Chapter 1

Higher-order Additive GPs

In **??**, we saw how additive structure in a GP prior enabled long-range extrapolation in multivariate problems. In general, models of the form

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

are widely used in machine learning and statistics, because they are relatively easy to fit and interpret. Examples include logistic regression, linear regression, Generalized Linear Models (Nelder and Wedderburn, 1972) and Generalized Additive Models (GAM) (Hastie and Tibshirani, 1990).

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

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

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 makes it difficult to generalize to new combinations of input variables.

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

$$g(y) = f(x_1, x_2) + f(x_2, x_3) + f(x_1, x_3) \tag{1.3}$$

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

possible products of one-dimensional kernels.

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

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

1.1 Additive Kernels

We now give a precise definition of additive kernels. We first assign each dimension $i \in \{1 \dots D\}$ a one-dimensional *base kernel* $k_i(x_i, x_i')$. We then define the first order, second order and nth order additive kernel as:

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

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

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

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

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

covariance function:

$$k_{add_D}(\mathbf{x}, \mathbf{x}') = \sigma_D^2 \prod_{d=1}^{D} k_d(x_d, x'_d) \tag{1.8}$$

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

$$k_{add_D}(\mathbf{x}, \mathbf{x}') = \sigma_D^2 \prod_{d=1}^{D} k_d(x_d, x'_d) = \sigma_D^2 \prod_{d=1}^{D} \exp\Big(-\frac{(x_d - x'_d)^2}{2l_d^2} \Big) = \sigma_D^2 \exp\Big(-\sum_{d=1}^{D} \frac{(x_d - x'_d)^2}{2l_d^2} \Big) \tag{1.9}$$

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

1.1.1 Parameterization

The only design choice necessary in specifying an additive kernel is the selection of a one-dimensional base kernel for each input dimension. Any parameters (such as length-scales) of the base kernels can be learned as usual by maximizing the marginal likelihood of the training data.

In addition to the parameters of each dimension-wise kernel, additive kernels are equipped with a set of D parameters $\sigma_1^2 \dots \sigma_D^2$ controlling how much variance we assign to each order of interaction. These "order variance" parameters have a useful interpretation: the dth order variance hyperparameter controls how much of the target function's variance comes from interactions of the dth order. Table 1.1 shows examples of the variance contributed by the different orders of interaction, learned on real datasets.

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

Because the variance parameters can specify which degrees of interaction are important, the additive GP can capture many different types of structure. If the function we are modeling is decomposable into a sum of low-dimensional functions, our model can discover this fact and exploit it. Discovering such structure allows long-range extrapolation, as we saw in **??**. If low-dimensional additive structure is not present, the kernel parameters can specify a suitably flexible model, with interactions between as many

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

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

variables as necessary.

1.1.2 Efficient Evaluation of Additive Kernels

An additive kernel over D inputs with interactions up to order n has $O(2^n)$ terms. Naïvely summing over these terms quickly becomes intractable. In this section, we show how one can evaluate the sum over all terms in $O(D^2)$.

To efficiently compute the additive kernel, we exploit the fact that the nth order additive kernel corresponds to the nth *elementary symmetric polynomial* (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

$$k_{\text{add}_0}(\mathbf{x}, \mathbf{x}') = e_0(k_1, k_2, k_3, k_4) = 1 \tag{1.10}$$

$$k_{\text{add}_1}(\mathbf{x}, \mathbf{x}') = e_1(k_1, k_2, k_3, k_4) = k_1 + k_2 + k_3 + k_4 \tag{1.11}$$

$$k_{\text{add}_2}(\mathbf{x}, \mathbf{x}') = e_2(k_1, k_2, k_3, k_4) = k_1k_2 + k_1k_3 + k_1k_4 + k_2k_3 + k_2k_4 + k_3k_4 \tag{1.12}$$

$$k_{\text{add}_3}(\mathbf{x}, \mathbf{x}') = e_3(k_1, k_2, k_3, k_4) = k_1k_2k_3 + k_1k_2k_4 + k_1k_3k_4 + k_2k_3k_4 \tag{1.13}$$

$$k_{\text{add}_4}(\mathbf{x}, \mathbf{x}') = e_4(k_1, k_2, k_3, k_4) = k_1k_2k_3k_4 \tag{1.14}$$

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

nomials:

$$k_{\text{add}_n}(\mathbf{x}, \mathbf{x}') = e_n(k_1, \ldots, k_D) = \frac{1}{n}\sum_{a=1}^{n}(-1)^{(a-1)}e_{n-a}(k_1, \ldots, k_D)\sum_{i=1}^{D}k_i^a \tag{1.15}$$

The Newton-Girard formulae have time complexity $\mathcal{O}(D^2)$, while computing a sum over an exponential number of terms. Note that the same sum can be computed in time

Conveniently, we can use the same trick to efficiently compute all of the necessary derivatives of the additive kernel with respect to the base kernels. We merely need to remove the kernel of interest from each term of the polynomials:

$$\frac{\partial k_{add_n}}{\partial k_j} = e_{n-1}(k_1, \ldots, k_{j-1}, k_{j+1}, \ldots k_D) \tag{1.16}$$

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

1.1.3 Computational Cost

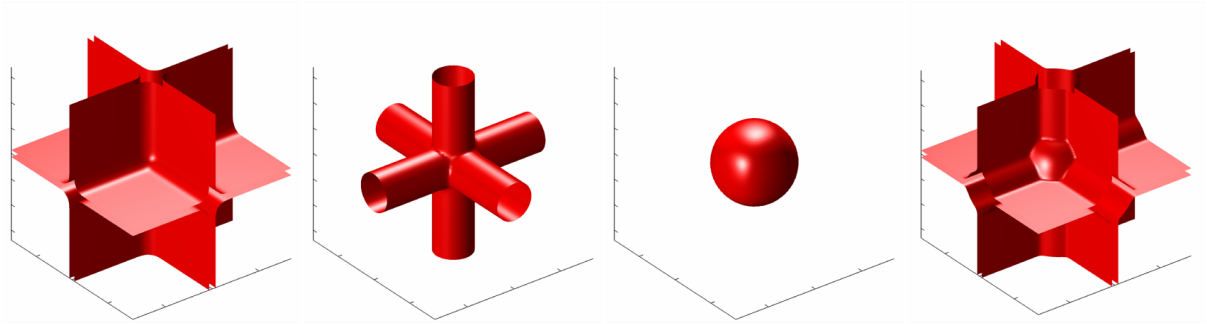
The computational cost of evaluating the Gram matrix of a product kernel (such as the SE kernel) is $\mathcal{O}(N^2D)$, while the cost of evaluating the Gram matrix of the additive kernel is $\mathcal{O}(N^2DR)$, where R is the maximum degree of interaction allowed (up to D). In higher dimensions, this can be a significant cost, even relative to the fixed $\mathcal{O}(N^3)$ cost of inverting the Gram matrix. However, as our experiments show, typically only the first few orders of interaction are important for modeling a given function; hence if one is computationally limited, one can simply limit the maximum degree of interaction without losing much accuracy.

1.2 Non-local Interactions

By far the most popular kernels for regression and classification tasks on continuous data are the SE kernel, and the Matérn kernels. These kernels depend only on the scaled Euclidean distance between two points, both having the form:

$$k(\mathbf{x}, \mathbf{x}') = g\left(\sum_{d=1}^{D}\left(\frac{x_d - x'_d}{l_d}\right)^2\right) \tag{1.17}$$

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



1st-order terms $k_1 + k_2 + k_3$ | 2nd-order terms $k_1k_2 + k_2k_3 + k_1k_3$ | 3rd-order terms $k_1k_2k_3$ Squared-exp kernel | All interactions Additive kernel

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

Additive kernels have a much more complex structure, and allow extrapolation based on distant parts of the input space, without spreading the mass of the kernel over the whole space. For example, additive kernels of the second order allow strong non-local interactions between any points which are similar in any two input dimensions. Figure 1.1 provides a geometric comparison between squared-exponential kernels and additive kernels in 3 dimensions.

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

1.3 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.

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

$$f(\mathbf{x}) = \frac{1}{K}\boldsymbol{\alpha}^\mathsf{T}\mathbf{h}(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K}\alpha_i h_i(\mathbf{x}) \tag{1.18}$$

$$\implies f(\mathbf{x}) \overset{K\to\infty}{\sim} \mathcal{GP}\Big(\mathbb{E}\left[\boldsymbol{\alpha}^\mathsf{T}\mathbf{h}(\mathbf{x})\right], \sigma^2_{\boldsymbol{\alpha}}\mathbf{h}(\mathbf{x})^\mathsf{T}\mathbf{h}(\mathbf{x}')\Big) \tag{1.19}$$

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

1.3.1 Dropout on Hidden Units

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

$$r_i \overset{\text{iid}}{\sim} \text{Ber}(p) \qquad \mathbb{V}[r_i\alpha_i] = p\sigma^2_\alpha \,. \tag{1.20}$$

Because equation (1.19) is a result of the central limit theorem, it does not depend on the form of the distribution on $\boldsymbol{\alpha}$, only its mean and variance. Thus, dropping out features of an infinitely-wide MLP does not change the model at all, except to rescale the output variance, since no individual feature can have more than infinitesimal contribution to the network output. Indeed, multiplying all weights by $p^{-1/2}$, the initial variance is restored:

$$\mathbb{V}\left[\frac{1}{p^{\frac{1}{2}}}r_i\alpha_i\right] = \frac{p}{p}\sigma^2_\alpha = \sigma^2_\alpha \,. \tag{1.21}$$

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

1.3.2 Dropout on Inputs

In a GP with kernel $k(\mathbf{x}, \mathbf{x}') = \prod_{d=1}^{D} k_d(x_d, x'_d)$, exact averaging over all possible ways of dropping out inputs with probability $1/2$ results in a mixture of GPs, each depending on only a subset of the inputs:

$$p\left(f(\mathbf{x})\right) = \frac{1}{2^D} \sum_{\mathbf{r} \in \{0,1\}^D} \mathrm{GP}\left(f(\mathbf{x}) \,|\, 0, \prod_{d=1}^{D} k_d(x_d, x'_d)^{r_d}\right) \tag{1.22}$$

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

First, if the kernel on each dimension has the form $k_d(x_d, x'_d) = g\left(\frac{x_d - x'_d}{w_d}\right)$, as does the SE kernel, then any input dimension can be dropped out by setting its lengthscale w_d to ∞. Thus, dropout on the inputs of a GP corresponds to a spike-and-slab prior on lengthscales, with each dimension independently having $w_d = \infty$ with probability $1/2$.

Another way to understand the resulting prior is to note that the dropout mixture (1.22) has the same covariance as

$$f(\mathbf{x}) \sim \mathrm{GP}\left(0, \frac{1}{2^{-2D}} \sum_{\mathbf{r} \in \{0,1\}^D} \prod_{d=1}^{D} k_d(x_d, x'_d)^{r_d}\right) \tag{1.23}$$

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

1.4 Related Work

Since additive models are a relatively natural and easy-to-analyze model class, the literature on similar model classes is extensive. We try to give a broad overview in this section.

Additive GPs

Since the non-local structure capturable by additive kernels is necessarily axis-aligned, we can naturally consider that an initial transformation of the input space might allow us to recover non-axis aligned additivity in functions. This avenue was explored by Gilboa

et al. (2013), who developed a linearly-transformed first-order additive GP model, called projection-pursuit GP regression. They further showed that inference in this model was possible in $\mathcal{O}(N)$ time.

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

Plate (1999) constructed an additive GP using only the first-order and Dth-order terms. This model is motivated by the desire to trade off the interpretability of first-order models with the flexibility of full-order models. Our experiments show that often, the intermediate degrees of interaction contribute most of the variance.

A related functional ANOVA GP model (Kaufman 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. (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.

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 learn a weighted sum over an exponential number of kernels which can be computed in polynomial time. This method chooses among *hull* of kernels, defined as a subset of all terms such that if $\prod_{j\in J} k_j(\mathbf{x},\mathbf{x}')$ is included in the subset, then so are all terms $\prod_{j\in J/i} k_j(\mathbf{x},\mathbf{x}')$, for all $i \in J$. HKL computes the sum over all orders in $\mathcal{O}(D)$ time by the formula:

$$k_a(\mathbf{x},\mathbf{x}') = v^2 \prod_{d=1}^{D} (1 + \alpha k_d(x_d, x'_d)) \tag{1.24}$$

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

Figure 1.2 contrasts the HKL model class with the additive GP model. Neither method is strictly more flexibile than the other. The main difficulty with the approach of Bach (2009) is that the kernel parameters are hard to set, other than by cross-validation.

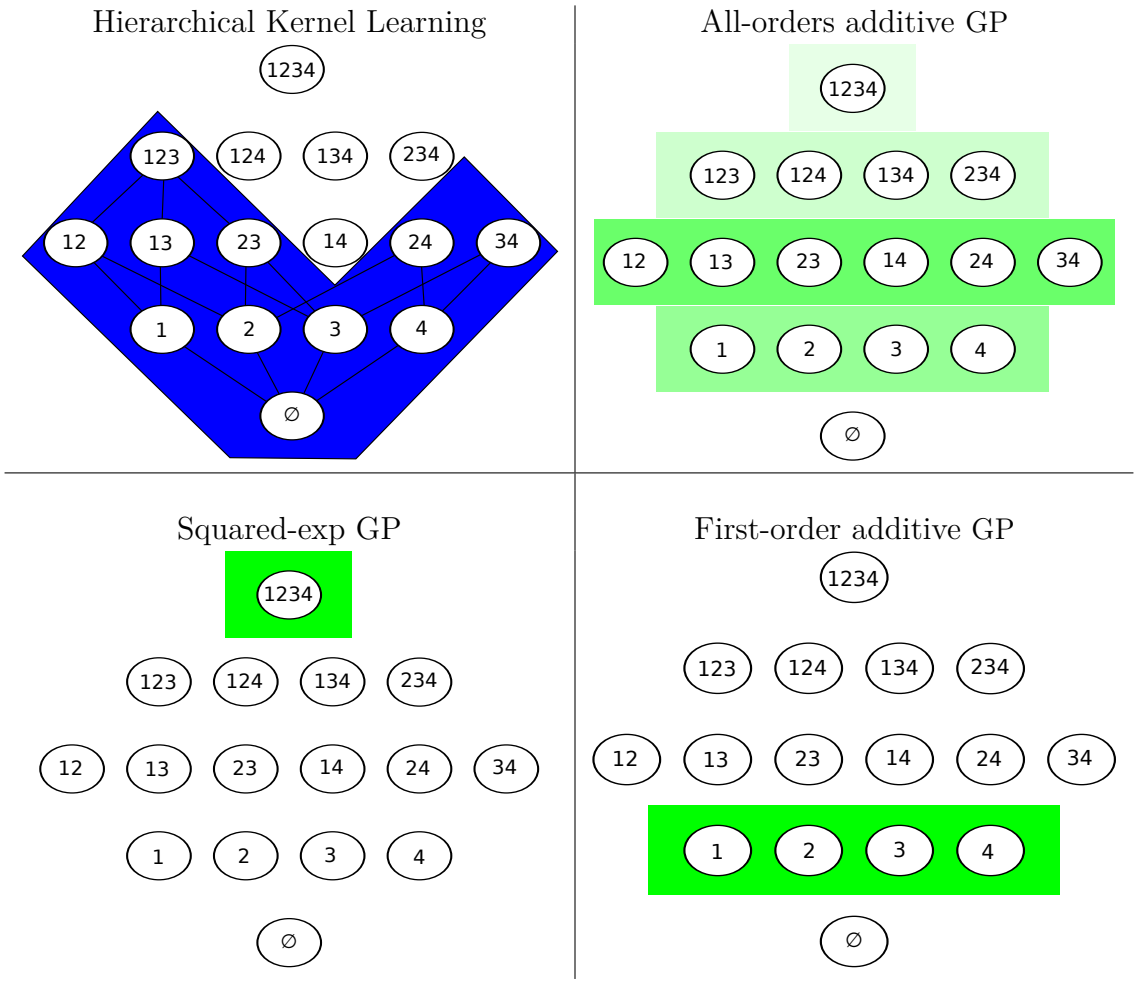


Fig. 1.2 A comparison of different additive model classes. Nodes represent different interaction terms, ranging from first-order to fourth-order interactions. Coloured boxes represent the weightings of different terms. *Top left:* HKL can select a hull of interaction terms, but must use a pre-determined weighting over those terms. *Top right:* the additive GP model can weight each order of interaction separately. *Bottom row:* Neither HKL nor the additive model dominate one another in terms of flexibility, however the SE-GP and first-order additive GP models are special cases of the all-orders additive GP.

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 D orders with only one term "of appropriate order", presumably because of the difficulty of setting the parameters of an SVM. 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, plus a sum of splines over all pairs of dimensions, all triplets, etc, with each individual interaction term having a separate weighting parameter. Because the number of terms to consider grows exponentially in the order, in practice, only terms of first and second order are usually considered.

This more general model class, in which each interaction term is estimated separately, is known in the physical sciences as High Dimensional Model Representation (HDMR). 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.

1.5 Experiments

Parameterization

A D-dimensional SE-ARD kernel has D lengthscale parameters and the output variance. An first-order additive SE model has $2 \times D$ parameters. A fully-parametrized model including all orders of interaction, with a seperate output variance for each scale will have $3 \times D$ parameters. Because each additional parameter increases the tendency to overfit, we recommend allowing only one kernel parameter per input dimension. In our experiments, we parametrized each one-dimensional kernel with only the lengthscale, fixing the output variance to be 1.

Methods

We compare six different methods. In the results tables below, GP Additive refers to a GP using the additive kernel with squared-exp base kernels. For speed, we limited the maximum order of interaction in the additive kernels to 10. GP-1st denotes an additive GP model with only first-order interactions - a sum of one-dimensional kernels. GP Squared-Exp is a GP model with a squared-exponential ARD kernel. HKL was run

using the all-subsets kernel, which corresponds to the same set of kernels as considered by the additive GP with a squared-exp base kernel.

For all GP models, we fit kernel parameters by the standard method of maximizing training-set marginal likelihood, using L-BFGS (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 done using Expectation Propagation (Minka, 2001).

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

1.5.1 Datasets

We compared these methods on a diverse set of 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

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

Bach Synthetic Dataset

In addition to standard UCI repository datasets, we generated a synthetic dataset following the same recipe as Bach (2009). This dataset was designed to demonstrate the advantages of HKL over GP-SE. It is generated by passing correlated Gaussian-distributed

inputs $x_1, x_2, \ldots, x_8$ through the quadratic function

$$f(\mathbf{x}) = \sum_{i=1}^{4} \sum_{j=1+1}^{4} x_i x_j + \epsilon \qquad \epsilon \sim \mathcal{N}(0, \sigma_\epsilon) \tag{1.25}$$

This dataset is well-modeled by an additive kernel which includes all two-way interactions over the first 4 variables, but does not depend on the extra 4 irrelevant inputs.

If the dataset is large enough, HKL can construct a hull around only the 16 cross-terms optimal for predicting the output. GP-SE, in contrast, can learn to ignore the noisy copy variables, but cannot learn to ignore the higher-order interactions between dimensions. However, a GP with an additive kernel can learn both to ignore irrelevant variables, and to ignore missing orders of interaction. With enough data, the marginal likelihood will favour hyperparameters specifying such a model.

1.5.2 Results

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

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**

The model with best performance on each dataset is in bold, along with all other models that were not significantly different under a paired t-test. The additive and structure search methods usually outperformed the other methods, especially on regression problems. The structure search outperforms the additive GP, but at the cost of a slow search over kernels.

The additive GP performed best on datasets well-explained by low orders of interaction, and approximately as well as the SE-GP model on datasets which were well explained by high orders of interaction (see table 1.1). Because the additive GP is a su-

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**

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

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

1.5.3 Source Code

Additive Gaussian processes are particularly appealing in practice because their use requires only the specification of the base kernel; all other aspects of GP inference remain the same. Note that we are also free to choose a different covariance function along each dimension.

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

1.6 Conclusion

In this chapter, we presented a parameterization of higher-order additive GPs allowing for tractable inference. Our experiments indicate that, to varying degrees, additive structure is present in real datasets. When it is present, modeling this structure allows our model to perform better than standard GP models. In the case where no such structure exists, the higher-order interaction terms present in the kernel can recover arbitrarily flexible models, as well.

The additive GP also affords extra interpretability: the variance parameters on each order of interaction indicate which sorts of structure are present the data.

References

Francis Bach. High-dimensional non-linear variable selection through hierarchical kernel learning. *CoRR*, abs/0909.0844, 2009. (pages 9, 12, and 14)

K. Bache and M. 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)

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)

M. Christoudias, R. Urtasun, and T. Darrell. Bayesian localized multiple kernel learning. *Technical report, EECS Department, University of California, Berkeley*, 2009. (page 9)

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

Elad Gilboa, Yunus Saatçi, and J Cunningham. Scaling multidimensional inference for structured Gaussian processes. In *Proceedings of the 30th International Conference on Machine Learning*, 2013. (page 8)

T.J. Hastie and R.J. Tibshirani. *Generalized additive models*. Chapman & Hall/CRC, 1990. (page 1)

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 6)

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

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

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

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 1)

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

T.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*. PhD thesis, University of Toronto, 2013. (page 6)

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 10)

V.N. Vapnik. *Statistical learning theory*, volume 2. Wiley New York, 1998. (page 10)

G. 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 (ICML-13)*, pages 118–126, 2013. (page 7)