Chapter 1

Additive Gaussian Processes

Section 1.7 showed how to learn the structure of a kernel by building it up piece-by-piece. This chapter presents an alternative approach: starting with many different types of structure in a kernel, adjusting kernel parameters to discard whatever structure is *not* present in the current dataset. The advantage of this approach is that we do not need to run an expensive discrete-and-continuous search in order to build a structured model, and implementation is simpler.

This model, which we call *additive Gaussian processes*, is a sum of functions of all possible combinations of input variables. This model can be specified by a weighted sum of all possible products of one-dimensional kernels.

There are 2^D combinations of D objects, so naïve computation of this kernel is intractable. Furthermore, if each term has different kernel parameters, fitting or integrating over so many parameters is difficult. To address these problems, we introduce a restricted parameterization of the kernel which allows efficient evaluation of all interaction terms, while still allowing a different weighting of each order of interaction. Empirically, this model has good predictive performance in regression tasks, and its parameters are relatively interpretable. This model also has an interpretation as an approximation to *dropout*, a recently-introduced regularization method for neural networks.

The work in this chapter was done in collaboration with Hannes Nickisch and Carl Rasmussen, who derived and coded up the additive kernel. My role in the project was to examine the properties of the resulting model, clarify the connections to existing methods, to create all figures and run all experiments. That work was published in Duvenaud et al. (2011). The connection to dropout regularization in section 1.4 is an independent contribution.

1.1 Different types of multivariate additive structure

Section 1.7 showed how additive structure in a GP prior enabled extrapolation in multivariate regression problems. In general, models of the form

$$f(\mathbf{x}) = g\Big(f(x_1) + f(x_2) + \cdots + f(x_D)\Big) \tag{1.1}$$

are widely used in machine learning and statistics, partly for this reason, and partly because they are relatively easy to fit and interpret. Examples include logistic regression, linear regression, generalized linear models (Nelder and Wedderburn, 1972) and generalized additive models (Hastie and Tibshirani, 1990).

At the other end of the spectrum are models which allow the response to depend on all input variables simultaneously, without any additive decomposition:

$$f(\mathbf{x}) = f(x_1, x_2, \ldots, x_D) \tag{1.2}$$

An example would be a GP with an SE-ARD kernel. Such models are much more flexible than those having the form (1.1), but this flexibility can make it difficult to generalize to new combinations of input variables.

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

$$f(x_1, x_2, x_3) = f_{12}(x_1, x_2) + f_{23}(x_2, x_3) + f_{13}(x_1, x_3). \tag{1.3}$$

We call the number of input variables appearing in each term the *order* of that term. Models containing terms only of intermediate order such as (1.3) allow more flexibility than models of form (1.2) (first-order), but have more structure than those of form (1.1) (D-th order).

Capturing the low-order additive structure present in a function can be expected to improve predictive accuracy. However, if the function being learned depends in some way on an interaction between all input variables, a Dth-order term is required in order for the model to be consistent.

1.2 Defining additive kernels

To define the additive kernels introduced in this chapter, we first assign each dimension $i \in \{1 \dots D\}$ a one-dimensional *base kernel* $k_i(x_i, x_i')$. Then the first order, second order and nth order additive kernels are defined as:

$$k_{add_1}(\mathbf{x}, \mathbf{x}') = \sigma_1^2 \sum_{i=1}^{D} k_i(x_i, x_i') \tag{1.4}$$

$$k_{add_2}(\mathbf{x}, \mathbf{x}') = \sigma_2^2 \sum_{i=1}^{D} \sum_{j=i+1}^{D} k_i(x_i, x_i') k_j(x_j, x_j') \tag{1.5}$$

$$k_{add_n}(\mathbf{x}, \mathbf{x}') = \sigma_n^2 \sum_{1 \le i_1 < i_2 < \ldots < i_n \le D} \left[\prod_{d=1}^{n} k_{i_d}(x_{i_d}, x_{i_d}') \right] \tag{1.6}$$

$$k_{add_D}(\mathbf{x}, \mathbf{x}') = \sigma_D^2 \sum_{1 \le i_1 < i_2 < \ldots < i_D \le D} \left[\prod_{d=1}^{D} k_{i_d}(x_{i_d}, x_{i_d}') \right] = \sigma_D^2 \prod_{d=1}^{D} k_d(x_d, x_d') \tag{1.7}$$

where D is the dimension of the input space, and σ_n^2 is the variance assigned to all nth order interactions. The nth-order kernel is a sum of $\binom{D}{n}$ terms. In particular, the Dth-order additive kernel has $\binom{D}{D} = 1$ term, a product of each dimension's kernel. In the case where each base kernel is a one-dimensional squared-exponential kernel, the Dth-order term corresponds to the multivariate squared-exponential kernel, also known as SE-ARD:

$$\prod_{d=1}^{D} \mathrm{SE}(x_d, x_d') = \prod_{d=1}^{D} \sigma_d^2 \exp\left(-\frac{(x_d - x_d')^2}{2\ell_d^2} \right) = \sigma_D^2 \exp\left(-\sum_{d=1}^{D} \frac{(x_d - x_d')^2}{2\ell_d^2} \right) \tag{1.8}$$

The full additive kernel is a sum of the additive kernels of all orders.

The only design choice necessary to specify an additive kernel is the selection of a one-dimensional base kernel for each input dimension. Parameters of the base kernels (such as length-scales $\ell_1, \ell_2, \ldots, \ell_D$) can be learned as per usual by maximizing the marginal likelihood of the training data.

1.2.1 Weighting different orders of interaction

In addition to the parameters of each dimension's kernel, additive kernels are equipped with a set of D parameters $\sigma_1^2 \dots \sigma_D^2$. These *order variance* parameters have a useful interpretation: the dth order variance parameter specifies how much of the target function's variance comes from interactions of the dth order.

Table 1.1 shows examples of the variance contributed by different orders of interaction, estimated on real datasets. These datasets are described in section 1.6.1.

Table 1.1: Percentage of variance contributed by each order of interaction of the additive model on different datasets. The maximum order of interaction is set to the input dimension or 10, whichever is smaller.

	Order of interaction									
Dataset	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
pima	0.1	0.1	0.1	0.3	1.5	**96.4**	1.4	0.0		
liver	0.0	0.2	**99.7**	0.1	0.0	0.0				
heart	**77.6**	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.1	22.0
concrete	**70.6**	13.3	13.8	2.3	0.0	0.0	0.0	0.0		
pumadyn-8nh	0.0	0.1	0.1	0.1	0.1	0.1	0.1	**99.5**		
servo	**58.7**	27.4	0.0	13.9						
housing	0.1	0.6	**80.6**	1.4	1.8	0.8	0.7	0.8	0.6	12.7

On different datasets, the dominant order of interaction estimated by the additive model varies widely. In some cases, the variance is concentrated almost entirely onto a single order of interaction. This may may be a side-effect of using the same lengthscales for all orders of interaction; lengthscales appropriate for low-dimensional regression might not be appropriate for high-dimensional regression.

1.2.2 Efficiently evaluating additive kernels

An additive kernel over D inputs with interactions up to order n has $O(2^n)$ terms. Naïvely summing these terms is intractable. One can exactly evaluate the sum over all terms in $O(D^2)$, while also weighting each order of interaction separately.

To efficiently compute the additive kernel, we exploit the fact that the nth order additive kernel corresponds to the nth *elementary symmetric polynomial* (Macdonald, 1998) of the base kernels, which we denote e_n. For example, if $\mathbf{x}$ has 4 input dimensions

($D = 4$), and if we use the shorthand notation $k_d = k_d(x_d, x'_d)$, then

$$k_{\text{add}_0}(\mathbf{x}, \mathbf{x}') = e_0(k_1, k_2, k_3, k_4) = 1 \tag{1.9}$$

$$k_{\text{add}_1}(\mathbf{x}, \mathbf{x}') = e_1(k_1, k_2, k_3, k_4) = k_1 + k_2 + k_3 + k_4 \tag{1.10}$$

$$k_{\text{add}_2}(\mathbf{x}, \mathbf{x}') = e_2(k_1, k_2, k_3, k_4) = k_1k_2 + k_1k_3 + k_1k_4 + k_2k_3 + k_2k_4 + k_3k_4 \tag{1.11}$$

$$k_{\text{add}_3}(\mathbf{x}, \mathbf{x}') = e_3(k_1, k_2, k_3, k_4) = k_1k_2k_3 + k_1k_2k_4 + k_1k_3k_4 + k_2k_3k_4 \tag{1.12}$$

$$k_{\text{add}_4}(\mathbf{x}, \mathbf{x}') = e_4(k_1, k_2, k_3, k_4) = k_1k_2k_3k_4 \tag{1.13}$$

The Newton-Girard formulas give an efficient recursive form for computing these polynomials.

$$k_{\text{add}_n}(\mathbf{x}, \mathbf{x}') = e_n(k_1, k_2, \ldots, k_D) = \frac{1}{n}\sum_{a=1}^{n}(-1)^{(a-1)}e_{n-a}(k_1, k_2, \ldots, k_D)\sum_{i=1}^{D}k_i^a \tag{1.14}$$

Each iteration has cost $\mathcal{O}(D)$, given the next-lowest polynomial.

Evaluation of derivatives

Conveniently, we can use the same trick to efficiently compute the necessary derivatives of the additive kernel with respect to the base kernels. This can be done by removing the base kernel of interest k_j from each term of the polynomials:

$$\frac{\partial k_{add_n}}{\partial k_j} = \frac{\partial e_n(k_1, k_2, \ldots, k_D)}{\partial k_j} = e_{n-1}(k_1, k_2, \ldots, k_{j-1}, k_{j+1}, \ldots k_D) \tag{1.15}$$

Equation (1.15) gives all terms that k_j is multiplied by in the original polynomial, which are exactly the terms required by the chain rule. These derivatives allow gradient-based optimization of the base kernel parameters with respect to the marginal likelihood.

Computational cost

The computational cost of evaluating the Gram matrix $k(\mathbf{X}, \mathbf{X})$ of a product kernel such as the SE-ARD scales as $\mathcal{O}(N^2D)$, while the cost of evaluating the Gram matrix of the additive kernel scales as $\mathcal{O}(N^2DR)$, where R is the maximum degree of interaction allowed (up to D). In high dimensions this can be a significant cost, even relative to the $\mathcal{O}(N^3)$ cost of inverting the Gram matrix. However, table 1.1 shows that sometimes only the first few orders of interaction contribute much variance. In those cases, one may be able to limit the maximum degree of interaction in order to save time, without

losing much accuracy.

1.3 Additive models allow non-local interactions

Commonly-used kernels such as the SE, RQ or Matérn kernels are *local* kernels, depending only on the scaled Euclidean distance between two points, all having the form:

$$k(\mathbf{x}, \mathbf{x}') = g\left(\sum_{d=1}^{D}\left(\frac{x_d - x'_d}{\ell_d}\right)^2\right) \tag{1.16}$$

for some function $g(\cdot)$. Bengio et al. (2006) argued that models based on local kernels are particularly susceptible to the *curse of dimensionality* (Bellman, 1956), and are generally unable to extrapolate away from the training data. Methods based solely on local kernels sometimes require training examples at exponentially-many combinations of inputs.

In contrast, additive kernels can allow extrapolation away from the training data. For example, additive kernels of second order give high covariance between function values at input locations which are similar in any two dimensions.

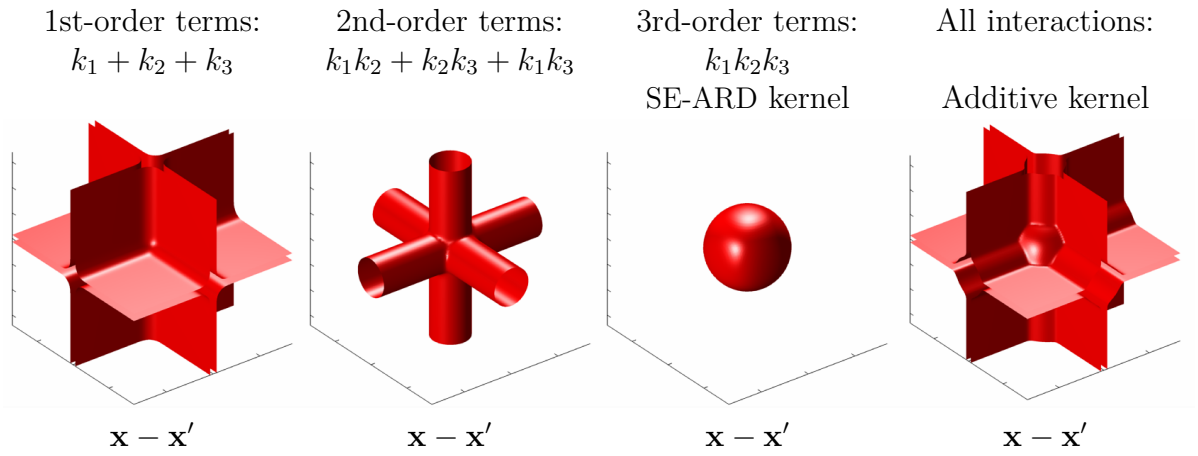

Figure 1.1: Isocontours of additive kernels in $D = 3$ dimensions. The Dth-order kernel only considers nearby points relevant, while lower-order kernels allow the output to depend on distant points, as long as they share one or more input value.

Figure 1.1 provides a geometric comparison between squared-exponential kernels and additive kernels in 3 dimensions. **??** contains an example of how additive kernels extrapolate differently than local kernels.

1.4 Dropout in Gaussian processes

Dropout is a recently-introduced method for regularizing neural networks (Hinton et al., 2012; Srivastava, 2013). Training with dropout entails independently setting to zero ("dropping") 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, the remaining weights are divided by p. Predictions are made by approximately averaging over all possible ways of dropping out neurons.

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 perform a similar analysis for GPs, examining the priors on functions that result from performing dropout in the one-hidden-layer neural network implicitly defined by a GP.

Recall from **??** that some GPs can be derived as infinitely-wide one-hidden-layer neural networks, with fixed activation functions $\mathbf{h}(\mathbf{x})$ and independent random weights $\mathbf{w}$ having zero mean and finite variance $\sigma_{\mathbf{w}}^2$:

$$f(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K} w_i h_i(\mathbf{x}) \implies f \overset{K\to\infty}{\sim} \mathcal{GP}\Big(0, \sigma_{\mathbf{w}}^2 \mathbf{h}(\mathbf{x})^\mathsf{T}\mathbf{h}(\mathbf{x}')\Big)\,. \tag{1.17}$$

1.4.1 Dropout on infinitely-wide hidden layers has no effect

First, we examine the prior obtained by dropping features from $\mathbf{h}(\mathbf{x})$ by setting weights in $\mathbf{w}$ to zero independently with probability p. For simplicity, we assume that $\mathbb{E}[\mathbf{w}] = \mathbf{0}$. If the weights w_i initially have finite variance σ_w^2 before dropout, then the weights after dropout (denoted by $r_i w_i$, where r_i is a Bernoulli random variable) will have variance:

$$r_i \overset{\text{iid}}{\sim} \text{Ber}(p) \qquad \mathbb{V}[r_i w_i] = p\sigma_w^2\,. \tag{1.18}$$

Because equation (1.17) is a result of the central limit theorem, it does not depend on the exact form of the distribution on $\mathbf{w}$, but only on its mean and variance. Thus the central limit theorem still applies. Performing dropout on the features of an infinitely-wide MLP does not change the resulting model at all, except to rescale the output variance. Indeed, dividing all weights by $\sqrt{p}$ restores the initial variance:

$$\mathbb{V}\left[\frac{1}{p^{\frac{1}{2}}} r_i w_i\right] = \frac{p}{p}\sigma_w^2 = \sigma_w^2 \tag{1.19}$$

in which case dropout on the hidden units has no effect at all. Intuitively, this is because no individual feature can have more than an infinitesimal contribution to the network output.

This result does not hold in neural networks having a finite number of hidden features with Gaussian-distributed weights, another model class that also gives rise to GPs.

1.4.2 Dropout on inputs gives additive covariance

One can also perform dropout on the D inputs to the GP. For simplicity, consider a stationary product kernel $k(\mathbf{x}, \mathbf{x}') = \prod_{d=1}^{D} k_d(x_d, x'_d)$ which has been normalized such that $k(\mathbf{x}, \mathbf{x}) = 1$, and a dropout probability of $p = 1/2$. In this case, the generative model can be written as:

$$\mathbf{r} = [r_1, r_2, \ldots, r_D], \quad \text{each } r_i \overset{\text{iid}}{\sim} \text{Ber}\left(\frac{1}{2}\right), \quad f(\mathbf{x})|\mathbf{r} \sim \mathcal{GP}\left(0, \prod_{d=1}^{D} k_d(x_d, x'_d)^{r_d}\right) \tag{1.20}$$

This is a mixture of 2^D GPs, each depending on a different subset of the inputs:

$$p\left(f(\mathbf{x})\right) = \sum_{\mathbf{r}} p\left(f(\mathbf{x})|\mathbf{r}\right) p(\mathbf{r}) = \frac{1}{2^D} \sum_{\mathbf{r}\in\{0,1\}^D} \mathcal{GP}\left(f(\mathbf{x}) \,\middle|\, 0, \prod_{d=1}^{D} k_d(x_d, x'_d)^{r_d}\right) \tag{1.21}$$

We present two results which might give intuition about this model.

First, if the kernel on each dimension has the form $k_d(x_d, x'_d) = g\left(\frac{x_d - x'_d}{\ell_d}\right)$, as does the SE kernel, then any input dimension can be dropped out by setting its lengthscale ℓ_d to ∞. In this case, performing dropout on the inputs of a GP corresponds to putting independent spike-and-slab priors on the lengthscales, with each dimension's distribution independently having "spikes" at $\ell_d = \infty$ with probability mass of $1/2$.

Another way to understand the resulting prior is to note that the dropout mixture (equation (1.21)) has the same covariance as an additive GP, scaled by a factor of 2^{-D}:

$$\text{cov}\left(\begin{bmatrix} f(\mathbf{x}) \\ f(\mathbf{x}') \end{bmatrix}\right) = \frac{1}{2^D} \sum_{\mathbf{r}\in\{0,1\}^D} \prod_{d=1}^{D} k_d(x_d, x'_d)^{r_d} \tag{1.22}$$

For dropout rates $p \neq 1/2$, the dth order terms will be weighted by $p^{(D-d)}(1-p)^d$. Therefore, performing dropout on the inputs of a GP gives a distribution with the same first two moments as an additive GP. This suggests an interpretation of additive GPs as an approximation to a mixture of models where each model only depends on a subset

of the input variables.

1.5 Related work

Since additive models are a relatively natural and easy-to-analyze model class, the literature on similar model classes is extensive. This section attempts to provide a broad overview.

Previous examples of additive GPs

The additive models considered in this chapter are axis-aligned, but transforming the input space allows one to recover non-axis aligned additivity. This model was explored by Gilboa et al. (2013), who developed a linearly-transformed first-order additive GP model, called projection-pursuit GP regression. They showed that inference in this model was possible in $\mathcal{O}(N)$ time.

Durrande et al. (2011) also examined properties of additive GPs, and proposed a layer-wise optimization strategy for kernel hyperparameters in these models.

Plate (1999) constructed an additive GP having only first-order and Dth-order terms, motivated by the desire to trade off the interpretability of first-order models with the flexibility of full-order models. However, table 1.1 shows that sometimes the intermediate degrees of interaction contribute most of the variance.

Kaufman and Sain (2010) used a closely related procedure called Gaussian process ANOVA to perform a Bayesian analysis of meteorological data using 2nd and 3rd-order interactions. They introduced a weighting scheme to ensure that each order's total contribution sums to zero. It is not clear if this weighting scheme permits the use of the Newton-Girard formulas to speed computation of the Gram matrix.

Hierarchical kernel learning

A similar model class was recently explored by Bach (2009) called hierarchical kernel learning (HKL). HKL uses a regularized optimization framework to build a weighted sum of an exponential number of kernels that can be computed in polynomial time. This method chooses among a *hull* of kernels, defined as a set of terms such that if $\prod_{j \in J} k_j(\mathbf{x}, \mathbf{x}')$ is included in the set, then so are all products of strict subsets of the same elements: $\prod_{j \in J/i} k_j(\mathbf{x}, \mathbf{x}')$, for all $i \in J$. HKL does not estimate a separate weighting parameter for each order.

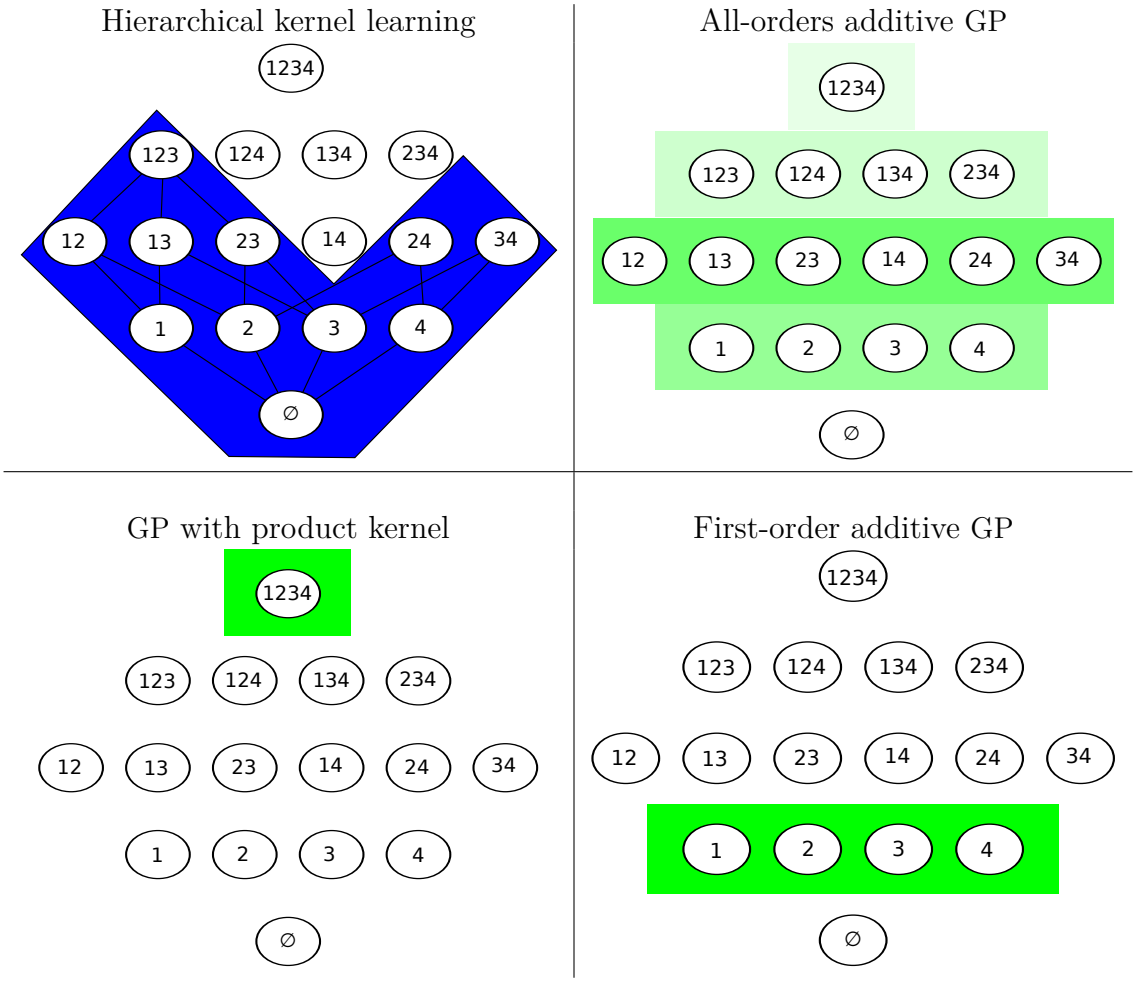

Figure 1.2: A comparison of different additive model classes of 4-dimensional functions. Circles represent different interaction terms, ranging from first-order to fourth-order interactions. Shaded boxes represent the relative weightings of different terms. *Top left:* HKL can select a hull of interaction terms, but must use a pre-determined weighting over those terms. *Top right:* the additive GP model can weight each order of interaction separately, but weights all terms equally within each order. *Bottom row:* GPs with product kernels (such as the SE-ARD kernel) and first-order additive GP models are special cases of the all-orders additive GP, with all variance assigned to a single order of interaction.

Figure 1.2 contrasts the HKL model class with the additive GP model. Neither model class encompasses the other. The main difficulty with this approach is that its parameters are hard to set other than by cross-validation.

Support vector machines

Vapnik (1998) introduced the support vector ANOVA decomposition, which has the same form as our additive kernel. They recommend approximating the sum over all interactions with only one set of interactions "of appropriate order", presumably because of the difficulty of setting the parameters of an SVM. This is an example of a model choice which can be automated in the GP framework.

Stitson et al. (1999) performed experiments which favourably compared the predictive accuracy of the support vector ANOVA decomposition against polynomial and spline kernels. They too allowed only one order to be active, and set parameters by cross-validation.

Other related models

A closely related procedure from Wahba (1990) is smoothing-splines ANOVA (SS-ANOVA). An SS-ANOVA model is a weighted sum of splines along each dimension, 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, only terms of first and second order are usually considered in practice.

This more general model class, in which each interaction term is estimated separately, is known in the physical sciences as high dimensional model representation (HDMR). Rabitz and Aliş (1999) review some properties and applications of this model class.

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

1.6 Regression and classification experiments

Choosing the base kernel

An additive GP using a separate SE kernel on each input dimension will have $3 \times D$ effective parameters. Because each additional parameter increases the tendency to overfit,

in our experiments we fixed each one-dimensional kernel's output variance to be 1, and only estimated the lengthscale of each kernel.

Methods

We compared six different methods. In the results tables below, GP Additive refers to a GP using the additive kernel with squared-exp base kernels. For speed, we limited the maximum order of interaction to 10. GP-1st denotes an additive GP model with only first-order interactions: a sum of one-dimensional kernels. GP Squared-exp is a GP using an SE-ARD kernel. HKL was run using the all-subsets kernel, which corresponds to the same set of interaction terms considered by GP Additive.

For all GP models, we fit kernel parameters 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 parameters, we fit a constant mean function to the data. In the classification experiments, approximate GP inference was performed using expectation propagation (Minka, 2001).

The regression experiments also compared against the structure search method from section 1.7, run up to depth 10, using only the SE and RQ base kernels.

1.6.1 Datasets

We compared the above methods on regression and classification datasets from the UCI repository (Bache and Lichman, 2013). Their size and dimension are given in tables 1.2 and 1.3:

Table 1.2: Regression dataset statistics

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

Bach synthetic dataset

In addition to standard UCI repository datasets, we generated a synthetic dataset using the same recipe as Bach (2009). This dataset was presumably designed to demonstrate

Table 1.3: Classification dataset statistics

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

the advantages of HKL over a GP using an SE-ARD kernel. It is generated by passing correlated Gaussian-distributed inputs $x_1, x_2, \ldots, x_8$ through a quadratic function

$$f(\mathbf{x}) = \sum_{i=1}^{4} \sum_{j=i+1}^{4} x_i x_j + \epsilon \qquad \epsilon \sim \mathcal{N}(0, \sigma_\epsilon)\,. \tag{1.23}$$

This dataset will presumably be well-modeled by an additive kernel which includes all two-way interactions over the first 4 variables, but does not depend on the extra 4 correlated nuisance inputs or the higher-order interactions.

1.6.2 Results

Tables 1.4 to 1.7 show mean performance across 10 train-test splits. Because HKL does not specify a noise model, it was not included in the likelihood comparisons.

On each dataset, the best performance is in boldface, along with all other performances not significantly different under a paired t-test. The additive and structure search methods usually outperformed the other methods, especially on regression problems.

The structure search outperforms the additive GP at the cost of a slower search over kernels. Structure search was on the order of 10 times slower than the additive GP, which was on the order of 10 times slower than GP-SE.

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 1.1). Because the additive GP is a superset of both the GP-1st model and the GP-SE model, instances where the additive GP performs slightly worse are presumably due to over-fitting, or due to the hyperparameter optimization becoming stuck in a local maximum. Performance of all GP models could be expected to benefit from approximately integrating over kernel parameters.

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

Table 1.4: Regression mean squared error

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

Table 1.5: Regression negative log-likelihood

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

Table 1.6: Classification percent error

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

Table 1.7: Classification negative log-likelihood

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

Source code

All of the experiments in this chapter were performed using the standard GPML toolbox, available at `http://wwww.gaussianprocess.org/gpml/code`. The additive kernel described in this chapter is included in GPML as of version 3.2. Code to perform all experiments in this chapter is available at `http://www.github.com/duvenaud/additive-gps`.

1.7 Conclusions

This chapter presented a tractable GP model consisting of a sum of exponentially-many functions, each depending on a different subset of the inputs. Our experiments indicate that, to varying degrees, such additive structure is useful for modeling real datasets. When it is present, modeling this structure allows our model to perform better than standard GP models. In the case where no such structure exists, the higher-order interaction terms present in the kernel can recover arbitrarily flexible models. The additive GP also affords some degree of interpretability: the variance parameters on each order of interaction indicate which sorts of structure are present the data, although they do not indicate which particular interactions explain the dataset.

The model class considered in this chapter is a subset of that explored by the structure search presented in section 1.7. Thus additive GPs can be considered a quick-and-dirty structure search, being strictly more limited in the types of structure that it can discover, but much faster and simpler to implement.

Closely related model classes have previously been explored, most notably smoothing-splines ANOVA and the support vector ANOVA decomposition. However, these models can be difficult to apply in practice because their kernel parameters, regularization penalties, and the relevant orders of interaction must be set by hand or by cross-validation. This chapter illustrates that the GP framework allows these model choices to be performed automatically.

References

Francis R. Bach. High-dimensional non-linear variable selection through hierarchical kernel learning. *arXiv preprint arXiv:0909.0844*, 2009. (pages 9, 12, and 13)

Kevin Bache and Moshe Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`. (page 12)

Pierre Baldi and Peter J. Sadowski. Understanding dropout. In *Advances in Neural Information Processing Systems*, pages 2814–2822, 2013. (page 7)

Richard Bellman. Dynamic programming and Lagrange multipliers. *Proceedings of the National Academy of Sciences of the United States of America*, 42(10):767, 1956. (page 6)

Yoshua Bengio, Olivier Delalleau, and Nicolas Le Roux. The curse of highly variable functions for local kernel machines. *Advances in Neural Information Processing Systems*, 18:107–114, 2006. ISSN 1049-5258. (page 6)

Nicolas Durrande, David Ginsbourger, and Olivier Roustant. Additive kernels for Gaussian process modeling. *arXiv preprint arXiv:1103.4023*, 2011. (page 9)

David Duvenaud, Hannes Nickisch, and Carl E. Rasmussen. Additive Gaussian processes. In *Advances in Neural Information Processing Systems 24*, pages 226–234, Granada, Spain, 2011. (page 1)

Elad Gilboa, Yunus Saatçi, and John Cunningham. Scaling multidimensional inference for structured Gaussian processes. In *Proceedings of the 30th International Conference on Machine Learning*, 2013. (page 9)

Trevor J. Hastie and Robert J. Tibshirani. *Generalized additive models*. Chapman & Hall/CRC, 1990. (page 2)

Geoffrey Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. (page 7)

Cari G. Kaufman and Stephan R. Sain. Bayesian functional ANOVA modeling using Gaussian process prior distributions. *Bayesian Analysis*, 5(1):123–150, 2010. (page 9)

Ian G. Macdonald. *Symmetric functions and Hall polynomials*. Oxford University Press, USA, 1998. ISBN 0198504500. (page 4)

Thomas P. Minka. Expectation propagation for approximate Bayesian inference. In *Uncertainty in Artificial Intelligence*, volume 17, pages 362–369, 2001. (page 12)

John Ashworth Nelder and Robert W.M. Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society. Series A (General)*, 135(3):370–384, 1972. (page 2)

Jorge Nocedal. Updating quasi-Newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980. (page 12)

Tony A. Plate. Accuracy versus interpretability in flexible modeling: Implementing a tradeoff using Gaussian process models. *Behaviormetrika*, 26:29–50, 1999. ISSN 0385-7417. (page 9)

Herschel Rabitz and Ömer F. Aliş. General foundations of high-dimensional model representations. *Journal of Mathematical Chemistry*, 25(2-3):197–233, 1999. (page 11)

Nitish Srivastava. Improving neural networks with dropout. Master's thesis, University of Toronto, 2013. (page 7)

Mark O. Stitson, Alex Gammerman, Vladimir Vapnik, Volodya Vovk, Chris Watkins, and Jason Weston. Support vector regression with ANOVA decomposition kernels. *Advances in kernel methods: Support vector learning*, pages 285–292, 1999. (page 11)

Vladimir N. Vapnik. *Statistical learning theory*, volume 2. Wiley New York, 1998. (page 11)

Grace Wahba. *Spline models for observational data*. Society for Industrial Mathematics, 1990. ISBN 0898712440. (page 11)

Sida Wang and Christopher Manning. Fast dropout training. In *Proceedings of the 30th International Conference on Machine Learning*, pages 118–126, 2013. (page 7)