Data-Driven Kernel Designs for Optimized Greedy Schemes: A Machine Learning Perspective

SIAM J. SCI. COMPUT.
Vol. 46, No. 1, pp. C101–C126

DATA-DRIVEN KERNEL DESIGNS FOR OPTIMIZED GREEDY SCHEMES: A MACHINE LEARNING PERSPECTIVE*

TIZIAN WENZEL†, FRANCESCO MARCHETTI‡, AND EMMA PERRACCHIONE§

Abstract. Thanks to their easy implementation via radial basis functions (RBFs), meshfree kernel methods have proved to be an effective tool for, e.g., scattered data interpolation, PDE collocation, and classification and regression tasks. Their accuracy might depend on a length scale hyperparameter, which is often tuned via cross-validation schemes. Here we leverage approaches and tools from the machine learning community to introduce two-layered kernel machines, which generalize the classical RBF approaches that rely on a single hyperparameter. Indeed, the proposed learning strategy returns a kernel that is optimized not only in the Euclidean directions, but that further incorporates kernel rotations. The kernel optimization is shown to be robust by using recently improved calculations of cross-validation scores. Finally, the use of greedy approaches, and specifically of the vectorial kernel orthogonal greedy algorithm (VKOGA), allows us to construct an optimized basis that adapts to the data. Beyond a rigorous analysis on the convergence of the so-constructed two-layered (2L)-KOGA, its benefits are highlighted on both synthesized and real benchmark datasets.

Key words. kernel interpolation, greedy methods, machine learning optimization

MSC codes. 65D15, 41A05, 68Q32

DOI. 10.1137/23M1551201

1. Introduction.

Kernel methods [13, 42] are an active field of research due to their wide applicability in several tasks such as machine learning, approximation theory, and numerical analysis. In order to increase the accuracy of kernel methods and to avoid instability issues at the same time, many researchers already have worked on the problem of selecting suitable center points, e.g., via greedy kernel methods, and on the computational issue of finding suitable values for the kernel hyperparameter, namely, the shape parameter in radial basis functions (RBFs) literature.

As far as the first item is concerned, assuming that a set of measurements sampled at multivariate scattered points is given, the basic idea of greedy kernel methods [46, 50] consists in selecting only a smaller subset (called *centers*) out of the large

*Submitted to the journal's Machine Learning Methods for Scientific Computing section February 3, 2023; accepted for publication (in revised form) October 6, 2023; published electronically February 1, 2024.

https://doi.org/10.1137/23M1551201

Funding: The work of the first author was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy EXC 2075 - 390740016 and by the BMBF under contract 05M20VSA. The work of the second author was supported by the Programma Operativo Nazionale (PON) "Ricerca e Innovazione" 2014-2020. The work of the third author was supported by the project "Physics-based AI for predicting extreme weather and space weather events (AIxtreme)" by the Compagnia di San Paolo (CUP: D33C22002220007). The research of the second and third authors was accomplished within the Italian Network on Approximation (RITA) and the thematic group on "Approximation Theory and Applications" (TAA) of the Italian Mathematical Union (UMI), and was partially funded by the GNCS-INδAM.

†Institute for Applied Analysis and Numerical Simulation, University of Stuttgart, Stuttgart, 70569, Germany (tizian.wenzel@mathematik.uni-stuttgart.de).

‡Dipartimento di Matematica "Tullio Levi-Civita," Università di Padova, Padova, Italy (francesco.marchetti@unipd.it).

§Dipartimento di Scienze Matematiche Giuseppe Luigi Lagrange, Politecnico di Torino, Torino, Italy (emma.perracchione@polito.it).

training set of scattered points to build the kernel model. For this, one starts with an empty set of centers and then adds one more center, step by step, according to some selection criterion [9, 11, 30, 39], until a predefined number of centers or some other stopping criterion is met. In this context, we will employ the vectorial kernel orthogonal greedy algorithm (VKOGA) [38, 49], which implements several greedy algorithms in an efficient way.

Moreover, besides greedy methods, we point out that in the literature there is quite a lot of work on selecting or optimizing a single hyperparameter for RBFs; see, e.g., [13, chapter 14] or [42] for a general overview and, e.g., [10, 14, 28] for specific instances that comprise heuristic approaches, cross-validation methods, or schemes based on maximum likelihood estimation. Such a hyperparameter determines the shape of the basis function, which can be more s*piky* or more *flat.* This radial scaling works fine as long as the data have similar scales along the Euclidean directions. Some improvement in this sense is provided by the so-called anisotropic kernels that need the selection of several scaling parameters, precisely as many as the problem dimension (see [13, chapter 3] for a general discussion on the topic). Unfortunately, anisotropic kernels in the classical sense do not allow any kernel rotation, and hence here we face the more challenging problem of learning the metric from data, thus allowing the presence of rotations. This issue is known as distance metric learning, and several algorithms, such as the collapsing classes method, have been developed to solve computational issues, e.g., in a k-nearest neighbor classification (see [1, 16, 17, 40]). A different approach to generating anisotropic bases is represented by the so-called variably scaled kernels, which demonstrated effectiveness in encoding steep gradients and discontinuities [3, 8, 34, 36].

More recent research tries to predict suitable kernel shape parameters with the help of machine learning methods; see, e.g., [29]. Another related approach is proposed in [31], where random Fourier features are used for automatic relevance determination of features; however, this does not incorporate rotated features. A more sophisticated tool, which can be seen as very deep kernels, is provided under the notion of *kernel flows;* see [32]. There the authors learn a nonparametric family of deep kernels of a given form via incremental data-dependent deformations, thus obtaining very deep (bottomless) kernels. Their optimization method is related to ours, but is based on a different error criterion. See also subsection 3.3 for some more comments on the relation. Moreover, [20] advances this approach by considering cross-validation criteria based on the maximal Lyapunov exponent or maximum mean discrepancy (MMD). Other approaches that are labeled *deep kernels* transform the considered input data with the help of a neural network and apply a kernel model (e.g., via a Gaussian process) on top of the prediction [48]. In contrast to these approaches, we make use of a two-layered kernel and give a clear interpretation in terms of a hyperparameter optimized kernel instead of using deep [48] or even "bottomless" kernels [32]. Another somewhat related topic is the *active subspaces* [6, 7]. Here the idea is to identify active subspaces, along which the target function changes, and inactive subspaces, along which the target function is invariant. However this is done using gradient information, which is usually not available in scattered data approximation. Furthermore, such active subspaces can in principle also be detected with our approach, as shown in section 3.

In the above setting our main contributions are

1. developing a two-layered kernel machine for *optimal* data-driven kernel designs, i.e., we learn the metric from the samples;
2. using such a kernel design to construct a sparse basis via greedy methods.

These two steps allow us to obtain more efficient kernel models for machine learning than previous state-of-the-art greedy methods. For the second step, we employ the VKOGA algorithm [38, 49], which implements several greedy algorithms in an efficient way. As we focus on the scalar valued output case, we denote our resulting two-layered greedy kernel machine as two-layered (2L)-KOGA. An extension of the method to the vectorial output case was done in [43].

As far as the first item above is concerned, we go beyond state-of-the art literature by using both kernel scalings and rotations and provide a helpful interpretation based on a deep kernel representer theorem as two-layered kernel. Finally, leveraging recent advancements on the computation of k-fold cross-validation scores, we provide efficient optimization procedures. Specifically, the classical hyperparameter is replaced by a more general linear mapping, and we prove that this turns out to be equivalent to a two-layered kernel machine.

We then analyze both theoretical and computational aspects of the proposed 2L-KOGA. The convergence analysis shows that we may obtain faster rates of convergence, and this depends on the singular values of the optimized first layer linear mapping matrix. Numerical evidence stresses that the use of the two-layered optimized kernel possibly significantly outperforms state-of-the art techniques, and we furthermore provide a practical criterion that allows us to discriminate when we should and should not expect some benefit. Such a criterion can be implemented already after the kernel optimization, i.e., before running the slightly more expensive greedy selection. Furthermore, we are also able to prove that the complexity of our approach turns out to be much cheaper than a straightforward cross-validation of kernel length scale parameters.

The outline of the paper is as follows. In section 2 we briefly review the basics of kernel methods and we introduce the main tools needed to develop the 2L-KOGA scheme. Section 3 is entirely focused on the two-layered kernel machines, from both a theoretical and a computationally oriented viewpoint. Numerical experiments with both synthetic and real datasets are carried out in section 4. Conclusions and possible further developments are presented in section 5.

2. Background information. In subsection 2.1 we review the basics of kernel methods [13, 42], while in subsection 2.2 we briefly introduce *greedy* kernel methods [46, 50].

2.1. Kernel approximation. Let $\Omega \subset \mathbb{R}^d$, $d \geq 1, d \in \mathbb{N}$, be a nonempty set, and let us introduce a symmetric kernel function $\kappa : \Omega \times \Omega \longrightarrow \mathbb{R}$. For a given set of N scattered data $X_N = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\} \subseteq \Omega$, we define the associated kernel matrix K_N whose entries are given by $(\mathsf{K}_N)_{ij} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $i, j = 1, \ldots, N$. We remark that if the kernel matrix is positive definite for any set of pairwise distinct scattered data, then the kernel is said to be strictly positive definite.

To each strictly positive definite kernel we can associate a unique reproducing kernel Hilbert space (RKHS) $N_\kappa(\Omega)$ equipped with an inner product $\langle \cdot, \cdot \rangle$. The RKHS is also known as *native space,* and it contains functions $f : \Omega \longrightarrow \mathbb{R}$ for which κ acts as a reproducing kernel, i.e.,

- $\kappa(\cdot, \boldsymbol{x}) \in N_\kappa(\Omega) \;\; \forall \boldsymbol{x} \in \Omega$,
- $f(\boldsymbol{x}) = \langle f, \kappa(\cdot, \boldsymbol{x}) \rangle \;\; \forall \boldsymbol{x} \in \Omega, \, \forall f \in N_\kappa(\Omega)$.

Given any set of pairwise distinct interpolation points $X_N \subseteq \Omega$ and an associated set of function values $f \in N_\kappa(\Omega)$, $F_N = \{f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_N)\} = \{f_1, \ldots, f_N\} \subseteq \mathbb{R}$, the well-known kernel representer theorem [22, 41] states that there exists a unique minimum-norm interpolant $s_{X_N} \in N_\kappa(\Omega)$ of the form

$$s_{X_N}(\cdot) = \sum_{i=1}^{N} \alpha_i \kappa(\cdot, \boldsymbol{x}_i). \tag{2.1}$$

The coefficients $\{\alpha_i\}_{i=1}^N$ of the kernel-based interpolant are determined by imposing the interpolation conditions $s_{X_N}(\boldsymbol{x}_i) = f_i$ for all $i = 1, \dots, N$, thus by solving the linear system

$$\mathsf{K}_N \boldsymbol{\alpha} = \boldsymbol{f}, \tag{2.2}$$

where $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_N)^\intercal$ and $\boldsymbol{f} = (f_1, \dots, f_N)^\intercal$. If one strives for approximation instead of interpolation, a regularization parameter $\lambda > 0$ can be incorporated and the coefficients $\{\alpha_i\}_{i=1}^N$ are obtained by solving instead

$$(\mathsf{K}_N + \lambda \mathsf{I}_N)\boldsymbol{\alpha} = \boldsymbol{f}. \tag{2.3}$$

A particular class of kernels are so-called translational invariant kernels, for which there exists a function $\Phi : \mathbb{R}^d \longrightarrow \mathbb{R}$ such that the kernel can be written as

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = \Phi(\varepsilon \cdot (\boldsymbol{x} - \boldsymbol{y})),$$

whereby we already included a so-called *shape* or *length scale* parameter $\varepsilon > 0$.

The choice of the shape or length scale parameter ε from (2.4), which affects the localization of the basis functions $\Phi(\varepsilon \cdot (\boldsymbol{x} - \boldsymbol{x_i}))$ around the respective center $\boldsymbol{x_i}$, is a critical issue. Because of the influence of this hyperparameter in the reconstruction process, many optimization and searching strategies have been studied for its fine tuning [5, 15]; a review of different techniques is proposed, e.g., in [13, chapter 14].

In the following, we will generalize the concept of optimizing the hyperparameter ε, and in doing so, with abuse of notation, we will formally omit the dependence of the kernel on ε. An important subclass of translational kernels is given by radial basis function (RBF) kernels, for which there exists a univariate radial basis function $\phi : \mathbb{R}_+ \longrightarrow \Omega$, which might depend on a positive and real scale parameter ε, such that

$$(\mathsf{K}_N)_{i,j} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \Phi(\varepsilon \cdot (\boldsymbol{x}_i - \boldsymbol{x}_j)) = \phi(\varepsilon \cdot \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2) \tag{2.4}$$

with the Euclidean norm $\|\cdot\|_2$, whereby in general it is also possible to use different distance metrics from the Euclidean one.

The function $\Phi(\boldsymbol{x}) \equiv \phi(\|\boldsymbol{x}\|)$ from (2.4) allows us to characterize the native space $N_\kappa(\Omega)$ in terms of Sobolev spaces: Assume that the decay of the Fourier transform of $\Phi : \mathbb{R}^d \longrightarrow \mathbb{R}$ with $\Phi \in L^1(\mathbb{R}^d)$ can be characterized by a decay rate $\tau > d/2$ and constants $c_\Phi, C_\Phi > 0$ as

$$c_\Phi (1 + \|\boldsymbol{\omega}\|_2^2)^{-\tau} \leq \hat{\Phi}(\boldsymbol{\omega}) \leq C_\Phi (1 + \|\boldsymbol{\omega}\|_2^2)^{-\tau} \quad \forall \boldsymbol{\omega} \in \mathbb{R}^d. \tag{2.5}$$

If, additionally, the domain Ω has a Lipschitz boundary, then the native space $N_\kappa(\Omega)$ can be shown to be norm-equivalent to the Sobolev space $H^\tau(\Omega)$, i.e., $N_\kappa(\Omega) \asymp H^\tau(\Omega)$ [42, Corollary 10.48].

The interpolant s_{X_N} from (2.1) can be equivalently defined as the orthogonal projection $\Pi_{V(X_N)}$ of f onto the linear subspace $V(X_N) = \operatorname{span}\{\kappa(\cdot, \boldsymbol{x}_i), \boldsymbol{x}_i \in X_N\}$, i.e.,

$$s_{X_N} = \Pi_{V(X_N)}(f) = \sum_{i=1}^{N} c_i \kappa(\cdot, \boldsymbol{x}_i).$$

Classical pointwise error bounds for kernel-based interpolants are of the form

$$|f(\boldsymbol{x}) - s_{X_N}(\boldsymbol{x})| \leq P_{X_N}(\boldsymbol{x})\|f - s_{X_N}\|_{N_\kappa(\Omega)} \equiv P_{X_N}(\boldsymbol{x})\|r_{X_N}\|_{N_\kappa(\Omega)}, \ \ \boldsymbol{x} \in \Omega, \ \ f \in N_\kappa(\Omega), \tag{2.6}$$

where r_{X_N} denotes the residual, i.e., $r_{X_N} = f - s_{X_N}$, and P_{X_N}, known as the *power function,* is defined as

$$P_{X_N}(\boldsymbol{x}) = \|\kappa(\cdot, \boldsymbol{x}) - \Pi_{V(X_N)}(\kappa(\cdot, \boldsymbol{x}))\|_{N_\kappa(\Omega)}. \tag{2.7}$$

Other error indicators are based on the so-called fill distance, which is given by

$$h_{X_N} = h_{\Omega, X_N} = \sup_{\boldsymbol{x} \in \Omega} \left(\min_{\boldsymbol{x}_k \in X_N} \|\boldsymbol{x} - \boldsymbol{x}_k\|_2 \right) \tag{2.8}$$

and indicates how well Ω is filled out by data points. Then the pointwise error also suffices the following relation:

$$|f(\boldsymbol{x}) - s_{X_N}(\boldsymbol{x})| \leq C h_{X_N}^{\tau - d/2} ||f||_{N_\kappa(\Omega)}, \quad \boldsymbol{x} \in \Omega, \tag{2.9}$$

for $h_{X_N} \leq h_0$, where $\tau > d/2$ is the rate of the decay of $\hat{\Phi}$ from (2.5), i.e., depending on the smoothness of the kernel.

As we point out in subsection 2.2, such error estimates and in particular (2.6) can be used to define selection criteria for greedy center selection.

2.2. Greedy kernel methods. Building the kernel model on all data points $X_N \subseteq \Omega$, i.e., in the form given by (2.1), might be detrimental in some cases. For instance, if the dataset is too huge, computing the kernel matrix and solving the linear system of (2.2) is either too costly or even infeasible. Furthermore, in the case of surrogate modeling [38], one would like to deal with cheap and quickly evaluable models, thus aiming at small expansion sizes $n \ll N$. Due to its small expansion size, the greedy interpolant s_{X_n} can be understood as a sparse approximation of s_{X_N}. An established way to achieve this in the context of surrogate modeling is to select a meaningful subset $X_n \subset X_N$ of the training data X_N via greedy kernel methods [46, 50]. These are iterative schemes that start with an empty set $X_0 = \{\}$. Then for $n \geq 1$, at the nth step the set X_n is defined as $X_n = X_{n-1} \cup \{\boldsymbol{x}_n\}$ and $\boldsymbol{x}_n$ is such that

$$\boldsymbol{x}_n := \operatorname*{argmax}_{\boldsymbol{x} \in X_N \setminus X_{n-1}} \eta^{(n)}(\boldsymbol{x}),$$

using some error indicator $\eta^{(n)} : \Omega \longrightarrow \mathbb{R}$.

In the kernel literature, the following criteria are frequently used [9, 30, 39], and they make use of either the residual $r_{X_n} \equiv f - s_{X_n}$ (see (2.6)) or the power function (see (2.7)) or both:

1. P-greedy: $\eta_P^{(n)}(\boldsymbol{x}) = P_{X_n}(\boldsymbol{x})$,
2. f-greedy: $\eta_f^{(n)}(\boldsymbol{x}) = |r_{X_n}(\boldsymbol{x})|$,
3. f/P-greedy: $\eta_{f/P}^{(n)}(\boldsymbol{x}) = |r_{X_n}(\boldsymbol{x})|/P_{X_n}(\boldsymbol{x})$.

The convergence rates for the P-greedy algorithm were analyzed in [37, 44, 47]. Based on these works, the P-greedy, f-greedy, and f/P-greedy algorithms were recently unified within the scale of so-called *β-greedy algorithms* and also analyzed in terms of their convergence rates [46]. Especially target data-dependent algorithms like the f-greedy provide a faster rate of convergence, thus usually yielding more accurate

(or cheaper) models. These faster convergence rates of the f-greedy models motivate their use later on in the numerical experiments in section 4.

For practical implementation, the algorithms stop as soon as a predefined maximal expansion size is obtained or a predefined accuracy threshold $\eta^{(n)} \leq \delta$ or some stability measure is reached. An efficient implementation of these greedy kernel algorithms with the selection criteria from above was provided in a matrix free way under the notion VKOGA [38, 49]. For our approach, which will be introduced in section 3, we build on this VKOGA implementation.

3. The 2L-KOGA. As already mentioned in the introduction, in order to obtain an efficient and effective kernel model, one can

1. select suitable center points $\{\boldsymbol{x}_i\}_{i=1}^n$, for example, via greedy kernel methods as elaborated in subsection 2.2;
2. use a suitable kernel κ with tuned hyperparameters, as presented in subsection 2.1.

With regard to the first, the best known greedy selection strategies in terms of convergence rates are target data-dependent algorithms, especially the f-greedy algorithm [46].

In the following we do not want to further investigate the greedy selection strategies, but instead focus on optimal kernel design for use in conjunction with a subsequent greedy center selection as implemented in VKOGA.

Precisely, we advance in two ways. First, in subsection 3.1 we generalize the hyperparameters of (2.4) to any arbitrary linear mapping and subsequently show how this setup can be seen as a two-layered kernel machine. Second, in subsection 3.3 we introduce a machine learning inspired strategy for the optimization of the two-layered kernel machines. In particular this optimization approach is much more time efficient than cross-validating several kernel shape parameters; see also subsection 3.4.2.

Additionally, an analysis of the first layer and a convergence analysis for the greedy selection are discussed in subsection 3.2. Furthermore, in subsection 3.4 we comment on implementation details as well as on the complexity of the introduced 2L-KOGA.

3.1. Hyperparameter optimized kernels as two-layered kernel machines. As elaborated in subsection 2.1, usually only one single length scale parameter ε is used within RBF kernels. However, especially in dimensions $d \gg 1$, different directions within the data might be unequally relevant. Furthermore, as those directions do not necessarily need to be aligned with the Euclidean ones, it is advisable to also incorporate possible rotations and transformations of the input space into the kernel. This can be done by using a $b \times d$ matrix $\mathsf{A}_{\boldsymbol{\theta}}$,

$$\mathsf{A}_{\boldsymbol{\theta}} = \begin{pmatrix} \theta_1^1 & \dots & \theta_1^d \\ \vdots & \ddots & \vdots \\ \theta_b^1 & \dots & \theta_b^d \end{pmatrix},$$

and then considering the kernel

$$\kappa_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{y}) = \kappa(\mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{x}, \mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{y}). \tag{3.1}$$

For the special choice of $\mathsf{A}_{\boldsymbol{\theta}} = \varepsilon \cdot \mathsf{I}_d$, where I_d is the $d \times d$ identity matrix, we obtain the classical RBF setting, while if $\mathsf{A}_{\boldsymbol{\theta}} = \operatorname{diag}(\varepsilon_1, \dots, \varepsilon_d)$ we recover the so-called anisotropic kernels. As the $b \cdot d$ hyperparameters within the matrix $\mathsf{A}_{\boldsymbol{\theta}}$ can be

optimized, in the following we generalize the concept of anisotropic kernels by learning the optimal kernel design, i.e., the matrix $\mathsf{A}_{\boldsymbol{\theta}}$. In the following, we will mostly focus on the $d \times d$ case, i.e., $b = d$.

Now we want to point out that this hyperparameter optimized kernel can be naturally understood as a two-layered kernel according to the deep kernel representer theorem. Precisely, according to [2, eq. (10)], with slightly modified notation, a deep L-layered kernel looks like

$$\mathcal{K}^L(\boldsymbol{x}, \boldsymbol{y}) = K_L(f_{L-1} \circ \ldots \circ f_1(\boldsymbol{x}), f_{L-1} \circ \ldots \circ f_1(\boldsymbol{y})),$$

with intermediate mappings

$$f_i(\cdot) = \sum_{j=1}^{N} \alpha_j^{(i)} K_i(\cdot, f_{i-1} \circ \cdots \circ f_1(\boldsymbol{x}_j)). \tag{3.2}$$

For the special case $L = 2$ (thus two-layered), and using an RBF kernel κ as outer kernel K_L, we obtain

$$\mathcal{K}^2(\boldsymbol{x}, \boldsymbol{y}) = \kappa(f_1(\boldsymbol{x}), f_1(\boldsymbol{y})). \tag{3.3}$$

Now we use a linear kernel for the first layer mapping f_1 (3.2); more precisely, we choose a $d \times d$ matrix valued linear kernel as done in [45, section 3.2]:

$$k_1(\boldsymbol{x}, \boldsymbol{y}) := k_{\text{lin}}(\boldsymbol{x}, \boldsymbol{y}) \equiv \langle \boldsymbol{x}, \boldsymbol{y} \rangle_{\mathbb{R}^d} \cdot \mathsf{I}_d.$$

In that manner we obtain

$$f_1(\cdot) = \sum_{j=1}^{N} k_{\text{lin}}(\cdot, \boldsymbol{x}_j) \alpha_j^{(1)}. \tag{3.4}$$

Here we can leverage the following Theorem 3.1 from [45, Proposition 2], which allows us to describe all possible mappings for kernel mappings (3.4).

THEOREM 3.1. *A linear mapping, i.e., a mapping* $\mathbb{R}^d \longrightarrow \mathbb{R}^d, \boldsymbol{x} \mapsto \mathsf{A}\boldsymbol{x}$ *with* $\mathsf{A} \in \mathbb{R}^{d \times d}$ *can be realized as a kernel mapping*

$$s : \mathbb{R}^d \longrightarrow \mathbb{R}^d, \boldsymbol{x} \mapsto \sum_{i=1}^{N} \alpha_i k(\boldsymbol{x}, \boldsymbol{z}_i), \ \alpha_i \in \mathbb{R}^d,$$

with given centers $\{\boldsymbol{z}_i\}_{i=1}^N \subset \mathbb{R}^d$ *by using a matrix valued linear kernel* $k_{lin}(\boldsymbol{x}, \boldsymbol{y}) \cdot \mathsf{I}_b = \langle \boldsymbol{x}, \boldsymbol{y} \rangle_{\mathbb{R}^d} \cdot \mathsf{I}_d$*, iff the span of the center points* $\boldsymbol{z}_i$, $i = 1, \ldots, N$*, is a superset of the row space of the matrix* A.

We remark that if the center matrix $[\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N] \in \mathbb{R}^{d \times N}$ has rank d, then the span of the center points is always a superset for the row space of any matrix $\mathsf{A} \in \mathbb{R}^{d \times d}$. Then, we can formalize the following corollary.

COROLLARY 3.2. *Let* $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\} \subset \mathbb{R}^d$ *such that the data matrix* $[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N] \in \mathbb{R}^{d \times N}$ *has rank* d*. Then the kernel* $\kappa_{\boldsymbol{\theta}}$ *from* (3.1) *is an instance of a two-layered kernel according to the deep kernel representer* [2, *Theorem* 1].

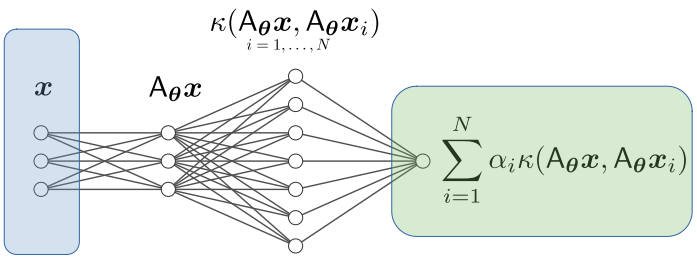

FIG. 1. *Visualization of the evaluation at $\boldsymbol{x}$ of the presented kernel machine. The input layer is highlighted in blue, while the output layer is framed in green. In this example, $b = d = 3$ and $N = 7$. (Color figure available online.)*

Proof. Using Theorem 3.1 applied to (3.4), we obtain

$$\sum_{j=1}^{N} k_{\mathrm{lin}}(\boldsymbol{x}, \boldsymbol{x}_j)\alpha_j^{(1)} = \mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{x}.$$

Thus, the general two-layered kernel from (3.3) specializes to

$$\mathcal{K}^2(\boldsymbol{x}, \boldsymbol{y}) = \kappa(f_1(\boldsymbol{x}), f_1(\boldsymbol{y})) = \kappa(\mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{x}, \mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{y}),$$

which is exactly the hyperparameter tunable kernel $\kappa_{\boldsymbol{\theta}}$ from (3.1). ∎

The evaluation of the proposed model is visualized in Figure 1, making use of common neural network layout structures: our framework includes two hidden layers, and the first one is used to map the input evaluation point $\boldsymbol{x}$ via the learned matrix $\mathsf{A}_{\boldsymbol{\theta}}$.

Again we want to emphasize that this two-layered kernel is indeed a generalization of standard shape parameter tuned kernels. If we choose $\mathsf{A}_{\boldsymbol{\theta}}$ as a scaled identity matrix, we directly reobtain the well-known shape parameter tuned kernel model because (3.1) boils down to

$$\sum_{j=1}^{N} \alpha_j^{(2)} \kappa(\mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{x}, \mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{y}) = \sum_{j=1}^{N} \alpha_j^{(2)} \kappa(\varepsilon \mathsf{I}_d \boldsymbol{x}, \varepsilon \mathsf{I}_d \boldsymbol{y}) = \sum_{j=1}^{N} \alpha_j^{(2)} \kappa(\varepsilon \boldsymbol{x}, \varepsilon \boldsymbol{y}).$$

This especially also means that the flat limits of kernels (i.e., $\varepsilon \to 0$; see, e.g., [13, Chapter 14]) can be realized with our frameworks of two-layered kernels.

The general structure of the kernel from (3.3) also suggests using other kinds of mappings to transform the base kernel κ. As our greedy approach aims at constructing a *reduced-data* model, it is heading in the same direction as well-established machine learning techniques such as, e.g., autoencoders [19], that map the data onto a feature space whose tuned dimensionality is typically smaller than the original one. However, our scheme mainly aims at reducing the cardinality of the dataset without acting *directly* on the dimensionality of the feature space. Indeed, the first layer of 2L-KOGA is employed to learn *active* spatial directions with respect to the dataset, and this may not lead to some dimensionality reduction. Moreover, while autoencoder models are often designed as deep networks composed by many parameters, and are deeply trained and tailored on the data at disposal, in 2L-KOGA the first layer is built to be an auxiliary *light* linear model, which may enhance the performance of the subsequent greedy algorithm. Therefore, the numerical experiments presented in

section 4 aim at comparing the 2L-KOGA with standard (greedy) kernel approaches, rather than with general dimensionality reduction techniques.

Sticking to light linear mappings as transformations within the kernel, the consideration of two layers is enough: The use of several linear mappings can always be reduced to a single layer, as the product of matrices $A_1 \cdot \ldots \cdot A_L$ is simply another matrix.

3.2. Theoretical analysis. In this subsection we first of all want to analyze and interpret the meaning of the first kernel layer, allowing us to incorporate linear transformations of the input. Subsequently we provide some preliminary convergence rate analysis for the subsequent greedy selection.

3.2.1. Analysis of the first layer. In order to understand the impact of the first kernel layer on the performance of the overall kernel machine, we analyze the matrix $\mathsf{A}_{\boldsymbol{\theta}}$ via its singular value decomposition given by

$$\mathsf{A}_{\boldsymbol{\theta}} = \mathsf{U}\Sigma\mathsf{V}^\intercal$$

with orthogonal matrices $\mathsf{U}, \mathsf{V} \in \mathbb{R}^{d\times d}$ respectively consisting of the left and right singular vectors and the diagonal matrix $\Sigma \in \mathbb{R}^{d\times d}$ with the nonnegative singular values on the diagonal. Denoting the columns of U, V by, respectively, $\boldsymbol{u}_i$ and $\boldsymbol{v}_i, i = 1, \ldots, d$, we have

$$\mathsf{A}_{\boldsymbol{\theta}}\mathsf{V} = \mathsf{U}\Sigma \quad \Leftrightarrow \quad \mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{v}_i = \sigma_i \boldsymbol{u}_i \quad \forall i = 1, \ldots, d. \tag{3.5}$$

For a given input $\boldsymbol{x} \in \mathbb{R}^d$, which can be decomposed as $\boldsymbol{x} = \sum_{i=1}^d \langle \boldsymbol{x}, \boldsymbol{v}_i\rangle_{\mathbb{R}^d} \boldsymbol{v}_i$, we obtain

$$\begin{aligned}
\mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{x} &= \sum_{i=1}^d \langle \boldsymbol{x}, \boldsymbol{v}_i\rangle_{\mathbb{R}^d} \mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{v}_i = \sum_{i=1}^d \langle \boldsymbol{x}, \boldsymbol{v}_i\rangle_{\mathbb{R}^d} \sigma_i \boldsymbol{u}_i \\
\Rightarrow \mathsf{A}_{\boldsymbol{\theta}}(\boldsymbol{x} - \tilde{\boldsymbol{x}}) &= \sum_{i=1}^d (\langle \boldsymbol{x}, \boldsymbol{v}_i\rangle_{\mathbb{R}^d} - \langle \tilde{\boldsymbol{x}}, \boldsymbol{v}_i\rangle_{\mathbb{R}^d}) \sigma_i \boldsymbol{u}_i \\
\Rightarrow \|\mathsf{A}_{\boldsymbol{\theta}}(\boldsymbol{x} - \tilde{\boldsymbol{x}})\|^2_{\mathbb{R}^d} &= \sum_{i=1}^d (\langle \boldsymbol{x}, \boldsymbol{v}_i\rangle_{\mathbb{R}^d} - \langle \tilde{\boldsymbol{x}}, \boldsymbol{v}_i\rangle_{\mathbb{R}^d})^2 \sigma_i^2. \qquad (3.6)
\end{aligned}$$

The last line, i.e., (3.6), is of importance, as the RBF kernel κ from (3.1) in the second layer of the kernel only requires distances as an input. Therefore we can see that the matrix $\mathsf{U} \in \mathbb{R}^{d\times d}$ does not matter at all for the 2L-KOGA model, because it is not seen due to the radiality of the RBF kernel. In particular it would be possible to set $\mathsf{U} := \mathsf{V}$, such that the matrix $\mathsf{A}_{\boldsymbol{\theta}}$ is even symmetric. However enforcing symmetry of the matrix $\mathsf{A}_{\boldsymbol{\theta}}$ during the optimization step indeed impedes the performance of the optimization; in particular, the resulting matrix $\mathsf{A}_{\boldsymbol{\theta}}$ is frequently not as good as when one uses a nonsymmetric optimization.

From (3.6) we see that the distance is scaled along the directions provided by the right singular vectors of $\mathsf{A}_{\boldsymbol{\theta}}$ by the corresponding singular value $\sigma_i \geq 0$. In particular, if a singular value σ_i is very small or even zero, this means that data along the directions of the corresponding right singular vectors $\boldsymbol{v}_i$ is squeezed or even mapped to the same point.

3.2.2. Convergence analysis. The convergence analysis for (greedy) kernel interpolation is usually done either in the number of (greedily) selected points [37, 46] or in terms of the fill distance [42, Chap. 11], which was defined in (2.8). Both of those approaches focus on the asymptotic rate of the decay of the error.

Given a two-layered kernel $\kappa_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{y}) = \kappa(\mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{x}, \mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{x})$, we can derive convergence rates by considering this kernel as a standard radial basis function kernel acting on the transformed domain $\mathsf{A}_{\boldsymbol{\theta}}\Omega \subset \mathbb{R}^{\operatorname{rank}(\mathsf{A})}$. We will distinguish the two cases $\operatorname{rank}(\mathsf{A}_{\boldsymbol{\theta}}) = d$ and $\operatorname{rank}(\mathsf{A}_{\boldsymbol{\theta}}) < d$. For the first case, Theorem 3.3 shows that we obtain at least the same rate of convergence as when one uses the standard kernel κ instead of the two-layered kernel $\kappa_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{y})$. For the second case, Theorem 3.4 shows that the convergence rate is indeed increased, depending on the number of singular values of the matrix $\mathsf{A}_{\boldsymbol{\theta}}$ equal to zero.

THEOREM 3.3. *Consider an RBF kernel κ that satisfies (2.5) with $\tau > d/2$ on a bounded Lipschitz domain $\Omega \subset \mathbb{R}^d$. Consider $f \in N_\kappa(\Omega)$ and the kernel interpolant s_{X_N} using the two-layered kernel $\kappa_{\boldsymbol{\theta}}$ with $\operatorname{rank}(\mathsf{A}_{\boldsymbol{\theta}}) = d$.*

Then the following pointwise error estimate holds:

$$|f(\boldsymbol{x}) - s_{X_N}(\boldsymbol{x})| \leq C h_{X_N}^{\tau - d/2}.$$

For asymptotically equidistributed points $h_{X_N} \asymp N^{-1/d}$ it then holds that

$$|f(\boldsymbol{x}) - s_{X_N}(\boldsymbol{x})| \leq C N^{1/2 - \tau/d}.$$

Proof. We consider the two-layered kernel $\kappa_{\boldsymbol{\theta}}$ as a standard RBF kernel κ applied to the transformed data $\mathsf{A}_{\boldsymbol{\theta}} X_N$ from the transformed domain $\mathsf{A}_{\boldsymbol{\theta}}\Omega$. As it holds that $\operatorname{rank}(\mathsf{A}_{\boldsymbol{\theta}}) = d$, we have $\dim(\mathsf{A}_{\boldsymbol{\theta}}\Omega) = \dim(\Omega) = d$. By standard Sobolev arguments (as $x \mapsto \mathsf{A}_{\boldsymbol{\theta}} x$ is just a linear full rank transformation), we have $H^\tau(\Omega) \asymp H^\tau(\mathsf{A}_{\boldsymbol{\theta}}\Omega)$, in particular $f \circ A_{\boldsymbol{\theta}}^{-1} \in H^\tau(\mathsf{A}_{\boldsymbol{\theta}}\Omega)$. Furthermore, due to $\operatorname{rank}(\mathsf{A}_{\boldsymbol{\theta}}) = d$, all the singular values of $\mathsf{A}_{\boldsymbol{\theta}}$ are positive, i.e., it holds that $s_{\min}(\mathsf{A}_{\boldsymbol{\theta}})\|x\|_2 \leq \|\mathsf{A}_{\boldsymbol{\theta}} x\|_2 \leq s_{\max}(\mathsf{A}_{\boldsymbol{\theta}})\|x\|_2$ or in short $\|\cdot\|_2 \asymp \|\mathsf{A}_{\boldsymbol{\theta}} \cdot\|_2$. Therewith we obtain for the respective fill distances

$$\begin{aligned} h_{\mathsf{A}_{\boldsymbol{\theta}}\Omega, \mathsf{A}_{\boldsymbol{\theta}} X_N} &\equiv \sup_{\tilde{\boldsymbol{x}} \in \mathsf{A}_{\boldsymbol{\theta}}\Omega} \left(\min_{\tilde{\boldsymbol{x}}_k \in \mathsf{A}_{\boldsymbol{\theta}} X_N} \|\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{x}}_k\|_2 \right) \\ &= \sup_{\boldsymbol{x} \in \Omega} \left(\min_{\boldsymbol{x}_k \in X_N} \|\mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{x} - \mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{x}_k\|_2 \right) \\ &\asymp \sup_{\boldsymbol{x} \in \Omega} \left(\min_{\boldsymbol{x}_k \in X_N} \|\boldsymbol{x} - \boldsymbol{x}_k\|_2 \right) = h_{\Omega, X_N}. \end{aligned}$$

Hence we can make use of the error bound of (2.9) to derive the final statement. ∎

From the previous proof it is obvious to see that we can obtain a faster rate of convergence, as soon as the fill distance decays faster. An asymptotically faster decay of the fill distance is only possible if the dimension of the underlying input domain is effectively reduced. This is the case iff $\operatorname{rank}(\mathsf{A}_{\boldsymbol{\theta}}) < d$, because then $\dim(\mathsf{A}_{\boldsymbol{\theta}}\Omega) < \dim(\Omega)$.

In order to avoid technical discussion on RBFs and the corresponding Fourier transforms, we focus for the following theorem on the class of Matérn kernels, which are also used throughout the numerical experiments of section 4. In their general form they are given as [12, section 4.4]

$$\Phi(\boldsymbol{x}) = \frac{K_{\tau - d/2}(\|\boldsymbol{x}\|)\|\boldsymbol{x}\|^{\tau - d/2}}{2^{\tau - 1}\Gamma(\tau)}, \quad \tau > d/2, \tag{3.7}$$

whereby K_ν is the modified Bessel function of second order. The corresponding Fourier transform is given as

$$\hat{\Phi}(\boldsymbol{\omega}) = (1 + \|\boldsymbol{\omega}\|^2)^{-\tau},$$

i.e., as in (2.5) but with $c_\Phi = C_\Phi = 1$.

THEOREM 3.4. *Consider a Matérn kernel κ of (3.7) with $\tau > d/2$ on a bounded Lipschitz domain $\Omega \subset \mathbb{R}^d$. Consider $f \in N_\kappa(\Omega)$ and the kernel interpolant s_{X_N} using the two-layered kernel $\kappa_{\boldsymbol{\theta}}$ such that* $\operatorname{rank}(\mathsf{A}_{\boldsymbol{\theta}}) =: d_{\mathit{eff}} < d$.

Assume that $f \in N_\kappa(\Omega)$ is invariant along the subspace $\operatorname{Null}(\mathsf{A}_{\boldsymbol{\theta}}) \subset \mathbb{R}^d$, *i.e., $f(\boldsymbol{x}) = f(\boldsymbol{x}')$ for any $\boldsymbol{x}, \boldsymbol{x}' \in \Omega$ with $\boldsymbol{x} - \boldsymbol{x}' \in \operatorname{Null}(A_{\boldsymbol{\theta}})$. For points $X_n \subset \Omega$ such that $h_{\mathsf{A}_{\boldsymbol{\theta}}\Omega, \mathsf{A}_{\boldsymbol{\theta}} X_n} \asymp n^{-1/d_{\mathit{eff}}}$ it then holds that*

$$|f(\boldsymbol{x}) - s_{X_n}(\boldsymbol{x})| \le C n^{\frac{d}{2d_{\mathit{eff}}} - \frac{\tau}{d_{\mathit{eff}}}}.$$

Note that it holds that $\frac{d}{2d_{\text{eff}}} - \frac{\tau}{d_{\text{eff}}} < \frac{1}{2} - \frac{\tau}{d} < 0$ due to $\tau > d/2$, i.e., the decay rate is faster than in Theorem 3.3:

$$\frac{d}{2d_{\text{eff}}} - \frac{\tau}{d_{\text{eff}}} - \left(\frac{1}{2} - \frac{\tau}{d}\right) = \frac{d(d - d_{\text{eff}}) - 2\tau(d - d_{\text{eff}})}{2dd_{\text{eff}}} = \frac{(d - 2\tau)(d - d_{\text{eff}})}{2dd_{\text{eff}}} < 0.$$

Proof. Define $\mathcal{N} := \operatorname{Null}(\mathsf{A}_{\boldsymbol{\theta}})$ and consider the orthogonal projector $\Pi_{\mathcal{N}^\perp} : \mathbb{R}^d \longrightarrow \mathcal{N}^\perp$. The mapping $\mathsf{A}_{\boldsymbol{\theta}}^{\mathcal{N}^\perp} : \mathcal{N}^\perp \longrightarrow \mathrm{R}(\mathsf{A}_{\boldsymbol{\theta}}), \boldsymbol{x} \mapsto \mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{x}$ is now full rank and thus invertible. Consider $\boldsymbol{x} \in \Omega$ and decompose $\boldsymbol{x} = \boldsymbol{x}_\parallel + \boldsymbol{x}_\perp$ with $\boldsymbol{x}_\parallel \in \mathcal{N}, \boldsymbol{x}_\perp \in \mathcal{N}^\perp$. Using the invariance assumption on f along $\mathcal{N}$, we have

$$\begin{aligned}
|(f - s_{X_n})(\boldsymbol{x})| &= \left| f(\boldsymbol{x}_\parallel + \boldsymbol{x}_\perp) - \sum_{j=1}^n \alpha_j^{(n)} \kappa(\mathsf{A}_{\boldsymbol{\theta}}(\boldsymbol{x}_\parallel + \boldsymbol{x}_\perp), \mathsf{A}_{\boldsymbol{\theta}}(\boldsymbol{x}_{j,\parallel} + \boldsymbol{x}_{j,\perp})) \right| \\
&= \left| (f(\boldsymbol{x}_\perp) - \sum_{j=1}^n \alpha_j^{(n)} \kappa(\mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{x}_\perp, \mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{x}_{j,\perp}) \right| \\
&= \left| (f \circ (\mathsf{A}_{\boldsymbol{\theta}}^{\mathcal{N}^\perp})^{-1})(\mathsf{A}_{\boldsymbol{\theta}}^{\mathcal{N}^\perp}\boldsymbol{x}_\perp) - \sum_{j=1}^n \alpha_j^{(n)} \kappa(\mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{x}_\perp, \mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{x}_{j,\perp}) \right|. \qquad (3.8)
\end{aligned}$$

As $\dim(\mathsf{A}_{\boldsymbol{\theta}}\Omega) \equiv d_{\text{eff}}$, the native space $N_\kappa(\mathsf{A}_{\boldsymbol{\theta}}\Omega)$ is now norm-equivalent to the Sobolev space $H^{\tau'}(\mathsf{A}_{\boldsymbol{\theta}}\Omega)$ of smaller smoothness $\tau' = \tau - \frac{d - d_{\text{eff}}}{2} < \tau$; see (3.7). Therefore we obtain $f \circ (\mathsf{A}_{\boldsymbol{\theta}}^{\mathcal{N}^\perp})^{-1} \in H^{\tau'}(\mathsf{A}_{\boldsymbol{\theta}}\Omega) \asymp N_\kappa(\mathsf{A}_{\boldsymbol{\theta}}\Omega)$.

Furthermore, $\sum_{j=1}^n \alpha_j^{(n)} \kappa(\cdot, \mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{x}_{j,\perp}) \in H^{\tau'}(\mathsf{A}_{\boldsymbol{\theta}}\Omega)$ and due to the kernel interpolation condition it holds that

$$\sum_{j=1}^n \alpha_j^{(n)} \kappa(\boldsymbol{y}_i, \mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{x}_{j,\perp}) = (f \circ (\mathsf{A}_{\boldsymbol{\theta}}^{\mathcal{N}^\perp})^{-1})(\boldsymbol{y}_i)$$

for all $\boldsymbol{y}_i \in \{\mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{x}_i \,|\, \boldsymbol{x}_i \in X_n\}$. Therefore we can leverage (2.9) to bound the error as

$$\left| (f \circ (\mathsf{A}_{\boldsymbol{\theta}}^{\mathcal{N}^\perp})^{-1})(\boldsymbol{y}) - \sum_{j=1}^n \alpha_j^{(n)} \kappa(\boldsymbol{y}, \mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{x}_{j,\perp}) \right| < C h_{\mathsf{A}_{\boldsymbol{\theta}}\Omega, \mathsf{A}_{\boldsymbol{\theta}} X_n}^{\tau' - d_{\text{eff}}/2}, \qquad \boldsymbol{y} \in \{\mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{x} \,|\, \boldsymbol{x} \in \Omega\}$$

$$\Leftrightarrow \quad \left| (f \circ (\mathsf{A}_{\boldsymbol{\theta}}^{\mathcal{N}^\perp})^{-1})(\mathsf{A}_{\boldsymbol{\theta}}^{\mathcal{N}^\perp}\boldsymbol{x}_\perp) - \sum_{j=1}^n \alpha_j^{(n)} \kappa(\mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{x}_\perp, \mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{x}_{j,\perp}) \right| < C h_{\mathsf{A}_{\boldsymbol{\theta}}\Omega, \mathsf{A}_{\boldsymbol{\theta}} X_n}^{\tau' - d_{\text{eff}}/2}, \qquad \boldsymbol{x} \in \Omega,$$

such that in conjunction with (3.8) we obtain

$$|(f - s_{X_n})(\boldsymbol{x})| \leq C h_{\mathsf{A}_{\boldsymbol{\theta}}\Omega, \mathsf{A}_{\boldsymbol{\theta}} X_n}^{\tau' - d_{\text{eff}}/2}, \qquad \boldsymbol{x} \in \Omega.$$

Finally using $h_{\mathsf{A}_{\boldsymbol{\theta}}\Omega, \mathsf{A}_{\boldsymbol{\theta}} X_n} \asymp n^{-1/d_{\text{eff}}}$ which is possible due to $\dim(\mathsf{A}_{\boldsymbol{\theta}}\Omega) \equiv d_{\text{eff}}$ we obtain the desired statement:

$$-\frac{1}{d_{\text{eff}}} \cdot \left(\tau' - \frac{d_{\text{eff}}}{2}\right) = -\frac{\tau - \frac{d - d_{\text{eff}}}{2}}{d_{\text{eff}}} + \frac{1}{2} = -\frac{\tau}{d_{\text{eff}}} + \frac{d}{2d_{\text{eff}}}. \qquad \square$$

Theorem 3.4 can be leveraged in the following way. Given $f \in N_\kappa(\Omega)$, which is invariant in some directions, the two-layered kernel $\kappa_{\boldsymbol{\theta}}$ can be chosen such that the null space $\text{Null}(\mathsf{A}_{\boldsymbol{\theta}})$ of the matrix $\mathsf{A}_{\boldsymbol{\theta}}$ coincides with this invariant subspace. Then, as $\text{rank}(\mathsf{A}_{\boldsymbol{\theta}}) \equiv d_{\text{eff}} < d$, the fill distance $h_{\mathsf{A}_{\boldsymbol{\theta}}\Omega, \mathsf{A}_{\boldsymbol{\theta}} X_n}$ can decay as $h_{\mathsf{A}_{\boldsymbol{\theta}}\Omega, \mathsf{A}_{\boldsymbol{\theta}} X_n} \asymp n^{-1/d_{\text{eff}}}$ for suitable chosen points X_n (namely, such that $\mathsf{A}_{\boldsymbol{\theta}} X_n$ is asymptotically equidistant within $\mathsf{A}_{\boldsymbol{\theta}}\Omega$). As remarked above, this provides a faster convergence rate, thus providing a benefit of using the two-layered kernel. Section 4 shows that our used optimization approach is indeed capable of (approximately) finding those invariant directions, i.e., inactive subspaces of the considered function $f \in N_\kappa(\Omega)$.

In order to avoid too many technical details, we do not show convergence results for the f-greedy algorithm using the two-layered kernel $\kappa_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{y})$, albeit we use it later on in section 4. However, we remark that roughly speaking the convergence analysis of the f-greedy algorithm from [46] is based on convergence rates for the P-greedy algorithm, i.e., those rates from Theorems 3.3 and 3.4. Its analysis shows an additional convergence rate of $\log(N)N^{-1/2}$ for the f-greedy algorithm by making use of its target data-dependent selection criterion. Therefore we expect that the use of the f-greedy algorithm within Theorem 3.3, respectively, Theorem 3.4 provides faster convergence rates, namely, by the additional factor of $\log(N)N^{-1/2}$.

As can also be seen in the numerical experiments in section 4, we want to point out that our 2L-KOGA approach does not only necessarily provide benefits in terms of the asymptotic convergence rate, but also in terms of improvements for finite datasets. This can be frequently observed if singular values of the final optimized matrix $\mathsf{A}_{\boldsymbol{\theta}}$ are not exactly zero, but close to zero.

3.3. Loss function for optimization. Learning the kernel $\kappa_{\boldsymbol{\theta}}$ is equivalent to *optimizing* the matrix $\mathsf{A}_{\boldsymbol{\theta}}$; thus $\kappa_{\boldsymbol{\theta}}$ is a parametric model that depends on $b \times d$ parameters belonging to some space $\Theta \subseteq \mathbb{R}^{b \times d}$. We aim at optimizing the kernel by minimizing a loss function that depends on the input and target data:

$$\min_{\mathsf{A}_{\boldsymbol{\theta}} \in \Theta} \ell(X_N, F_N, \mathsf{A}_{\boldsymbol{\theta}}). \tag{3.9}$$

For the actual optimization we will make use of well-known optimization strategies from the machine learning community [19], in particular gradient based optimization and the use of mini-batches; more details on the optimization and implementation will be given in subsection 3.4. In the following, we will focus on the structure of the loss function and on its use in conjunction with the optimization via mini-batches.

The loss consists in the cross-validation (CV) error, which is an established criterion in the scientific community to assess the effectiveness of a model [18]. However, instead of directly evaluating the CV error on X_N with respect to F_N using the kernel $\kappa_{\boldsymbol{\theta}}$, we evaluate the k-fold cross-validation error on so-called *mini-batches.* This is necessary and even beneficial, as the evaluation of k-fold cross-validation scores is time

consuming on large datasets and full batch learning is known to be detrimental for generalization in machine learning tasks [19]. Mini-batches are randomly drawn subsets $(X_{\text{batch}}, F_{\text{batch}})$ of input data with corresponding target values of size $n_{\text{batch}} \ll N$ from the large dataset X_N with corresponding target data F_N. A commonly used value, which is later employed in the numerical experiments, is $n_{\text{batch}} = 64$.

As mentioned in the introduction, this strategy is related to the optimization approaches in [20, 32], where mini-batches are also used. However there the optimization criterion was based on the premise that a kernel must be good "if the number of points used to interpolate the data can be halved without significant loss in accuracy"; see [20, eq. (6)]. In contrast, we use for every mini-batch the k-fold CV error by taking advantage of an efficient implementation proposed by Rippa for the case $k = n_{\text{batch}}$ (leave-one-out CV (LOOCV)) [35] and then extended to the general $1 < k < n_{\text{batch}}$ framework in [26]. In the following, we briefly outline this strategy, which we refer to as extended Rippa's algorithm (ERA).

Let $k \in \mathbb{N}$, $1 < k \leq n_{\text{batch}}$, be the number of folds used for the k-fold validation scheme, and suppose, for simplicity, that $p = n_{\text{batch}}/k \in \mathbb{N}$. Then, for each fold $j = 1, \ldots, p$, let us split the mini-batch X_{batch} into a training set $T_{n_{\text{batch}}-p}$ of cardinality $n_{\text{batch}} - p$ and a validation set V_p, so that $X_{n_{\text{batch}}} = T_{n_{\text{batch}}-p} \cup V_p$ and $T_{n_{\text{batch}}-p} \cap V_p = \emptyset$. Let us denote by $\boldsymbol{r} = \boldsymbol{r}^{(j)} = (r_1, \ldots, r_p)^\intercal$, $r_i \in \{1, \ldots, n_{\text{batch}}\}$, the vector of distinct validation indices for a given fold, i.e., $V_p = \{\boldsymbol{x}_{r_i}, i = 1, \ldots, p\}$. Then, we are interested in computing the residual vector $\boldsymbol{e_r} = \boldsymbol{e}_{\boldsymbol{r}^{(j)}} = (e_1, \ldots, e_p)$ whose components are $e_i = s_{T_{n_{\text{batch}}-p}}(\boldsymbol{x}_{r_i}) - f(\boldsymbol{x}_{r_i})$, $\boldsymbol{x}_{r_i} \in V_p$, with $s_{T_{n_{\text{batch}}-p}}$ the interpolant constructed upon the training set $T_{n_{\text{batch}}-p}$. A standard application of the k-fold CV scheme would require the inversion of k different $(n_{\text{batch}} - p) \times (n_{\text{batch}} - p)$ linear systems of the form (2.2), leading to a complexity cost of about $\mathcal{O}(n_{\text{batch}}^3 k)$. Fortunately, letting

$$\mathsf{K}^{\boldsymbol{\theta}}_{n_{\text{batch}}} = \begin{pmatrix} \kappa_{\boldsymbol{\theta}}(\boldsymbol{x}_1, \boldsymbol{x}_1) & \ldots & \kappa_{\boldsymbol{\theta}}(\boldsymbol{x}_1, \boldsymbol{x}_{n_{\text{batch}}}) \\ \vdots & \ddots & \vdots \\ \kappa_{\boldsymbol{\theta}}(\boldsymbol{x}_{n_{\text{batch}}}, \boldsymbol{x}_1) & \ldots & \kappa_{\boldsymbol{\theta}}(\boldsymbol{x}_{n_{\text{batch}}}, \boldsymbol{x}_{n_{\text{batch}}}) \end{pmatrix},$$

in [26], the author proved that $\boldsymbol{e_r}$ is the unique solution of the linear system

$$(\mathsf{K}^{\boldsymbol{\theta}}_{n_{\text{batch}}})^{-1}_{V_p} \boldsymbol{e_r} = \boldsymbol{c_r}, \tag{3.10}$$

where $(\mathsf{K}^{\boldsymbol{\theta}}_{n_{\text{batch}}})^{-1}_{V_p}$ is the submatrix of the $n_{\text{batch}} \times n_{\text{batch}}$ inverse kernel matrix $(\mathsf{K}^{\boldsymbol{\theta}}_{n_{\text{batch}}})^{-1}$, built on $\kappa_{\boldsymbol{\theta}}$, which is defined by restricting to the validation indices, and $\boldsymbol{c_r}$ are the components of the solution vector from (2.2) for which $i \in \boldsymbol{r}$. Hence, such an implementation requires a total complexity cost which is about $\mathcal{O}(n_{\text{batch}}^3) + \mathcal{O}(n_{\text{batch}}^3/k^2)$. Finally, we point out that any norm of the vector $\boldsymbol{e_r}$ can be used to have an a priori error estimate, because Rippa's scheme is independent of the used norm in the end. In the following we will consider the squared two-norm, and we will apply Tikhonov regularization with a parameter λ to (3.10).

Thus finally we are optimizing the kernel $\kappa_{\boldsymbol{\theta}}$ by minimizing the following loss function from (3.9):

$$\ell_{\lambda,p}(X_N, F_N, \mathsf{A}_{\boldsymbol{\theta}}) := \sum_{j=1}^{p} \|\boldsymbol{e}_{\boldsymbol{r}^{(j)}}\|_2^2. \tag{3.11}$$

The ERA may be further sped up by allowing a stochastic approximation, which, however, affects the exactness of the scheme [25]. While ERA, as Rippa's scheme,

Algorithm 3.1. SGD optimization of the kernel $\kappa_{\boldsymbol{\theta}} \equiv \kappa(\mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{x}, \mathsf{A}_{\boldsymbol{\theta}}\boldsymbol{y})$.

Input: Data (X_N, F_N), base kernel κ, k-fold parameter k, learning rate μ, regularization parameter λ
Output: Optimized matrix $\mathsf{A}_{\boldsymbol{\theta}}$

$\mathsf{A}_{\boldsymbol{\theta}} \leftarrow \mathrm{diag}(1, \dots, 1)$ `/* Initialization of` $\mathsf{A}_{\boldsymbol{\theta}}$ `*/`
for $n_{\text{epoch}} = 1, \dots, \max_{\text{epoch}}$ **do**
  $L_{\text{epoch}} \leftarrow 0$
  SHUFFLE (X_N, F_N)
  for $n_{\text{epoch}} = 1, \dots, \max_{\text{epoch}}$ **do**
    $(X_{\text{batch}}, F_{\text{batch}}) \leftarrow$ GET_BATCH$((X_N, F_N))$
    $L = \ell_\lambda(X_{\text{batch}}, F_{\text{batch}}, \mathsf{A}_{\boldsymbol{\theta}}, k)$ `/* Using (3.11) */`
    $\mathsf{A}_{\boldsymbol{\theta}} \leftarrow \mathsf{A}_{\boldsymbol{\theta}} - \mu \cdot \frac{\partial L}{\partial \mathsf{A}_{\boldsymbol{\theta}}}$ `/* Gradient descent update */`
    $L_{\text{epoch}} \leftarrow L_{\text{epoch}} + L$
  end for
  EARLY_STOPPING(L_{epoch})
end for

was originally designed for the tuning of the shape parameter, we employ it as a more general *error indicator* as recently done in [4, 27]. This allows us to find a good two-layered kernel, which can be used for subsequent kernel approximation tasks. In the next subsection we provide more details on the optimization and implementation.

3.4. Optimization, implementation, and computational complexity. This section is devoted to the optimization and implementation of the two-layered kernel and the investigation of its computational complexity.

3.4.1. Optimization and implementation. In subsection 3.3 we elaborated on the (family of) loss functions which we employ for the optimization of the kernel $\kappa_{\boldsymbol{\theta}}$. Here we give some more details on the actual optimization procedure. Based on the computed loss values, we employ an iterative gradient based optimization using the adaptive *Adam optimizer* [23] (instead of plain stochastic gradient descent (SGD)) and *early stopping* [19] on the accumulated loss values during one epoch. The implementation is done in Python leveraging the deep learning framework `pytorch` [33] and especially making use of the autodifferentiation for computing the gradients. The code (with its extension to the vectorial case [43]) is implemented as an extension of the VKOGA software package [38] and can be found at https://gitlab.mathematik.uni-stuttgart.de/pub/ians-anm/2L-VKOGA.

The algorithm used for the optimization of the two-layered kernel is depicted in Algorithm 3.1 for the case of plain SGD. For the use of the Adam optimizer, the weights are more sophistically updated and we do not delve into details concerning this. The crucial step of the optimization, namely, the computation of the gradients of the loss with respect to the matrix $\mathsf{A}_{\boldsymbol{\theta}}$, is conveniently handled by PyTorch via automatic differentiation. To improve the numerical stability of the algorithm, a Tikhonov regularization is added to the kernel matrix as elaborated in subsection 3.3. The early stopping criterion in line 11 stops the optimization if the loss does not decay further, thus avoiding unnecessary further optimization steps.

3.4.2. Computational complexity. In this section, we briefly analyze the complexity of our proposed 2L-KOGA approach against a standard cross-validation

approach. Furthermore we comment on the speed up of using Rippa's and extended Rippa's scheme as described in subsection 3.3 for the kernel optimization:

As our 2L-KOGA approach is based on a gradient descent optimization using up to $\max_{\text{epoch}}$ epochs with each N/n_{batch} iterations using batches of size n_{batch}, it requires

$$\mathcal{O}(n_{\text{epoch}} \cdot N/n_{\text{batch}} \cdot n_{\text{batch}}^3) = \mathcal{O}(n_{\text{epoch}} \cdot N \cdot n_{\text{batch}}^2)$$

operations for the calculation of the CV errors on the small matrices. The subsequent run of VKOGA up to an expansion size of n_{vkoga} is typically of order $\mathcal{O}(n_{\text{vkoga}}^2 \cdot N)$; thus the overall complexity of the 2L-KOGA is

$$\mathcal{O}(n_{\text{epoch}} \cdot N \cdot n_{\text{batch}}^2) + \mathcal{O}(n_{\text{vkoga}}^2 \cdot N).$$

For typical values of $n_{\text{batch}}, n_{\text{epoch}}, n_{\text{vkoga}}$ such as $n_{\text{batch}} = 64, n_{\text{epoch}} = 10, n_{\text{vkoga}} = 1000$, which were used for the numerical experiments in section 4, it holds that $n_{\text{epoch}} \cdot N \cdot n_{\text{batch}}^2 < n_{\text{vkoga}}^2 \cdot N$, i.e., there is only a small computational overhead for the kernel optimization before running VKOGA.

This is in contrast to a straightforward shape parameter cross-validation using VKOGA. Indeed, full cross-validation for the matrix $\mathsf{A}_{\boldsymbol{\theta}}$ is infeasible, because validating n_{CV} many values for d^2 many matrix entries requires an effort of $n_{\text{CV}}^{d^2}$, which is exponential in the number of parameters (curse of dimensionality). Also, a full cross-validation for a classical anisotropic kernel, i.e., just for the d diagonal entries of $\mathsf{A}_{\boldsymbol{\theta}}$, is infeasible, as this requires n_{CV}^d runs. Thus, a cross-validation of n_{CV} many shape parameters takes the effort

$$\mathcal{O}(n_{\text{CV}} \cdot n_{\text{vkoga}}^2 \cdot N),$$

which elucidates the additional factor n_{CV}. Hence we conclude that our approach is favorable and cheaper than the classical CV implementation, in particular for large N.

Especially for large space dimension d only a cross-validated shape parameter is likely inferior to a whole optimized matrix (in terms of the explored parameter space $\Theta \subset \mathbb{R}^{d \times d}$).

4. Numerical experiments. In this section we provide three different kinds of numerical experiments. First, in subsection 4.1 we use our two-layered approach for the efficient approximation of given functions. Second, in subsection 4.2 we show the applicability of two-layered kernels for sparse surrogate modeling on real world machine learning datasets. Finally, in subsection 4.3 we investigate the optimization of the first layer, i.e., the matrix $\mathsf{A}_{\boldsymbol{\theta}}$, with the help of the extended Rippa's formula. The code to rerun the numerical experiments can be found at https://gitlab.mathematik.uni-stuttgart.de/pub/ians-anm/paper-2023-data-driven-kernel-designs.

4.1. Function approximation on the unit cube. As a first class of example we highlight the benefits of the proposed 2L-KOGA for function approximation. For this, we picked functions where different behaviors can be seen. We consider the unit cube $\Omega = [0,1]^d$ for $d = 5, 6, 7$, discretized with $N = 50000$ uniformly randomly selected points. The corresponding target data are given as the evaluation of the function

$$f_d(\boldsymbol{x}) = \begin{cases} \mathrm{e}^{-4\left(\sum_{j=1}^5 \boldsymbol{x}_i - 0.5\right)^2} & \text{for } d = 5, \\ \mathrm{e}^{-4\sum_{j=1}^6 (\boldsymbol{x}_i - 0.5)^2} + 2|\boldsymbol{x}_1 - 0.5| & \text{for } d = 6, \\ \mathrm{e}^{-\sum_{j=1}^7 (\boldsymbol{x}_i - 0.5)^2} + \mathrm{e}^{-9\sum_{j=1}^2 (\boldsymbol{x}_i - 0.3)^2} & \text{for } d = 7. \end{cases}$$

TABLE 1

Optimization and VKOGA runtime for f_5, f_6, and f_7. The optimization of the two-layered kernel takes approximately as much time as the runtime of VKOGA. The optimized two-layered kernel is several orders of magnitude more accurate, in terms of mean squared error (MSE), than a standard kernel model when using the same expansion size.

	f_5	f_6	f_7
Kernel optimization	8.936s	9.173s	9.817s
Average VKOGA runtime	8.087s	8.329s	8.787s
2L-KOGA kernel MSE	$4.351 \cdot 10^{-8}$	$5.553 \cdot 10^{-4}$	$2.730 \cdot 10^{-3}$
Standard kernel MSE	$9.009 \cdot 10^{-3}$	$4.174 \cdot 10^{-3}$	$6.261 \cdot 10^{-3}$

While the first function f_5 clearly possesses an active subspace along the direction $(1,1,1,1,1)^\intercal \in \mathbb{R}^5$, this does not hold for f_6 and f_7. However, for f_6 the kink exists only in the x_1 direction, while for f_7 the second bump depends only on the first two variables.

For the approximation we use as a base kernel the Matérn kernel $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\|\boldsymbol{x} - \boldsymbol{y}\|/\sqrt{d})$, but we remark that the results are qualitatively the same when using other Matérn kernels.

We compare the approximation given by f-greedy (up to 250 centers) of the 2L-KOGA introduced in section 3 with a standard hyperparameter tuned kernel method.

- The two-layered kernel is optimized for 25 epochs using a batch size of 64 and the Adam optimizer with an initial learning rate of $5 \cdot 10^{-3}$. A regularization of 10^{-5} was added to stabilize the numerical calculation of Rippa's formula in (3.10).
- The standard Matérn kernel was used with 10 logarithmically equally spaced shape parameters ε between 0.05 and 10, i.e., $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\varepsilon \cdot \|\boldsymbol{x} - \boldsymbol{y}\|/\sqrt{d})$.

The results are visualized in Figure 2 and furthermore listed in Table 1: In the left column of Figure 2 we note that the 2L-KOGA performs better than any of the hyperparameter tuned kernels. There seem to be two cases:

- For f_5 (top), the convergence rate of the two-layered kernel is significantly faster than the convergence rates of the standard kernels. Just in the beginning the convergence rate seems to decrease. The faster convergence rate can be explained by the final optimized matrix $\mathsf{A}_{\boldsymbol{\theta}}$, which has eigenvalues

$$\lambda_1 = 2.6623, \lambda_2 = 1.5540 \cdot 10^{-2}, \lambda_3 = 1.8592 \cdot 10^{-3},$$
$$\lambda_4 = 2.4576 \cdot 10^{-4}, \lambda_5 = -4.1398 \cdot 10^{-3},$$

  i.e., one major eigenvalue and four more eigenvalues which are significantly smaller. The eigenvector associated to the largest eigenvalue is given by

$$\boldsymbol{v}_{\lambda_1} = (0.4453, 0.4418, 0.4364, 0.4461, 0.4480)^\intercal,$$

  which is close to (a multiple) of $(1,1,1,1,1)^\intercal$ and thus quite well aligned with the active subspace direction of f_5.
- For both f_6 and f_7 the convergence rate seems to be the same; however, the prefactor is smaller, i.e., the two-layered approach is consistently better by some factor. Neither f_6 nor f_7 has an active subspace, nevertheless still there exist "more important directions" for the approximation, which are found by the optimization of the matrix $\mathsf{A}_{\boldsymbol{\theta}}$ of the first layer.

In all the cases the optimized two-layered kernel is better suited for the approximation of the given target function than any kernel with a tuned single hyperparameter

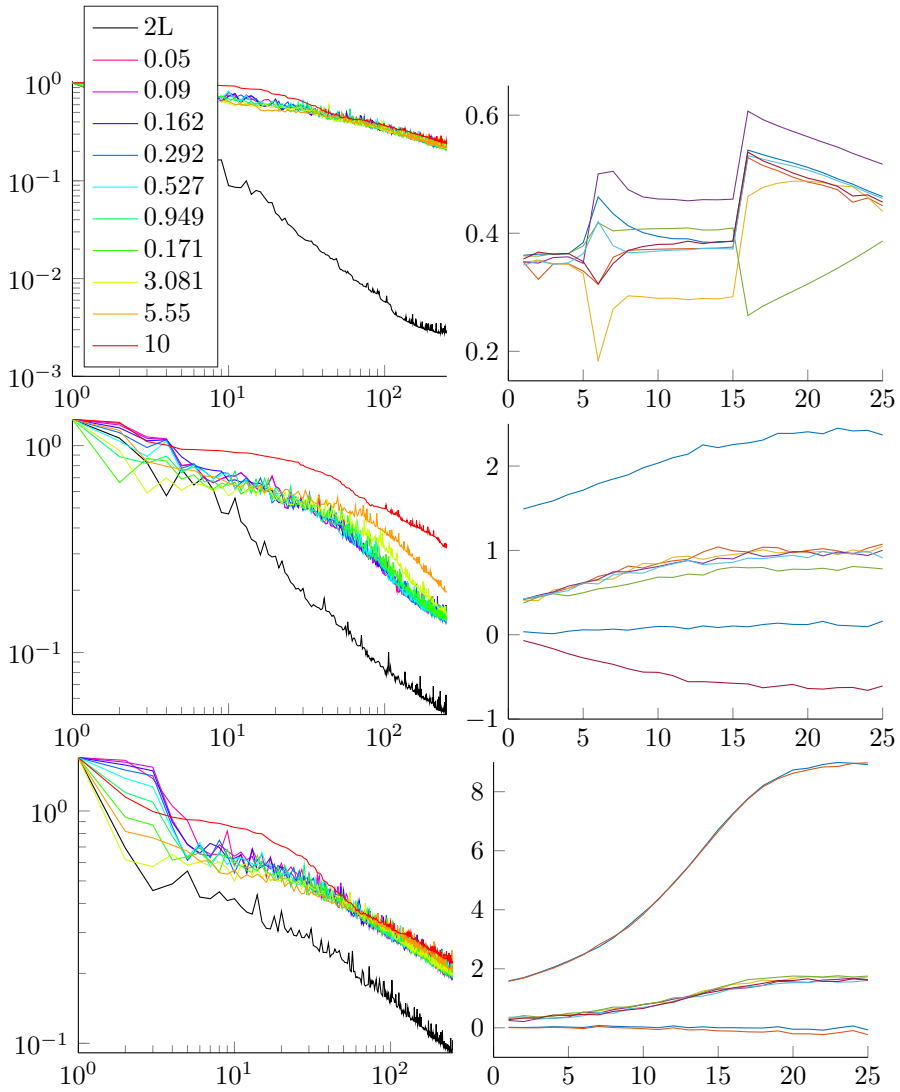

FIG. 2. *Top: 5D experiment; middle: 6D experiment; bottom: 7D experiment. Left: Visualization of the* $\|\cdot\|_{L^\infty(\Omega)}$ *error (y-axis) in the number of greedily selected points (x-axis) for both the standard kernel and the 2L-KOGA. Right: Visualization of the change of several matrix entries of the matrix* $\mathsf{A}_{\boldsymbol{\theta}}$ *(y-axis) during the optimization epochs (x-axis).*

in view of the preasymptotic range. These results on function approximation also raise the demand for further theoretical work, to better understand the faster convergence rates with approximation theory results.

We remark that the results do not qualitatively change if some small noise is added on the target data, i.e., to the function evaluations $f_d(\boldsymbol{x})$. The case of presence of noise is discussed in subsection 4.2 as well as subsection 4.3, as the machine learning datasets considered in those sections are naturally affected by noise.

4.2. Machine learning datasets. In order to show the usability of our 2L-KOGA method also on real world datasets, we compare our method to standard

TABLE 2

Overview of the runtimes (mean and standard deviation over five reruns) for the standard one-layered kernel model and the two-layered kernel models. The time required for the two-layered kernel optimization is only a small fraction of the subsequent greedy runtime.

Short name	1L greedy [s]	2L optimization [s]	2L greedy [s]
ct	264.34 ± 1.06	53.04 ± 10.95	264.01 ± 0.54
sgemm	509.16 ± 6.85	67.63 ± 5.57	511.77 ± 6.50
wecs	173.63 ± 0.73	22.26 ± 3.09	179.48 ± 0.45
mlr_knn_rng	338.69 ± 1.66	86.58 ± 13.98	339.61 ± 1.10
fried	94.92 ± 0.61	10.57 ± 2.18	94.76 ± 0.44
kegg_undir_uci	149.76 ± 0.86	18.77 ± 1.62	150.19 ± 1.03
online_video	158.68 ± 0.71	18.69 ± 1.91	158.72 ± 0.60
diamonds	128.79 ± 0.53	15.54 ± 1.40	128.66 ± 0.49
stock	133.17 ± 0.80	15.53 ± 2.65	133.18 ± 0.31
sarcos	106.09 ± 0.57	11.56 ± 1.42	106.71 ± 0.81
query_agg_count	410.40 ± 3.08	52.21 ± 2.41	396.19 ± 4.32
road_network	848.32 ± 10.30	108.23 ± 6.19	815.47 ± 3.15

hyperparameter tuned approaches on 12 out of 15 regression datasets which were used in [21]. We excluded the three datasets *methane, poker*, and *protein* because those datasets are not suitable for sparse kernel models. These datasets have input dimensionality ranging from 2 to 379 and sampling sizes between 8153 and 300000; see Table 3. For more details on the used datasets, we refer the reader to [21, Tables E.1 and E.2].

We note that the motivation of surrogate modeling might not be intrinsically given on those datasets; however, they still provide a meaningful benchmark for comparing our method with standard cross-validated kernel models.

We used the same setup for our comparison as in subsection 4.1, with the small modification that we used a regularization of 10^{-3} for the stabilization of the numerical calculation of Rippa's formula in (3.11) and a regularization of $\lambda = 10^{-4}$ (see (2.3)) within the greedy approximation. These higher regularizations are due to the fact that these real world datasets are possibly noisy. Note that these hyperparameters were used out of the box without any fine tuning or adaption to any specific datasets, which shows the general applicability of the approach. The experiments were rerun five times for different training test splits in order to mitigate the randomness of the split. We note that we did not rerun the (partly randomized, due to batch selection) kernel optimization several times within a given training test split. This is discussed in more detail in subsection 4.3, which highlights the robustness and stability of our optimization procedure.

The results for the different datasets are visualized in Figures 3 and 4, where the decay of the test mean squared error (MSE) is displayed for one out of the five reruns. We opted to not include the error bars, because they provided very limited further insights but made the visualizations of the results more difficult. The timings of the experiments are summarized in Table 2.

The results for the 12 datasets are ordered from *better* to *worse* (from the 2L-KOGA point of view) by using the mean relative improvement of the 2L-KOGA approach in comparison to the best hyperparameter tuned kernel as a criterion.

- For the six datasets *ct, sgemm, wecs, mlr_knn_rng, fried*, and *kegg_undir_uci* one can observe that the 2L-KOGA approach is quite consistently better than any hyperparameter tuned kernel. Especially for, e.g., *ct* or *sgemm,* the 2L-KOGA approach with around 50 centers can already reach the same accuracy

TABLE 3
Overview on the machine learning datasets which were used.

Short name	Number of examples	Number of features
ct	53500	379
sgemm	241600	14
wecs	72000	48
mlr_knn_rng	111753	132
fried	40768	10
kegg_undir_uci	64608	27
online_video	68784	26
diamonds	53940	29
stock	59049	9
sarcos	44484	21
query_agg_count	200000	4
road_network	434874	2

as the best hyperparameter tuned kernel model with 1000 centers. We note that in some datasets (e.g., in *fried*) one can observe an overall saturation of the accuracy for any method. However, the 2L-KOGA approach achieves this saturation already with a much smaller expansion size.
One should note that this does not necessarily imply that the 2L-KOGA approach is cheaper to evaluate, because the linear mapping of the first layer is possibly costly, especially, e.g., for the 379-dimensional *ct* dataset. Though for the 14-dimensional *sgemm* dataset we can for sure expect a benefit.

- For the six datasets *diamonds, sarcos, stock, road_network, online_video*, and *query_agg_count,* the 2L-KOGA approach is not consistently better: One can observe alternating performances, e.g., for the *diamonds, sarcos*, and *online_video* dataset, where the 2L-KOGA approach provides better models for small expansion sizes, but no longer for large expansion sizes. For *stock* and *road_network* the 2L-KOGA approach is (asymptotically) on par with the hyperparameter tuned method. Only for the *query_agg_count* does the 2L-KOGA approach seem to be consistently inferior to well-chosen hyperparameter tuned kernels. A possible remedy in view of the alternating behavior might be larger or even varying batch sizes during training or other optimization objectives. We leave these points for future research, while in the following we try to give some explanations about the fact that the 2L-KOGA is not necessarily better than a well-chosen hyperparameter tuned model.

We can again leverage the eigenvalues of the optimized first layer matrix $\mathsf{A}_{\boldsymbol{\theta}}$: A standard kernel can be seen as using the identity matrix and therefore having all eigenvalues equal to one, meaning that all directions in the Euclidean space are of equal importance. A matrix $\mathsf{A}_{\boldsymbol{\theta}}$ with only a few large singular values might, on the other hand, indicate that there are more important directions in the dataset. Therefore we use the *cumulative power*

$$m \mapsto \frac{\sum_{i=1}^{m} |s_i(\mathsf{A}_{\boldsymbol{\theta}})|}{\sum_{i=1}^{d} |s_i(\mathsf{A}_{\boldsymbol{\theta}})|}, \quad 1 \le m \le d, \tag{4.1}$$

as a criterion (whereby the singular values $s_i(\mathsf{A}_{\boldsymbol{\theta}})$ are ordered from the largest to the smallest one according to their absolute value) that allows us to understand how much power is clustered in the top singular values. The behavior of the quantity in (4.1) is depicted in Figure 5, whereby n/d is used for the x-axis. This allows us to

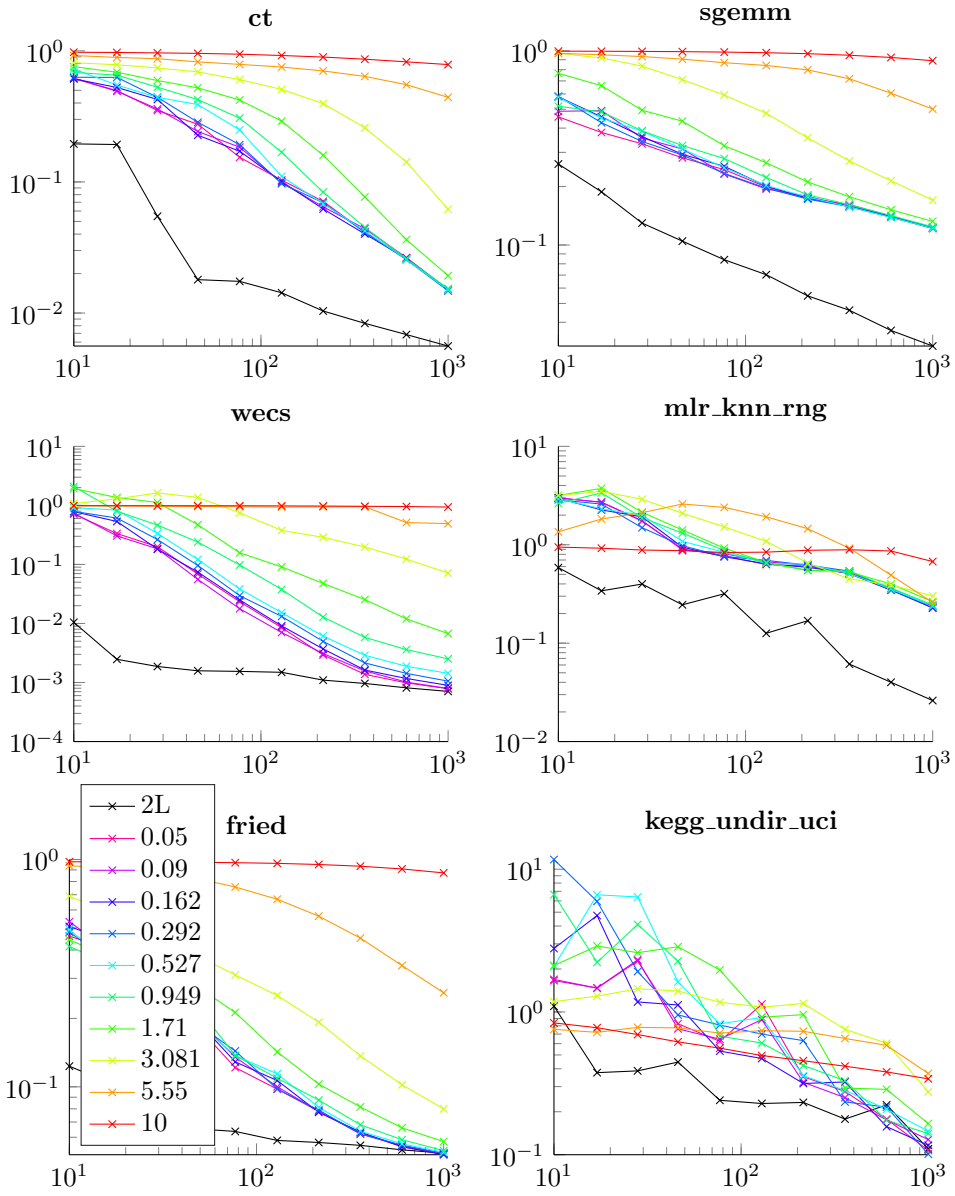

FIG. 3. *Visualization of the test MSE (y-axis) over the number of greedily selected centers (x-axis) for the first six datasets. The black line shows the 2L-KOGA, while the colored lines show the use of standard kernels with different length scale parameters ε. (Color figure available online.)*

compare datasets with different dimension d. One can see that the 2L-KOGA models are better if the quantity (4.1) increases quickly, i.e., if a lot of power is captured by a few large singular values, which is the case, for example, for *ct* or *sgemm*. On the other hand, a slow increase (e.g., for *query_agg_count* or *online_video*) is linked to the fact that we have no benefit in using the 2L-KOGA approach. Therefore the decay of the absolute values of the singular values $s_i(\mathsf{A}_{\boldsymbol{\theta}})$, or especially the existence of particular large singular values, can be used as a criterion to assess whether or not the kernel optimization is likely to improve the kernel model.

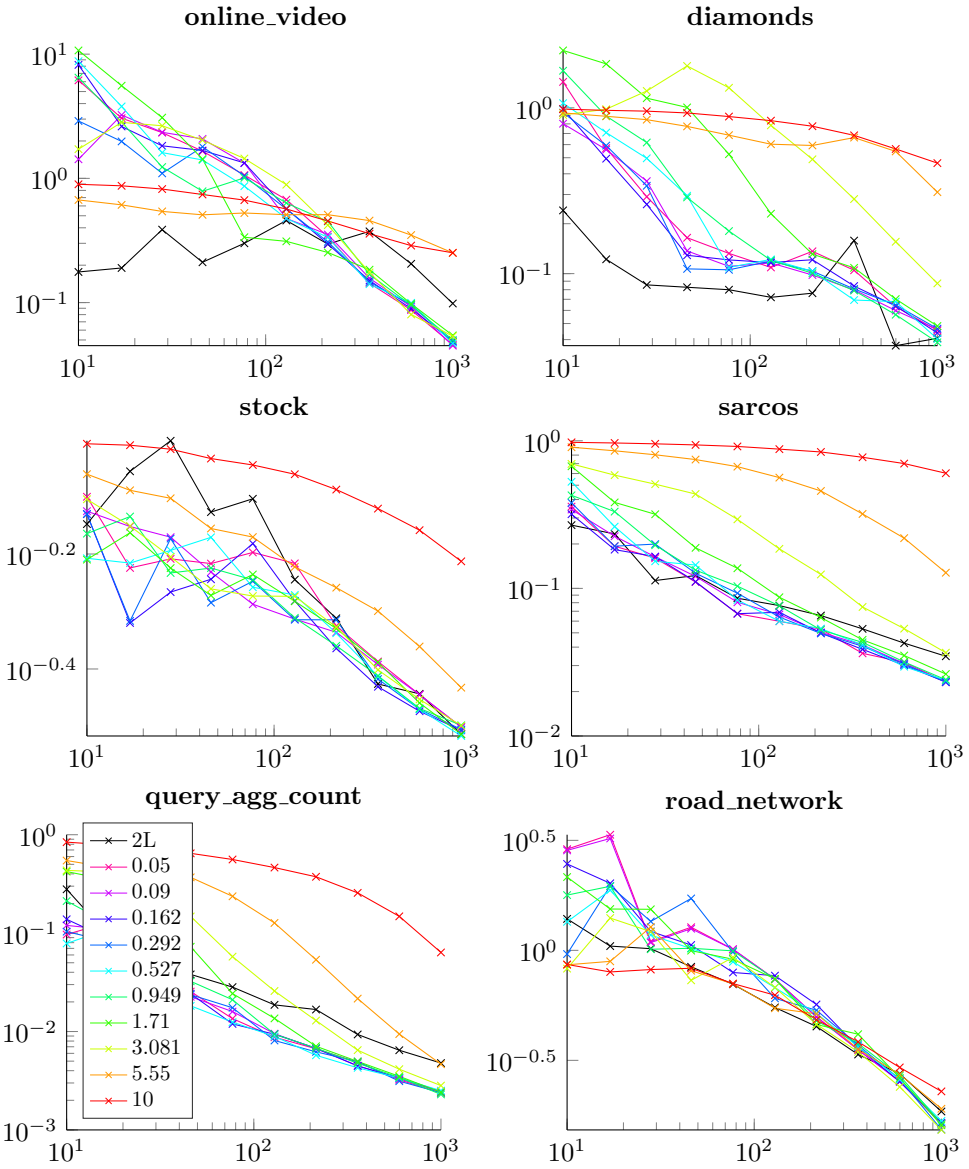

FIG. 4. *Visualization of the test MSE (y-axis) over the number of greedily selected centers (x-axis) for the last six datasets. The black line shows the 2L-KOGA, while the colored lines show the use of standard kernels with different length scale parameters ε. (Color figure available online.)*

Based on this observation it might make sense to use and optimize a $b \times d$ matrix with $b < d$ instead of the full $d \times d$ matrix. We leave this idea to future research.

4.3. Stability of the kernel optimization. In order to verify that the 2L-KOGA approach indeed uses a stable data-driven kernel optimization, we employ different criteria for the kernel optimization. While subsection 4.2 relied on LOOCV (i.e., k-fold cross-validation for $k = 64$, which was the batch size) for the optimization of the kernel, here we rerun the same experiments and make use of different values of k for our cross-validation, leveraging the efficient implementation described in subsec-

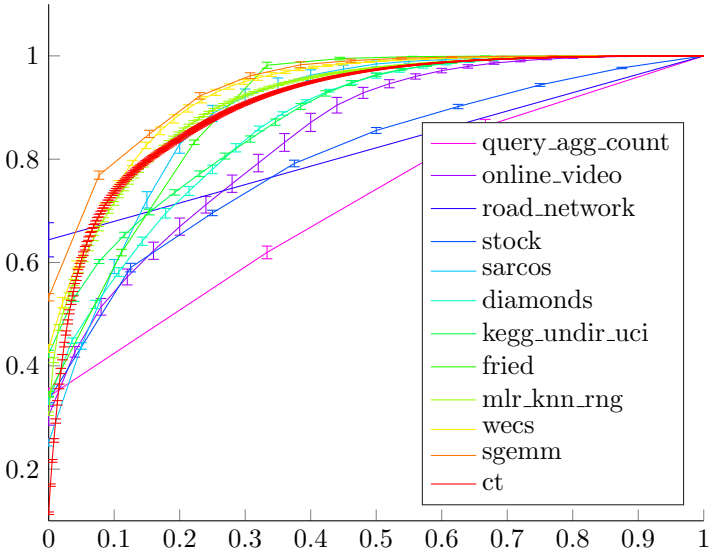

FIG. 5. *Visualization of the ratio* $\sum_{i=1}^{m} |s_i(\mathsf{A}_{\boldsymbol{\theta}})| / \sum_{i=1}^{d} |s_i(\mathsf{A}_{\boldsymbol{\theta}})|$ *(mean and standard deviation over five reruns) (y-axis) over the ratio* m/d *of considered singular values (x-axis). In conjunction with Figures* 3 *and* 4 *one can see that the 2L-KOGA approach works better when the ratio of the singular values approaches the value one more quickly.*

tion 3.3. This is motivated by the fact that it is a priori unclear whether leave-one-out cross-validation is a suitable choice for k-fold cross-validation, or whether one should rather use a different value.

In Figure 6 the results for three out of the twelve datasets are depicted. In the left column, the decay of the singular values is shown. In the right column, the principal angle between subspaces according to [24] is plotted. In detail, we depicted the principal angle between the subspace spanned by the first n right singular vectors of the optimized matrix $\mathsf{A}_{\boldsymbol{\theta}}$ for some method with the corresponding subspace of a benchmark method (for which we here used the LOOCV method, i.e., $k = 64$). For a precise definition of the principal angle, see [24, section 1]. For comparison reasons, also the corresponding quantities for the standard approach are computed, i.e., all the singular values are equal to one as we have $\mathsf{A}_{\boldsymbol{\theta}} = \mathsf{I}_d$.

1. From the plots of the singular value decays we can infer that the singular value distributions are always very close to each other, in particular for large singular values. Those large singular values are more important, as data in the direction of singular vectors to larger singular values are stretched, while those in the direction of smaller singular values are compressed. Furthermore, one can clearly observe the difference of the classical approach, where all the singular values have the same value 1, i.e., all the directions within the dataset have equal importance.
2. From the plots of the overlap we can observe that the right singular vectors to the largest singular values of the matrix $\mathsf{A}_{\boldsymbol{\theta}}$ are quite aligned (i.e., small angles) for all the optimization methods. Only for larger subspaces, which, however, also correspond to less important singular values, the angle starts to grow.

We remark that the plots for other datasets are qualitatively similar and showing them does not provide further insights.

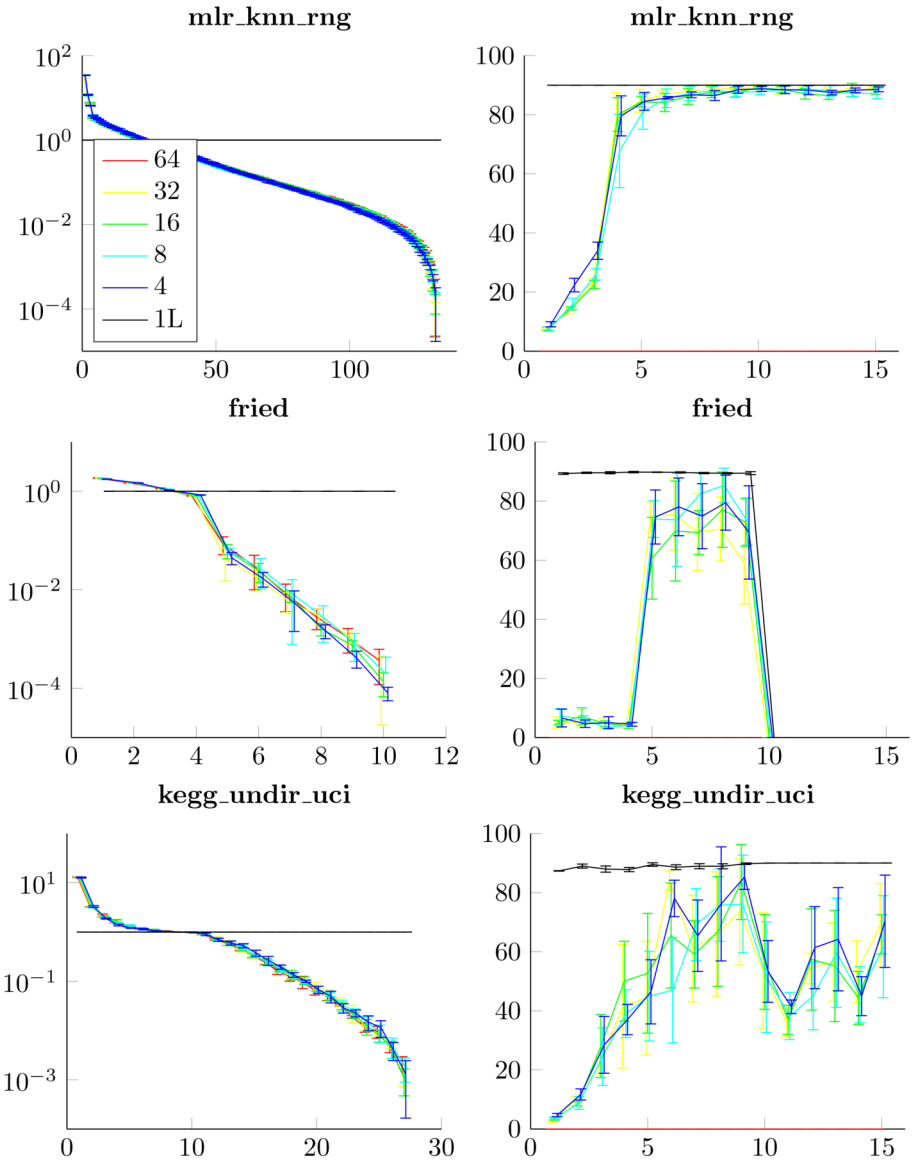

FIG. 6. *Analysis of the optimized matrices* $\mathsf{A}_{\boldsymbol{\theta}}$ *for three datasets. Left: Visualization of the singular values of the matrix* $\mathsf{A}_{\boldsymbol{\theta}}$. *Right: Visualization of the principal angle (y-axis) between subspaces spanned by the first n right singular vectors (x-axis). We set the number of folds as* $n_{\text{fold}} = 64$ *as reference and compare to* $n_{\text{fold}} \in \{64, 32, 16, 8, 4\}$ *and no optimization (i.e., the classical VKOGA). Errorbars are used to visualize the mean and standard deviation for five reruns.*

Since both the (large) singular values and the corresponding singular vectors are similar, we infer that the optimized matrices $\mathsf{A}_{\boldsymbol{\theta}}$ are quite close to each other in the sense of detecting the same features and applying the same scalings to those features. The differences within the smaller singular values and corresponding singular vectors actually do not negatively impact the performance of the kernel model, as the f-greedy selection criterion can accommodate for this. In fact the test error decays

(i.e., Figures 3 and 4) when using those slightly different matrices match almost perfectly (not shown here). We note that the optimized matrices $\mathsf{A}_{\boldsymbol{\theta}}$ are not necessarily close to each other, which is related to the irrelevance of the right singular vectors for the overall kernel model as elaborated in subsection 3.2.1.

All in all these experiments using different k-fold cross-validation parameters for our kernel optimization showed that the used optimization procedure is stable. This was desired, because we strove to obtain a data adapted kernel.

For the experiments in subsection 4.2 we simply used $k = 64$ (batch size) for the optimization as it is slightly more time-efficient than using another value of k. However, also the use of another k fold cross-validation parameter is not detrimental, as the time consumption for the kernel optimization is always smaller than the time required for the greedy selection; see also Table 1. This was, however, possible only by making use of the efficient implementation due to the extended Rippa's algorithm; see (3.10).

5. Conclusion and outlook. In this paper we introduced a machine learning way of choosing kernel hyperparameters, which is done by a stochastic gradient descent optimization. For the optimization, we leveraged a recent efficient way to compute k-fold cross-validation errors and compared their performances. Especially those hyperparameter optimized kernels can be seen as a two-layered kernel machine, i.e., a deep kernel. The method was used in conjunction with greedy methods to select proper data points in order to obtain sparse models. Fundamental analysis on the proposed method was provided as well as experiments on synthetic and real world data, which highlight the benefits of the approach.

Future work will aim at combining the kernel optimization with the greedy selection procedure and generalizing the first layer map. Instead of using only linear kernels which give rise to linear mappings, the use of nonlinear kernels seems appealing.

Acknowledgments. The first author thanks Gabriele Santin and Bernard Haasdonk for discussions.

REFERENCES

[1] F. Aiolli and M. Donini, *Learning anisotropic RBF kernels*, in International Conference on Artificial Neural Networks, Springer, 2014, pp. 515–522.

[2] B. Bohn, C. Rieger, and M. Griebel, *A representer theorem for deep kernel learning*, J. Mach. Learn. Res., 20 (2019), pp. 2302–2333.

[3] C. Campi, F. Marchetti, and E. Perracchione, *Learning via variably scaled kernels*, Adv. Comput. Math., 47 (2021), pp. 1–23.

[4] R. Cavoretto, *Adaptive LOOCV-based kernel methods for solving time-dependent BVPs*, Appl. Math. Comput., 429 (2022), 127228.

[5] R. Cavoretto, A. De Rossi, M. S. Mukhametzhanov, and Y. D. Sergeyev, *On the search of the shape parameter in radial basis functions using univariate global optimization methods*, J. Global Optim., 79 (2021), pp. 305–327.

[6] P. G. Constantine, *Active Subspaces: Emerging Ideas for Dimension Reduction in Parameter Studies*, SIAM, Philadelphia, 2015, https://doi.org/10.1137/1.9781611973860.

[7] P. G. Constantine, E. Dow, and Q. Wang, *Active subspace methods in theory and practice: Applications to kriging surfaces*, SIAM J. Sci. Comput., 36 (2014), pp. A1500–A1524, https://doi.org/10.1137/130916138.

[8] S. De Marchi, F. Marchetti, and E. Perracchione, *Jumping with variably scaled discontinuous kernels (VSDKs)*, BIT, 60 (2020), pp. 441–463.

[9] S. De Marchi, R. Schaback, and H. Wendland, *Near-optimal data-independent point locations for radial basis function interpolation*, Adv. Comput. Math., 23 (2005), pp. 317–330.

[10] T. A. Driscoll and B. Fornberg, *Interpolation in the limit of increasingly flat radial basis functions*, Comput. Math. Appl., 43 (2002), pp. 413–422.

[11] S. Dutta, M. W. Farthing, E. Perracchione, G. Savant, and M. Putti, *A greedy non-intrusive reduced order model for shallow water equations*, J. Comput. Phys., 439 (2021), 110378.
[12] G. E. Fasshauer, *Meshfree Approximation Methods with MATLAB*, Interdiscip. Math. Sci. 6, World Scientific, 2007.
[13] G. E. Fasshauer and M. J. McCourt, *Kernel-Based Approximation Methods Using MATLAB,* Interdiscip. Math. Sci. 19, World Scientific, 2015.
[14] B. Fornberg and G. Wright, *Stable computation of multiquadric interpolants for all values of the shape parameter*, Comput. Math. Appl., 48 (2004), pp. 853–867.
[15] B. Fornberg and J. Zuev, *The Runge phenomenon and spatially variable shape parameters in RBF interpolation*, Comput. Math. Appl., 54 (2007), pp. 379–398.
[16] A. Globerson and S. Roweis, *Metric learning by collapsing classes*, in Advances in Neural Information Processing Systems 18, MIT Press, 2005, pp. 451–458.
[17] J. Goldberger, G. E. Hinton, S. Roweis, and R. R. Salakhutdinov, *Neighbourhood components analysis*, in Advances in Neural Information Processing Systems 17, MIT Press, 2004, pp. 513–520.
[18] G. H. Golub, M. Heath, and G. Wahba, *Generalized cross-validation as a method for choosing a good ridge parameter*, Technometrics, 21 (1979), pp. 215–223.
[19] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016; available online at http://www.deeplearningbook.org.
[20] B. Hamzi and H. Owhadi, *Learning dynamical systems from data: A simple cross-validation perspective, part* I*: Parametric kernel flows*, Phys. D, 421 (2021), 132817.
[21] D. Holzmüller, V. Zaverkin, J. Kästner, and I. Steinwart, *A framework and benchmark for deep batch active learning for regression*, J. Mach. Learn. Res., 24 (2023), 164.
[22] G. S. Kimeldorf and G. Wahba, *A correspondence between Bayesian estimation on stochastic processes and smoothing by splines*, Ann. Math. Statist., 41 (1970), pp. 495–502.
[23] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, preprint, arXiv:1412.6980, 2014.
[24] A. V. Knyazev and M. E. Argentati, *Principal angles between subspaces in an A-based scalar product: Algorithms and perturbation estimates*, SIAM J. Sci. Comput., 23 (2002), pp. 2008–2040, https://doi.org/10.1137/S1064827500377332.
[25] L. Ling and F. Marchetti, *A stochastic extended Rippa's algorithm for LpOCV*, Appl. Math. Lett., 129 (2022), 107955.
[26] F. Marchetti, *The extension of Rippa's algorithm beyond LOOCV*, Appl. Math. Lett., 120 (2021), 107262.
[27] F. Marchetti and E. Perracchione, *Efficient reduced basis algorithm (ERBA) for kernel-based approximation*, J. Sci. Comput., 91 (2022), 41.
[28] M. McCourt, *Using Gaussian eigenfunctions to solve boundary value problems*, Adv. Appl. Math. Mech., 5 (2013), pp. 569–594.
[29] F. N. Mojarrad, M. H. Veiga, J. S. Hesthaven, and P. Öffner, *A new variable shape parameter strategy for RBF approximation using neural networks*, Comput. Math. Appl., 143 (2023), pp. 151–168.
[30] S. Müller, *Komplexität und Stabilität von kernbasierten Rekonstruktionsmethoden*, Ph.D. thesis, Niedersächsische Staats- und Universitätsbibliothek Göttingen, Göttingen, Germany, 2009.
[31] M. P. Otto and R. Izbicki, *RFFNet: Scalable and Interpretable Kernel Methods via Random Fourier Features*, preprint, arXiv:2211.06410, 2022.
[32] H. Owhadi and G. R. Yoo, *Kernel flows: From learning kernels from data into the abyss*, J. Comput. Phys., 389 (2019), pp. 22–47.
[33] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., *PyTorch: An imperative style, high-performance deep learning library*, in Advances in Neural Information Processing Systems 32, Curran Associates, 2019, pp. 8026–8037.
[34] E. Perracchione, A. M. Massone, and M. Piana, *Feature augmentation for the inversion of the Fourier transform with limited data*, Inverse Problems, 37 (2021), 105001.
[35] S. Rippa, *An algorithm for selecting a good value for the parameter c in radial basis function interpolation*, Adv. Comput. Math., 11 (1999), pp. 193–210.
[36] M. Rossini, *Interpolating functions with gradient discontinuities via variably scaled kernels*, Dolomites Res. Notes Approx., 11 (2018).
[37] G. Santin and B. Haasdonk, *Convergence rate of the data-independent P-greedy algorithm in kernel-based approximation*, Dolomites Res. Notes Approx., 10 (2017), pp. 68–78, pp. 3–14.

[38] G. Santin and B. Haasdonk, *Kernel methods for surrogate modeling*, Model Order Reduction, 1 (2019), pp. 311–354.
[39] R. Schaback and H. Wendland, *Adaptive greedy techniques for approximate solution of large RBF systems*, Numer. Algorithms, 24 (2000), pp. 239–254.
[40] S. Shalev-Shwartz, Y. Singer, and A. Y. Ng, *Online and batch learning of pseudo-metrics*, in Proceedings of the Twenty-First International Conference on Machine Learning, 2004, p. 94–101.
[41] G. Wahba, *Spline Models for Observational Data*, SIAM, Philadelphia, 1990, https://doi.org/10.1137/1.9781611970128.
[42] H. Wendland, *Scattered Data Approximation*, Cambridge Monogr. Appl. Comput. Math. 17, Cambridge University Press, Cambridge, 2005.
[43] T. Wenzel, B. Haasdonk, H. Kleikamp, M. Ohlberger, and F. Schindler, *Application of Deep Kernel Models for Certified and Adaptive RB-ML-ROM Surrogate Modeling*, to appear, 2023. Accepted for LSSC. 2023 proceedings.
[44] T. Wenzel, G. Santin, and B. Haasdonk, *A novel class of stabilized greedy kernel approximation algorithms: Convergence, stability and uniform point distribution*, J. Approx. Theory, 262 (2021), 105508.
[45] T. Wenzel, G. Santin, and B. Haasdonk, *Universality and Optimality of Structured Deep Kernel Networks*, preprint, arXiv:2105.07228, 2021.
[46] T. Wenzel, G. Santin, and B. Haasdonk, *Analysis of target data-dependent Greedy Kernel algorithms: Convergence rates for f-, $f \cdot P$-, and f/P-Greedy*, Constr. Approx., 57 (2023), pp. 45–74.
[47] T. Wenzel, G. Santin, and B. Haasdonk, *Stability of Convergence Rates: Kernel Interpolation on Non-Lipschitz Domains*, preprint, arXiv:2203.12532, 2022.
[48] A. G. Wilson, Z. Hu, R. Salakhutdinov, and E. P. Xing, *Deep kernel learning*, in Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, A. Gretton and C. C. Robert, eds., PMLR 51, 2016, pp. 370–378.
[49] D. Wirtz and B. Haasdonk, *A vectorial kernel orthogonal greedy algorithm*, Dolomites Res. Notes Approx., 6 (2013), pp. 83–100.
[50] D. Wirtz, N. Karajan, and B. Haasdonk, *Surrogate modeling of multiscale models using kernel methods*, Internat. J. Numer. Methods Engrg., 101 (2015), pp. 1–28.