Regression with Linear Factored Functions

Wendelin Böhmer(✉) and Klaus Obermayer

Neural Information Processing Group, Technische Universität Berlin,
Sekr. MAR5-6, Marchstr. 23, 10587 Berlin, Germany
{wendelin,oby}@ni.tu-berlin.de
http://www.ni.tu-berlin.de

Abstract. Many applications that use empirically estimated functions face a *curse of dimensionality*, because integrals over most function classes must be approximated by sampling. This paper introduces a novel *regression*-algorithm that learns *linear factored functions* (LFF). This class of functions has structural properties that allow to analytically solve certain integrals and to calculate point-wise products. Applications like *belief propagation* and *reinforcement learning* can exploit these properties to break the curse and speed up computation. We derive a regularized greedy optimization scheme, that learns factored basis functions during training. The novel regression algorithm performs competitively to *Gaussian processes* on benchmark tasks, and the learned LFF functions are with 4-9 factored basis functions on average very compact.

Keywords: Regression · Factored functions · Curse of dimensionality

1 Introduction

This paper introduces a novel regression-algorithm, which performs competitive to *Gaussian processes*, but yields *linear factored functions* (LFF). These have outstanding properties like analytical *point-wise products* and *marginalization*.

Regression is a well known problem, which can be solved by many non-linear architectures like *kernel methods* (Shawe-Taylor and Cristianini 2004) or *neural networks* (Haykin 1998). While these perform well, the estimated functions often suffer a *curse of dimensionality* in later applications. For example, computing an integral over a neural network or kernel function requires to sample the entire input space. Applications like *belief propagation* (Pearl 1988) and *reinforcement learning* (Kaelbling et al. 1996), on the other hand, face large input spaces and require therefore efficient computations. We propose LFF for this purpose and showcase its properties in comparison to kernel functions.

1.1 Kernel Regression

In the last 20 years, kernel methods like *support vector machines* (SVM, Boser et al. 1992; Vapnik 1995) have become a de facto standard in various practical applications. This is mainly due to a sparse representation of the learned classifiers with so called *support vectors* (SV). The most popular kernel method for

A. Appice et al. (Eds.): ECML PKDD 2015, Part I, LNAI 9284, pp. 119–134, 2015.
DOI: 10.1007/978-3-319-23528-8_8

regression, *Gaussian processes* (GP, see Bishop 2006; Rasmussen and Williams 2006), on the other hand, requires as many SV as training samples. Sparse versions of GP aim thus for a small subset of SV. Some select this set based on constraints similar to SVM (Tipping 2001; Vapnik 1995), while others try to conserve the spanned linear function space (*sparse GP*, Csató and Opper 2002; Rasmussen and Williams 2006). There exist also attempts to construct new SV by averaging similar training samples (e.g. Wang et al. 2012).

Well chosen SV for regression are usually not sparsely concentrated on a decision boundary as they are for SVM. In fact, many practical applications report that they are distributed uniformly in the input space (e.g. in Böhmer et al. 2013). Regression tasks restricted to a small region of the input space may tolerate this, but some applications require predictions everywhere. For example, the *value function* in reinforcement learning must be generalized to each state. The number of SV required to *represent* this function equally well in each state grows exponentially in the number of input-space dimensions, leading to Bellman's famous curse of dimensionality (Bellman 1957).

Kernel methods derive their effectiveness from linear optimization in a non-linear *Hilbert space* of functions. Kernel-functions parameterized by SV are the non-linear *basis functions* in this space. Due to the functional form of the kernel, this can be a very ineffective way to select basis functions. Even in relatively small input spaces, it often takes hundreds or thousands SV to approximate a function sufficiently. To alleviate the problem, one can construct complex kernels out of simple prototypes (see a recent review in Gönen and Alpaydın 2011).

1.2 Factored Basis Functions

Diverging from all above arguments, this article proposes a more radical approach: to construct the non-linear basis functions directly during training, without the detour over kernel functions and support vectors. This poses two main challenges: to select a *suitable functions space* and to *regularize the optimization* properly. The former is critical, as a small set of basis functions must be able to approximate any target function, but should also be easy to compute in practice.

We propose *factored functions* $\psi_i = \prod_k \psi_i^k \in \mathcal{F}$ as basis functions for regression, and call the linear combination of m of those bases a *linear factored function* $f \in \mathcal{F}^m$ (LFF, Section 3). For example, *generalized linear models* (Nelder and Wedderburn 1972) and *multivariate adaptive regression splines* (MARS, Friedman 1991) are both LFF. Computation remains feasible by using *hinge functions* $\psi_i^k(x_k) = \max(0, x_k - c)$ and restricting the scope of each factored function ψ_i. In contrast, we assume the general case without restrictions to functions or scope.

Due to their structure, LFF can solve certain integrals analytically and allow very efficient computation of point-wise products and marginalization. We show that our LFF are universal function approximators and derive an appropriate *regularization* term. This regularization promotes smoothness, but also retains a high degree of variability in densely sampled regions by linking smoothness to uncertainty about the sampling distribution. Finally, we derive a novel regression algorithm for LFF based on a greedy optimization scheme.

Functions learned by this algorithm (Algorithm 1, see pages 125 and 133) are very compact (between 3 and 12 bases on standard benchmarks) and perform competitive with Gaussian processes (Section 4). The paper finishes with a discussion of the computational possibilities of LFF in potential areas of application and possible extensions to *sparse regression* with LFF (Section 5).

2 Regression

Let $\{\boldsymbol{x}_t \in \mathcal{X}\}_{t=1}^n$ be a set of n *input samples*, i.i.d. drawn from an input set $\mathcal{X} \subset \mathrm{I\!R}^d$. Each so called "training sample" is *labeled* with a real number $\{y_t \in \mathrm{I\!R}\}_{t=1}^n$. *Regression* aims to find a function $f : \mathcal{X} \to \mathrm{I\!R}$, that predicts the labels to all (previously unseen) test samples as well as possible. Labels may be afflicted by *noise* and f must thus approximate the mean label of each sample, i.e., the function $\mu : \mathcal{X} \to \mathrm{I\!R}$. It is important to notice that *conceptually* the noise is introduced by two (non observable) sources: noisy labels y_t and noisy samples $\boldsymbol{x}_t$. The latter will play an important role for regularization. We define the conditional distribution χ of observable samples $\boldsymbol{x} \in \mathcal{X}$ given the non-observable "true" samples $\boldsymbol{z} \in \mathcal{X}$, which are drawn by a distribution ξ. In the limit of infinite samples, the *least squares* cost-function $\mathcal{C}[f|\chi, \mu]$ can thus be written as

$$\lim_{n\to\infty} \inf_f \frac{1}{n} \sum_{t=1}^n \Big(f(\boldsymbol{x}_t) - y_t\Big)^2 \quad = \quad \inf_f \iint \xi(d\boldsymbol{z})\, \chi(d\boldsymbol{x}|\boldsymbol{z}) \Big(f(\boldsymbol{x}) - \mu(\boldsymbol{z})\Big)^2 . \quad (1)$$

The cost function $\mathcal{C}$ can never be computed *exactly*, but *approximated* using the training samples[1] and assumptions about the unknown noise distribution χ.

3 Linear Factored Functions

Any non-linear function can be expressed as a linear function $f(\boldsymbol{x}) = \boldsymbol{a}^\top \boldsymbol{\psi}(\boldsymbol{x})$, $\forall \boldsymbol{x} \in \mathcal{X}$, with m non-linear basis functions $\psi_i : \mathcal{X} \to \mathrm{I\!R}$, $\forall i \in \{1 \ldots, m\}$. In this section we will define *linear factored functions* (LFF), that have *factored basis functions* $\psi_i(\boldsymbol{x}) := \psi_i^1(x_1) \cdot \ldots \cdot \psi_i^d(x_d) \in \mathcal{F}$, a regularization method for this function class and an algorithm for regression with LFF.

3.1 Function Class

We define the class of linear factored functions $f \in \mathcal{F}^m$ as a linear combination (with linear parameters $\boldsymbol{a} \in \mathrm{I\!R}^m$) of m factored basis functions $\psi_i : \mathcal{X} \to \mathrm{I\!R}$

[1] The distribution ξ of "true" samples $\boldsymbol{z}$ can *not* be observed. We approximate in the following ξ with the training-sample distribution. This may be justified if the sample-noise χ is comparatively small. Although not strictly rigorous, the presented formalism helps to put the regularization derived in Proposition 2 into perspective.

(with parameters $\{\mathbf{B}^k \in \mathbb{R}^{m_k \times m}\}_{k=1}^d$):

$$f(\boldsymbol{x}) \;:=\; \boldsymbol{a}^\top \boldsymbol{\psi}(\boldsymbol{x}) \;:=\; \boldsymbol{a}^\top \Big[\prod_{k=1}^d \boldsymbol{\psi}^k(x_k)\Big] \;:=\; \sum_{i=1}^m a_i \prod_{k=1}^d \sum_{j=1}^{m_k} B_{ji}^k \, \phi_j^k(x_k)\,. \qquad (2)$$

LFF are formally defined in Appendix A. In short, a basis function ψ_i is the *point-wise product* of one-dimensional functions ψ_i^k in each input dimension k. These are themselves constructed as linear functions of a corresponding one-dimensional base $\{\phi_j^k\}_{j=1}^{m_k}$ over that dimension and ideally can approximate arbitrary functions[2]. Although each factored function ψ_i is very restricted, a linear combination of them can be very powerful:

Corollary 1. *Let $\mathcal{X}_k$ be a bounded continuous set and ϕ_j^k the j'th Fourier base over $\mathcal{X}_k$. In the limit of $m_k \to \infty, \forall k \in \{1, \ldots, d\}$, holds $\mathcal{F}^\infty = L^2(\mathcal{X}, \vartheta)$.*

Strictly this holds in the limit of infinitely many basis functions ψ_i, but we will show empirically that there exist close approximations with a small number m of factored functions. One can make similar statements for other bases $\{\phi_j^k\}_{j=1}^\infty$. For example, for Gaussian kernels one can show that the space $\mathcal{F}^\infty$ is in the limit equivalent to the corresponding *reproducing kernel Hilbert space $\mathcal{H}$*.

LFF offer some structural advantages over other universal function approximation classes like neural networks or reproducing kernel Hilbert spaces. Firstly, the *inner product* of two LFF in $L^2(\mathcal{X}, \vartheta)$ can be computed as products of one-dimensional integrals. For some bases[3], these integrals can be calculated analytically without any sampling. This could in principle break the curse of dimensionality for algorithms that have to approximate these inner products numerically. For example, input variables can be *marginalized* (integrated) out analytically (Equation 9 on Page 130). Secondly, the *point-wise product* of two LFF is a LFF as well[4] (Equation 10 on Page 131). See Appendix A for details. These properties are very useful, for example in *belief propagation* (Pearl 1988) and *factored reinforcement learning* (Böhmer and Obermayer 2013).

3.2 Constraints

LFF have some degrees of freedom that can impede optimization. For example, the norm of $\psi_i \in \mathcal{F}$ does not influence function $f \in \mathcal{F}^m$, as the corresponding linear coefficients a_i can be scaled accordingly. We can therefore introduce the *constraints* $\|\psi_i\|_\vartheta = 1, \forall i$, without restriction to the function class. The factorization of inner products (see Appendix A on Page 130) allows us furthermore to rewrite the constraints as $\|\psi_i\|_\vartheta = \prod_k \|\psi_i^k\|_{\vartheta^k} = 1$. This holds as long as the product is one, which exposes another unnecessary degree of freedom. To

[2] Examples are Fourier bases, Gaussian kernels or hinge-functions as in MARS.

[3] E.g. Fourier bases for continuous, and Kronecker-delta bases for discrete variables.

[4] One can use the trigonometric product-to-sum identities for Fourier bases or the Kronecker delta for discrete bases to construct LFF from a point-wise product without changing the underlying basis $\{\{\phi_i^k\}_{i=1}^{m_k}\}_{k=1}^d$.

finally make the solution unique (up to permutation), we define the constraints as $\|\psi_i^k\|_{\vartheta^k} = 1, \forall k, \forall i$. Minimizing some $\mathcal{C}[f]$ w.r.t. $f \in \mathcal{F}^m$ is thus equivalent to

$$\inf_{f \in \mathcal{F}^m} \mathcal{C}[f] \qquad \text{s.t.} \quad \|\psi_i^k\|_{\vartheta^k} \;=\; 1\,, \quad \forall k \in \{1, \ldots, d\}\,, \quad \forall i \in \{1, \ldots, m\}\,. \tag{3}$$

The *cost function* $\mathcal{C}[f|\chi, \mu]$ of Equation 1 with the constraints in Equation 3 is equivalent to *ordinary least squares* (OLS) w.r.t. linear parameters $\boldsymbol{a} \in \mathrm{I\!R}^m$. However, the optimization problem is *not* convex w.r.t. the parameter space $\{\mathbf{B}^k \in \mathrm{I\!R}^{m_k \times m}\}_{k=1}^d$, due to the nonlinearity of products.

Instead of tackling the global optimization problem induced by Equation 3, we propose a *greedy* approximation algorithm. Here we optimize at iteration $\hat{\imath}$ one linear basis function $\psi_{\hat{\imath}} =: g =: \prod_k g^k \in \mathcal{F}$, with $g^k(x_k) =: \boldsymbol{b}^{k\top} \boldsymbol{\phi}^k(x_k)$, at a time, to fit the residual $\mu - f$ between the true *mean label* function $\mu \in L^2(\mathcal{X}, \vartheta)$ and the current regression estimate $f \in \mathcal{F}^{\hat{\imath}-1}$, based on all $\hat{\imath} - 1$ previously constructed factored basis functions $\{\psi_i\}_{i=1}^{\hat{\imath}-1}$:

$$\inf_{g \in \mathcal{F}} \mathcal{C}[f + g|\chi, \mu] \qquad \text{s.t.} \quad \|g^k\|_{\vartheta^k} = 1\,, \quad \forall k \in \{1, \ldots, d\}\,. \tag{4}$$

3.3 Regularization

Regression with any powerful function class requires regularization to avoid overfitting. Examples are *weight decay* for neural networks (Haykin 1998) or parameterized *priors* for Gaussian processes. It is, however, not immediately obvious how to regularize the parameters of a LFF and we will derive a regularization term from a Taylor approximation of the cost function in Equation 1.

We aim to enforce smooth functions, especially in those regions our knowledge is limited due to a lack of training samples. This *uncertainty* can be expressed as the *Radon-Nikodym derivative*[5] $\frac{\vartheta}{\xi} : \mathcal{X} \to [0, \infty)$ of our factored measure ϑ (see Appendix A) w.r.t. the sampling distribution ξ. Figure 1 demonstrates at the example of a uniform distribution ϑ how $\frac{\vartheta}{\xi}$ reflects our empirical knowledge of the input space $\mathcal{X}$.

We use this uncertainty to modulate the *sample noise distribution* χ in Equation 1. This means that frequently sampled regions of $\mathcal{X}$ shall yield low, while scarcely sampled regions shall yield high variance. Formally, we assume $\chi(d\boldsymbol{x}|\boldsymbol{z})$ to be a Gaussian probability measure over $\mathcal{X}$ with mean $\boldsymbol{z}$ and a *covariance matrix* $\boldsymbol{\Sigma} \in \mathrm{I\!R}^{d \times d}$, scaled by the local uncertainty in $\boldsymbol{z}$ (modeled as $\frac{\vartheta}{\xi}(\boldsymbol{z})$):

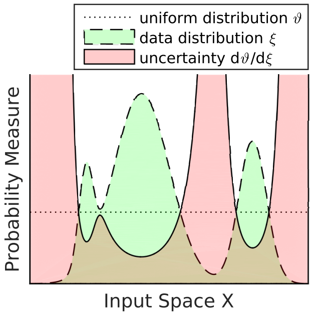

Fig. 1. We interpret the Radon-Nikodym derivative $\frac{d\vartheta}{d\xi}$ as *uncertainty measure* for our knowledge of $\mathcal{X}$. Regularization enforces smoothness in uncertain regions.

[5] Technically we have to assume that ϑ is *absolutely continuous* in respect to ξ. For "well-behaving" distributions ϑ, like the uniform or Gaussian distributions we discuss in Appendix A, this is equivalent to the assumption that in the limit of infinite samples, each sample $\boldsymbol{z} \in \mathcal{X}$ will *eventually* be drawn by ξ.

$$\int \chi(d\boldsymbol{x}|\boldsymbol{z})(\boldsymbol{x}-\boldsymbol{z}) = \mathbf{0}\,, \qquad \int \chi(d\boldsymbol{x}|\boldsymbol{z})(\boldsymbol{x}-\boldsymbol{z})(\boldsymbol{x}-\boldsymbol{z})^\top = \tfrac{\vartheta}{\xi}(\boldsymbol{z}) \cdot \boldsymbol{\Sigma}\,, \qquad \forall \boldsymbol{z} \in \mathcal{X}\,. \tag{5}$$

In the following we assume without loss of generality[6] the matrix $\boldsymbol{\Sigma}$ to be diagonal, with the diagonal elements called $\sigma_k^2 := \Sigma_{kk}$.

Proposition 2. *Under the assumptions of Equation 5 and a diagonal covariance matrix $\boldsymbol{\Sigma}$, the first order Taylor approximation of the cost $\mathcal{C}$ in Equation 4 is*

$$\tilde{\mathcal{C}}[g] \quad := \quad \underbrace{\|g - (\mu - f)\|_\xi^2}_{\text{sample-noise free cost}} \; + \; \sum_{k=1}^{d} \sigma_k^2 \underbrace{\|\tfrac{\partial}{\partial x_k} g + \tfrac{\partial}{\partial x_k} f\|_\vartheta^2}_{\text{smoothness in dimension } k} \,. \tag{6}$$

Proof: see Appendix C on Page 132. □

Note that the approximated cost $\tilde{\mathcal{C}}[g]$ consists of the sample-noise free cost (measured w.r.t. training distribution ξ) and d regularization terms. Each term prefers functions that are smooth[7] in one input dimension. This enforces smoothness everywhere, but allows exceptions where enough data is available. To avoid a cluttered notation, in the following we will use the symbol $\nabla_k f := \frac{\partial}{\partial x_k} f$.

3.4 Optimization

Another advantage of cost function $\tilde{\mathcal{C}}[g]$ is that one can optimize one factor function g^k of $g(\boldsymbol{x}) = g^1(x_1) \cdot \ldots \cdot g^d(x_d) \in \mathcal{F}$ at a time, instead of time consuming *gradient descend* over the entire parameter space of g. To be more precise:

Proposition 3. *If all but one factor function g^k are considered constant, Equation 6 has an analytical solution. If $\{\phi_j^k\}_{j=1}^{m_k}$ is a Fourier base, $\sigma_k^2 > 0$ and $\vartheta \ll \xi$, then the solution is also unique.*

Proof: see Appendix C on Page 133. □

One can give similar guarantees for other bases, e.g. Gaussian kernels. Note that Proposition 3 does *not* state that the optimization problem has a unique solution in $\mathcal{F}$. Formal convergence statements are not trivial and empirically the parameters of g do not converge, but evolve around orbits of equal cost instead. However, since the optimization of *any* g^k cannot increase the cost, any sequence of improvements will converge to (and stay in) a *local minimum*. This implies a *nested* optimization approach, that is formulated in Algorithm 1 on Page 133:

- An *inner loop* that optimizes one factored basis function $g(\boldsymbol{x}) = g^1(x_1) \cdot \ldots \cdot g^d(x_d)$ by selecting an input dimension k in each iteration and solve Equation 6 for the corresponding g^k. A detailed derivation of the optimization steps of

[6] Non-diagonal covariance matrices $\boldsymbol{\Sigma}$ can be cast in this framework by projecting the input samples into the eigenspace of $\boldsymbol{\Sigma}$ (thus diagonalizing the input) and use the corresponding eigenvalues λ_k instead of the regularization parameters σ_k^2's.

[7] Each regularization term is measured w.r.t. the factored distribution ϑ. We also tested the algorithm without consideration of "uncertainty" $\frac{\vartheta}{\xi}$, i.e., by measuring each term w.r.t. ξ. As a result, regions outside the hypercube containing the training set were no longer regularized and predicted arbitrary (often extreme) values.

Algorithm 1. (abstract) – a detailed version can be found on Page 133

```
while new factored basis function can improve solution do
  initialize new basis function $g$ as constant function
  while optimization improves cost in Equation 6 do
    for random input dimension $k$ do
      calculate optimal solution for $g^k$ without changing $g^l, \forall l \neq k$
    end for
  end while  // new basis function $g$ has converged
  add $g$ to set of factored basis functions and solve OLS
end while  // regression has converged
```

the inner loop is given in Appendix B on Page 131. The choice of k influences the solution in a non-trivial way and further research is needed to build up a rationale for any meaningful decision. For the purpose of this paper, we assume k to be chosen randomly by permuting the order of updates.

The *computational complexity* of the inner loop is $\mathcal{O}(m_k^2 n + d^2 m_k m)$. Memory complexity is $\mathcal{O}(d\, m_k m)$, or $\mathcal{O}(d\, m_k n)$ with the optional cache speedup of Algorithm 1. The loop is repeated for random k until the cost-improvements of all dimensions k fall below some small ϵ.

- After convergence of the inner loop in (outer) iteration $\hat{\imath}$, the new basis function is $\psi_{\hat{\imath}} := g$. As the basis has changed, the linear parameters $\boldsymbol{a} \in \mathbb{R}^{\hat{\imath}}$ have to be readjusted by solving the ordinary least squares problem

$$\boldsymbol{a} \; = \; (\mathbf{\Psi}\mathbf{\Psi}^\top)^{-1}\mathbf{\Psi}\boldsymbol{y} \,, \quad \text{with} \;\; \mathit{\Psi}_{it} \; := \; \psi_i(\boldsymbol{x}_t) \,, \; \forall i \in \{1, \dots, \hat{\imath}\} \,, \; \forall t \in \{1, \dots, n\} \,.$$

  We propose to stop the approximation when the newly found basis function $\psi_{\hat{\imath}}$ is no longer *linearly independent* of the current basis $\{\psi_i\}_{i=1}^{\hat{\imath}-1}$. This can for example be tested by comparing the *determinant* $\det(\frac{1}{n}\mathbf{\Psi}\mathbf{\Psi}^\top) < \varepsilon$, for some very small ε.

4 Empirical Evaluation

In this section we will evaluate the novel LFF regression Algorithm 1, printed in detail on Page 133. We will analyze its properties on low dimensional toy-data, and compare its performance with sparse and traditional Gaussian processes (GP, see Bishop 2006; Rasmussen and Williams 2006).

4.1 Demonstration

To showcase the novel Algorithm 1, we tested it on an artificial two-dimensional regression toy-data set. The $n = 1000$ training samples were drawn from a noisy spiral and labeled with a sinus. The variance of the Gaussian sample-noise grew with the spiral as well:

$$\boldsymbol{x}_t \; = \; 6\tfrac{t}{n} \begin{bmatrix} \cos\left(6\tfrac{t}{n}\pi\right) \\ \sin\left(6\tfrac{t}{n}\pi\right) \end{bmatrix} + \mathcal{N}\Big(\mathbf{0}, \tfrac{t^2}{4n^2}\mathbf{I}\Big) \,, \quad y_t \; = \; \sin\Big(4\tfrac{t}{n}\pi\Big) \,, \quad \forall t \in \{1, \dots, n\} \,. \quad (7)$$

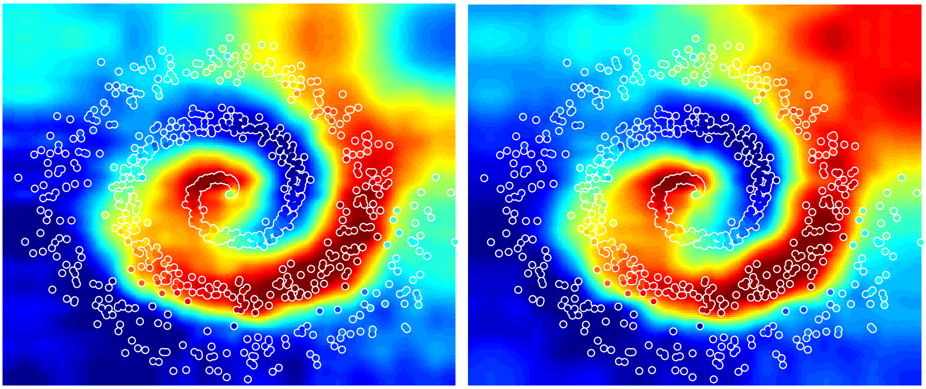

Fig. 2. Two LFF functions learned from the same 1000 training samples (white circles). The color inside a circle represents the training label. Outside the circles, the color represents the prediction of the LFF function. The differences between both functions are rooted in the randomized order in which the factor functions g^k are updated. However, the similarity of the sampled region indicates that poor initial choices can be compensated by subsequently constructed basis functions.

Figure 2 shows one training set plotted over two learned[8] functions $f \in \mathcal{F}^m$ with $m = 21$ and $m = 24$ factored basis functions, respectively. Regularization constants were in both cases $\sigma_k^2 = 0.0005, \forall k$. The differences between the functions stem from the randomized order in which the factor functions g^k are updated. Note that the sampled regions have similar predictions. Regions with strong differences, for example the upper right corner, are never seen during training.

In all our experiments, Algorithm 1 always converged. Runtime was mainly influenced by the input dimensionality ($\mathcal{O}(d^2)$), the number of training samples ($\mathcal{O}(n)$) and the eventual number of basis functions ($\mathcal{O}(m)$). The latter was strongly correlated with approximation quality, i.e., bad approximations converged fast. Cross-validation was therefore able to find good parameters efficiently and the resulting LFF were always very similar near the training data.

4.2 Evaluation

We compared the regression performance of LFF and GP with cross-validation on five regression benchmarks from the *UCI Manchine Learning Repository*[9]:

- The *concrete compressive strength* data set (*concrete*, Yeh 1998) consists of $n = 1030$ samples with $d = 8$ dimensions describing various concrete

[8] Here (and in the rest of the paper), each variable was encoded with 50 Fourier cosine bases. We tested other sizes as well. Few cosine bases result effectively in a low-pass filtered function, whereas every experiment with more than 20 or 30 bases behaved very similar. We tested up to $m_k = 1000$ bases and did not experience over-fitting.

[9] https://archive.ics.uci.edu/ml/index.html

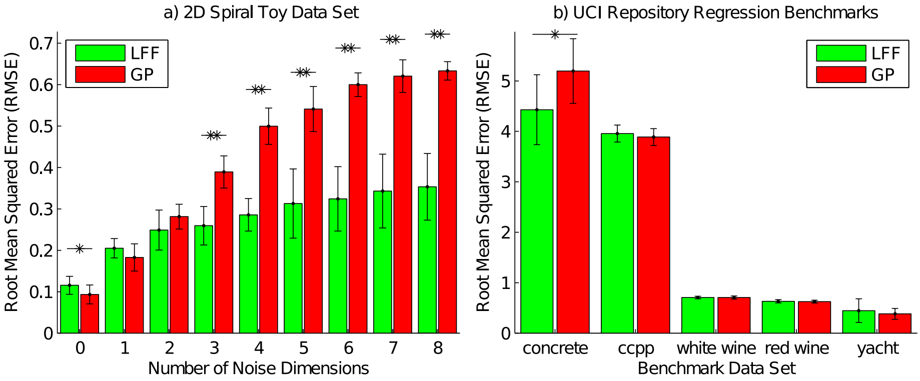

Fig. 3. Mean and standard deviation within a 10-fold cross-validation of a) the toy data set with additional independent noise input dimensions and b) all tested UCI benchmark data sets. The stars mark *significantly* different distribution of RMSE over all folds in both a *paired-sample t-test* and a *Wilcoxon signed rank test*. Significance levels are: one star $p < 0.05$, two stars $p < 0.005$.

mixture-components. The target variable is the real-valued compression strength of the mixture after it hardened.

- The *combined cycle power plant* data set (*ccpp*, Tüfekci 2014) consists of $n = 9568$ samples with $d = 4$ dimensions describing 6 years worth of measurements from a combined gas and steam turbine. The real-valued target variable is the energy output of the system.
- The *wine quality* data set (Cortez et al. 2009) consists of two subsets with $d = 11$ dimensions each, which describe physical attributes of various white and red wines: the set contains $n = 4898$ samples of *white wine* and $n = 1599$ samples of *red wine*. The target variable is the estimated wine quality on a discrete scale from 0 to 10.
- The *yacht hydrodynamics* data set (*yacht*, Gerritsma et al. 1981) consists of $n = 308$ samples with $d = 6$ dimensions describing parameters of the *Delft yacht hull* ship-series. The real-valued target variable is the residuary resistance measured in full-scale experiments.

To demonstrate the advantage of factored basis functions, we also used the 2d-spiral toy-data set of the previous section with a varying number of additional input dimensions. Additional values were drawn i.i.d. from a Gaussian distribution and are thus independent of the target labels. As the input space $\mathcal{X}$ grows, kernel methods will increasingly face the curse of dimensionality during training.

Every data-dimension (except the labels) have been translated and scaled to zero mean and unit-variance before training. Hyper-parameters were chosen w.r.t. the mean of a 10-fold cross-validation. LFF-regression was tested for the uniform noise-parameters $\sigma_k^2 \in \{10^{-10}, 10^{-9.75}, 10^{-9.5}, \dots, 10^{10}\}, \forall k$, i.e. for 81 different hyper-parameters. GP were tested with Gaussian kernels $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\frac{1}{2\bar{\sigma}^2}\|\boldsymbol{x} - \boldsymbol{y}\|_2^2)$ using kernel parameters $\bar{\sigma} \in \{10^{-1}, 10^{-3/4}, 10^{-1/2}, \dots, 3\}$ and prior-parameters $\beta \in \{10^{-2}, 10^{-1}, \dots, 10^{10}\}$ (Bishop 2006, see for the

Table 1. 10-fold cross-validation RMSE for benchmark data sets with d dimensions and n samples, resulting in m *basis functions*. The cross-validation took h hours.

DATA SET	$\boldsymbol{d}$	$\boldsymbol{n}$	**#SV**	**RMSE LFF**	**RMSE GP**	$\boldsymbol{m}$ **LFF**	$\boldsymbol{h}$ **LFF**	$\boldsymbol{h}$ **GP**
Concrete	8	1030	927	$\mathbf{4.429 \pm 0.69}$	5.196 ± 0.64	4.2 ± 0.8	3.00	0.05
CCPP	4	9568	2000	3.957 ± 0.17	3.888 ± 0.17	8.8 ± 2.0	1.96	1.14
White Wine	11	4898	2000	0.707 ± 0.02	0.708 ± 0.03	4.2 ± 0.4	4.21	0.69
Red Wine	11	1599	1440	0.632 ± 0.03	0.625 ± 0.03	4.7 ± 0.7	3.25	0.13
Yacht	6	308	278	0.446 ± 0.23	0.383 ± 0.11	4.2 ± 0.6	0.43	0.005

definition), i.e. for 221 different hyper-parameter combinations. The number of support vectors in standard GP equals the number of training samples. As this is not feasible for larger data sets, we used the MP-MAH algorithm (Böhmer et al. 2012) to select a uniformly distributed subset of 2000 training samples for sparse GP (Rasmussen and Williams 2006).

Figure 3a demonstrates the advantage of factored basis functions over kernel methods during training. The plot shows the *root mean squared errors*[10] (RMSE) of the two dimensional spiral toy-data set with an increasing number of independent noise dimensions. GP solves the initial task better, but clearly succumbs to the curse of dimensionality, as the size of the input space $\mathcal{X}$ grows. LFF, on the other hand, significantly overtake GP from 3 noise dimensions on, as the factored basis functions appear to be less affected by the curse. Another difference to GP is that decreasing performance automatically yields less factored basis functions (from 19.9 ± 2.18 with 0, to 6.3 ± 0.48 bases with 8 noise dimensions).

Figure 3b and Table 1 show that our LFF algorithm performs on all evaluated real-world benchmark data sets comparable to (sparse) GP. RMSE distributions over all folds were statistically indistinguishable, except for an advantage of LFF regression in the concrete compressive strength data set ($p < 0.01$ in a *t-test* and $p < 0.02$ in a *signed rank test*). As each basis function requries many iterations to converge, LFF regression runs considerably longer than standard approaches. However, LFF require between 3 and 12 factored basis functions to achieve the *same* performance as GP with 278-2000 kernel basis functions.

5 Discussion

We presented a novel algorithm for regression, which constructs factored basis functions during training. As *linear factored functions* (LFF) can in principle approximate any function in $L^2(\mathcal{X}, \vartheta)$, a regularization is necessary to avoid over-fitting. Here we rely on a regularization scheme that has been motivated by a Taylor approximation of the least-squares cost function with (an approximation of) virtual sample-noise. RMSE performance appears comparable to Gaussian

[10] RMSE are not a common performance metric for GP, which represent a *distribution* of solutions. However, RMSE reflect the objective of regression and are well suited to compare our algorithm with the *mean* of a GP.

processes on real-world benchmark data sets, but the factored representation is considerably more compact and seems to be less affected by distractors.

At the moment, LFF optimization faces two challenges. (i) The optimized cost function is not convex, but the local minimum of the solution may be controlled by selecting the next factor function to optimize. For example, MARS successively adds factor functions. Generalizing this will require further research, but may also allow some performance guarantees. (ii) The large number of inner-loop iterations make the algorithm slow. This problem should be mostly solved by addressing (i), but finding a trade-off between approximation quality and runtime may also provide a less compact shortcut with similar performance.

Preliminary experiments also demonstrated the viability of LFF in a *sparse regression* approach. Sparsity refers here to a limited number of input-dimensions that affect the prediction, which can be implemented by adjusting the sample-noise parameters σ_k^2 during training for each variable $\mathcal{X}_k$ individually. This is of particular interest, as factored functions are ideally suited to represent sparse functions and are in principle *unaffected* by the curse of dimensionality in function representation. Our approach modified the cost function to enforce LFF functions that were constant in all noise-dimensions. We did not include our results in this paper, as choosing the first updated factor functions g^k poorly resulted in basis functions that rather fitted noise than predicted labels. When we enforce sparseness, this initial mistake can afterwards no longer be rectified by other basis functions, in difference to the presented Algorithm 1. However, if this can be controlled by a sensible order in the updates, the resulting algorithm should be much faster and more robust than the presented version.

There are many application areas that may exploit the structural advantages of LLF. In *reinforcement learning* (Kaelbling et al. 1996), one can exploit the factorizing inner products to break the curse of dimensionality of the state space (Böhmer and Obermayer 2013). Factored transition models also need to be learned from experience, which is essentially a sparse regression task. Another possible field of application are *junction trees* (for Bayesian inference, see e.g. Bishop 2006) over continuous variables, where sparse regression may estimate the conditional probabilities. In each node one must also marginalize out variables, or calculate the point-wise product over multiple functions. Both operations can be performed analytically with LFF, the latter at the expense of more basis functions in the resulting LFF. However, one can use our framework to *compress* these functions after multiplication. This would allow junction-tree inference over mixed continuous and discrete variables.

In summary, we believe our approach to approximate functions by constructing non-linear factored basis functions (LFF) to be very promising. The presented algorithm performs comparable with Gaussian processes, but appears less sensitive to large input spaces than kernel methods. We also discussed some potential extensions for sparse regression that should improve upon that, in particular on runtime, and gave some fields of application that would benefit greatly from the algebraic structure of LFF.

Acknowledgments. The authors thank Yun Shen and the anonymous reviewers for their helpful comments. This work was funded by the *German science foundation* (DFG) within SPP 1527 *autonomous learning.*

Appendix A LFF Definition and Properties

Let $\mathcal{X}_k$ denote the subset of $\mathbb{R}$ associated with the k'th variable of input space $\mathcal{X} \subset \mathbb{R}^d$, such that $\mathcal{X} := \mathcal{X}_1 \times \ldots \times \mathcal{X}_d$. To avoid the curse of dimensionality in this space, one can integrate w.r.t. a *factored probability measure* ϑ, i.e. $\vartheta(d\boldsymbol{x}) = \prod_{k=1}^d \vartheta^k(dx_k), \int \vartheta^k(dx_k) = 1, \forall k$. For example, ϑ^k could be uniform or Gaussian distributions over $\mathcal{X}_k$ and the resulting ϑ would be a uniform or Gaussian distribution over the input space $\mathcal{X}$.

A function $g : \mathcal{X} \to \mathbb{R}$ is called a *factored function* if it can be written as a product of one-dimensional *factor functions* $g^k : \mathcal{X}_k \to \mathbb{R}$, i.e. $g(\boldsymbol{x}) = \prod_{k=1}^d g^k(x_k)$. We only consider factored functions g that are twice integrable w.r.t. measure ϑ, i.e. $g \in L^2(\mathcal{X}, \vartheta)$. Note that not all functions $f \in L^2(\mathcal{X}, \vartheta)$ are factored, though. Due to *Fubini's theorem* the d-dimensional inner product between two factored functions $g, g' \in L^2(\mathcal{X}, \vartheta)$ can be written as the product of d one-dimensional inner products:

$$\langle g, g' \rangle_\vartheta = \int \vartheta(d\boldsymbol{x})\, g(\boldsymbol{x})\, g'(\boldsymbol{x}) = \int \prod_{k=1}^d \vartheta^k(dx_k)\, g^k(dx_k)\, g'^k(dx_k) = \prod_{k=1}^d \langle g^k, g'^k \rangle_{\vartheta^k} \,.$$

This trick can be used to solve the integrals at the heart of many least-squares algorithms. Our aim is to *learn* factored basis functions ψ_i. To this end, let $\{\phi_j^k : \mathcal{X}_k \to \mathbb{R}\}_{j=1}^{m_k}$ be a well-chosen[11] (i.e. universal) basis on $\mathcal{X}_k$, with the space of linear combinations denoted by $\mathcal{L}_\phi^k := \{\boldsymbol{b}^\top \boldsymbol{\phi}^k | \boldsymbol{b} \in \mathbb{R}^{m_k}\}$. One can thus approximate factor functions of ψ_i in $\mathcal{L}_\phi^k$, i.e., as linear functions

$$\psi_i^k(x_k) \quad := \quad \sum_{j=1}^{m_k} B_{ji}^k\, \phi_j^k(x_k) \quad \in \quad \mathcal{L}_\phi^k \,, \qquad\qquad \mathbf{B}^k \;\in\; \mathbb{R}^{m_k \times m} \,. \tag{8}$$

Let $\mathcal{F}$ be the space of all factored basis functions ψ_i defined by the factor functions ψ_i^k above, and $\mathcal{F}^m$ be the space of all linear combinations of those m factored basis functions (Equation 2).

Marginalization of LFF can be performed analytically with Fourier bases ϕ_j^k and uniform distribution ϑ (many other bases can be analytically solved as well):

$$\int \vartheta^l(dx_l)\, f(\boldsymbol{x}) = \sum_{i=1}^m \Big(a_i \sum_{j=1}^{m_l} B_{ji}^l \underbrace{\langle \phi_j^l, 1 \rangle_{\vartheta^l}}_{\text{mean of } \phi_j^l} \Big) \Big[\prod_{k \neq l}^d \psi_i^k \Big] \overset{\text{Fourier}}{=} \sum_{i=1}^m \underbrace{a_i B_{1i}^l}_{\text{new } a_i} \Big[\prod_{k \neq l}^d \psi_i^k \Big] \tag{9}$$

[11] Examples for continuous variables $\mathcal{X}_k$ are Fourier cosine bases $\phi_j^k(x_k) \sim \cos\big((j-1)\,\pi\, x_k\big)$, and Gaussian bases $\phi_j^k(x_k) = \exp\big(\frac{1}{2\sigma^2}(x_k - s_{kj})^2\big)$. Discrete variables may be represented with Kronecker-delta bases $\phi_j^k(x_k = i) = \delta_{ij}$.

Using the trigonometric *product-to-sum* identity $\cos(x)\cdot\cos(y) = \frac{1}{2}\big(\cos(x-y) + \cos(x+y)\big)$, one can also compute the point-wise product between two LFF f and $\bar f$ with cosine-Fourier base (solutions to other Fourier bases are less elegant):

$$\begin{aligned} \tilde f(\boldsymbol{x}) \;&:=\; f(\boldsymbol{x}) \cdot \bar f(\boldsymbol{x}) \\ &\overset{\text{Fourier}}{=} \sum_{i,j=1}^{m\bar m} \underbrace{a_i \, \bar a_j}_{\text{new } \tilde a_t} \prod_{k=1}^{d} \sum_{l=1}^{2m_k} \overbrace{\Big(\tfrac{1}{2} \sum_{q=1}^{l-1} B^k_{qi} \, \bar B^k_{(l-q)j} + \tfrac{1}{2} \sum_{q=l+1}^{m_k} B^k_{qi} \, \bar B^k_{(q-l)j} \Big)}^{\text{new } \tilde B^k_{lt}} \phi^k_l(x_k)\,, \end{aligned} \tag{10}$$

where $t := (i-1)\,\bar m + j$, and $B^k_{ji} := 0, \forall j > m_k$, for both f and $\bar f$. Note that this increases the number of basis functions $\tilde m = m\bar m$, and the number of bases $\tilde m_k = 2m_k$ for each respective input dimension. The latter can be counteracted by *low-pass filtering*, i.e., by setting $\tilde B^k_{ji} := 0, \forall j > m_k$.

Appendix B Inner Loop Derivation

Here we will optimize the problem in Equation 6 for one variable $\mathcal{X}_k$ at a time, by describing the update step $g^k \leftarrow g'^k$. This is repeated with randomly chosen variables k, until convergence of the cost $\tilde{\mathcal{C}}[g]$, that is, until all possible updates decrease the cost less than some small ϵ.

Let in the following $\mathbf{C}^k := \langle \boldsymbol{\phi}^k, \boldsymbol{\phi}^{k\top} \rangle_{\vartheta^k}$ and $\dot{\mathbf{C}}^k := \langle \nabla_k \boldsymbol{\phi}^k, \nabla_k \boldsymbol{\phi}^{k\top} \rangle_{\vartheta^k}$ denote covariance matrices, and $\boldsymbol{R}^k_l := \frac{\partial}{\partial \boldsymbol{b}^k} \langle \nabla_l g, \nabla_l f \rangle_\vartheta$ denote the derivative of one regularization term. Note that for some choices of bases $\{\phi^k_j\}_{j=1}^{m_k}$, one can compute the covariance matrices analytically before the main algorithm starts, e.g. Fourier cosine bases have $C^k_{ij} = \delta_{ij}$ and $\dot C^k_{ij} = (i-1)^2 \, \pi^2 \, \delta_{ij}$.
The approximated cost function in Equation 6 is

$$\tilde{\mathcal{C}}[g] \;=\; \|g\|^2_\xi - 2\langle g, \mu - f\rangle_\xi + \|\mu - f\|^2_\xi + \sum_{k=1}^{d} \sigma^2_k \Big(\|\nabla_k g\|^2_\vartheta + 2\langle \nabla_k g, \nabla_k f \rangle_\vartheta + \|\nabla_k f\|^2_\vartheta \Big)\,.$$

The non-zero gradients of all inner products of this equation w.r.t. parameter vector $\boldsymbol{b}^k \in \mathbb{R}^{m_k}$ are

$$\begin{aligned} \tfrac{\partial}{\partial \boldsymbol{b}^k} \langle g, g \rangle_\xi &= 2\,\langle \boldsymbol{\phi}^k \cdot \prod_{l \neq k} g^l, \prod_{l \neq k} g^l \cdot \boldsymbol{\phi}^{k\top} \rangle_\xi \boldsymbol{b}^k\,, \\ \tfrac{\partial}{\partial \boldsymbol{b}^k} \langle g, \mu - f \rangle_\xi &= \langle \boldsymbol{\phi}^k \cdot \prod_{l \neq k} g^l, \mu - f \rangle_\xi\,, \\ \tfrac{\partial}{\partial \boldsymbol{b}^k} \langle \nabla_l g, \nabla_l g \rangle_\vartheta &= \tfrac{\partial}{\partial \boldsymbol{b}^k} \langle \nabla_l g^l, \nabla_l g^l \rangle_{\vartheta^l} \prod_{s \neq l} \overbrace{\langle g^s, g^s \rangle_{\vartheta^s}}^{1} \quad = \quad 2\,\delta_{kl}\, \dot{\mathbf{C}}^k \boldsymbol{b}^k\,, \end{aligned}$$

$$\boldsymbol{R}^k_l \quad := \quad \tfrac{\partial}{\partial \boldsymbol{b}^k} \langle \nabla_l g, \nabla_l f \rangle_\vartheta = \begin{cases} \dot{\mathbf{C}}^k \mathbf{B}^k \Big[\boldsymbol{a} \cdot \prod\limits_{s \neq k} \mathbf{B}^{s\top} \mathbf{C}^s \boldsymbol{b}^s \Big] & \text{, if} \quad k = l \\ \mathbf{C}^k \mathbf{B}^k \Big[\boldsymbol{a} \cdot \mathbf{B}^{l\top} \dot{\mathbf{C}}^l \boldsymbol{b}^l \cdot \prod\limits_{s \neq k \neq l} \mathbf{B}^{s\top} \mathbf{C}^s \boldsymbol{b}^s \Big] & \text{, if} \quad k \neq l \end{cases}\,.$$

Setting this to zero yields the unconstrained solution g^k_{uc},

$$\boldsymbol{b}^k_{uc} = \Big(\overbrace{\langle \boldsymbol{\phi}^k \cdot \prod_{l \neq k} g^l, \prod_{l \neq k} g^l \cdot \boldsymbol{\phi}^{k\top} \rangle_\xi + \sigma^2_k \dot{\mathbf{C}}^k}^{\text{regularized covariance matrix } \bar{\mathbf{C}}^k} \Big)^{-1} \Big(\langle \boldsymbol{\phi}^k \cdot \prod_{l \neq k} g^l, \mu - f \rangle_\xi - \sum_{l=1}^{d} \boldsymbol{R}^k_l \, \sigma^2_l \Big)\,. \tag{11}$$

However, these parameters do not satisfy to the constraint $\|g'^k\|_{\vartheta^k} \overset{!}{=} 1$, and have to be normalized:

$$\boldsymbol{b}'^k \quad := \quad \frac{\boldsymbol{b}^k_{uc}}{\|g^k_{uc}\|_{\vartheta^k}} \quad = \quad \frac{\boldsymbol{b}^k_{uc}}{\sqrt{\boldsymbol{b}^{k\top}_{uc}\mathbf{C}^k\boldsymbol{b}^k_{uc}}} \,. \tag{12}$$

The inner loop finishes when for all k the improvement[12] from g^k to g'^k drops below some very small threshold ϵ, i.e. $\tilde{\mathcal{C}}[g] - \tilde{\mathcal{C}}[g'] < \epsilon$. Using $g'^l = g^l, \forall l \neq k$, one can calculate the left hand side:

$$\begin{aligned}\tilde{\mathcal{C}}[g] - \tilde{\mathcal{C}}[g'] &= \|g\|^2_\xi - \|g'\|^2_\xi - 2\langle g - g', \mu - f\rangle_\xi \\ &\quad + \sum_{l=1}^{d}\sigma_l^2 \Big[\underbrace{\|\nabla_l g\|^2_\vartheta}_{\boldsymbol{b}^{l\top}\dot{\mathbf{C}}^l\boldsymbol{b}^l} - \underbrace{\|\nabla_l g'\|^2_\vartheta}_{\boldsymbol{b}'^{l\top}\dot{\mathbf{C}}^l\boldsymbol{b}'^l} - 2\underbrace{\langle \nabla_l g - \nabla_l g', \nabla_l f\rangle_\vartheta}_{(\boldsymbol{b}^k - \boldsymbol{b}'^k)^\top \boldsymbol{R}^k_l}\Big] \qquad (13) \\ &= 2\langle g - g', \mu - f\rangle_\xi + \boldsymbol{b}^{k\top}\bar{\mathbf{C}}^k\boldsymbol{b}^{k\top} - \boldsymbol{b}'^{k\top}\bar{\mathbf{C}}^k\boldsymbol{b}'^{k\top} - 2(\boldsymbol{b}^k - \boldsymbol{b}'^k)^\top\Big(\sum_{l=1}^{d}\boldsymbol{R}^k_l\sigma_l^2\Big)\,.\end{aligned}$$

Appendix C Proofs of the Propositions

Proof of Proposition 2: The 1st order Taylor approximation of any $g, f \in L^2(\mathcal{X}, \xi\chi)$ around $\boldsymbol{z} \in \mathcal{X}$ is $f(\boldsymbol{x}) = f(\boldsymbol{z} + \boldsymbol{x} - \boldsymbol{z}) \approx f(\boldsymbol{z}) + (\boldsymbol{x} - \boldsymbol{z})^\top \boldsymbol{\nabla} f(\boldsymbol{z})$. For the Hilbert space $L^2(\mathcal{X}, \xi\chi)$ we can thus approximate:

$$\begin{aligned}\langle g, f\rangle_{\xi\chi} &= \int \xi(d\boldsymbol{z}) \int \chi(d\boldsymbol{x}|\boldsymbol{z})\, g(\boldsymbol{x})\, f(\boldsymbol{x}) \\ &\approx \int \xi(d\boldsymbol{z})\Big(g(\boldsymbol{z})\, f(\boldsymbol{z}) \textstyle\int \overbrace{\xi(d\boldsymbol{x}|\boldsymbol{z})}^{1} + g(\boldsymbol{z}) \textstyle\int \overbrace{\chi(d\boldsymbol{x}|\boldsymbol{z})\,(\boldsymbol{x} - \boldsymbol{z})}^{\mathbf{0} \text{ due to (eq.5)}}{}^\top \boldsymbol{\nabla} f(\boldsymbol{z}) \\ &\quad + \textstyle\int \underbrace{\chi(d\boldsymbol{x}|\boldsymbol{z})\,(\boldsymbol{x} - \boldsymbol{z})}_{\mathbf{0} \text{ due to (eq.5)}}{}^\top \boldsymbol{\nabla} g(\boldsymbol{z})\, f(\boldsymbol{z}) + \boldsymbol{\nabla} g(\boldsymbol{z})^\top \textstyle\int \underbrace{\chi(d\boldsymbol{x}|\boldsymbol{z})\,(\boldsymbol{x} - \boldsymbol{z})(\boldsymbol{x} - \boldsymbol{z})^\top}_{\frac{\vartheta}{\xi}(\boldsymbol{z})\cdot\boldsymbol{\Sigma} \text{ due to (eq.5)}} \boldsymbol{\nabla} f(\boldsymbol{z})\Big) \\ &= \langle g, f\rangle_\xi + \sum_{k=1}^{d}\sigma_k^2\, \langle \nabla_k g, \nabla_k f\rangle_\vartheta\,.\end{aligned}$$

Using this twice and the zero mean assumption (Eq. 5), we can derive:

$$\begin{aligned}\inf_{g\in\mathcal{F}} \mathcal{C}[f + g|\chi, \mu] &\equiv \inf_{g\in\mathcal{F}} \iint \xi(d\boldsymbol{z})\,\chi(d\boldsymbol{x}|\boldsymbol{z})\Big(g^2(\boldsymbol{x}) - 2\,g(\boldsymbol{x})\,\big(\mu(\boldsymbol{z}) - f(\boldsymbol{x})\big)\Big) \\ &= \inf_{g\in\mathcal{F}} \langle g, g\rangle_{\xi\chi} + 2\langle g, f\rangle_{\xi\chi} - 2\int \xi(d\boldsymbol{z})\,\mu(\boldsymbol{z}) \int \chi(d\boldsymbol{x}|\boldsymbol{z})\, g(\boldsymbol{x}) \\ &\approx \inf_{g\in\mathcal{F}} \langle g, g\rangle_\xi - 2\langle g, \mu - f\rangle_\xi + \sum_{k=1}^{d}\sigma_k^2\Big(\langle \nabla_k g, \nabla_k g\rangle_\vartheta + 2\langle \nabla_k g, \nabla_k f\rangle_\vartheta\Big) \\ &\equiv \inf_{g\in\mathcal{F}} \|g - (\mu - f)\|^2_\xi + \sum_{k=1}^{d}\sigma_k^2\|\nabla_k g + \nabla_k f\|^2_\vartheta \quad = \quad \tilde{\mathcal{C}}[g]\,.\end{aligned}$$

□

[12] Anything simpler does not converge, as the parameter vectors often evolve along chaotic orbits in $\mathbb{R}^{m_k}$.

Proof of Proposition 3: The analytical solution to the optimization problem in Equation 6 is derived in Appendix B and has a unique solution if the matrix $\bar{\mathbf{C}}^k$, defined in Equation 11, is of full rank:

$$\bar{\mathbf{C}}^k \quad := \quad \langle \boldsymbol{\phi}^k \cdot \prod_{l \neq k} g^l, \prod_{l \neq k} g^l \cdot \boldsymbol{\phi}^{k\top} \rangle_\xi \quad + \quad \sigma_k^2 \dot{\mathbf{C}}^k \,.$$

For Fourier bases the matrix $\dot{\mathbf{C}}^k$ is diagonal, with $\dot{C}_{11}^k$ being the only zero entry. $\bar{\mathbf{C}}^k$ is therfore full rank if $\sigma_k^2 > 0$ and $\bar{C}_{11}^k > 0$. Because ϑ is *absolutely continuous* w.r.t. ξ, the constraint $\|g^l\|_\vartheta = 1, \forall l$, implies that there exist no g^l that is zero on *all* training samples. As the first Fourier base is a constant, $\langle \phi_1^k \cdot \prod_{l \neq k} g^l, \prod_{l \neq k} g^l \cdot \phi_1^k \rangle_\xi > 0$ and the matrix $\bar{\mathbf{C}}^k$ is therefore of full rank. □

Algorithm 1. (detailed) – LFF-Regression

```
Input:   $\mathbf{X} \in \mathrm{I\!R}^{d \times n}$,   $\boldsymbol{y} \in \mathrm{I\!R}^n$,   $\boldsymbol{\sigma}^2 \in \mathrm{I\!R}^d$   $\epsilon, \varepsilon \in \mathrm{I\!R}$
$\mathbf{C}^k := \langle \boldsymbol{\phi}^k, \boldsymbol{\phi}^k \rangle_{\vartheta^k}$,   $\dot{\mathbf{C}}^k := \langle \nabla\boldsymbol{\phi}^k, \nabla\boldsymbol{\phi}^k \rangle_{\vartheta^k}$,   $\forall k$        // analytical covariance matrices
$\mathit{\Phi}_{jt}^k := \boldsymbol{\phi}_j^k(X_{kt})$,   $\forall k, \forall j, \forall t$        // optional cache of sample-expansion
$\boldsymbol{f} := \mathbf{0} \in \mathrm{I\!R}^n$;   $\boldsymbol{a} := \emptyset$;   $\mathbf{B}^k := \emptyset, \ \forall k$;   $\boldsymbol{\Psi} := \infty$        // initialize empty $f \in \mathcal{F}^0$
while det$\left(\frac{1}{n}\boldsymbol{\Psi}\boldsymbol{\Psi}^\top\right) > \varepsilon$ do
  $\boldsymbol{b}^k := \mathbf{1}^k \in \mathrm{I\!R}^{m_k}, \ \forall k$;   $\boldsymbol{g}^k := \mathbf{1} \in \mathrm{I\!R}^n, \ \forall k$        // initialize all $g^k$ as constant
  $\boldsymbol{h} := \boldsymbol{\infty} \in \mathrm{I\!R}^d$        // initialize estimated improvement
  while max$(\boldsymbol{h}) > \epsilon$ do
    for $k$ in randperm$(1, \ldots, d)$ do
      $\boldsymbol{R}_k := \dot{\mathbf{C}}^k \mathbf{B}^k [\boldsymbol{a} \cdot \prod_{s \neq k} \mathbf{B}^{s\top} \mathbf{C}^s \boldsymbol{b}^s]$        // $\boldsymbol{R}_k = \frac{\partial}{\partial \boldsymbol{b}^k} \langle \nabla_k g, \nabla_k f \rangle_\vartheta$
      $\boldsymbol{R}_l := \mathbf{C}^k \mathbf{B}^k [\boldsymbol{a} \cdot \mathbf{B}^{l\top} \dot{\mathbf{C}}^l \boldsymbol{b}^l \cdot \prod_{s \neq k \neq l} \mathbf{B}^{s\top} \mathbf{C}^s \boldsymbol{b}^s]$,   $\forall l \neq k$        // $\boldsymbol{R}_l = \frac{\partial}{\partial \boldsymbol{b}^k} \langle \nabla_l g, \nabla_l f \rangle_\vartheta$
      $\bar{\mathbf{C}} := \boldsymbol{\Phi}^k \Big[\boldsymbol{\Phi}^{k\top} \cdot \prod_{l \neq k} (\boldsymbol{g}^l)^2 \mathbf{1}^\top\Big] + \sigma_k^2 \dot{\mathbf{C}}^k$        // regularized cov. matrix (eq. 11)
      $\boldsymbol{b}' := \bar{\mathbf{C}}^{-1}\Big(\boldsymbol{\Phi}^k \Big[(\boldsymbol{y} - \boldsymbol{f}) \cdot \prod_{l \neq k} \boldsymbol{g}^l\Big] - \mathbf{R}\boldsymbol{\sigma}^2\Big)$        // unconstrained $g_{uc}^k$ (eq. 11)
      $\boldsymbol{b}' := \boldsymbol{b}' \,/\, \sqrt{\boldsymbol{b}'^\top \mathbf{C}^k \boldsymbol{b}'}$        // enforce constraints (eq. 12)
      $h_k := \frac{2}{n} (\boldsymbol{b}^k - \boldsymbol{b}')^\top \Big(\boldsymbol{\Phi}^k \Big[(\boldsymbol{y} - \boldsymbol{f}) \cdot \prod_{l \neq k} \boldsymbol{g}^l\Big]\Big)$        // approximate $2\langle g - g', \mu - f \rangle_\xi$
      $h_k := h_k + \boldsymbol{b}^k \bar{\mathbf{C}} \boldsymbol{b}^k - \boldsymbol{b}' \bar{\mathbf{C}} \boldsymbol{b}' - 2(\boldsymbol{b}^k - \boldsymbol{b}')^\top \mathbf{R}\boldsymbol{\sigma}^2$        // cost improvement (eq. 13)
      $\boldsymbol{b}^k := \boldsymbol{b}'$;   $\boldsymbol{g}^k := \boldsymbol{\Phi}^{k\top} \boldsymbol{b}^k$        // update factor function $g^k$
    end for   // end function $g^k$ update
  end while   // end inner loop: cost function converged and thus $g$ optimized
  $\mathbf{B}^k := [\mathbf{B}^k, \boldsymbol{b}^k], \ \forall k$;   $\boldsymbol{\Psi} := \Big[\prod_{k=1}^{d} \mathbf{B}^{k\top} \boldsymbol{\Phi}^k\Big]$        // adding $g$ to the bases functions of $f$
  $\boldsymbol{a} := (\boldsymbol{\Psi}\boldsymbol{\Psi}^\top)^{-1} \boldsymbol{\Psi}\boldsymbol{y}$;     $\boldsymbol{f} := \boldsymbol{\Psi}^\top \boldsymbol{a}$        // project $\mu$ onto new bases
end while     // end outer loop: new $g$ no longer linear independent, thus $f \approx \mu$
Output:   $\boldsymbol{a} \in \mathrm{I\!R}^m$,   $\{\mathbf{B}^k \in \mathrm{I\!R}^{m_k \times m}\}_{k=1}^d$        // return parameters of $f \in \mathcal{F}^m$
```

References

Bellman, R.E.: Dynamic programming. Princeton University Press (1957)

Bishop, C.M.: Pattern Recognition and Machine Learning. Springer-Verlag New York Inc, Secaucus (2006). ISBN 0387310738

Böhmer, W., Obermayer, K.: Towards structural generalization: Factored approximate planning. ICRA Workshop on Autonomous Learning (2013). http://autonomous-learning.org/wp-content/uploads/13-ALW/paper_1.pdf

Böhmer, W., Grünewälder, S., Nickisch, H., Obermayer, K.: Generating feature spaces for linear algorithms with regularized sparse kernel slow feature analysis. Machine Learning **89**(1–2), 67–86 (2012)

Böhmer, W., Grünewälder, S., Shen, Y., Musial, M., Obermayer, K.: Construction of approximation spaces for reinforcement learning. Journal of Machine Learning Research **14**, 2067–2118 (2013)

Boser, B.E., Guyon, I.M., Vapnik, V.N.: A training algorithm for optimal margin classifiers. In: Proceedings of the Fifth Annual Workshop on Computational Learning Theory, pp. 144–152 (1992)

Cortez, P., Cerdeira, A., Almeida, F., Matos, T., Reis, J.: Modeling wine preferences by data mining from physicochemical properties. Decision Support Systems **47**(4), 547–553 (2009)

Csató, L., Opper, M.: Sparse on-line Gaussian processes. Neural Computation **14**(3), 641–668 (2002)

Friedman, J.H.: Multivariate adaptive regression splines. The Annals of Statistics **19**(1), 1–67 (1991)

Gerritsma, J., Onnink, R., Versluis, A.: Geometry, resistance and stability of the delft systematic yacht hull series. Int. Shipbuilding Progress **28**, 276–297 (1981)

Gönen, M., Alpaydın, E.: Multiple kernel learning algorithms. Journal of Machine Learning Research 12, 2211–2268 (2011). ISSN 1532–4435

Haykin, S.: Neural Networks: A Comprehensive Foundation, 2nd edn. Prentice Hall (1998). ISBN 978-0132733502

Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: a survey. Journal of Artificial Intelligence Research **4**, 237–285 (1996)

Nelder, J.A., Wedderburn, R.W.M.: Generalized linear models. Journal of the Royal Statistical Society, Series A, General **135**, 370–384 (1972)

Pearl, J.: Probabilistic reasoning in intelligent systems. Morgan Kaufmann (1988)

Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. MIT Press (2006)

Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press (2004)

Tipping, M.E.: Sparse bayesian learning and the relevance vector machine. Journal of Machine Learning Research **1**, 211–244 (2001). ISSN 1532–4435

Tüfekci, P.: Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods. International Journal of Electrical Power & Energy Systems **60**, 126–140 (2014)

Vapnik, V.N.: The Nature of Statistical Learning Theory. Springer (1995)

Wang, Z., Crammer, K., Vucetic, S.: Breaking the curse of kernelization: Budgeted stochastic gradient descent for large-scale svm training. Journal of Machine Learning Research **13**(1), 3103–3131 (2012). ISSN 1532–4435

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