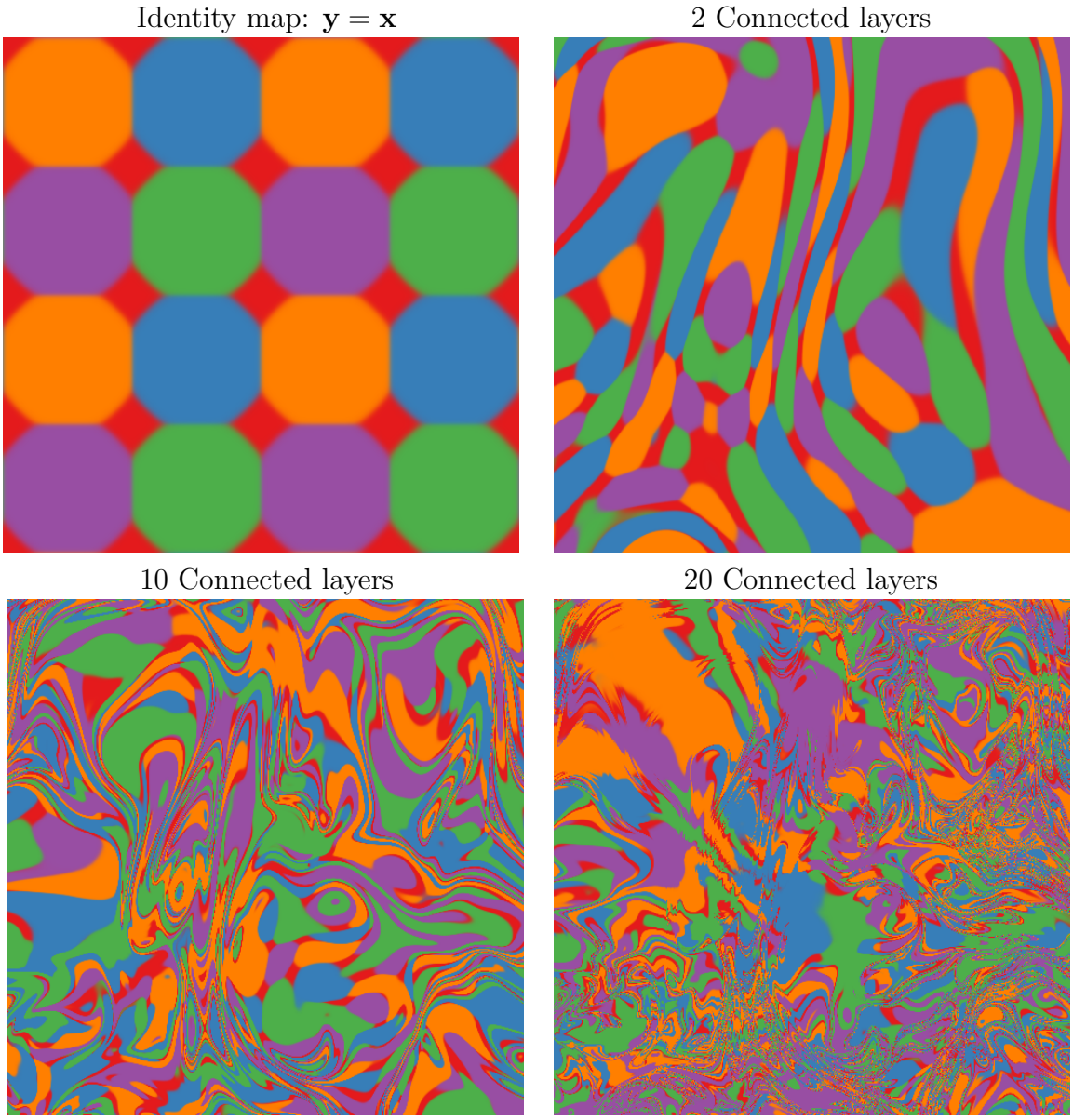


Fig. 7.11 Feature mapping of a deep GP with each layer connected to the input $\mathbf{x}$. Just as the densities in figure 7.9 remained locally two-dimensional even after many transformations, in this mapping there are often two directions that one can move locally in $\mathbf{x}$ to in order to change the values of $\mathbf{f}(\mathbf{x})$. This means that the prior puts mass on representations which sometimes depend on all aspects of the input. Compare to figure 7.6.

7.7 Conclusions

In this work, we attempted to gain insight into the properties of deep neural networks by characterizing the sorts of functions likeley to be obtained under different choices of

priors on compositions of functions.

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

We also examined properties of deep kernels, corresponding to arbitrarily many compositions of fixed features. Finally, we derived models obtained by performing dropout on Gaussian processes, finding a tractable approximation to exact dropout in GPs.