Chapter 1

Expressing Structure with Kernels

In this chapter, we'll show how to use kernels to build many different kinds of models of functions. By combining a few simple kernels through addition and multiplication, we'll be able to express many different kinds of structure: additivity, symmetry, periodicity, interactions between variables, and changepoints. We'll also show several ways to encode group invariants into kernels. Combining kernels in these simple ways will give us a rich, open-ended language of models.

1.1 Definition

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

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

$$\text{Cov}\left(f(\mathbf{x}), f(\mathbf{x}')\right) = k(\mathbf{x}, \mathbf{x}') \tag{1.1}$$

Colloquially, kernels are often said to specify the similarity between two objects. This is a slightly misleading statement in this context, since what is actually being specified is the similarity between two values of a *function* over objects. The kernel specifies which functions are likely under the GP prior, which in turn determines the generalization properties of the model.

1.2   A Few Basic Kernels

To begin understanding the types of structures expressible by GPs, we'll start by briefly examining the types of structure encoded by a diverse set of commonly used kernels: the squared-exponential (SE), periodic (Per), and linear (Lin) kernels. These kernels are defined in Table 1.1.
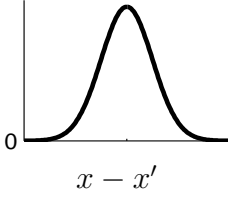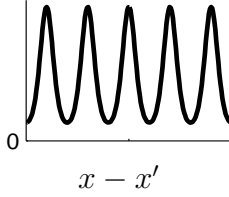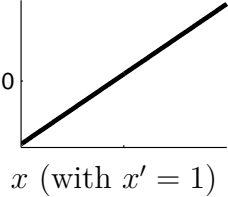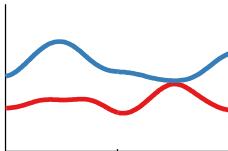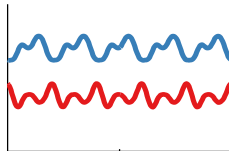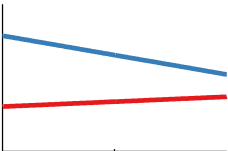
Kernel name:	Squared-exp (SE)	Periodic (Per)	Linear (Lin)
$k(x, x') =$	$\exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$	$\exp\left(-\frac{2}{\ell^2}\sin^2\left(\pi\frac{x-x'}{p}\right)\right)$	$(x-c)(x'-c)$
Plot of kernel:			
	$x - x'$	$x - x'$	x (with $x' = 1$)
	$\downarrow$	$\downarrow$	$\downarrow$
Samples from prior:			
Type of structure:	local variation	repeating structure	linear functions

Table 1.1 Examples of structures expressible by some basic kernels.

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

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

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

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

1.3   Combining Kernels

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

1.3.1   Notation

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

$$k_a + k_b = k_a(\mathbf{x}, \mathbf{x}') + k_b(\mathbf{x}, \mathbf{x}') \tag{1.2}$$

$$k_a \times k_b = k_a(\mathbf{x}, \mathbf{x}') \times k_b(\mathbf{x}, \mathbf{x}') \tag{1.3}$$

All of the basic kernels we considered in section 1.2 are one-dimensional, but kernels over multidimensional inputs can be constructed by adding and multiplying between kernels on different dimensions. The dimension on which a kernel operates is denoted by a subscripted integer. For example, SE_2 represents an SE kernel over the second dimension of $\mathbf{x}$. To remove clutter, we'll usually refer to a kernels without specifying their parameters.

1.3.2   Combining Properties through Multiplication

Multiplying two positive-definite kernels together always results in another positive-definite kernel. But what properties do these new kernels have? Figure 1.1 shows some interesting kernels that one can obtain by multiplying two basic kernels together.

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

Lin × Lin	SE × Per	Lin × SE	Lin × Per
x (with $x' = 1$)	$x - x'$	x (with $x' = 1$)	x (with $x' = 1$)
↓	↓	↓	↓
quadratic functions	locally periodic	increasing variation	growing amplitude

Fig. 1.1 Examples of one-dimensional structures expressible by multiplying kernels. Plots have same meaning as in figure 1.1.

Polynomial Regression

By multiplying together T linear kernels, we obtain a prior on polynomials of degree T. Column 1 of fig. 1.1 shows a quadratic kernel.

Locally Periodic Functions

In univariate data, multiplying a kernel by SE gives a way of converting global structure to local structure. For example, Per corresponds to exactly periodic structure, whereas Per×SE corresponds to locally periodic structure, as shown in column 2 of fig. 1.1.

Functions with Growing Amplitude

Multiplying by a linear kernel means that the marginal standard deviation of the function being modeled will grow linearly away from the origin. Columns 3 and 4 of fig. 1.1 show two examples.

More Combinations

We can multiply any number of kernels together in this way to produce kernels combining several high-level properties. For example, the kernel SE×Lin×Per is a prior on functions which are locally periodic with linearly growing amplitude. We'll see examples of real datasets with this kind of structure in section 1.10.

1.3.3   Building Flexible Multidimensional Models

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

$$\text{SE-ARD}(\mathbf{x}, \mathbf{x}') = \prod_{d=1}^{D} \exp\left(-\frac{1}{2}\frac{(x_d - x_d')^2}{\ell_d^2}\right) = \exp\left(-\frac{1}{2}\sum_{d=1}^{D}\frac{(x_d - x_d')^2}{\ell_d^2}\right) \qquad (1.4)$$

Figure 1.2 illustrates the SE-ARD kernel in two dimensions.

$$\text{SE}_1(x_1, x_1') \quad \times \quad \text{SE}_2(x_2, x_2') \quad = \quad \text{SE}_1 \times \text{SE}_2 \qquad \begin{array}{c} f(x_1, x_2) \text{ drawn from} \\ \mathcal{GP}(0, \text{SE}_1 \times \text{SE}_2) \end{array}$$
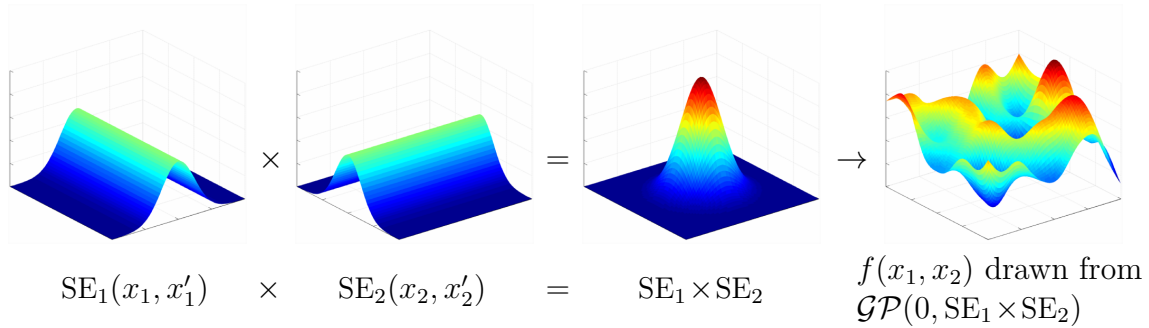
Fig. 1.2 A product of two one-dimensional kernels gives rise to a prior on functions which depend on both dimensions.

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

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

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

Lin + Per	SE + Per	SE + Lin	SE$^{(\text{long})}$ + SE$^{(\text{short})}$

x (with $x' = 1$) $x - x'$ x (with $x' = 1$) $x - x'$

↓   ↓   ↓   ↓

periodic plus trend   periodic plus noise   linear plus variation   slow & fast variation

Table 1.2 Examples of one-dimensional structures expressible by adding kernels. Rows have the same meaning as in table 1.1. SE$^{(\text{long})}$ denotes a SE kernel whose lengthscale is long relative to that of SE$^{(\text{short})}$

1.4   Modeling Sums of Functions

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

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

$$f_a \; \sim \; \mathcal{GP}(\mu_a, k_a) \tag{1.5}$$

$$f_b \; \sim \; \mathcal{GP}(\mu_b, k_b) \tag{1.6}$$

Then the sum of those functions is simply another GP:

$$f_a + f_b \; \sim \; \mathcal{GP}(\mu_a + \mu_b, k_a + k_b). \tag{1.7}$$

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

$$k_1(x_1, x_1') \qquad\qquad k_2(x_2, x_2') \qquad\qquad k_1(x_1, x_1') + k_2(x_2, x_2')$$

$$\downarrow \qquad\qquad\qquad \downarrow \qquad\qquad\qquad \downarrow$$

$$f_1(x_1) \sim \mathcal{GP}\,(0, k_1) \qquad f_2(x_2) \sim \mathcal{GP}\,(0, k_2) \qquad f_1(x_1) + f_2(x_2)$$
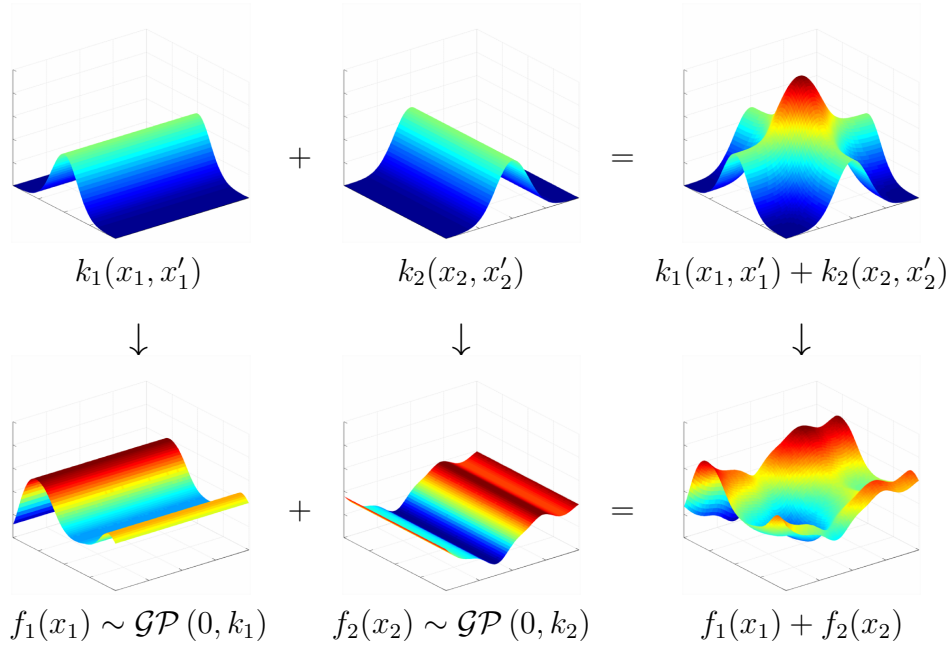
Fig. 1.3 A sum of two orthogonal one-dimensional kernels. *Top row:* An additive kernel is any sum of kernels. *Bottom row:* A draw from an additive kernel corresponds to a sum of draws from independent GP priors, each having the corresponding kernel.

1.4.1   Additivity Across Multiple Dimensions

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

$$f(x_1, x_2) \;\sim\; \mathcal{GP}(\mathbf{0}, k_1(x_1, x_1') + k_2(x_2, x_2')) \tag{1.8}$$

Then this is equivalent to the model

$$f_1(x_1) \;\sim\; \mathcal{GP}(\mathbf{0}, k_1(x_1, x_1')) \tag{1.9}$$

$$f_2(x_2) \;\sim\; \mathcal{GP}(\mathbf{0}, k_2(x_2, x_2')) \tag{1.10}$$

$$f(x_1, x_2) = f_1(x_1) + f_2(x_2) \tag{1.11}$$

Figure 1.3 illustrates a decomposition of this form. Note that the product of two kernels does not have an analogous interpretation as the product of two functions.

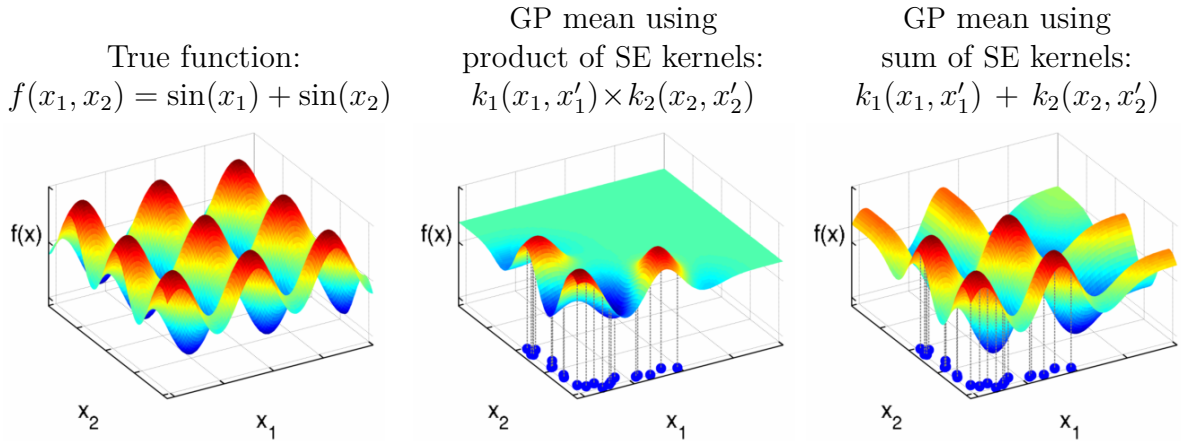Unfinished draft – compiled on Tuesday 13$^{\text{th}}$ May, 2014 at 19:45

Fig. 1.4 Inference in functions with additive structure. When the function being modeled has additive structure, we can exploit this fact to extrapolate far from the training data. The product kernel allows a different function value for every combination of inputs, and so is uncertain about function values away from the training data.

1.4.2 Long-range Extrapolation through Additivity

Additive structure often allows us to make predictions far from the training data. Figure 1.4 compares the extrapolations made by additive versus non-additive GP models, conditioned on data from a sum of two axis-aligned sine functions, evaluated in a small, L-shaped area. In this example, the additive model is able to correctly predict the height of the function at unseen combinations of inputs. The product-kernel model is more flexible, and so remains uncertain about the function away from the data.

These types of additive models have been well-explored in the statistics literature. For example, generalized additive models (Hastie and Tibshirani, 1990) have seen wide adoption. In high dimensions, we can also consider sums of functions of more than one dimension. Section 1.10 considers this model class in more detail.

1.4.3 Example: An Additive Model of Concrete Strength

To illustrate how additive kernels give rise to interpretable models, we'll build an additive model of the strength of concrete as a function of the amount of 7 different ingredients, plus the age of the concrete (Yeh, 1998).
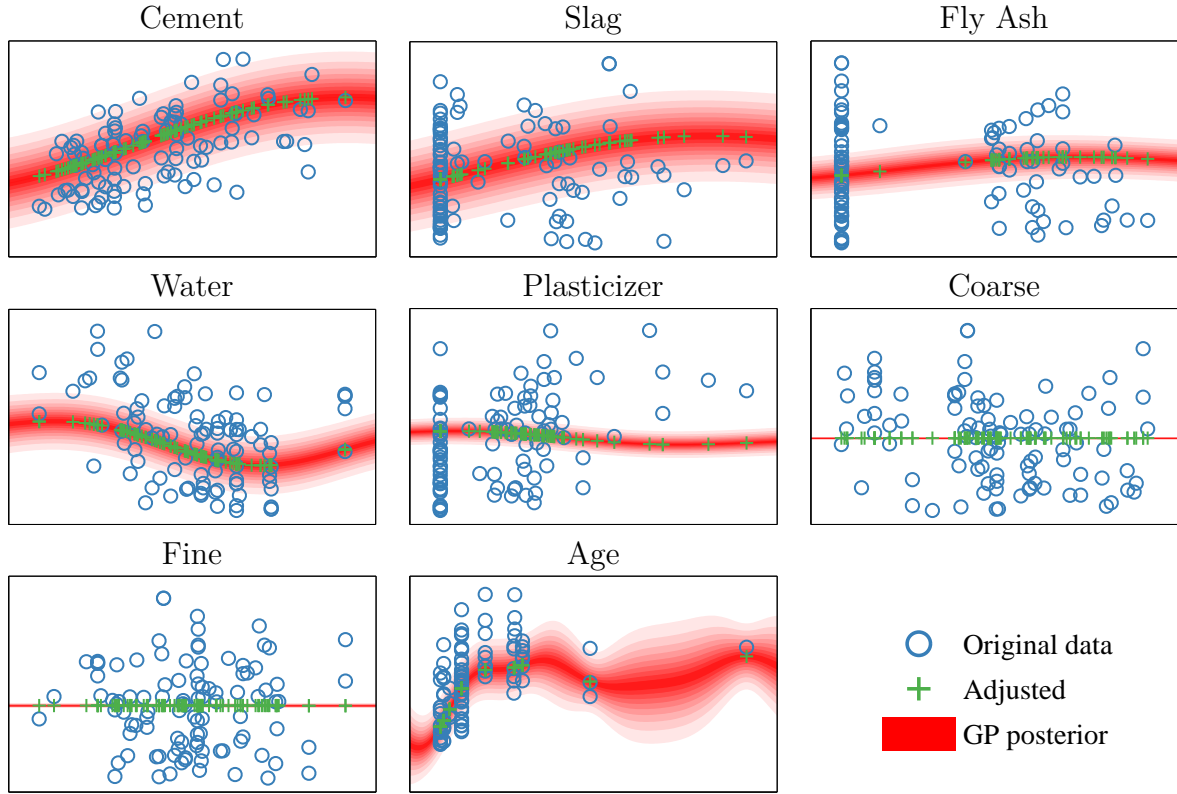
Fig. 1.5 By considering only one-dimensional terms of the additive kernel, we can plot the predictive distribution as a function of each dimension separately. Blue points indicate the original data, green points are data after the average contribution from all other terms has been subtracted. The vertical axis is the same for all plots.

Our simple additive model looks like

$$f(\mathbf{x}) = f_1(\text{cement}) + f_2(\text{slag}) + f_3(\text{fly ash}) + f_4(\text{water})$$
$$+ f_5(\text{plasticizer}) + f_6(\text{coarse}) + f_7(\text{fine}) + f_8(\text{age}) + \text{noise} \qquad (1.12)$$

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

Figure 1.5 shows the marginal posterior distribution of each of the 8 components in equation (1.12). We can see that the parameters controlling the variance of two of the components, Coarse and Fine, were set to zero, meaning that the marginal likelihood preferred a parsimonious model which did not depend on these dimensions. This is an example of the automatic sparsity that arises by maximizing marginal likelihood in GP

models, and another example of automatic relevance determination (ARD) (Neal, 1995).

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

1.4.4 Posterior Variance of Additive Components

Here we derive the posterior variance and covariance of all of the additive components of a GP. These formulas allow us to make plots such as fig. 1.5.

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

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

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

$$(1.13)$$

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

$$\mathbf{K}_1 = k_1(\mathbf{X}, \mathbf{X}) \tag{1.14}$$

$$\mathbf{K}_1^\star = k_1(\mathbf{X}^\star, \mathbf{X}) \tag{1.15}$$

$$\mathbf{K}_1^{\star\star} = k_1(\mathbf{X}^\star, \mathbf{X}^\star) \tag{1.16}$$

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

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

$$
\mathbf{K}_1^{\star\star} - {\mathbf{K}_1^\star}^{\mathsf{T}} (\mathbf{K}_1 + \mathbf{K}_2)^{-1} \mathbf{K}_1^\star \Big) \tag{1.17}
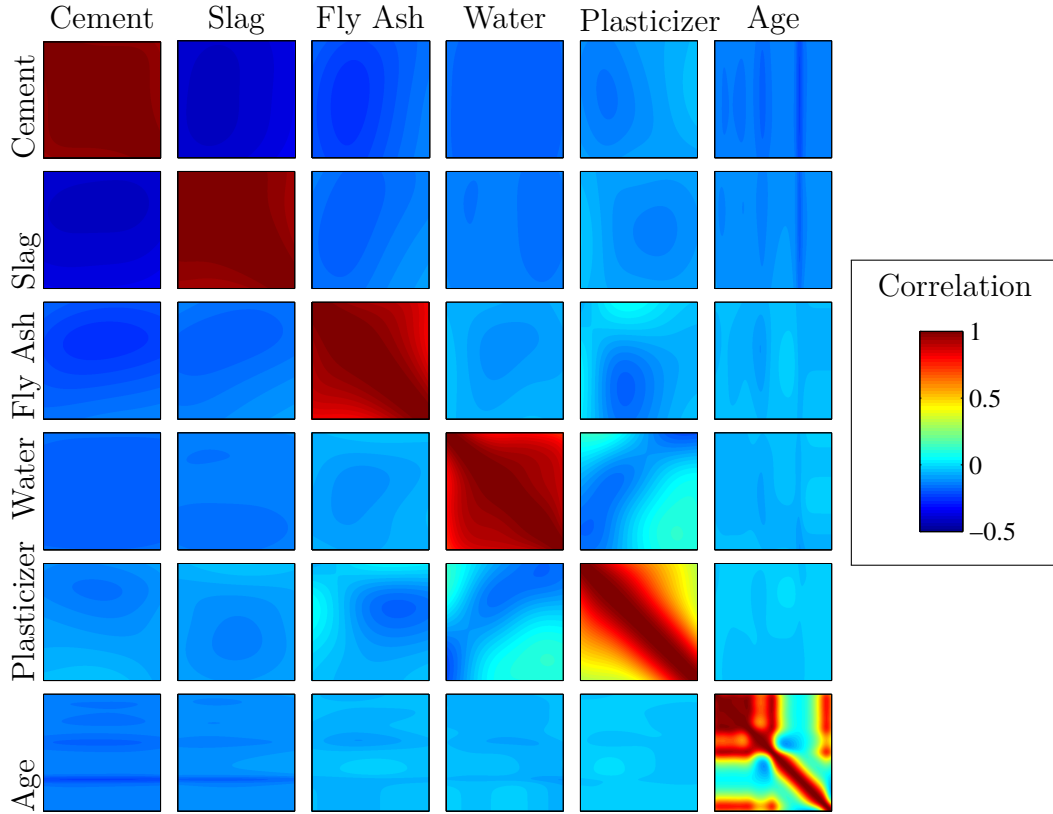$$

Fig. 1.6 Posterior correlations between the components explaining the concrete dataset. Each plot shows the additive model's posterior correlations between two components, plotted over the domain of the data ±5%. Red indicates high correlation, teal indicates no correlation, and blue indicates negative correlation. Plots on the diagonal show posterior correlations within each component.

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

Posterior Covariance of Additive Components

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

$$\text{cov}\left[\boldsymbol{f}_1(\mathbf{X}^\star), \boldsymbol{f}_2(\mathbf{X}^\star) | \boldsymbol{f}(\mathbf{X})\right] = -\mathbf{K}_1^{\star^\mathsf{T}}(\mathbf{K}_1 + \mathbf{K}_2)^{-1}\mathbf{K}_2^\star \tag{1.18}$$

Figure 1.6 shows the posterior correlation between all non-zero components of the concrete model. Most of the correlation occurs within components, but there is also negative correlation between the "Cement" and "Slag" variables. This reflects an ambiguity in the model about which one of these functions is high and the other low. Dimensions 'Coarse' and 'Fine' are not shown, because their variance was zero.

1.5   Changepoints

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

$$\mathrm{CP}(k_1, k_2)(x, x') = \sigma(x)k_1(x, x')\sigma(x') + (1 - \sigma(x))k_2(x, x')(1 - \sigma(x')) \qquad (1.19)$$

which can be written in shorthand as

$$\mathrm{CP}(k_1, k_2) = k_1 \times \boldsymbol{\sigma} \, + \, k_2 \times \bar{\boldsymbol{\sigma}} \qquad (1.20)$$

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

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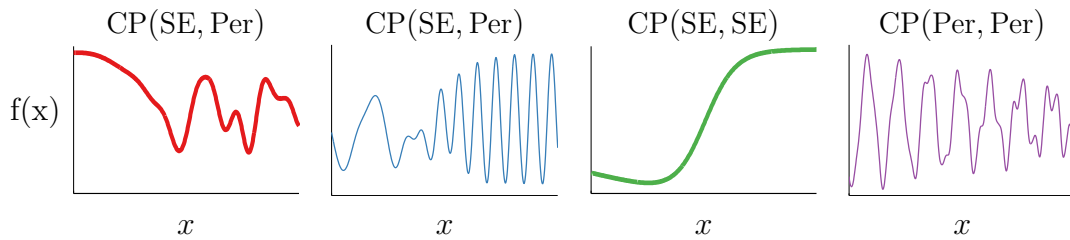

Fig. 1.7 Draws from different priors on using changepoint kernels, constructed by adding and multiplying together base kernels with sigmoidal functions.

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

1.5.1 Multiplication by a Known Function

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

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

(1.21)

1.6 Feature Representation

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

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

(1.22)

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

In some cases, $\mathcal{X}$ can even be the infinite-dimensional feature mapping of another kernel. Composing feature maps in this way leads to *deep kernels*, a topic explored in section 1.10.

1.6.1 Relation to Linear Regression

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

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

(1.23)

The link between Gaussian processes, linear regression, and neural networks is explored further in section 1.10.

1.6.2   Feature-space view of Combining Kernels

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

$$k_a(\mathbf{x}, \mathbf{x}') = \mathbf{a}(\mathbf{x})^\mathsf{T}\mathbf{a}(\mathbf{x}') \tag{1.24}$$

$$k_b(\mathbf{x}, \mathbf{x}') = \mathbf{b}(\mathbf{x})^\mathsf{T}\mathbf{b}(\mathbf{x}') \tag{1.25}$$

and we consider their addition, then

$$k_a(\mathbf{x}, \mathbf{x}') + k_b(\mathbf{x}, \mathbf{x}) = \mathbf{a}(\mathbf{x})^\mathsf{T}\mathbf{b}(\mathbf{x}') + \mathbf{a}(\mathbf{x})^\mathsf{T}\mathbf{b}(\mathbf{x}') \tag{1.26}$$

$$= \begin{bmatrix} \mathbf{a}(\mathbf{x}) \\ \mathbf{b}(\mathbf{x}) \end{bmatrix}^\mathsf{T} \begin{bmatrix} \mathbf{a}(\mathbf{x}') \\ \mathbf{b}(\mathbf{x}') \end{bmatrix} \tag{1.27}$$

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

We can examine kernel multiplication in a similar way:

$$k_a(\mathbf{x}, \mathbf{x}') \times k_b(\mathbf{x}, \mathbf{x}') = \Big[\mathbf{a}(\mathbf{x})^\mathsf{T}\mathbf{a}(\mathbf{x}')\Big] \times \Big[\mathbf{b}(\mathbf{x})^\mathsf{T}\mathbf{b}(\mathbf{x}')\Big] \tag{1.28}$$

$$= \sum_i a_i(\mathbf{x})a_i(\mathbf{x}') \times \sum_j b_j(\mathbf{x})b_j(\mathbf{x}') \tag{1.29}$$

$$= \sum_i \sum_j a_i(\mathbf{x})a_i(\mathbf{x}')b_j(\mathbf{x})b_j(\mathbf{x}') \tag{1.30}$$

$$= \sum_{i,j} \Big[a_i(\mathbf{x})b_j(\mathbf{x})\Big]\Big[a_i(\mathbf{x}')b_j(\mathbf{x}')\Big] \tag{1.31}$$

$$= \mathrm{vec}\left(\mathbf{a}(\mathbf{x}) \otimes \mathbf{b}(\mathbf{x}')\right)^\mathsf{T} \mathrm{vec}\left(\mathbf{a}(\mathbf{x}) \otimes \mathbf{b}(\mathbf{x}')\right) \tag{1.32}$$

In other words, the features of $k_a \times k_b$ are just the Cartesian product (all possible
combinations) of the original two sets of features. For example, the Cartesian product
of the features of two one-dimensional SE kernels covers the plane with two-dimensional
radial-basis functions of the form

$$h_{ij}(x_1, x_2) \propto \exp\left(-\frac{1}{2}\frac{(x_1 - c_i)^2}{2\ell_1^2}\right) \exp\left(-\frac{1}{2}\frac{(x_2 - c_j)^2}{2\ell_2^2}\right) \tag{1.33}$$

1.7   Expressing Symmetries and Invariants

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

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

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

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

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

$$k(g(\mathbf{x}), g(\mathbf{x}')) = k(\mathbf{x}, \mathbf{x}'), \quad \forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}, \quad \forall g, g' \in G \tag{1.34}$$

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

$$g_1(f(x, y)) = f(x, y) \qquad \text{(identity)} \tag{1.35}$$

$$g_2(f(x, y)) = f(y, x) \qquad \text{(swap)} \tag{1.36}$$

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

1.7.1 Three Recipes for Group Invariance

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

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

$$k_{\text{sum}}(\mathbf{x}, \mathbf{x}') = \sum_{g, \in G} \sum_{g' \in G} k(g(\mathbf{x}), g'(\mathbf{x}')) \tag{1.37}$$

Additive method   Projection method   Product method

$$\begin{array}{ccc} \mathrm{SE}(x,y,x',y') & \mathrm{SE}(\min(x,y),\max(x,y), & \mathrm{SE}(x,y,x',y') \\ +\,\mathrm{SE}(x,y,y',x') & \min(x',y'),\max(x',y')) & \times\,\mathrm{SE}(x,y,y',x') \end{array}$$
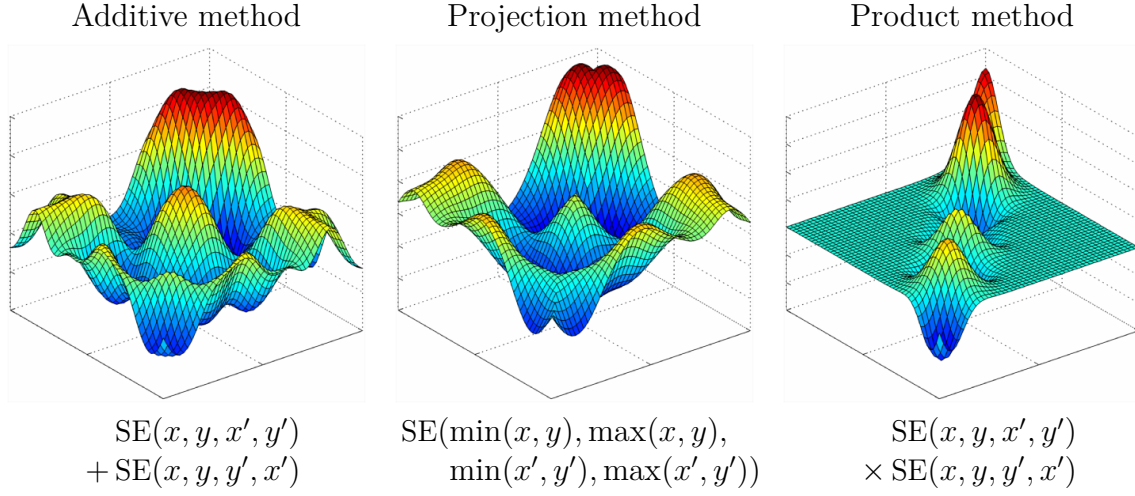
Fig. 1.8 Three methods of introducing symmetry, illustrated through draws from the corresponding priors. All three methods introduce a different type of nonstationarity.

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

$$k_{\mathrm{switch}}(\mathbf{x},\mathbf{x}') = \sum_{g,\in G_{\mathrm{swap}}} \sum_{g'\in G_{\mathrm{swap}}} k(g(\mathbf{x}),g'(\mathbf{x}')) \tag{1.38}$$

$$= k(x,y,x',y') + k(x,y,y',x') + k(y,x,x',y') + k(y,x,y',x') \tag{1.39}$$

For stationary kernels, some pairs of elements in this sum will be identical, and can be ignored. Figure 1.8(a) shows a draw from a GP prior with an SE kernel symmetrized in this way. This construction has the property that the marginal variance is doubled near $x = y$, which may or may not be desirable.

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

$$k_{\mathrm{proj}}(\mathbf{x},\mathbf{x}') = k(A_G(\mathbf{x}),A_G(\mathbf{x}')) \tag{1.40}$$

For the group G_{swap}, a fundamental domain is $\{x,y : x < y\}$, which can be mapped to using $A_{G_{\mathrm{swap}}}(x,y) = \big[\min(x,y),\max(x,y)\big]$. Constructing a kernel using this method introduces a non-differentiable "seam" along $x = y$, as shown in fig. 1.8(b). The projection method also works for infinite groups, as we shall see below.

3. **Multiply over the Orbit.** Ryan P. Adams (personal communication) suggests a construction enforcing invariants through products over the orbits:

$$k_{\text{sum}}(\mathbf{x}, \mathbf{x}') = \prod_{g, \in G} \prod_{g' \in G} k(g(\mathbf{x}), g'(\mathbf{x}')) \tag{1.41}$$

This method can sometimes produce GP priors with zero variance in some regions of the space, as in fig. 1.8(c).

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

1.7.2 Periodicity

Periodicity in a one-dimensional function corresponds to the invariance

$$f(x) = f(x + \tau) \tag{1.42}$$

where τ is the period.

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

$$\text{Per}(x, x') = \text{SE}\left(\left[\sin(x), \cos(x)\right], \left[\sin(x'), \cos(x')\right]\right) \tag{1.43}$$

achieves the invariance in Equation (1.42). Simple algebra reduces this kernel to the form shown in Table 1.1.

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

1.7.3 Symmetry About Zero

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

$$f(x) = f(-x) \tag{1.44}$$

using the sum over orbits method, by the transform

$$k_{\text{reflect}}(x, x') = k(x, x') + k(x, -x') + k(-x, x') + k(-x, -x') \qquad (1.45)$$

1.7.4 Translation Invariance in Images

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

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

$$f\left(\ \blacksquare\ \right) = f\left(\ \blacksquare\ \right) \qquad (1.46)$$

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

$$\text{shift}(\mathbf{x}, i) = \left[x_{\text{mod}(i+1,D)}, x_{\text{mod}(i+2,D)}, \ldots, x_{\text{mod}(i+D,D)} \right]^{\mathsf{T}} \qquad (1.47)$$

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

$$k_{\text{invariant}}(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{D} \sum_{j=1}^{D} k(\text{shift}(\mathbf{x}, i), \text{shift}(\mathbf{x}, j)), \qquad (1.48)$$

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

The extension to two dimensions, $\text{shift}(\mathbf{x}, i, j)$ is straightforward, but notationally cumbersome. Kondor (2008) built a more elaborate kernel between images, approximately invariant to both translation and rotation by using the projection method.
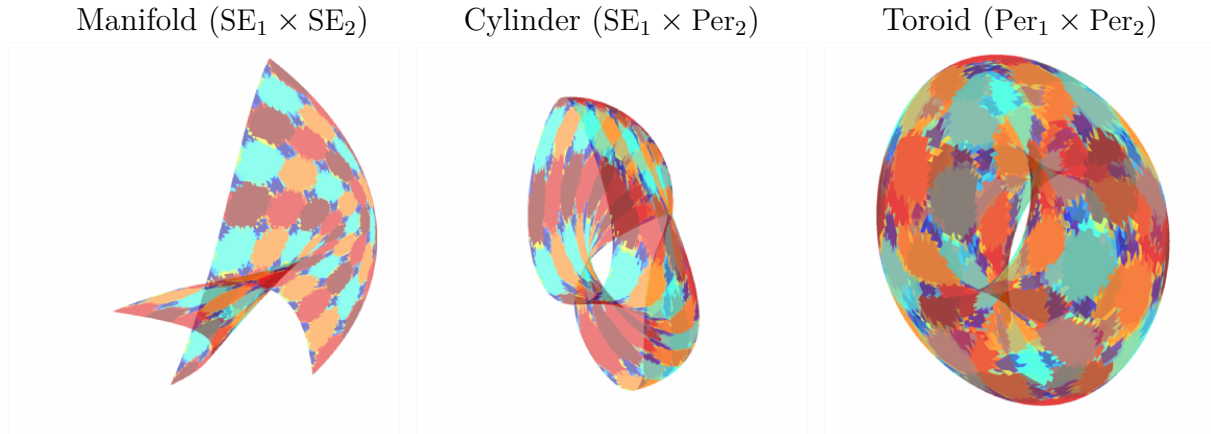
Manifold ($\text{SE}_1 \times \text{SE}_2$)　　　Cylinder ($\text{SE}_1 \times \text{Per}_2$)　　　Toroid ($\text{Per}_1 \times \text{Per}_2$)

Fig. 1.9 Generating 2D manifolds with different topological structures. By enforcing that the functions mapping from $\mathbb{R}^2$ to $\mathbb{R}^3$ obey the appropriate symmetries, the surfaces created have the corresponding topologies, ignoring self-intersections.

1.8　Generating Topological Manifolds

In this section we give a geometric illustration of the symmetries encoded by different compositions of kernels. The work presented in this section is based on a collaboration with David Reshef, Roger Grosse, Josh Tenenbaum, and Zoubin Ghahramani.

Priors on functions exhibiting symmetries can be used to create a prior on topological manifolds, by warping a simply-connected surface into a higher-dimensional space using a mapping expressing symmetries. To build a prior on 2-dimensional manifolds embedded in 3-dimensional space, we simply need a prior on mappings from $\mathbb{R}^2$ to $\mathbb{R}^3$, which we can construct using 3 independent functions $[f_1(\mathbf{x}), f_2(\mathbf{x}), f_3(\mathbf{x})]$, each mapping from $\mathbb{R}^2$ to $\mathbb{R}$. GP priors on the mapping functions implictly give rise to a prior on warped surfaces. Symmetries in $[f_1, f_2, f_3]$ will connect different parts of the manifolds, giving rise to non-trivial topologies on the sampled surfaces. Figure 1.9 shows 2D meshes warped into 3D by functions drawn from GP priors with different kernels, giving rise to a variety of different topologies.

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

Draw from GP prior:
$\mathrm{Per}(x_1, x_1') \times \mathrm{Per}(x_2, x_2')$
$+ \mathrm{Per}(x_1, x_2') \times \mathrm{Per}(x_2, x_1')$

Möbius strip
drawn from manifold prior

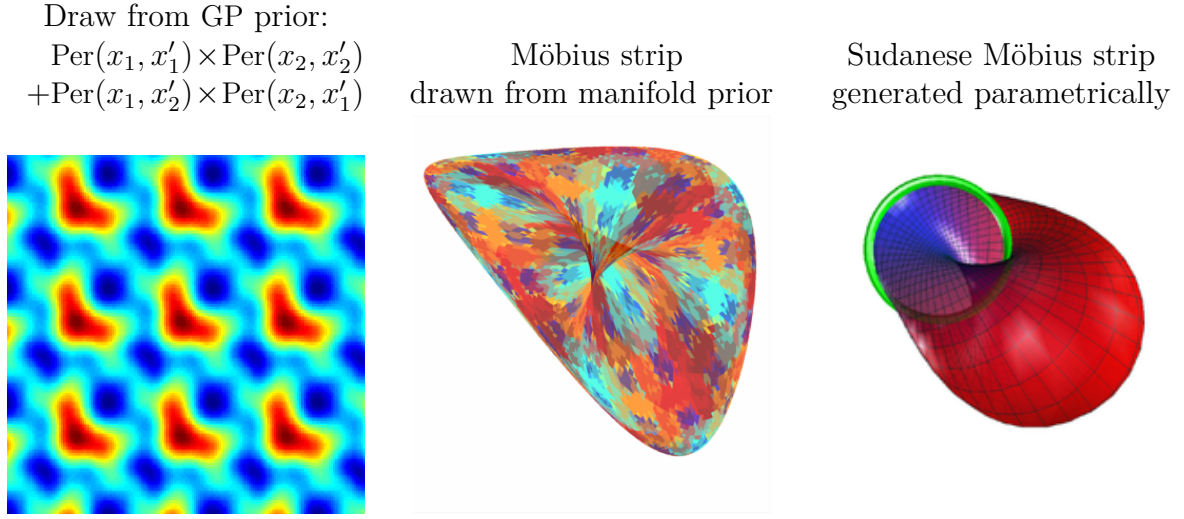Sudanese Möbius strip
generated parametrically

Fig. 1.10 Generating Möbius strips. *Left:* A function drawn from a GP prior obeying the same symmetries as a Möbius strip. *Center:* By enforcing that the functions mapping from $\mathbb{R}^2$ to $\mathbb{R}^3$ obey the appropriate symmetries, surfaces sampled from the prior have topology corresponding to a Möbius strip. Surfaces generated this way do not have the familiar shape of a flat surface connected to itself with a half-twist. Instead, they tend to look like *Sudanese* Möbius strips (Lerner and Asimov, 1984), whose edge has a circular shape. *Right:* A Sudanese projection of a Möbius strip. Image adapted from (Commons, 2005).

1.8.1   Möbius Strips

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

$$f(x, y) = f(x, y + \tau_y) \tag{1.49}$$

$$f(x, y) = f(x + \tau_x, y) \tag{1.50}$$

$$f(x, y) = f(y, x) \tag{1.51}$$

Moving along the diagonal $x = y$ of a function drawn from the corresponding GP prior is equivalent to moving along the edge of a notional Möbius strip which has had the function mapped on to its surface. Figure 1.10a shows an example of a function drawn from such a prior.

Figure 1.10(b) shows an example of a 2D mesh mapped to 3D by functions drawn from such a prior. This surface doesn't resemble the typical representation of a Möbius strip, but instead resembles an embedding known as the Sudanese Möbius strip (Lerner and Asimov, 1984), shown in Figure 1.10(c).

1.9 Kernels on Categorical Variables

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

$$k_{\text{categorical}}(x, x') = \text{SE-ARD}(\text{one-of-k}(x), \text{one-of-k}(x')) \tag{1.52}$$

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

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

$$\text{SE-full}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}\mathbf{x}^{\mathsf{T}}\mathbf{L}\mathbf{x}'\right) \tag{1.53}$$

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

1.10 Building a Kernel in Practice

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

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

Source Code

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

References

Richard Bellman. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences of the United States of America*, 42(10):767, 1956.
(page 5)

Salomon Bochner. *Lectures on Fourier integrals*, volume 42. Princeton University Press, 1959.
(page 13)

Wikimedia Commons. Stereographic projection of a Sudanese Möbius band, 2005. URL http://commons.wikimedia.org/wiki/File:MobiusSnail2B.png.
(page 20)

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20 (3):273–297, 1995.
(page 10)

D. Ginsbourger, O. Roustant, and N. Durrande. Invariances of random fields paths, with applications in Gaussian process regression. Technical Report arXiv:1308.1359 [math.ST], August 2013.
(pages 15 and 16)

David Ginsbourger, Xavier Bay, Olivier Roustant, and Laurent Carraro. Argumentwise invariant kernels for the approximation of invariant functions. In *Annales de la Faculté de Sciences de Toulouse*, number 3, 2012.
(page 15)

Noah D Goodman, Tomer D Ullman, and Joshua B Tenenbaum. Learning a theory of causality. *Psychological review*, 118(1):110, 2011.
(page 5)

T.J. Hastie and R.J. Tibshirani. *Generalized additive models*. Chapman & Hall/CRC, 1990.
(page 8)

Imre Risi Kondor. *Group theoretical methods in machine learning*. PhD thesis, Columbia University, 2008.
(pages 15 and 18)

N. Lawrence. Probabilistic non-linear principal component analysis with gaussian process latent variable models. *The Journal of Machine Learning Research*, 6:1783–1816, 2005. (page 19)

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361, 1995. (page 18)

D. Lerner and D. Asimov. The Sudanese Möbius band. In *SIGGRAPH Electronic Theatre*, 1984. (page 20)

David J.C. MacKay. Introduction to gaussian processes. *NATO ASI Series F Computer and Systems Sciences*, 168:133–166, 1998. (page 17)

James Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, pages 415–446, 1909. (page 13)

Charles A Micchelli, Yuesheng Xu, and Haizhang Zhang. Universal kernels. *The Journal of Machine Learning Research*, 7:2651–2667, 2006. (page 5)

Ha Quang Minh, Partha Niyogi, and Yuan Yao. Mercer's theorem, feature maps, and smoothing. In *Learning theory*, pages 154–168. Springer, 2006. (page 13)

Radford M Neal. *Bayesian learning for neural networks*. PhD thesis, University of Toronto, 1995. (page 10)

Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *Conference on Uncertainty in AI*, pages 689–690. IEEE, 2011.

I-C Yeh. Modeling of strength of high-performance concrete using artificial neural networks. *Cement and Concrete research*, 28(12):1797–1808, 1998. (page 8)