

Structured Deep Kernel Networks for Data-Driven Closure Terms of Turbulent Flows

Tizian Wenzel^{1,5}, Marius Kurz², Andrea Beck³, Gabriele Santin⁴[0000-0001-6959-1070], and Bernard Haasdonk¹

¹ Institute for Applied Analysis and Numerical Simulation, University of Stuttgart, Germany

² Institute of Aerodynamics und Gas Dynamics, University of Stuttgart, Germany

³ Laboratory of Fluid Dynamics and Technical Flows, Otto-von-Guericke-Universität Magdeburg

⁴ Digital Society Center, Bruno Kessler Foundation, Trento, Italy

⁵ Corresponding author

Abstract. Standard kernel methods for machine learning usually struggle when dealing with large datasets. We review a recently introduced Structured Deep Kernel Network (SDKN) approach that is capable of dealing with high-dimensional and huge datasets - and enjoys typical standard machine learning approximation properties.

We extend the SDKN to combine it with standard machine learning modules and compare it with Neural Networks on the scientific challenge of data-driven prediction of closure terms of turbulent flows. We show experimentally that the SDKNs are capable of dealing with large datasets and achieve near-perfect accuracy on the given application.

Keywords: Machine Learning · Structured Deep Kernel Networks · Closure Terms · Turbulent Flows

1 Introduction

In modern science and engineering there is an increasing demand for efficient and reliable data-based techniques. On the one hand, an unprecedented amount of data is nowadays available from various sources, notably complex computer simulations. On the other hand, traditional model-based methods are often required to integrate the additional information acquired through measurements.

A particularly interesting situation is the case of surrogate modeling [8], where the underlying ground-truth is some engineering model represented by a computational simulation, which is accessible but expensive to execute. In this setting the accurate simulation provides a discrete set of input-output pairs, and the resulting data-based model, or surrogate, can be used to replace to some extent the full simulation in order to simplify its understanding and analysis.

In this paper we use a recently proposed Structured Deep Kernel Network, which is build on a representer theorem for deep kernel learning [3], and extend and apply it to multivariate regression using time-series data. The technique is applied to the prediction of closure terms for turbulent flows, where a comparison

with standard neural network techniques is drawn. The results show the full flexibility of the proposed setup while archiving near-perfect accuracy.

The paper is organized as follows. To begin with, we recall in Section 1.1 and Section 1.2 some background information about machine learning and Artificial Neural Networks (ANNs). In Section 2 the novel Structured Deep Kernel Network (SDKN) is reviewed and combined with standard machine learning modules. Section 3 provides background information on our application setting and the need for machine learning techniques. The subsequent Section 4 explains the numerical experiments and their results as well as the practicability of the SDKN. Section 5 summarizes the results and provides an outlook.

1.1 Regression in Machine Learning

Machine learning for regression tasks is usually posed as an optimization problem. For given data $\mathcal{D} := (x_i, y_i)_{i=1}^n$ with $x_i \in \mathbb{R}^{d_{\text{in}}}$, $y_i \in \mathbb{R}^{d_{\text{out}}}$, the goal is to find a function f which approximates $f(x_i) = y_i$. Mathematically speaking, this refers to minimizing a loss functional \mathcal{L}_D . In case of a mean-squared error (MSE) loss, this means searching for

$$f^* := \arg \min_{f \in \mathcal{H}} \mathcal{L}_D(f), \quad \mathcal{L}_D(f) = \frac{1}{n} \sum_{i=1}^n \|y_i - f(x_i)\|_2^2 + \lambda \cdot \mathcal{R}(f) \quad (1)$$

over a suitable space of functions \mathcal{H} , whereby $\mathcal{R}(f)$ is a function-dependent regularization term with regularization parameter $\lambda \in \mathbb{R}$. The space of functions \mathcal{H} is usually parametrized, i.e. we have $f(x) = f(x, \theta)$ for some parameters θ .

We consider here two very popular techniques that define f , namely Artificial Neural Networks (ANNs) and kernel methods. ANNs enjoy favorable properties for high dimensional data approximation due to representation learning. The other approach is given by kernel methods, which have a sound theoretical background, however struggle when dealing with large and high-dimensional datasets.

1.2 Neural Networks

ANNs (see e.g. [5]) are a prevalent class of ansatz functions in machine learning. The common feedforward ANN architecture consists of L consecutive layers, whereby layer l transforms its input $x \in \mathbb{R}^{d_{l-1}}$ according to

$$f_l(x) = \sigma(W_l x + b_l), \quad (2)$$

with a weight matrix $W_l \in \mathbb{R}^{d_l \times d_{l-1}}$ and a bias vector $b \in \mathbb{R}^{d_l}$. The weight matrices $\{W_l\}_{l=1, \dots, L}$ and bias vectors $\{b_l\}_{l=1, \dots, L}$ for all the layers are parameters of the ANN, which are obtained by solving the optimization task in Eq. (1). A common choice for the non-linear activation function $\sigma(\cdot)$ is the rectified linear unit (ReLU, $\sigma(\cdot) = \max(\cdot, 0)$), which is applied element-wise. The output of a layer is then passed as input to the succeeding layer. The feedforward ANN can thus be written as a concatenation of the individual layers $f(x) = f_L \circ \dots \circ f_1(x)$.

In contrast to standard kernel methods which will be recalled in the next section, the basis of the ANN is not determined a priori, but depends on its weights

and biases. ANNs can thus learn a suitable basis automatically from the data they are trained on. Moreover, the concatenation of non-linear transformations allows the ANN to recombine the non-linear basis functions of the preceding layers to increasingly complex basis functions in the succeeding layers, which renders ANNs highly suitable for high-dimensional data.

Numerous variants of the feedforward ANN have been proposed for particular kinds of data and applications. For sequential data, recurrent neural networks (RNNs) have established themselves as state of the art. A common representative of RNNs is the GRU architecture proposed in [4].

2 Structured Deep Kernel networks

Here, we present a brief summary of kernel methods and the architecture presented in [10]. For a thorough discussion, we refer the reader to that reference, as our goal here is to present an extension of the original SDKN to sequential data.

Standard kernel methods rely on the use of possibly strictly positive definite kernels like the Gaussian kernel

$$k(x, z) = \exp(-\|x - z\|_2^2). \quad (3)$$

In this framework, a standard Representer Theorem [6] simplifies the loss minimization of Eq. (1) and states that a loss-minimizing function can be found in a subspace $V_n := \text{span}\{k(\cdot, x_i), i = 1, \dots, n\}$ spanned by the data, i.e.

$$f^*(\cdot) = \sum_{j=1}^n \alpha_j k(\cdot, x_j). \quad (4)$$

The corresponding coefficients $\alpha_j \in \mathbb{R}^{d_{\text{out}}}$, $j = 1, \dots, n$ can then be found by solving a finite dimensional and convex optimization problem. This is the basis to provide both efficient solution algorithms and approximation-theoretical results. However the choice of a fixed kernel $k(x, y)$ restricts these standard kernel methods, as the feature representation of the data is implicitly determined by the choice of k . Furthermore, assembling and solving the linear system to determine the coefficients α_j poses further problems in the big data regime.

In order to overcome these shortcoming, a deep kernel representer theorem [3] has been leveraged in [10] to introduce Structured Deep Kernel Networks, which alleviate these obstacles by putting kernel methods into a structured multilayer setup: The deep kernel representer theorem considers a concatenation of L functions as $f = f_L \circ \dots \circ f_1$ and states that the problem can be restricted to

$$f_l^*(\cdot) = \sum_{j=1}^n \alpha_{lj} k_l(\cdot, f_{l-1}^* \circ \dots \circ f_1^*(x_j)), \quad l = 2, \dots, L \quad (5)$$

with $\alpha_{lj} \in \mathbb{R}^{d_l}$. This is a generalization of the standard representation of Eq. (4) to the multilayer setting. The paper [10] proceeds by choosing non-standard kernels:

1. For l odd, vector-valued linear kernels are picked: $k_{\text{lin}}(x, z) = \langle x, z \rangle_{\mathbb{R}^{d_{l-1}}} \cdot I_{d_l}$, whereby $I_{d_l} \in \mathbb{R}^{d_l \times d_l}$ is the identity matrix.

- For l even, single-dimensional kernels are used, that use single components $x^{(i)}, z^{(i)}$ of the vectors $x, z \in \mathbb{R}^{d_{l-1}}$: $k_s(x, z) = \text{diag}(k(x^{(1)}, z^{(1)}), \dots, k(x^{(d)}, z^{(d)}))$ for some standard kernel $k: \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$, e.g. the Gaussian from Eq. (3).

This choice of kernels introduces a structure which is depicted in Figure 1, which motivates the naming *Structured Deep Kernel Networks* and allows for comparison with neural networks: The linear kernels of the odd layers give rise to fully connected layers (without biases), while even layers with their single-dimensional kernels can be viewed as *optimizable activation function layers*. In order to obtain a sparse model and alleviate possible overfitting issues, we only use an expansion size of e.g. $M = 5 \ll n$ within the sum of Eq. (5), which amounts to fix the remaining coefficients to zero. Even for this sparse representation and special choice of kernels, it is proven in [10] that the SDKNs satisfy universal approximation properties, legitimizing their use for machine learning tasks.

The proposed setup is sufficiently flexible such that it can be combined with standard neural network modules: For the application described in Section 3 we incorporate GRU-modules by using them in between two activation function layers. By doing so, we can make use of the time-dependence within the input data.

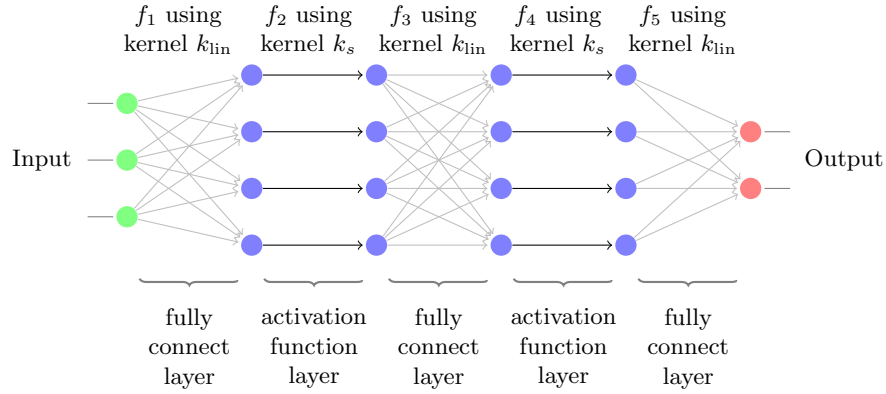


Fig. 1. Visualization of the Structured Deep Kernel Network. Gray arrows refer to layers using the linear kernel, while black arrows refer to layers using the single dimensional kernel layer. The braces below the layers indicate similarities to neural networks.

3 Turbulence Closure Problem

The evolution of compressible fluid flows (see e.g. [9]) is governed by the Navier-Stokes equations, which can be written in short as

$$U_t + R(F(U)) = 0, \quad (6)$$

with U as the vector of conserved variables. U_t denotes the derivation with respect to time and $R(\cdot)$ denotes the divergence operator applied to the non-linear fluxes $F(U)$. Most engineering flows of interest exhibit turbulent behavior.

Such turbulent flows are inherently chaotic dynamical systems with a multiscale character. The direct numerical simulation (DNS) of turbulence is thus only feasible for simple geometries and low Reynolds numbers, which correspond to a small range of active flow scales. The framework of large eddy simulation (LES) addresses these restrictions by resolving only the large energy-containing scales by applying a low-pass filter $\overline{(\cdot)}$ to the underlying Navier-Stokes equations. However, the filtered flux term $\overline{R(F(U))}$ is generally unknown, since it depends on the full solution U . To this end, the coarse-scale solution is typically advanced in time by some appropriate numerical discretization $\tilde{R}(\overline{U})$, which then yields

$$\overline{U}_t + \tilde{R}(\overline{U}) = \underbrace{\tilde{R}(\overline{U}) - \overline{R(F(U))}}_{\text{perfect LES closure}}. \quad (7)$$

Solving these filtered equations becomes feasible in terms of computational cost, but the right-hand side of Eq. (7) exhibits the unknown *closure terms*, which describe the effects of the unresolved small-scale dynamics on the resolved scales.

The task of turbulence modeling can thus be stated as finding some model M which recovers the closure terms solely from the filtered flow field:

$$\left(\tilde{R}(\overline{U}) - \overline{R(F(U))}\right) \approx M(\overline{U}). \quad (8)$$

A myriad of different models have been proposed in literature over the last decades to derive the mapping M based on mathematical and physical reasoning. While the accuracy of the closure model is crucial for obtaining a reliable description of the flow field, no universal and generally *best* model has been identified to date. Therefore, increasing focus is laid upon finding this mapping M from data by leveraging the recent advances in machine learning. See [2] for an extensive review. In that reference, machine learning is used to directly recover the unknown flux term $\overline{R(F(U))} = f(\overline{U})$ without positing any prior assumptions on the functional form of the underlying mapping $f(\cdot)$.

4 Numerical application

In the present work, the dataset described in [1,7] (to which we refer for further details on the following configurations) is used as training set for the machine learning algorithms. This dataset is based on high-fidelity DNS of decaying homogeneous isotropic turbulence (DHIT). The coarse-scale quantities according to Eq. (7) are obtained by applying three different LES filters to the DNS solution: A global Fourier cutoff filter (“Fourier”), a local L_2 -projection filter onto piecewise polynomials (“projection”) and a local Top-hat filter, as shown in Figure 2. The latter two are based on the typical representations of the solution in discontinuous Galerkin (DG) and Finite-Volume (FV) schemes, respectively.

For each filter, the corresponding dataset comprises nearly 30 million samples and a separate DHIT simulation is used as blind testing set. Since turbulence is a time-dependent phenomenon, the input features for each training sample are chosen as a time series of the filtered three-dimensional velocity vector $\overline{v}^{(i)}$,

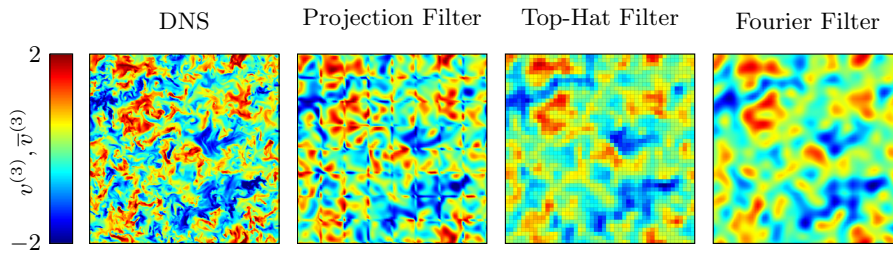


Fig. 2. Two-dimensional slices of the three-dimensional z-velocity field $v^{(3)}$ for the high-fidelity DNS and for the corresponding filtered velocity fields $\bar{v}^{(3)}$ of the three investigated LES filters. Each slice of the filtered flow field contains 48^2 solution points.

$i = 1, 2, 3$ at a given point in space. The target quantity is the three-dimensional closure term $\overline{R(F(U))}^{(i)}$, $i = 1, 2, 3$ in the last timestep of the given series. To investigate the influence of temporal resolution, a variety of different sampling strategies are examined, which are given in Table 1. Before training, the input features of the training dataset were normalized to zero mean and unit variance.

Table 1. Different time series for training. N_{seq} denotes the number of time instances per sample and Δt_{seq} is the time increment between two successive time instances.

	GRU1	GRU2	GRU3
N_{seq}	3	10	21
Δt_{seq}	10^{-3}	10^{-4}	10^{-4}

The ANN architecture from [7] was used as baseline model, which incorporates a GRU layer to leverage the temporal information in the data. Such a GRU layer was also introduced into the SDKN framework, which clearly demonstrates its modularity. In order to obtain a fair comparison of both the SDKN and ANN approach, both models exhibit mostly the same setup: The structure of the networks is given via their input dimension and output dimension of 3 each and the dimensions of the inner layers are chosen as 32, 64, 48, 24. This amounts to a total of 31400 optimizable parameters for the ANN and 32240 for the SDKN. The slight difference is related to the additional parameters α_{lj} within the single-dimensional kernel layers, see Eq. (5) for l even, and the unused bias-parameters within the SDKN. Concerning the the results in Table 2, the Gaussian kernel from Eq. (3) was used for the single-dimensional kernels within the SDKN, but we remark that the use of other kernels yields qualitatively the same results. We further remark that both models are within the underparametrized regime, as the number of parameters is significantly below the number of training samples.

The Adam optimizer was used for training with an initial learning rate of 10^{-3} , which was then halved after each 10 epochs for the NN and after each 5 epochs for the SDKN. The NN was optimized for 50 epochs and the SDKN for 25 epochs. This distinction keeps the overall optimization time comparable, since the SDKN optimization is about a factor of 2 more time-consuming per epoch.

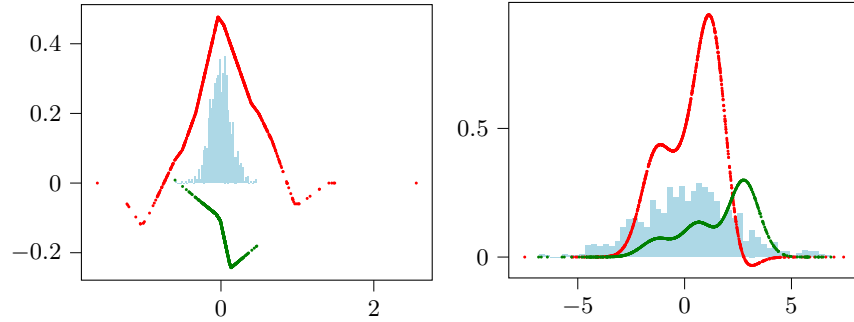


Fig. 3. Visualization of exemplary single-dimensional kernel function mappings before (red) and after optimization (green). The histograms indicate the distribution of the training data after optimization. Right: Gaussian kernel as defined in Eq. (3). Left: Wendland kernel of order 0, which is defined as $k(x, y) = \max(1 - \|x - y\|_2, 0)$. These mappings can be interpreted as optimizable activation functions, see Section 2.

This stems from the optimization of the additional parameters (e.g. the α_{lj} for l even in Eq. (5)) related to its setup. For training, the MSE loss from Eq. (1) was used without any regularization, since the use of a relatively small batch size (128) is likely to provide a sufficient regularization effect. We remark that both architecture and training are hyperparameter-optimized for the ANN setting, as this was the model of choice in [7]. The experiments were run on an Nvidia GTX1070 using PyTorch implementations of the models. The optimization took about 12 GPU-hours each.

Table 2 summarizes the results for the different datasets and cases: Both the final MSE-loss as well as the cross-correlation is given. The reported cross-correlations match the ones reported in [7], thus validating their results: Even in our runs, the ANN using the GRU2 setup performed badly on the DG dataset with only a final cross-correlation of 0.8163, which is way worse than the other listed cross-correlations. The SDKN reached at least the same test accuracies, even without any further hyperparameter optimization. Especially there is no drop in accuracy for the prediction related to the DG dataset in conjunction with the GRU2 case, as it can be observed in Table 2: The SDKN reaches a cross-correlation of 0.9989 which is en par with the other GRU-setups, in contrast to the performance of the ANN setup. This might indicate that the SDKN is less likely to get stuck in a local minima compared to the ANN.

5 Conclusion and outlook

In this paper an extension of a recently proposed Structured Deep Kernel Network (SDKN) setup to time-series data was introduced. The proposed SDKN model was compared to standard Neural Network models on the challenging task of data-driven prediction of closure terms for turbulence modeling. With help of machine learning models, significant speed ups in the simulation of turbulent flows can be achieved. It was shown numerically that the SDKN can reach

Table 2. Overview of the results on the test set after optimization of the Neural Network (NN) and the Structured Deep Kernel Network (SDKN): Cross-correlation (left) and Loss (right).

Cross-Correlation				MSE-Loss					
		GRU1	GRU2	GRU3		GRU1	GRU2	GRU3	
Projection	ANN	0.9989	0.8163	0.9989	Projection	ANN	3.235e-01	4.996e+01	3.233e-01
	SDKN	0.9989	0.9989	0.9988		SDKN	3.253e-01	3.261e-01	3.368e-01
Top-Hat	ANN	0.9992	0.9992	0.9992	Top-Hat	ANN	3.155e-02	2.917e-02	2.888e-02
	SDKN	0.9991	0.9992	0.9992		SDKN	3.222e-02	2.989e-02	2.893e-02
Fourier	ANN	0.9992	0.9993	0.9993	Fourier	ANN	1.179e-02	9.737e-03	9.452e-03
	SDKN	0.9992	0.9993	0.9993		SDKN	1.177e-02	1.007e-02	9.587e-03

the same near-perfect accuracy as hyper-parameter optimized ANNs for several variants of the task at the same training cost.

The mathematical background which the SDKN was built on seems promising for further theoretical analysis. From the application point of view, the goal is to optimize the SDKN only for a few epochs and then use the optimized kernel in conjunction with standard shallow kernel models, for which efficient and reliable methods exist. This is expected to significantly speed up the training phase of the surrogate model.

Acknowledgements: The authors acknowledge the funding of the project by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany’s Excellence Strategy - EXC 2075 - 390740016 and funding by the BMBF project ML-MORE.

References

1. Beck, A., Flad, D., Munz, C.D.: Deep neural networks for data-driven LES closure models. *Journal of Computational Physics* **398**, 108910 (2019)
2. Beck, A., Kurz, M.: A perspective on machine learning methods in turbulence modeling. *GAMM-Mitteilungen* (2021)
3. Bohn, B., Rieger, C., Griebel, M.: A representer theorem for deep kernel learning. *J. Mach. Learn. Res.* **20**, 64–1 (2019)
4. Cho, K., van Merriënboer, B., Bahdanau, D., Bengio, Y.: On the properties of neural machine translation: Encoder-decoder approaches. In: *Proc. of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*. pp. 103 – 111. Association for Computational Linguistics, Stroudsburg (2014)
5. Goodfellow, I., Bengio, Y., Courville, A.: *Deep Learning*. MIT Press (2016)
6. Kimeldorf, G.S., Wahba, G.: A correspondence between Bayesian estimation on stochastic processes and smoothing by splines. *Ann. Math. Statist.* **41**(2), 495–502 (1970)
7. Kurz, M., Beck, A.: A machine learning framework for LES closure terms. arXiv preprint arXiv:2010.03030 (2020)
8. Santin, G., Haasdonk, B.: Kernel methods for surrogate modeling. ArXiv 1907.10556 (2019), <https://arxiv.org/abs/1907.10556>
9. Serrin, J.: *Mathematical Principles of Classical Fluid Mechanics*, pp. 125–263. Springer Berlin Heidelberg, Berlin, Heidelberg (1959)
10. Wenzel, T., Santin, G., Haasdonk, B.: Structured Deep Kernel Networks (2021), in preparation