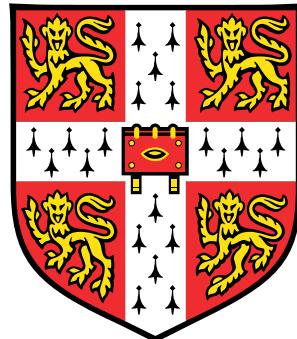


Structured Gaussian Process Models



David Kristjanson Duvenaud

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

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

Contents	xi
List of Figures	xv
List of Tables	xix
Nomenclature	xix
1 Introduction	1
1.1 Derivation of Component Marginal Variance	1
2 Expressing Structure through Kernels	3
2.1 Nonparametric Inference via Bayesian Quadrature	3
2.2 Introduction	3
2.3 Bayesian Quadrature	4
2.3.1 BQ as an inference method	6
2.4 Guidelines for Constructing a Kernel	6
2.4.1 Convergence	7
2.5 Inference in the Indian Buffet Process	8
2.5.1 A kernel between binary matrices	9
2.5.2 Computing z_n and the prior variance	9
2.6 Infinite Mixture Models	10
2.6.1 A Kernel Between Multinomial Distributions	10
2.6.2 Infinite Mixture of Gaussians	11
2.7 Related Work	12
2.8 Experiments	12
2.8.1 IBP Experiments	12
2.8.2 DP Mixture Experiments	13

2.9	Discussion	14
2.9.1	Appropriateness of the GP prior	14
2.10	Conclusions	15
3	Additive Gaussian Processes	17
3.1	Introduction	17
3.2	Gaussian Process Models	18
3.3	Additive Kernels	19
3.3.1	Parameterization	20
3.3.2	Interpretability	21
3.3.3	Efficient Evaluation of Additive Kernels	22
3.3.4	Computation	22
3.4	Related Work	23
3.4.1	Hierarchical Kernel Learning	23
3.4.2	ANOVA Procedures	24
3.4.3	Non-local Interactions	25
3.5	Experiments	26
3.5.1	Synthetic Data	26
3.5.2	Experimental Setup	27
3.5.3	Results	27
3.6	Conclusion	29
3.6.1	Bach Synthetic Dataset	29
3.6.2	Polynomial Kernels	31
3.6.3	Future Work	31
4	Characterizing Deep Gaussian Process Models	33
4.1	Introduction	33
4.2	Relating deep neural nets and deep Gaussian processes	35
4.2.1	Single-layer models	35
4.2.2	Multiple hidden layers	36
4.3	Characterizing deep Gaussian processes	37
4.3.1	One-dimensional asymptotics	37
4.3.2	Distribution of the Jacobian	39
4.4	Formalizing a pathology	41
4.5	Fixing the pathology	43
4.6	Deep kernels	45

4.6.1	When are deep kernels useful models?	49
4.7	Dropout in Gaussian processes	51
4.7.1	Dropout on feature activations	51
4.7.2	Dropping out inputs	52
4.8	Related work	52
4.9	Conclusions	53
	References	55

List of Figures

2.1	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 $Z_{f,p}$.	5
2.2	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.	13
2.3	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, $dof = \infty$ is Gaussian. The shaded error represents uncertainty about the value of the marginal likelihood.	14
3.1	Additive kernels correspond to additive functions	19
3.2	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.	21

3.3 A comparison of different models. 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 separately. 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.	24
3.4 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.	26
3.5 Long-range inference in functions with additive structure.	26
4.1 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).	35
4.2 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.	38
4.3 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.	41
4.4 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.	42

4.5	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.	43
4.6	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 4.5 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}	44
4.7	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.	45
4.8	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 4.2.	46
4.9	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.	47
4.10	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.	47
4.11	Feature mapping of a deep GP with each layer connected to the input \mathbf{x} . Just as the densities in figure 4.9 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 4.6.	48
4.12	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.	50

4.13 GP draws using deep input-connected kernels.	50
---	----

List of Tables

3.1	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.	21
3.2	Regression Mean Squared Error	27
3.3	Regression Negative Log Likelihood	28
3.4	Classification Percent Error	28
3.5	Classification Negative Log Likelihood	28
3.6	Hyperparameters for bach synth c 200 dataset	31

Chapter 1

Introduction

1.1 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 $\mathbf{f} = \mathbf{f}_1 + \mathbf{f}_2$. If \mathbf{f}_1 and \mathbf{f}_2 are a priori independent, and $\mathbf{f}_1 \sim \text{gp}(\mu_1, k_1)$ and $\mathbf{f}_2 \sim \text{gp}(\mu_2, k_2)$, then

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

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

By the formula for Gaussian conditionals:

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

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

Gaussian is given by

$$\mathbf{f}_1^* | \mathbf{f} \sim \mathcal{N}\left(\mu_1^* + \mathbf{k}_1^{*\top} (\mathbf{K}_1 + \mathbf{K}_2)^{-1} (\mathbf{f} - \mu_1 - \mu_2), \mathbf{k}_1^{**} - \mathbf{k}_1^{*\top} (\mathbf{K}_1 + \mathbf{K}_2)^{-1} \mathbf{k}_1^*\right). \quad (1.3)$$

The covariance between the two components, conditioned on their sum is given by:

$$\text{cov}(\mathbf{f}_1^*, \mathbf{f}_2^*) | \mathbf{f} = -\mathbf{k}_1^{*\top} (\mathbf{K}_1 + \mathbf{K}_2)^{-1} \mathbf{k}_2^* \quad (1.4)$$

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 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.

2.2 Introduction

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

$$Z_{f,p} = \int f(\theta)p(\theta)d\theta \quad (2.1)$$

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 (2.1) is especially challenging, as the domain of integration 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.

2.3 Bayesian Quadrature

In contrast to mcmc, a procedure which computes (2.1) in the limit of infinite samples, Bayesian quadrature is a *model-based* integration method. This means that bq puts a 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 2.1 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

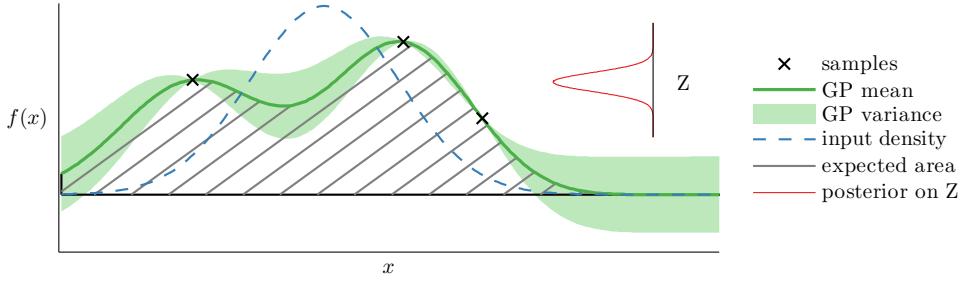


Fig. 2.1 An illustration of Bayesian Quadrature. The function $f(x)$ is sampled at a set of input locations. This induces a Gaussian process posterior distribution on f , which is integrated in closed form against the target density, $p(\mathbf{x})$. Since the amount of volume under f is uncertain, this gives rise to a (Gaussian) posterior distribution over $Z_{f,p}$.

after conditioning on function evaluations $\mathbf{y} = f(\theta_1) \dots f(\theta_N)$, we obtain:

$$p(\mathbf{f}(\theta_\star)|\mathbf{y}) = \mathcal{N}\left(\mathbf{f}(\theta_\star)|\bar{\mathbf{f}}(\theta_\star), \text{cov}(\theta_\star, \theta'_\star)\right) \quad (2.2)$$

where

$$\bar{\mathbf{f}}(\mathbf{x}_\star) = k(\theta_\star, \theta_s)K^{-1}\mathbf{y} \quad (2.3)$$

$$\text{cov}(\mathbf{x}_\star, \mathbf{x}'_\star) = k(\theta_\star, \theta_\star) - k(\theta_\star, \theta_s)\mathbf{K}^{-1}k(\theta_s, \theta_\star) \quad (2.4)$$

and $\mathbf{K}_{mn} = k(\theta_m, \theta_n)$. Conveniently, the gp posterior implies a closed form for the expectation and variance of (2.1):

$$\mathbb{E}[Z_{f,p}|\mathbf{y}] = \mathbb{E}_{\text{gp}(f|\mathbf{y})} \left[\int f(\theta)p(\theta)d\theta \right] = \left[\int k(\theta, \theta_s)p(\theta)d\theta \right] \mathbf{K}^{-1}\mathbf{y} = \mathbf{z}^\top \mathbf{K}^{-1}\mathbf{y} \quad (2.5)$$

$$\mathbb{V}[Z_{f,p}|\mathbf{y}] = \mathbb{V}_{\text{prior}}[Z_{f,p}] - \mathbf{z}^\top \mathbf{K}^{-1} \mathbf{z} \quad (2.6)$$

where

$$z_n = \int k(\theta, \theta_n)p(\theta)d\theta \quad (2.7)$$

$$\mathbb{V}_{\text{prior}}[Z_{f,p}] = \iint k(\theta, \theta')p(\theta)p(\theta')d\theta d\theta'. \quad (2.8)$$

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 (2.7) and (2.8) efficiently. We must also set or integrate out any parameters of k .

2.3.1 BQ as an inference method

If we assume that in (2.1), the form of $p(\theta)$ is such that we can compute z_n , then applying bq is straightforward. 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.

$$\mathbb{E}_{p(\theta|x)} [f(\theta)] = \int f(\theta)p(\theta|x)d\theta = \frac{1}{Z} \int f(\theta)p(x|\theta)p(\theta)d\theta \quad \text{where } Z = \int p(x|\theta)p(\theta)d\theta \quad (2.9)$$

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.

2.4 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 symmetric 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 $Z_{f,p}$.

2.4.1 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-dimensional 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 [Rasmussen and Williams \(2006\)](#). 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 $\mathbb{E}_{\text{gp}}[Z] = f(\theta_1)$ and $\mathbb{V}_{\text{gp}}[Z] = 0$. 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. \square

For a more detailed proof, see the supplementary material. The statements in this section also hold for the problem of gp regression in general, but are specially relevant for the problem of inferring likelihood functions over latent structures. This is for two reasons: First, likelihood functions over structures can typically be 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.

2.5 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) [1] 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 [1], 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

$$P(\mathbf{Z}|\alpha) = \prod_{k=1}^K \frac{\frac{\alpha}{K}\Gamma(m_k + \frac{\alpha}{K})\Gamma(N - m_k + 1)}{\Gamma(N + 1 + \frac{\alpha}{K})} \quad (2.10)$$

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 [1]. 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:

$$p(\mathbf{X}|\mathbf{Z}, \sigma_X, \sigma_A) = \frac{\exp\left\{-\frac{1}{2\sigma_X^2} \text{tr}(\mathbf{X}^T(\mathbf{I} - \mathbf{Z}(\mathbf{Z}^T\mathbf{Z} + \frac{\sigma_X^2}{\sigma_A^2}\mathbf{E})^{-1}\mathbf{Z}^T)\mathbf{X})\right\}}{(2\pi)^{ND/2}\sigma_X^{(N-K)D}\sigma_A^{KD}|\mathbf{Z}^T\mathbf{Z} + \frac{\sigma_X^2}{\sigma_A^2}\mathbf{I}|^{\frac{D}{2}}} \quad (2.11)$$

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 [...]

2.5.1 A kernel between binary matrices

Here we give a kernel which satisfies the guidelines given in section 2.4. Since the likelihood in (2.11) 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}' :

$$k(\mathbf{Z}, \mathbf{Z}') = \sum_{k=1}^K \sum_{k'=1}^{K'} \sum_{n=1}^N \mathbf{Z}_{n,k} \mathbf{Z}'_{n,k'} \quad (2.12)$$

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]

2.5.2 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, \dots, z_n , given by (2.7). 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 (2.10) and (2.12), we have:

$$z_n(\mathbf{Z}) = \sum_{\mathbf{Z}'} k(\mathbf{Z}, \mathbf{Z}') p(\mathbf{Z}'|\alpha) = \sum_{\mathbf{Z}'} \left[\sum_{n=1}^N \sum_{k=1}^K \sum_{k'=1}^{K'} \mathbf{Z}_{n,k} \mathbf{Z}'_{n,k'} \right] \left[\prod_{k^*=1}^K P(\mathbf{Z}'_{:,k^*}|\alpha) \right] \quad (2.13)$$

$$= \sum_{k=1}^K \sum_{n=1}^N \mathbf{Z}_{n,k} \frac{\alpha}{1 + \frac{\alpha}{K'}} = \alpha \sum_{k=1}^K \sum_{n=1}^N \mathbf{Z}_{n,k} \quad \text{as } K' \rightarrow \infty \quad (2.14)$$

See the supplementary material for longer derivations. We can interpret (2.14) 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 (2.8), which follows a similar derivation. This is the expected variance of Z before we have seen any data.

$$V_{\text{prior}} = \sum_{\mathbf{Z}'} \sum_{\mathbf{Z}} p(\mathbf{Z}|\alpha) k(\mathbf{Z}, \mathbf{Z}') p(\mathbf{Z}'|\alpha) = \frac{\alpha^2 N}{1 + \frac{\alpha}{K}} = \alpha^2 N \quad \text{as } K \rightarrow \infty \quad (2.15)$$

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.

2.6 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 (2.7) and (2.8) for this kernel, leaving unspecified the kernel between individual mixture elements. Then, we will continue the derivation for an infinite mixture of Gaussians.

2.6.1 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.

$$k(\pi, \theta, \pi', \theta') = \sum_i^{n_\theta} \sum_j^{n_{\theta'}} \pi_i \pi'_j k_a(\theta_i, \theta'_j) \quad (2.16)$$

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 (2.16).

If our mixture θ has n_θ components, then

$$\begin{aligned} z(\pi, \theta) &= \iint k(\pi, \theta, \pi', \theta') p(\theta') p(\pi') d\theta' d\pi' = \iint \left[\sum_{i=1}^{n_\theta} \sum_{j=1}^{n_{\theta'}} \pi_i \pi'_j k(\theta_i, \theta'_j) \right] \left[\prod_a^{n_{\theta'}} p(\theta'_a) \right] p(\pi') d\theta' d\pi' \\ &= \int \sum_{i=1}^{n_\theta} \sum_{j=1}^{n_{\theta'}} \pi_i \pi'_j \underbrace{\int k(\theta_i, \theta'_j) p(\theta'_j) d\theta'_j}_{z_a(\theta_i)} p(\pi') d\pi' = \sum_{i=1}^{n_\theta} \pi_i z_a(\theta_i) \end{aligned} \quad (2.17)$$

$$\mathbb{V}_{\text{prior}} [Z_{f,p}] = \iint z(\pi, \theta) p(\theta) p(\pi) d\theta d\pi = \underbrace{\int z(\theta_i) p(\theta_i) d\theta_i}_{V_a} \underbrace{\int p(\pi) \sum_{i=1}^{n_\theta} \pi_i d\pi}_{\text{sums to one}} = V_a \quad (2.18)$$

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 $n_\theta \rightarrow \infty$. 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.

2.6.2 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 ?:

$$p(x|\pi, \theta) = \sum_{i=1}^k \pi_i p(x|\theta_i) = \sum_{i=1}^k \pi_i \mathcal{N}(y|\mu_i, \Sigma_i) \quad (2.19)$$

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:

$$k(\mu, \Sigma, \mu', \Sigma') = \mathcal{N}(\mu | \mu', \Sigma_k) \quad (2.20)$$

where the entries of Σ_k are kernel parameters. Next, we give (2.7) and (2.8) for this kernel:

$$z_k(\mu_i, \Sigma_i) = \iint k(\mu_i, \Sigma_i, \mu'_j, \Sigma'_j) p(\mu'_j, \Sigma'_j) d\mu'_j d\Sigma'_j = \mathcal{N}(\mu_i | \lambda, \Sigma_k + \Sigma_p) \quad (2.21)$$

$$V_a = \int z_k(\mu_i, \Sigma_i) p(\mu_i, \Sigma_i) d\mu_i d\Sigma_i = \mathcal{N}(0 | 0, \Sigma_k + 2\Sigma_p) \quad (2.22)$$

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

2.7 Related Work

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

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

2.8.1 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 2.2 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,

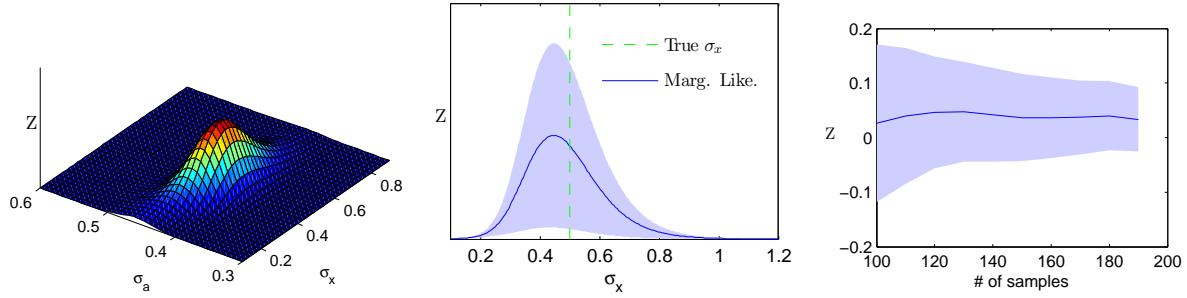


Fig. 2.2 Computing marginal likelihoods with BQ. Left: The marginal likelihood as a function of two nuisance parameters. Center: A slice of the 2-D marginal likelihood function, at $\sigma_A = 0.4$. The true value of σ_X lies roughly in the center of the likelihood function. Right: The marginal likelihood as a function of the number of evaluations of the likelihood function. The shaded error represents uncertainty about the value of the marginal likelihood.

etc. For example, the likelihood function (2.11) 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 2.2 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.

2.8.2 DP Mixture Experiments

Figure 2.2 demonstrates the use of bq to examine sensitivity to prior parameters.

Figures 2.2 and 2.3 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.

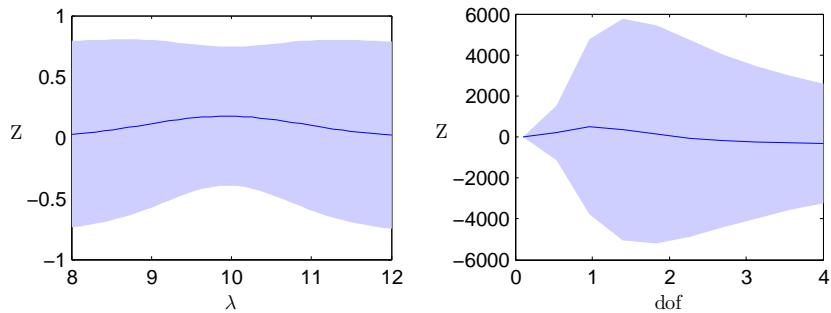


Fig. 2.3 A demonstration of computing marginal likelihoods. Left: The marginal likelihood versus the prior mean. Right: Marginal likelihood versus the degrees of freedom of the mixture components. $\text{dof} = 1$ is Cauchy, $\text{dof} = 2$ is Student's t, $\text{dof} = \infty$ is Gaussian. The shaded error represents uncertainty about the value of the marginal likelihood.

2.9 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.

2.9.1 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 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.

2.10 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 one setting to perform post-hoc sensitivity analysis of nuisance parameters.

Chapter 3

Additive Gaussian Processes

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 increased interpretability, as well as state-of-the-art predictive power in regression tasks.

3.1 Introduction

Most statistical regression models in use today are of the form: $g(y) = f(x_1) + f(x_2) + \dots + f(x_D)$. Popular examples include logistic regression, linear regression, and Generalized Linear Models [Nelder and Wedderburn \(1972\)](#). This family of functions, known as Generalized Additive Models (GAM) [Hastie and Tibshirani \(1990\)](#), are typically easy to fit and interpret. Some extensions of this family, such as smoothing-splines ANOVA [Wahba \(1990\)](#), 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: $y = f(x_1, x_2, \dots, x_D)$. 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 D th-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)[Bach \(2009\)](#). HKL explores a similar class of models, and sidesteps the 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 [Bach \(2009\)](#).

3.2 Gaussian Process Models

Gaussian processes are a flexible and tractable prior over functions, useful for solving regression and classification tasks[Rasmussen and Williams \(2006\)](#). 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 3.1 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 interac-

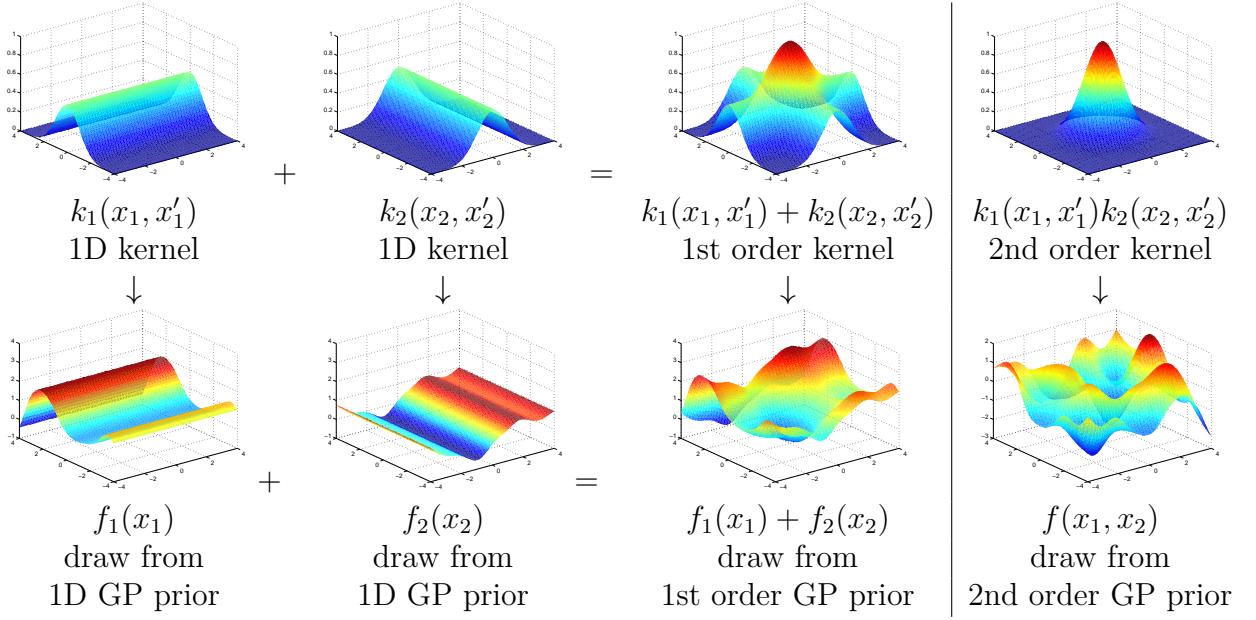


Fig. 3.1 A first-order additive kernel, and a product kernel. Left: a draw from a first-order additive kernel corresponds to a sum of draws from one-dimensional kernels. Right: functions drawn from a product kernel prior have weaker long-range dependencies, and less long-range structure.

tions. 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.

3.3 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 n th order additive kernel as:

$$k_{add_1}(\mathbf{x}, \mathbf{x}') = \sigma_1^2 \sum_{i=1}^D k_i(x_i, x'_i) \quad (3.1)$$

$$k_{add_2}(\mathbf{x}, \mathbf{x}') = \sigma_2^2 \sum_{i=1}^D \sum_{j=i+1}^D k_i(x_i, x'_i) k_j(x_j, x'_j) \quad (3.2)$$

$$k_{add_n}(\mathbf{x}, \mathbf{x}') = \sigma_n^2 \sum_{1 \leq i_1 < i_2 < \dots < i_n \leq D} \left[\prod_{d=1}^n k_{i_d}(x_{i_d}, x'_{i_d}) \right] \quad (3.3)$$

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

$$k_{add_D}(\mathbf{x}, \mathbf{x}') = \sigma_D^2 \prod_{d=1}^D k_d(x_d, x'_d) \quad (3.4)$$

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

$$k_{add_D}(\mathbf{x}, \mathbf{x}') = \sigma_D^2 \prod_{d=1}^D k_d(x_d, x'_d) = \sigma_D^2 \prod_{d=1}^D \exp\left(-\frac{(x_d - x'_d)^2}{2l_d^2}\right) = \sigma_D^2 \exp\left(-\sum_{d=1}^D \frac{(x_d - x'_d)^2}{2l_d^2}\right) \quad (3.5)$$

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

3.3.1 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 d th order variance hyperparameter controls how much of the target function’s variance comes from interactions of the d th order. Table 3.1 shows examples of normalized order variance hyperparameters learned on real datasets.

On different datasets, the dominant order of interaction estimated by the additive model varies widely. An additive GP with all of its variance coming from the 1st order is equivalent to a GAM; an additive GP with all its variance coming from the D th 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 3.5). If this is not the case, the hyperparameters can specify a suitably flexible model.

Table 3.1 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

3.3.2 Interpretability

As noted by PlatePlate (1999), 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 3.2.

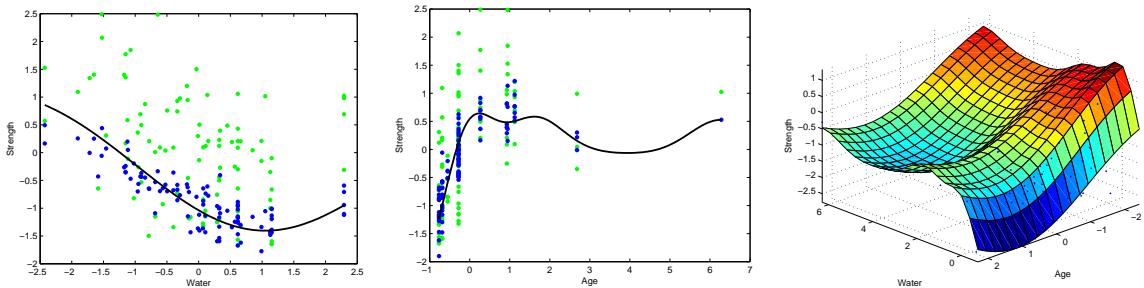


Fig. 3.2 Low-order functions on the concrete dataset. Left, Centre: By considering only first-order terms of the additive kernel, we recover a form of Generalized Additive Model, and can plot the corresponding 1-dimensional functions. Green points indicate the original data, blue points are data after 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.

3.3.3 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 n th order additive kernel corresponds to the n th *elementary symmetric polynomial*[Macdonald \(1998\)](#) [Stanley \(2001\)](#), 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 k th power sum: $s_k(z_1, z_2, \dots, z_D) = \sum_{i=1}^D z_i^k$, then

$$k_{add_n}(\mathbf{x}, \mathbf{x}') = e_n(z_1, \dots, z_D) = \frac{1}{n} \sum_{k=1}^n (-1)^{(k-1)} e_{n-k}(z_1, \dots, z_D) s_k(z_1, \dots, z_D) \quad (3.6)$$

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:

$$\frac{\partial k_{add_n}}{\partial z_j} = e_{n-1}(z_1, \dots, z_{j-1}, z_{j+1}, \dots, z_D) \quad (3.7)$$

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

3.3.4 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 toolbox¹; code to perform all experiments is available at the author’s website.²

3.4 Related Work

PlatePlate (1999) constructs a form of additive GP, but using only the first-order and D th 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 modelKaufman and Sain (2010) 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 Kaufman and Sain (2010) is disadvantageous.

Christoudias et al.Christoudias et al. (2009) 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.

3.4.1 Hierarchical Kernel Learning

BachBach (2009) 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

¹Available at <http://www.gaussianprocess.org/gpml/code/>

²Example code available at: <http://mlg.eng.cam.ac.uk/duvenaud/>

³In 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 $\prod_{j \in J \setminus i} k_j(\mathbf{x}, \mathbf{x}')$, for all $i \in J$. For details, see Bach (2009).

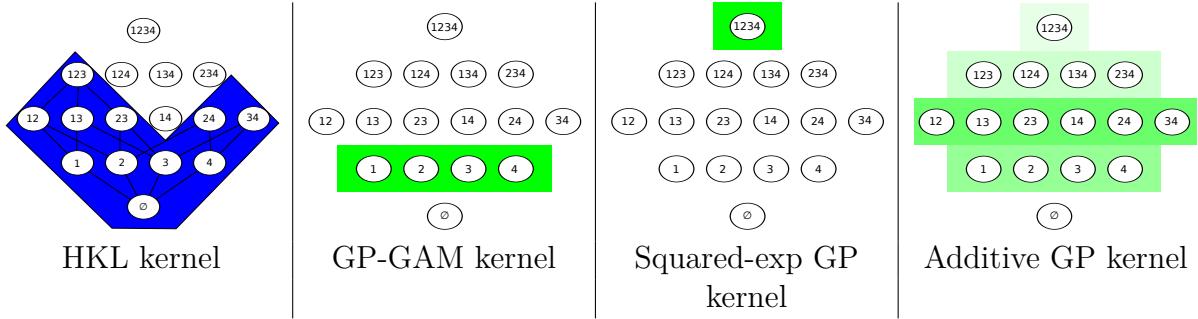


Fig. 3.3 A comparison of different models. Nodes represent different interaction terms, ranging from first-order to fourth-order interactions. Far left: HKL can select a hull of interaction terms, but must use a pre-determined weighting over those terms. Far right: the additive GP model can weight each order of interaction separately. 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.

each dimension’s kernel, and a pre-defined weighting over all terms, HKL performs model selection by searching over hulls of interaction terms. In Bach (2009), Bach also fixes the relative weighting between orders of interaction with a single term α , computing the sum over all orders by:

$$k_a(\mathbf{x}, \mathbf{x}') = v_D^2 \prod_{d=1}^D (1 + \alpha k_d(x_d, x'_d)) \quad (3.8)$$

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

Figure 3.3 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 Bach (2009) 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.

3.4.2 ANOVA Procedures

Vapnik Vapnik (1998) 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. Stitson et al. (1999) 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) [Wahba \(1990\)](#). 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.

3.4.3 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. \(2006a\)](#) argue that models based on squared-exponential kernels are particularly susceptible to the *curse of dimensionality*. They emphasize that the locality of the kernels means that these models cannot capture non-local structure. They argue that many functions that we care about have such structure. Methods based solely on local kernels will require training examples at all combinations of relevant inputs.

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 [3.4](#) provides a geometric comparison between squared-exponential kernels and additive kernels in 3 dimensions.

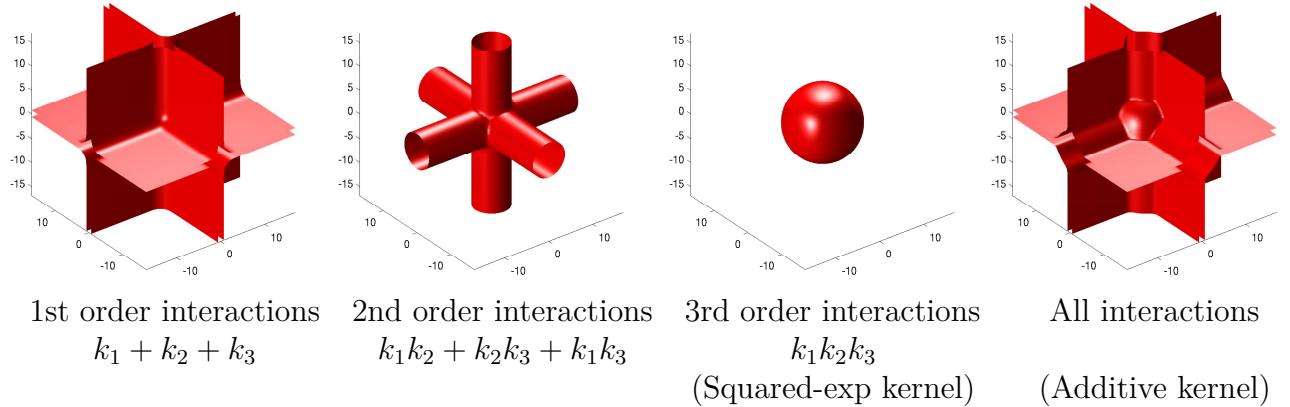


Fig. 3.4 Isocontours of additive kernels in 3 dimensions. The third-order kernel only considers nearby points relevant, while the lower-order kernels allow the output to depend on distant points, as long as they share one or more input value.

3.5 Experiments

3.5.1 Synthetic Data

Because additive kernels can discover non-local structure in data, they are exceptionally well-suited to problems where local interpolation fails. Figure 3.5 shows a dataset which

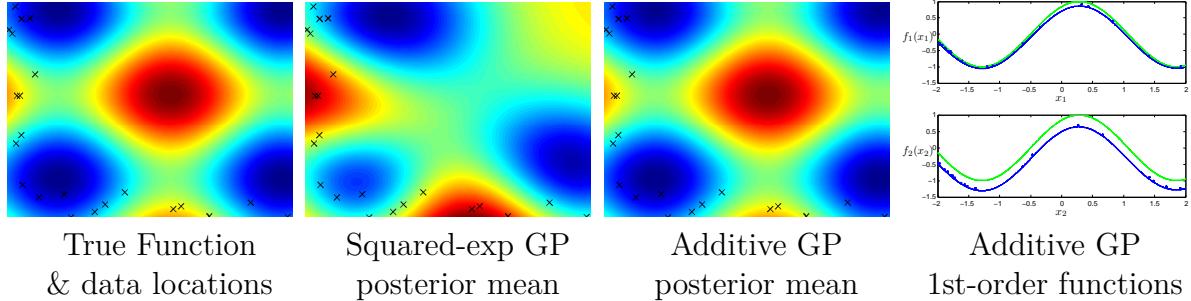


Fig. 3.5 Long-range inference in functions with additive structure.

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.

3.5.2 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. HKL⁴ 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 Nocedal (1980) 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 Minka (2001).

3.5.3 Results

Tables 3.2, 3.3, 3.4 and 3.5 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 3.2 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

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

⁴Code for HKL available at <http://www.di.ens.fr/~fbach/hkl/>

Table 3.3 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 3.4 Classification Percent Error

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

Table 3.5 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

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 3.1). 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.

3.6 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.

3.6.1 Bach Synthetic Dataset

In addition to standard UCI repository datasets, we generated a synthetic dataset following the same recipe as [Bach \(2009\)](#): 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 [Bach \(2009\)](#) 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.

Table 3.6 shows the hyperparameters learnt. Note that while GP-ARD increases the lengthscale of the irrelevant variables, GP-AS reduces the signal variance.

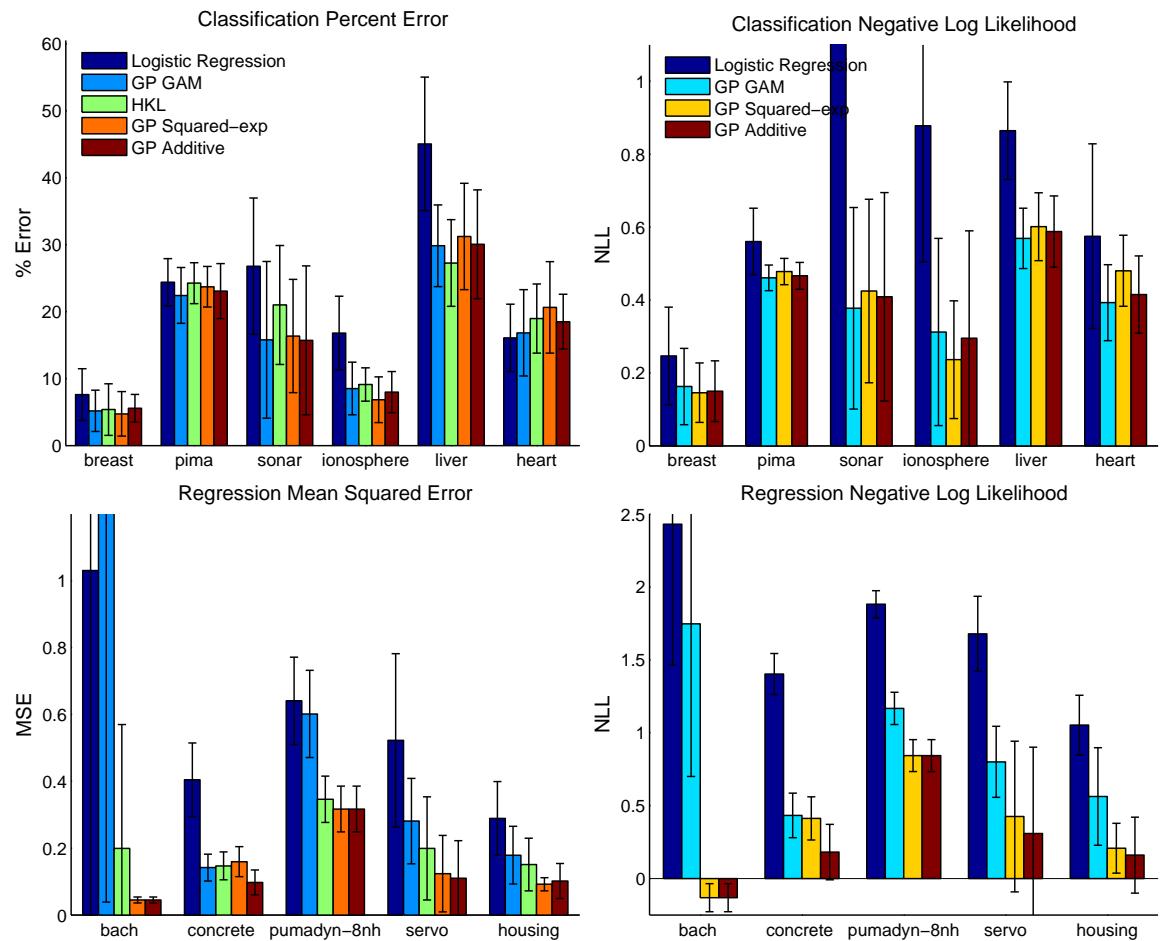


Table 3.6 Hyperparameters for bach synth c 200 dataset

Variable:	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
ARD Lengthscale	1.09	0.32	0.09	0.74	760.60	522.32	667.74	886.37
ADD Lengthscale	0.93	0.83	0.04	0.62	473.04	325.51	415.76	551.96
ADD Variance	15.48	21.02	0.01	60.93	0.67	0.67	0.67	0.67
Order of Interaction:	1	2	3	4	5	6	7	8
ADD Order Variance	0.0%	0.0%	0.0%	0.0%	0.0%	49.0%	0.1%	50.8%

3.6.2 Polynomial Kernels

A simple variation on the all-subsets kernel would be one which includes repeated terms, of the form

$$k_{h_n}(\mathbf{x}, \mathbf{x}') = v_n^2 \sum_{i_1, \dots, i_n} \prod_{d=1}^n k_{i_d}(x_{i_d}, x'_{i_d}) = v_n^2 \prod_{d=1}^n \sum_{i_1, \dots, i_n} k_{i_d}(x_{i_d}, x'_{i_d}) \quad (3.9)$$

This formulation allows repeated terms, such as x_i^2 and is equivalent to the polynomial kernel [Shawe-Taylor and Cristianini \(2004\)](#). However, this formulation cases the length scale of each dimension to vary with the order of interaction considered.

3.6.3 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.

A Gaussian process with an additive kernel corresponds to a prior over functions which are additive. If a function $\mathbf{f} = \mathbf{a} + \mathbf{b}$ is a sum of two functions \mathbf{a} and \mathbf{b} drawn from a Gaussian process prior with covariances \mathbf{K}_a and \mathbf{K}_b , then equivalently, \mathbf{f} is drawn from a Gaussian process prior with covariance $\mathbf{K}_f = \mathbf{K}_a + \mathbf{K}_b$. The converse also holds. If \mathbf{K}_a and \mathbf{K}_b are positive definite, then so is \mathbf{K}_f .

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

[Explain how squared exp somehow puts too much mass on complicated hypotheses, or is unable to represent simple hypotheses].

[Show differences in marginal likelihoods]

[Should we try learning a convex hull of features as well?]

[To discuss: Shawe-Taylor says in Remark 9.3 that we can control the relative weights of the degree monomials by just computing $\Pi(k(x, x') + R)$. If R is high then the high-degree monomials don't matter as much. Seems like we end up with a huge constant at the end, though...]

[Shawe-Taylor and Cristianini \(2004\)](#) define ANOVA kernels thusly: "The ANOVA kernel of degree d is like the all-subsets kernel except that it is restricted to subsets of the given cardinality d ."

Chapter 4

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.

4.1 Introduction

Much recent work on deep networks has focused on weight initialization (Martens, 2010), regularization (Lee et al., 2007) and network architecture (Gens and Domingos, 2013). 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 (Damianou and Lawrence, 2013). Deep gps are simply priors on compositions of vector-valued functions, where each output of each layer is drawn independently from a GP prior:

$$\mathbf{f}^{(1:L)}(\mathbf{x}) = \mathbf{f}^{(L)}(\mathbf{f}^{(L-1)}(\dots \mathbf{f}^{(2)}(\mathbf{f}^{(1)}(\mathbf{x})) \dots)) \quad (4.1)$$

$$\mathbf{f}_d^{(\ell)} \stackrel{\text{ind}}{\sim} \text{gp}\left(0, k_d^\ell(\mathbf{x}, \mathbf{x}')\right) \quad (4.2)$$

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.

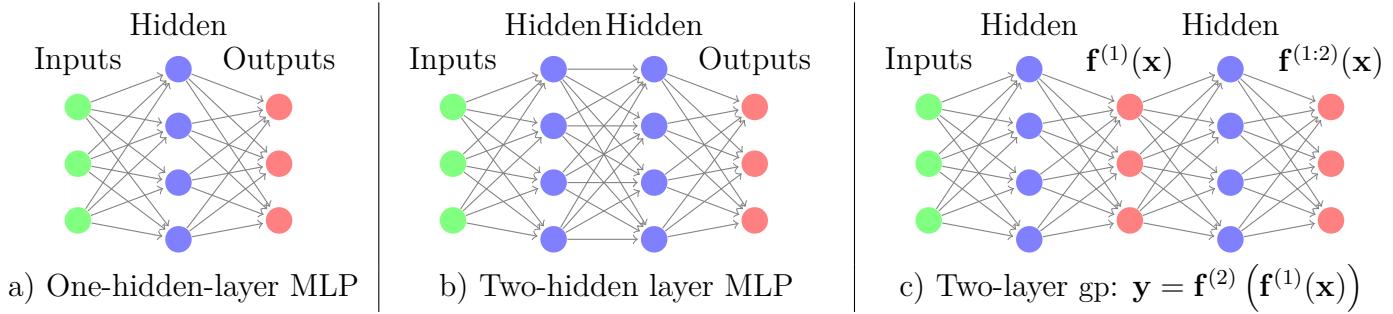


Fig. 4.1 GPs can be understood as one-hidden-layer MLP with infinitely many hidden units (a). There are two possible interpretations of deep GPs: We can consider the deep GP to be a neural network with a finite number of hidden units, each with a different non-parametric activation function (b). Alternatively, we can consider every second layer to be a random linear combination of an infinite number of fixed parametric hidden units (c).

4.2 Relating deep neural nets and deep Gaussian processes

4.2.1 Single-layer models

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

$$\mathbf{h}^{(1)}(\mathbf{x}) = \sigma(\mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}) \quad (4.3)$$

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:

$$f(\mathbf{x}) = \mathbf{V}^{(1)}\sigma(\mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}) = \mathbf{V}^{(1)}\mathbf{h}^{(1)}(\mathbf{x}) \quad (4.4)$$

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

There exists a correspondence between one-layer MLPs and GPs (Neal, 1995). 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

$$f(\mathbf{x}) = \frac{1}{K}\alpha^\top \mathbf{h}(\mathbf{x}) = \frac{1}{K} \sum_{i=1}^K \alpha_i h_i(\mathbf{x}), \quad (4.5)$$

with fixed features $[h_1(\mathbf{x}), \dots, h_K(\mathbf{x})]^\top = \mathbf{h}(\mathbf{x})$ and i.i.d. α 's with zero mean and fi-

nite variance σ^2 , the central limit theorem implies that as the number of features K grows, any two function values $f(\mathbf{x}), f(\mathbf{x}')$ have a joint distribution approaching $\mathcal{N}\left(0, \frac{\sigma^2}{K} \sum_{i=1}^K h_i(\mathbf{x})h_i(\mathbf{x}')\right)$. A joint Gaussian distribution between any set of function values is the definition of a Gaussian process.

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

We can also work backwards to derive a one-layer MLP from any gp. Mercer’s theorem implies that any positive-definite kernel function corresponds to an inner product of features: $k(\mathbf{x}, \mathbf{x}') = \mathbf{h}(\mathbf{x})^\top \mathbf{h}(\mathbf{x}')$. 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.

4.2.2 Multiple hidden layers

In an MLP, the ℓ th layer units are given by the recurrence

$$\mathbf{h}^{(\ell)}(\mathbf{x}) = \sigma \left(\mathbf{b}^{(\ell)} + \mathbf{W}^{(\ell)} \mathbf{h}^{(\ell-1)}(\mathbf{x}) \right) . \quad (4.6)$$

This architecture is shown in figure 4.1b. For example, if we extend the model given by (4.4) to have two layers of feature mappings, the resulting model is

$$f(\mathbf{x}) = \frac{1}{K} \alpha^\top \mathbf{h}^{(2)} \left(\mathbf{h}^{(1)}(\mathbf{x}) \right) . \quad (4.7)$$

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”:

$$k(\mathbf{x}, \mathbf{x}') = \left(\mathbf{h}^{(2)}(\mathbf{h}^{(1)}(\mathbf{x})) \right)^\top \mathbf{h}^{(2)}(\mathbf{h}^{(1)}(\mathbf{x}')) \quad (4.8)$$

These models, examined in section 4.6, imply a fixed representation as opposed to a prior over representations, which is what we wish to analyze in this paper.

To construct a neural network with fixed nonlinearities corresponding to a deep gp, one must introduce a second layer in between each infinitely-wide set of fixed basis functions, as in figure 4.1c. 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 (4.4) into (4.6) to get

$$\mathbf{h}^{(\ell)}(\mathbf{x}) = \sigma \left(\mathbf{b}^{(\ell)} + \mathbf{W}^{(\ell)} \mathbf{V}^{(\ell-1)} \mathbf{h}^{(\ell-1)}(\mathbf{x}) \right). \quad (4.9)$$

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.

4.3 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 4.4 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.

4.3.1 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:

$$k_{\text{SE}}(x, x') = \sigma^2 \exp \left(\frac{-(x - x')^2}{2w^2} \right). \quad (4.10)$$

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

By the chain rule, the derivative of a one-dimensional deep gp is simply a product of its (independent) derivatives. The distribution of the absolute value of this derivative is

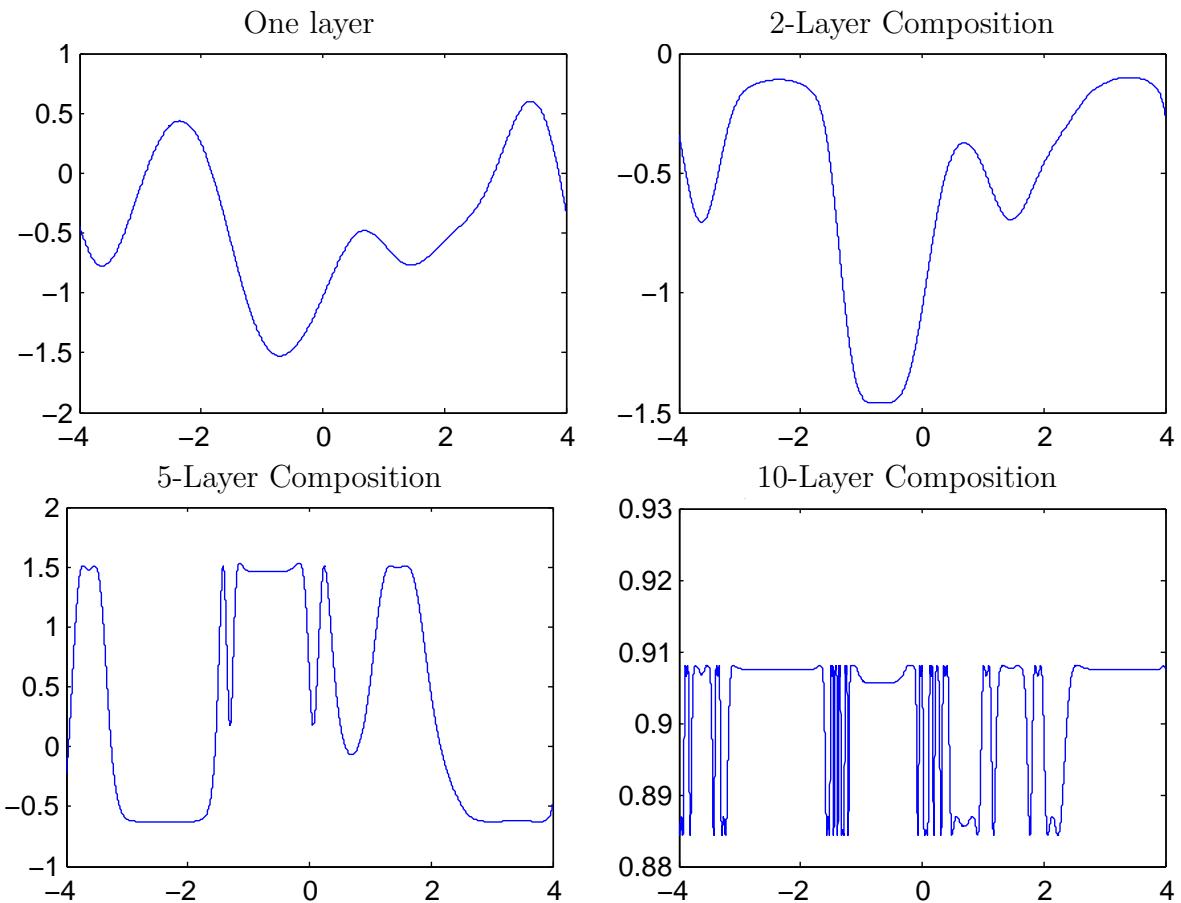


Fig. 4.2 One-dimensional draws from a deep GP prior. After a few layers, the functions begin to be either nearly flat, or highly varying, everywhere. This is a consequence of the distribution on derivatives becoming heavy-tailed.

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

$$\begin{aligned} m_{\log} &= \mathbb{E} \left[\log \left| \frac{\partial f(x)}{\partial x} \right| \right] = 2 \log \left(\frac{\sigma}{w} \right) - \log 2 - \gamma \\ v_{\log} &= \mathbb{V} \left[\log \left| \frac{\partial f(x)}{\partial x} \right| \right] = \frac{\pi^2}{4} + \frac{\log^2 2}{2} - \gamma^2 - \gamma \log 4 \\ &\quad + 2 \log \left(\frac{\sigma}{w} \right) \left[\gamma + \log 2 - \log \left(\frac{\sigma}{w} \right) \right] \end{aligned} \quad (4.11)$$

where $\gamma \approx 0.5772$ 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:

$$\begin{aligned} \log \left| \frac{\partial f^{(1:L)}(x)}{\partial x} \right| &= \sum_{\ell=1}^L \log \left| \frac{\partial f^{(\ell)}(x)}{\partial x} \right| \\ \implies \log \left| \frac{\partial f^{(1:L)}(x)}{\partial x} \right| &\stackrel{L \rightarrow \infty}{\sim} \mathcal{N}(Lm_{\log}, L^2v_{\log}) \end{aligned} \quad (4.12)$$

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 4.2 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.

4.3.2 Distribution of the Jacobian

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

Lemma 4.3.1. *The partial derivatives of a function mapping $\mathbb{R}^D \rightarrow \mathbb{R}$ 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 Solak et al. (2003):

$$\text{cov} \left(\frac{\partial f(\mathbf{x})}{\partial x_{d_1}}, \frac{\partial f(\mathbf{x})}{\partial x_{d_2}} \right) = \frac{\partial^2 k(\mathbf{x}, \mathbf{x}')}{\partial x_{d_1} \partial x'_{d_2}} \Big|_{\mathbf{x}=\mathbf{x}'} \quad (4.13)$$

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. \square

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

$$\begin{aligned} f(\mathbf{x}) &\sim \text{GP} \left(0, \sigma^2 \prod_{d=1}^D \exp \left(-\frac{1}{2} \frac{(x_d - x'_d)^2}{w_d^2} \right) \right) \\ \implies \text{cov} \left(\frac{\partial f(\mathbf{x})}{\partial x_{d_1}}, \frac{\partial f(\mathbf{x})}{\partial x_{d_2}} \right) &= \begin{cases} \frac{\sigma^2}{w_{d_1}^2} & \text{if } d_1 = d_2 \\ 0 & \text{if } d_1 \neq d_2 \end{cases} \end{aligned} \quad (4.14)$$

Lemma 4.3.2. *The Jacobian of a set of D functions $\mathbb{R}^D \rightarrow \mathbb{R}$ 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 (4.2), it follows that each row of J is independent. Lemma 4.3.1 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. \square

Theorem 4.3.3. *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 $\mathbf{f}^{(L)}(\mathbf{f}^{(L-1)}(\dots \mathbf{f}^{(3)}(\mathbf{f}^{(2)}(\mathbf{f}^{(1)}(\mathbf{x})))) \dots))$ is

$$J^{1:L}(\mathbf{x}) = J^L J^{(L-1)} \dots J^3 J^2 J^1. \quad (4.15)$$

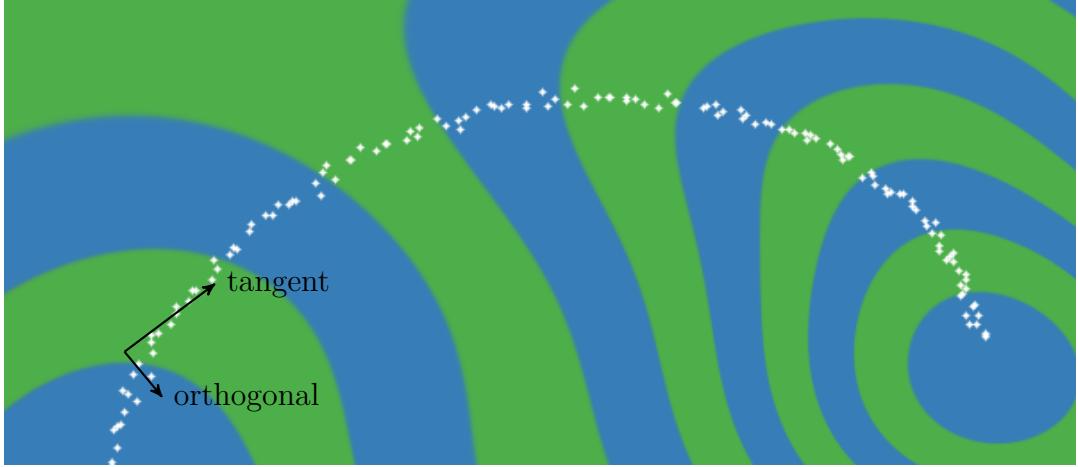


Fig. 4.3 Representing a 1-D manifold. Colors show the output of the computed representation as a function of the input space. The representation (blue & green) is invariant in directions orthogonal to the data manifold (white), making it robust to noise in those directions, and reducing the number of parameters needed to represent a datapoint. The representation also changes in directions tangent to the manifold, preserving information for later layers.

By lemma 4.3.2, each $J_{i,j}^\ell \stackrel{\text{ind}}{\sim} \mathcal{N}$, so the complete Jacobian is a product of independent Gaussian matrices, with each entry of each matrix drawn independently. \square

Theorem 4.3.3 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.

4.4 Formalizing a pathology

Rifai et al. (2011a) 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 4.3 visualizes this idea. We follow Rifai et al. (2011b) 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 4.4 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.

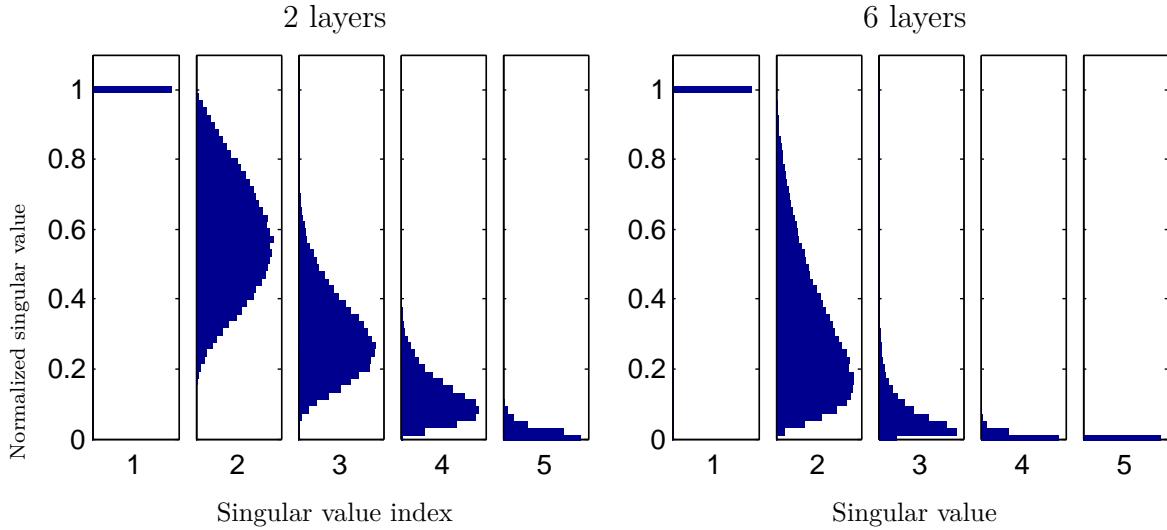


Fig. 4.4 The normalized singular value spectrum of the Jacobian of a deep GP. As the net gets deeper, the largest singular value dominates. This implies that with high probability, there is only one effective degree of freedom in the representation being computed.

Figure 4.5 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 unsuitable to model manifolds whose underlying dimensionality is greater than one.

To visualize this pathology in another way, figure 4.6 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.

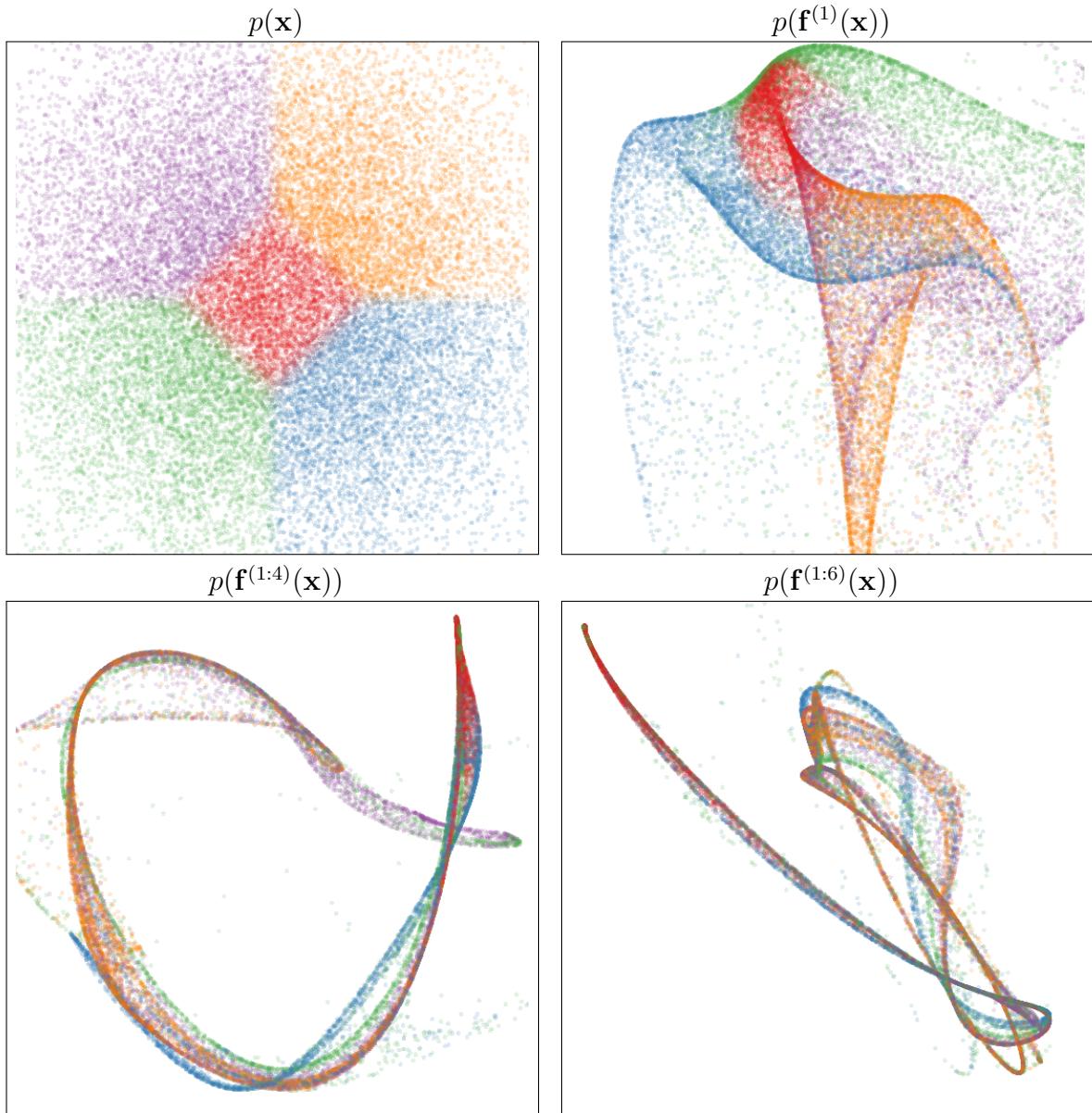


Fig. 4.5 Visualization of draws from a deep GP. A 2-dimensional Gaussian distribution (top left) is warped by successive functions drawn from a GP prior. As the number of layers increases, the density concentrates along one-dimensional filaments.

4.5 Fixing the pathology

Following a suggestion from [Neal \(1995\)](#), we can fix the pathologies exhibited in figures [4.5](#) and [4.6](#) 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 [4.7](#) shows a graphical representation of the two connectivity architec-

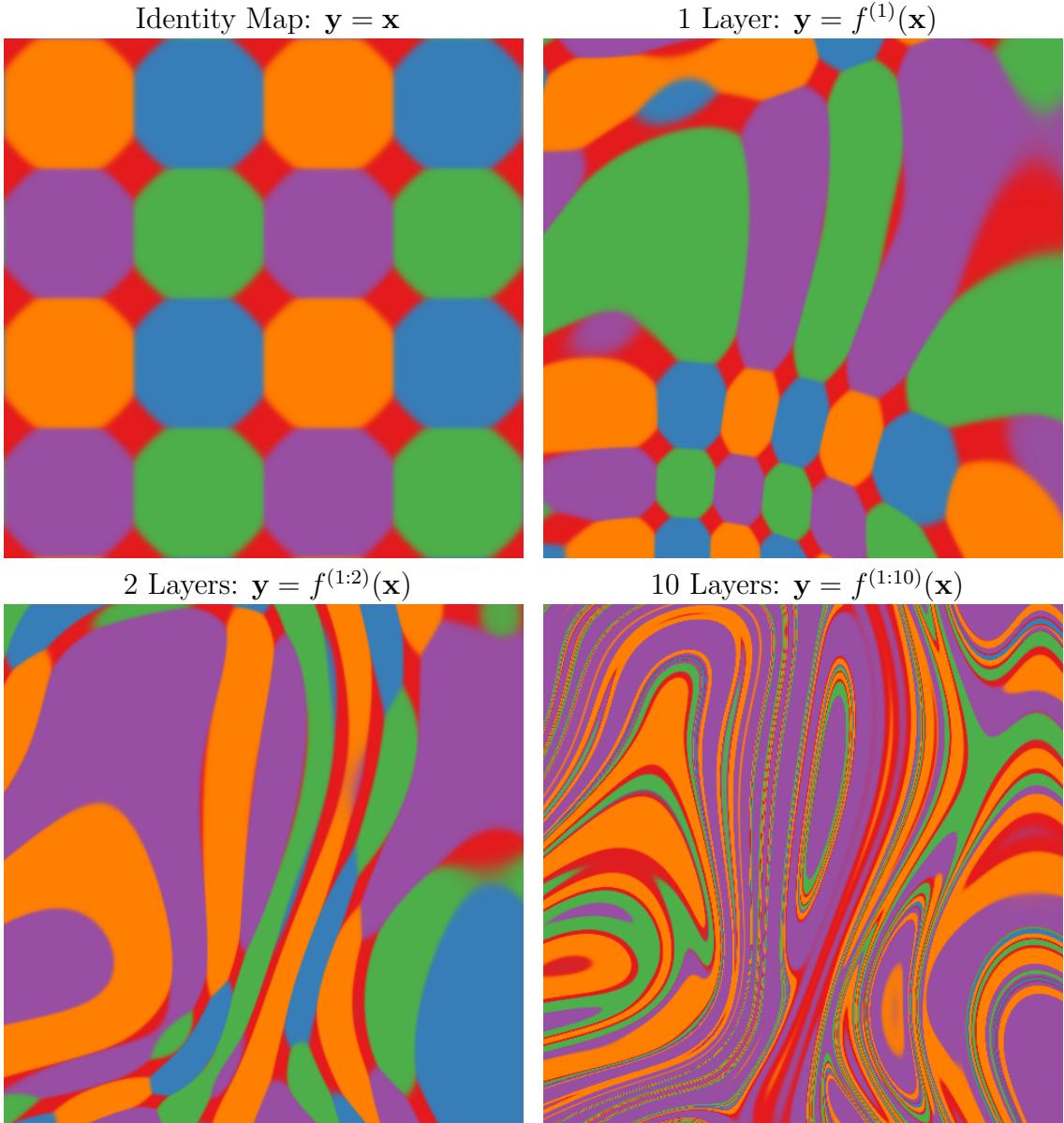
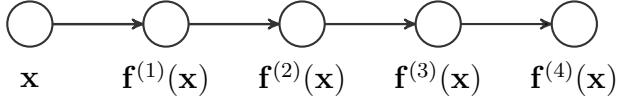
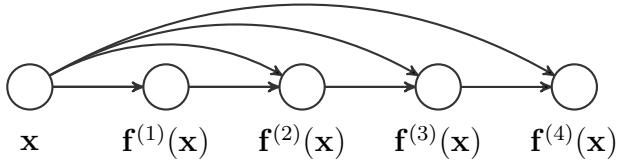


Fig. 4.6 Feature mapping of a deep GP. Colors correspond to the location $\mathbf{y} = \mathbf{f}(\mathbf{x})$ that each point is mapped to after being warped by a deep GP. The number of directions in which the color changes rapidly corresponds to the number of large singular values in the Jacobian. Just as the densities in figure 4.5 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} .

tures. Similar connections between non-adjacent layers can also be found in the primate visual cortex (Maunsell and van Essen, 1983). Formally, this functional dependence can



a) The standard MLP connectivity architecture.



b) Input-connected architecture.

Fig. 4.7 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.

be written as

$$\mathbf{f}^{(1:L)}(\mathbf{x}) = \mathbf{f}^{(L)} \left(\mathbf{f}^{(1:L-1)}(\mathbf{x}), \mathbf{x} \right), \quad \forall L \quad (4.16)$$

Draws from the resulting prior are shown in figures 4.8, 4.9 and 4.11. The Jacobian of the composed, input-connected deep function is defined by the recurrence

$$J^{1:L}(\mathbf{x}) = J^L \begin{bmatrix} J^{1:L-1} \\ I_D \end{bmatrix}. \quad (4.17)$$

Figure 4.10 shows that with this architecture, even 50-layer deep gps have well-behaved singular value spectra.

4.6 Deep kernels

Bengio et al. (2006b) 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 $x \rightarrow [\sin(x), \cos(x)]$, 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? Cho (2012) constructed kernels of this form, repeatedly applying

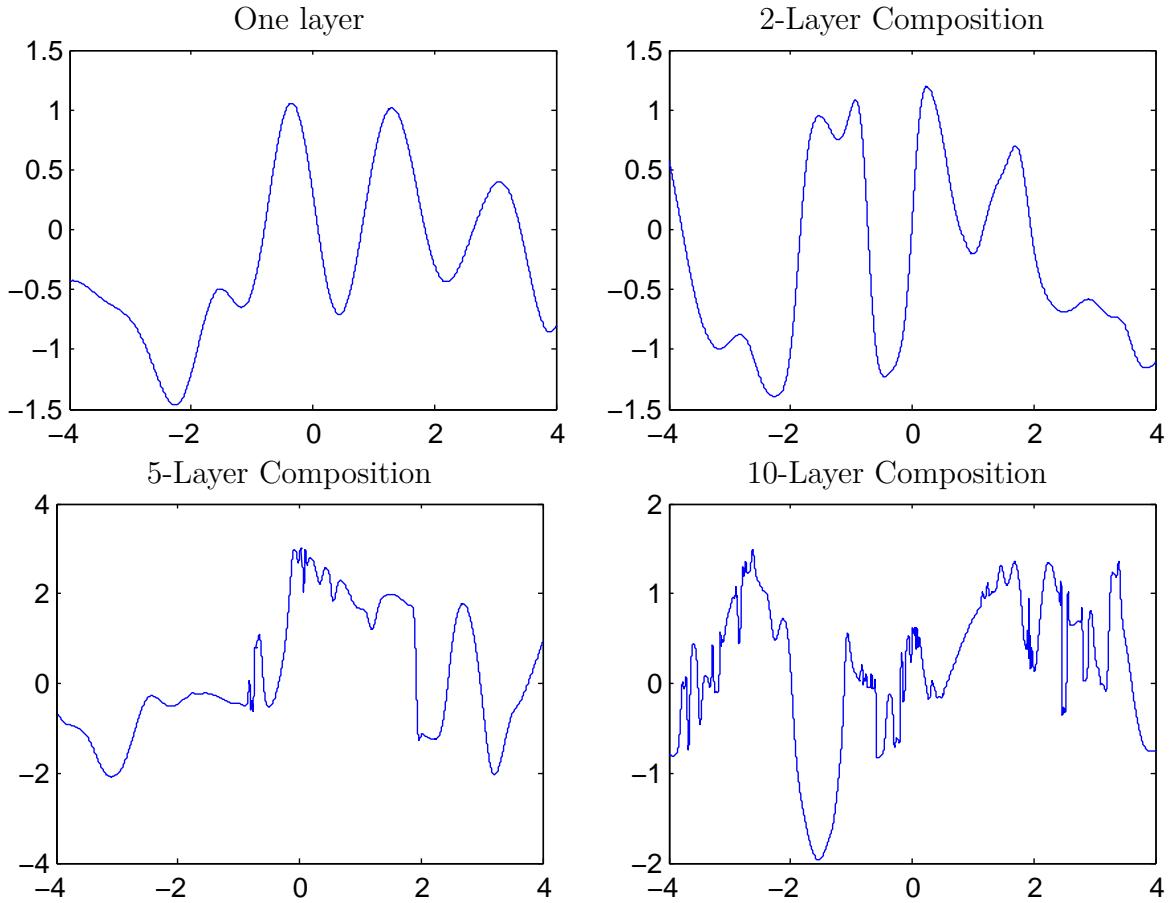


Fig. 4.8 Draws from a 1D deep GP prior with each layer connected to the input. Even after many layers, the functions remain smooth in some regions, while varying rapidly in other regions. Compare to standard-connectivity deep GP draws shown in figure 4.2.

multiple layers of feature mappings. We can compose the feature mapping of two kernels:

$$k_1(\mathbf{x}, \mathbf{x}') = \mathbf{h}_1(\mathbf{x})^\top \mathbf{h}_1(\mathbf{x}') \quad (4.18)$$

$$k_2(\mathbf{x}, \mathbf{x}') = \mathbf{h}_2(\mathbf{x})^\top \mathbf{h}_2(\mathbf{x}') \quad (4.19)$$

$$(k_1 \circ k_2)(\mathbf{x}, \mathbf{x}') = k_2(\mathbf{h}_1(\mathbf{x}), \mathbf{h}_1(\mathbf{x}')) \quad (4.20)$$

$$= [\mathbf{h}_2(\mathbf{h}_1(\mathbf{x}))]^\top \mathbf{h}_2(\mathbf{h}_1(\mathbf{x}')) \quad (4.21)$$

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

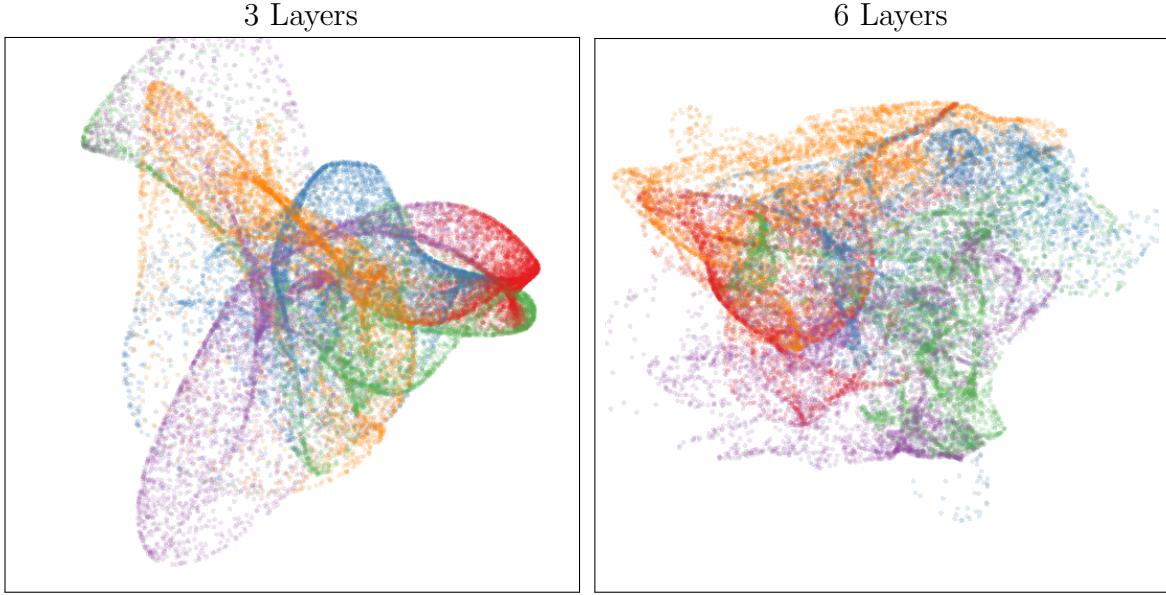


Fig. 4.9 Left: Densities defined by a draw from a deep GP, with each layer connected to the input \mathbf{x} . As depth increases, the density becomes more complex without concentrating along filaments.

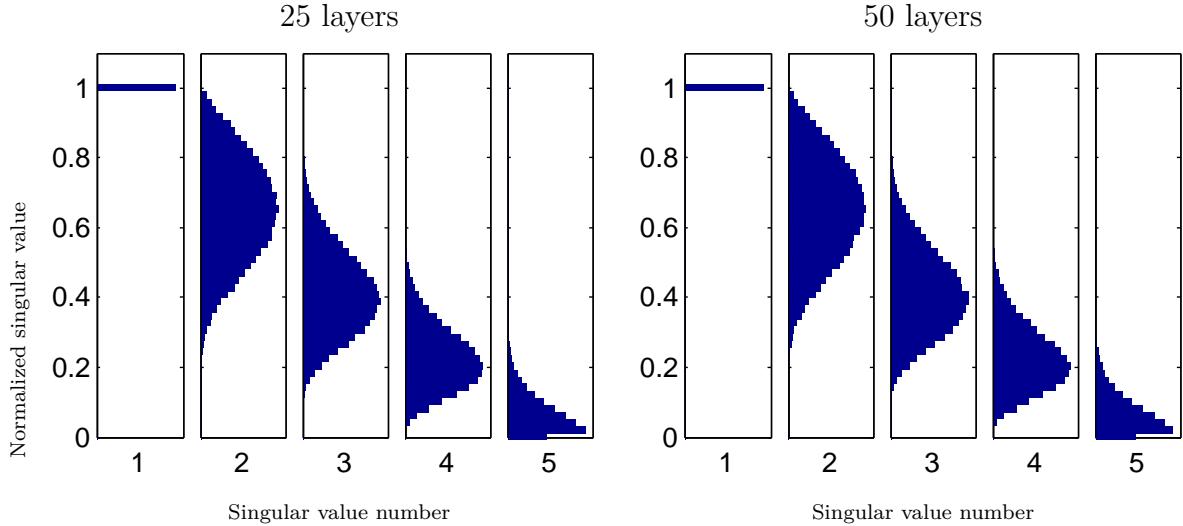


Fig. 4.10 The distribution of singular values drawn from 5-dimensional input-connected deep GP priors 25 and 50 layers deep. The singular values remain roughly the same scale as one another.

closed form:

$$\begin{aligned}
 k_{L+1}(\mathbf{x}, \mathbf{x}') &= k_{SE}(\mathbf{h}(\mathbf{x}), \mathbf{h}(\mathbf{x}')) = & (4.22) \\
 &= \exp\left(-\frac{1}{2}\|\mathbf{h}(\mathbf{x}) - \mathbf{h}(\mathbf{x}')\|_2^2\right) \\
 &= \exp\left(-\frac{1}{2}\left[\mathbf{h}(\mathbf{x})^\top \mathbf{h}(\mathbf{x}) - 2\mathbf{h}(\mathbf{x})^\top \mathbf{h}(\mathbf{x}') + \mathbf{h}(\mathbf{x}')^\top \mathbf{h}(\mathbf{x}')\right]\right) \\
 &= \exp\left(-\frac{1}{2}[k_L(\mathbf{x}, \mathbf{x}) - 2k_L(\mathbf{x}, \mathbf{x}') + k_L(\mathbf{x}', \mathbf{x}')]\right)
 \end{aligned}$$

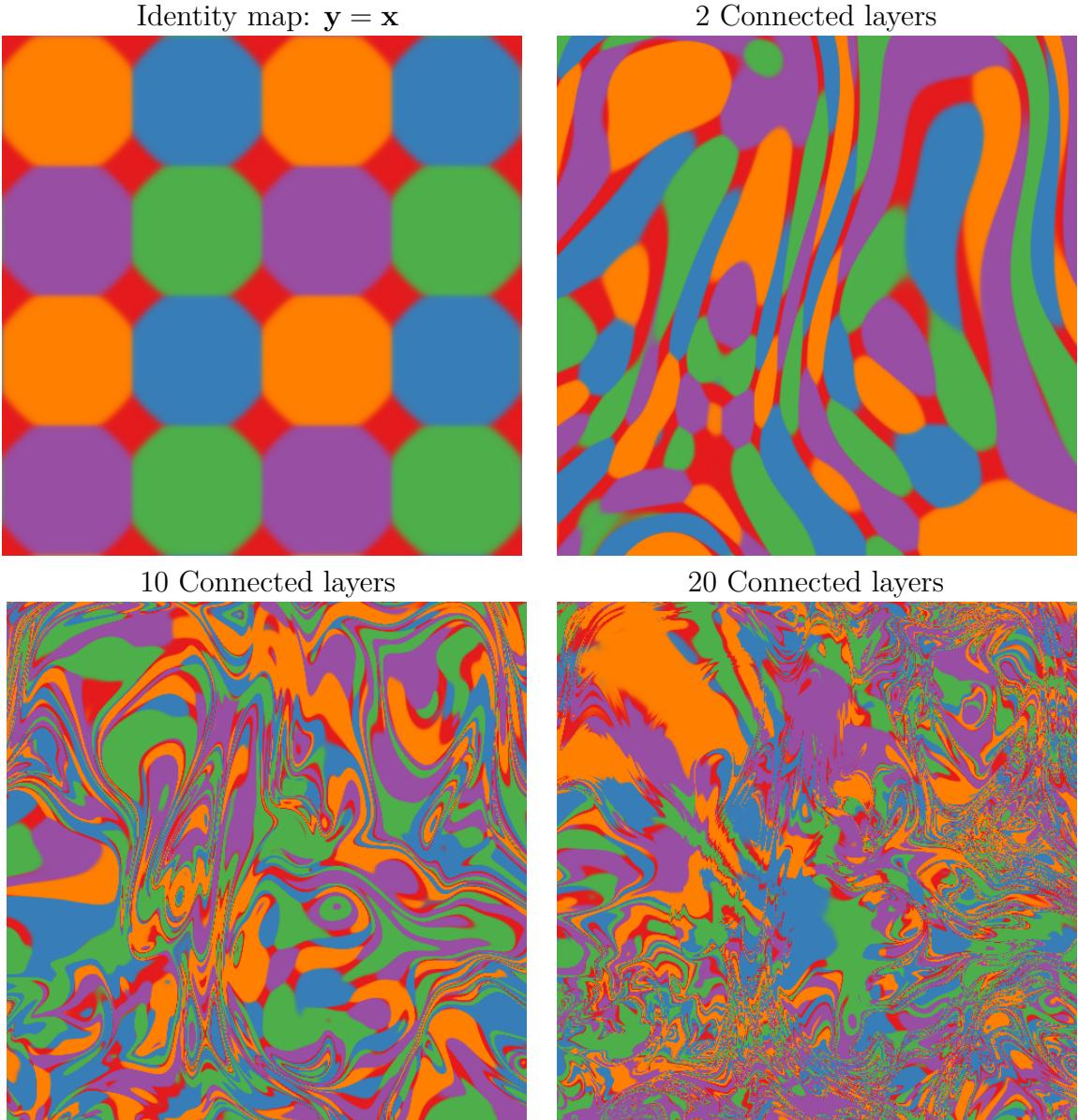


Fig. 4.11 Feature mapping of a deep GP with each layer connected to the input \mathbf{x} . Just as the densities in figure 4.9 remained locally two-dimensional even after many transformations, in this mapping there are often two directions that one can move locally in \mathbf{x} to in order to change 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 4.6.

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:

$$\begin{aligned} k_{L+1}(\mathbf{x}, \mathbf{x}') &= \exp\left(-\frac{1}{2}\left\|\begin{bmatrix} \mathbf{h}_L(\mathbf{x}) \\ \mathbf{x} \end{bmatrix} - \begin{bmatrix} \mathbf{h}_L(\mathbf{x}') \\ \mathbf{x}' \end{bmatrix}\right\|_2^2\right) \\ &= \exp\left(-\frac{1}{2}\left[k_L(\mathbf{x}, \mathbf{x}) - 2k_L(\mathbf{x}, \mathbf{x}') + k_L(\mathbf{x}', \mathbf{x}') - \|\mathbf{x} - \mathbf{x}'\|_2^2\right]\right) \end{aligned} \quad (4.23)$$

For the SE kernel, this repeated mapping satisfies

$$k_\infty(\mathbf{x}, \mathbf{x}') - \log(k_\infty(\mathbf{x}, \mathbf{x}')) = 1 + \frac{1}{2}\|\mathbf{x} - \mathbf{x}'\|_2^2 \quad (4.24)$$

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

4.6.1 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 (Duvenaud et al., 2013), and can enable many types of generalization, such as translation and rotation invariance in images (Kondor, 2008). Salakhutdinov and Hinton (2008) 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.

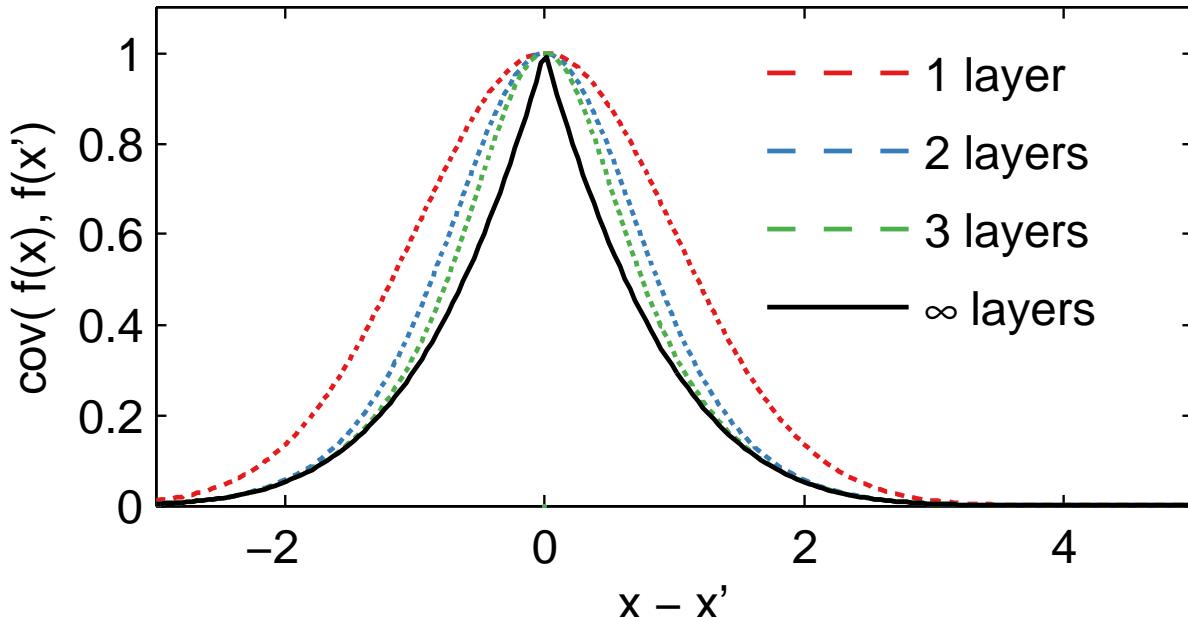


Fig. 4.12 Input-connected deep kernels. By connecting the inputs \mathbf{x} to each layer, the kernel can still depend on its input even after arbitrarily many layers of computation.

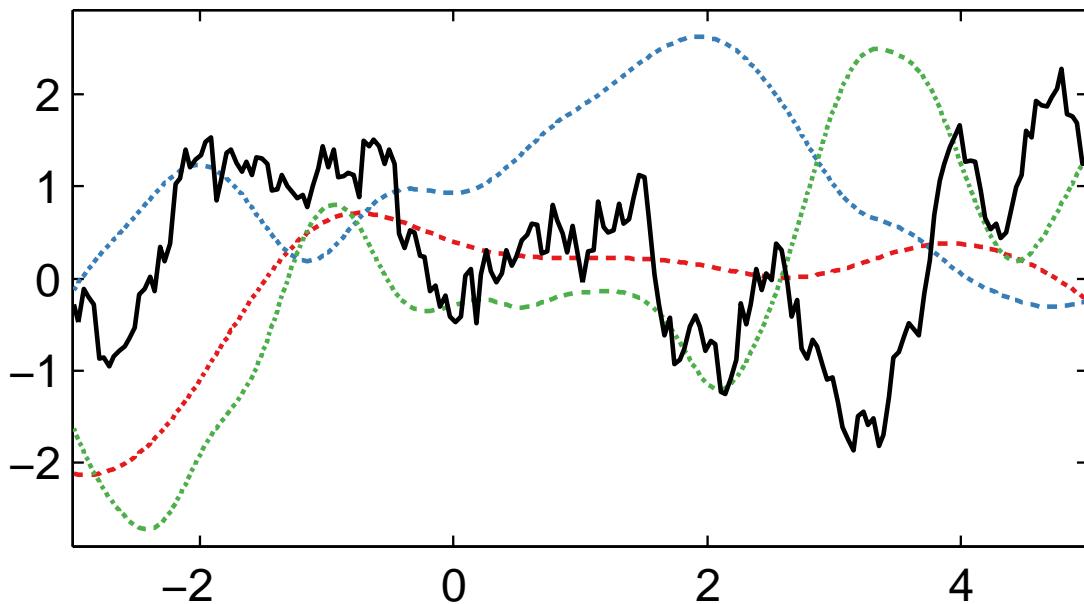


Fig. 4.13 GP draws using deep input-connected kernels.

4.7 Dropout in Gaussian processes

Dropout is a method for regularizing neural networks (Hinton et al., 2012; Srivastava, 2013). 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.

Baldi and Sadowski (2013) and Wang and Manning (2013) analyzed dropout in terms of the effective prior induced by this procedure in several models, such as linear and logistic regression. In this section, we examine the priors on functions that result from performing dropout in the one-hidden-layer neural network implicitly defined by a gp (equation (4.4)).

4.7.1 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:

$$r_i \stackrel{\text{iid}}{\sim} \text{Ber}(p) \quad \mathbb{E}[r_i \alpha_i] = p\mu, \mathbb{V}[r_i \alpha_i] = p^2\sigma^2. \quad (4.25)$$

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

$$\mathbb{E}\left[\frac{1}{p}r_i \alpha_i\right] = \frac{p}{p}\mu = \mu, \quad \mathbb{V}\left[\frac{1}{p}r_i \alpha_i\right] = \frac{p^2}{p^2}\sigma^2 = \sigma^2. \quad (4.26)$$

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.

4.7.2 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:

$$p(f(\mathbf{x})) = \frac{1}{2^D} \sum_{\mathbf{r} \in \{0,1\}^D} \text{gp} \left(0, \prod_{d=1}^D k_d(\mathbf{x}_d, \mathbf{x}'_d)^{r_d} \right) \quad (4.27)$$

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 (4.10), 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 (4.27) has the same covariance as

$$f(\mathbf{x}) \sim \text{gp} \left(0, \frac{1}{2^{-2D}} \sum_{\mathbf{r} \in \{0,1\}^D} \prod_{d=1}^D k_d(\mathbf{x}_d, \mathbf{x}'_d)^{r_d} \right) \quad (4.28)$$

A gp whose covariance is a sum of kernels corresponds to a sum of functions, each distributed according to a gp. Therefore, (4.28) describes a sum of 2^D functions, each depending on a different subset of the inputs. This model class was studied by Duvenaud et al. (2011), who showed that exact inference in these models can be performed in $\mathcal{O}(N^2 D^2 + N^3)$.

4.8 Related work

Deep gps were first proposed by Lawrence and Moore (2007). Variational inference in deep gps was developed by Damianou and Lawrence (2013), who also analyzed the effect of automatic relevance determination in that model.

Adams et al. (2010) 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 (Rippel and Adams, 2013) 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.

Recurrent networks Bengio et al. (1994) and Pascanu et al. (2012) analyze a related problem with gradient-based learning in recurrent nets, the “exploding-gradients” problem. Hermans and Schrauwen (2012) analyze deep kernels corresponding to recurrent neural networks.

Analysis of deep learning Montavon et al. (2010) 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 Saxe et al. (2011). Saxe et al. (2013) looked at the dynamics of learning in deep linear models, as a tractable approximation to deep neural networks.

4.9 Conclusions

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

First, we identified deep Gaussian processes as an easy-to-analyze model corresponding to multi-layer perceptrons 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.

References

- Ryan P Adams, Hanna M Wallach, and Zoubin Ghahramani. Learning the structure of deep sparse graphical models. In *International Conference on Artificial Intelligence and Statistics*, pages 1–8, 2010. (page 52)
- Francis Bach. High-dimensional non-linear variable selection through hierarchical kernel learning. *CoRR*, abs/0909.0844, 2009. (pages 18, 23, 24, 27, and 29)
- Pierre Baldi and Peter J Sadowski. Understanding dropout. In *Advances in Neural Information Processing Systems*, pages 2814–2822, 2013. (page 51)
- Y. Bengio, O. Delalleau, and N. Le Roux. The curse of highly variable functions for local kernel machines. *Advances in neural information processing systems*, 18:107, 2006a. ISSN 1049-5258. (page 25)
- Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166, 1994. (page 53)
- Yoshua Bengio, Olivier Delalleau, and Nicolas Le Roux. The curse of highly variable functions for local kernel machines. pages 107–114, 2006b. (page 45)
- Youngmin Cho. *Kernel methods for deep learning*. PhD thesis, University of California, San Diego, 2012. (page 45)
- M. Christoudias, R. Urtasun, and T. Darrell. Bayesian localized multiple kernel learning. 2009. (page 23)
- Andreas Damianou and Neil Lawrence. Deep Gaussian processes. In *Artificial Intelligence and Statistics*, pages 207–215, 2013. (pages 34 and 52)
- David Duvenaud, Hannes Nickisch, and Carl Edward Rasmussen. Additive Gaussian processes. In *Advances in Neural Information Processing Systems 24*, pages 226–234, Granada, Spain, 2011. (page 52)
- David Duvenaud, James Robert Lloyd, Roger Grosse, Joshua B. Tenenbaum, and Zoubin Ghahramani. Structure discovery in nonparametric regression through compositional kernel search. In *Proceedings of the 30th International Conference on Machine Learning*, June 2013. (page 49)

- Robert Gens and Pedro Domingos. Learning the structure of sum-product networks. In *International Conference on Machine learning*, 2013. (page 33)
- T.J. Hastie and R.J. Tibshirani. *Generalized additive models*. Chapman & Hall/CRC, 1990. (page 17)
- Michiel Hermans and Benjamin Schrauwen. Recurrent kernel machines: Computing with infinite echo state networks. *Neural Computation*, 24(1):104–133, 2012. (page 53)
- Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. (page 51)
- C.G. Kaufman and S.R. Sain. Bayesian functional anova modeling using gaussian process prior distributions. *Bayesian Analysis*, 5(1):123–150, 2010. (page 23)
- Imre Risi Kondor. *Group theoretical methods in machine learning*. PhD thesis, Columbia University, 2008. (page 49)
- Neil D Lawrence and Andrew J Moore. Hierarchical Gaussian process latent variable models. In *Proceedings of the 24th international conference on Machine learning*, pages 481–488. ACM, 2007. (page 52)
- Honglak Lee, Chaitanya Ekanadham, and Andrew Ng. Sparse deep belief net model for visual area v2. In *Advances in neural information processing systems*, pages 873–880, 2007. (page 33)
- I.G. Macdonald. *Symmetric functions and Hall polynomials*. Oxford University Press, USA, 1998. ISBN 0198504500. (page 22)
- James Martens. Deep learning via Hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning*, pages 735–742, 2010. (page 33)
- JH Maunsell and David C van Essen. The connections of the middle temporal visual area (mt) and their relationship to a cortical hierarchy in the macaque monkey. *The Journal of neuroscience*, 3(12):2563–2586, 1983. (page 44)
- T.P. Minka. Expectation propagation for approximate bayesian inference. In *Uncertainty in Artificial Intelligence*, volume 17, pages 362–369. Citeseer, 2001. (page 27)
- Grégoire Montavon, Dr Braun, and Klaus-Robert Müller. Layer-wise analysis of deep networks with Gaussian kernels. *Advances in Neural Information Processing Systems*, 23:1678–1686, 2010. (page 53)
- Radford M Neal. *Bayesian learning for neural networks*. PhD thesis, University of Toronto, 1995. (pages 35 and 43)
- J.A. Nelder and R.W.M. Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society. Series A (General)*, 135(3):370–384, 1972. (page 17)
- J. Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980. (page 27)

- Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *arXiv preprint arXiv:1211.5063*, 2012. (page 53)
- T.A. Plate. Accuracy versus interpretability in flexible modeling: Implementing a trade-off using gaussian process models. *Behaviormetrika*, 26:29–50, 1999. ISSN 0385-7417. (pages 21 and 23)
- C.E. Rasmussen and CKI Williams. Gaussian Processes for Machine Learning. *The MIT Press, Cambridge, MA, USA*, 2006. (pages 7 and 18)
- Salah Rifai, Grégoire Mesnil, Pascal Vincent, Xavier Muller, Yoshua Bengio, Yann Dauphin, and Xavier Glorot. Higher order contractive auto-encoder. In *Machine Learning and Knowledge Discovery in Databases*, pages 645–660. Springer, 2011a. (page 41)
- Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning*, pages 833–840, 2011b. (page 41)
- Oren Rippel and Ryan Prescott Adams. High-dimensional probability estimation with deep density models. *arXiv preprint arXiv:1302.5125*, 2013. (page 52)
- Ruslan Salakhutdinov and Geoffrey Hinton. Using deep belief nets to learn covariance kernels for Gaussian processes. In *Advances in Neural Information Processing Systems*, volume 20, 2008. (page 49)
- Andrew Saxe, Pang W Koh, Zhenghao Chen, Maneesh Bhand, Bipin Suresh, and Andrew Y Ng. On random weights and unsupervised feature learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1089–1096, 2011. (page 53)
- Andrew M Saxe, James L McClelland, and Surya Ganguli. Dynamics of learning in deep linear neural networks. 2013. (page 53)
- J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis*. Cambridge University Press, 2004. ISBN 0521813972. (pages 31 and 32)
- Ercan Solak, Roderick Murray-Smith, E. Solak, William Leithead, Carl Rasmussen, and Douglas Leith. Derivative observations in Gaussian process models of dynamic systems, 2003. (page 40)
- Nitish Srivastava. *Improving neural networks with dropout*. PhD thesis, University of Toronto, 2013. (page 51)
- R.P. Stanley. *Enumerative combinatorics*. Cambridge University Press, 2001. ISBN 0521789877. (page 22)
- M. Stitson, A. Gammerman, V. Vapnik, V. Vovk, C. Watkins, and J. Weston. Support vector regression with ANOVA decomposition kernels. *Advances in kernel methods—Support vector learning*, pages 285–292, 1999. (page 24)

- V.N. Vapnik. *Statistical learning theory*, volume 2. Wiley New York, 1998. (page 24)
- G. Wahba. *Spline models for observational data*. Society for Industrial Mathematics, 1990. ISBN 0898712440. (pages 17 and 25)
- Sida Wang and Christopher Manning. Fast dropout training. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 118–126, 2013. (page 51)