Journal of Scientific Computing (2021) 88:22
https://doi.org/10.1007/s10915-021-01532-w

Numerical Solution of the Parametric Diffusion Equation by Deep Neural Networks

Moritz Geist[1] · Philipp Petersen[2] · Mones Raslan[1] · Reinhold Schneider[1] · Gitta Kutyniok[3,4]

Received: 27 April 2020 / Revised: 25 March 2021 / Accepted: 18 May 2021 / Published online: 5 June 2021

Abstract
We perform a comprehensive numerical study of the effect of approximation-theoretical results for neural networks on practical learning problems in the context of numerical analysis. As the underlying model, we study the machine-learning-based solution of parametric partial differential equations. Here, approximation theory for fully-connected neural networks predicts that the performance of the model should depend only very mildly on the dimension of the parameter space and is determined by the intrinsic dimension of the solution manifold of the parametric partial differential equation. We use various methods to establish comparability between test-cases by minimizing the effect of the choice of test-cases on the optimization and sampling aspects of the learning problem. We find strong support for the hypothesis that approximation-theoretical effects heavily influence the practical behavior of learning problems in numerical analysis. Turning to practically more successful and modern architectures, at the end of this study we derive improved error bounds by focusing on convolutional neural networks.

Keywords Neural networks · Parametric diffusion equation · Numerical approximation · Neural network capacity

Mathematics Subject Classification 35J99 · 41A25 · 41A30 · 68T05 · 65N30

1 Introduction

This work studies the problem of numerically solving a specific parametric partial differential equation (PPDE) by training and applying neural networks (NNs). The central goal of the following exposition is to identify those key aspects of a parametric problem that render the problem harder or simpler to solve for methods based on NNs.

The underlying mathematical problem, the solution of PPDEs, is a standard problem in applied sciences and engineering. In this model, certain parts of a PDE such as the boundary conditions, the source terms, or the shape of the domain are controlled through a set of

Moritz Geist, Philipp Petersen and Mones Raslan have contributed equally to this work.

Extended author information available on the last page of the article

parameters, e.g., [30,56]. In some applications where PDEs need to be evaluated very often or in real-time, individually solving the underlying PDEs for each choice of parameters becomes computationally infeasible. In this case, it is advisable to invoke methods that leverage on the joint structure of all the individual problems. A typical approach is that of constructing a *reduced basis* associated with the problem. With respect to this basis, the computational complexity of solving the PPDE is then significantly reduced, e.g., [30,49,56,62].

Recently, as an alternative or to augment the reduced basis method, approaches were introduced that attempt to learn the parameter-to-solution map through methods of machine learning. We will provide a comprehensive overview of related approaches in Sect. 1.5. One approach is to train a NN to fit the discretized parameter-to-solution map, i.e., a map taking a parameter to a finite-element discretization of the solution of the associated PDEs. This approach has already been analyzed theoretically in [37] where it was shown from an approximation-theoretical point of view that the hardness of representing the parameter-to-solution map by NNs is determined by a highly problem-specific notion of complexity that depends (in some cases) only very mildly on the dimension of the parameter space.

In this work, *we study the problem of learning the discretized parameter-to-solution map in practice*. We hypothesize that the approximation-theoretical capacity of a NN architecture is one of the central factors in determining the practical difficulty level of the learning problem.

The motivation for this analysis is twofold: first, we regard this as a general analysis of the feasibility of approximation-theoretical arguments in the study of deep learning. Second, specifically for the problem of numerical solution of PPDEs, we consider it important to identify which characteristics of a parametric problem determine its practical hardness. This is especially relevant to identify in which areas the application of this model is appropriate. We outline these two points of motivation in Sect. 1.1. The design of the numerical experiment based on fully-connected neural networks is presented in Sect. 1.2, whereas we focus on convolutional neural networks in Sect. 1.3. Afterwards, we give a high-level report of our findings in Sect. 1.4.

1.1 Motivation

Below, we describe the two main reasons that motivate this numerical study.

1.1.1 Understanding of Deep Learning in General

A typical learning problem consists of an unknown data model, a hypothesis class, and an optimization procedure to identify the best fit in the hypothesis class to the observed (sampled) data, e.g., [17,18]. In a deep learning problem, the hypothesis class is the set of NNs with a specific architecture.

The approximation-theoretical point of view analyzes the trade-off between the capacity of the hypothesis class and the complexity of the data model. In this sense, this point of view describes only one aspect of the learning problem.

In the framework of approximation theory, there are precise ways to assess the hardness of an underlying problem. Concretely, this is done by identifying the rate by which the misfit between the hypothesis class and the data model decreases for sequences of growing hypothesis classes. For example, one common theme in the literature is the observation that for certain function classes NNs do not admit a curse of dimension, i.e., their approximation rates do not deteriorate exponentially fast with increasing input dimension, e.g., [7,54,67].

Another theme is that classes of smooth functions can be approximated more efficiently than classes of rougher functions, e.g., [45,46,50,52,75].

While these results offer some interpretation of why a certain problem should be harder or simpler, it is not clear how relevant these results are in practice. Indeed, there are at least three issues that call the approximation-theoretical explanation for a practical learning problem into question:

- *Tightness of the upper bounds* Approximation-theoretical bounds usually describe worst-case error estimates for whole classes of functions. For individual functions or subsets of these function classes, there is no guarantee that one could not achieve a significantly better approximation rate.
- *Optimization and sampling prevent approximation theoretical effect from materializing* As explained at the beginning of this section, the learning problem consists of multiple aspects, one of which is the ability of the hypothesis class to describe the data. Two further aspects are how well the sampling of the data model describes the true model and how well the optimization procedure performs in finding the best fit to the sampled data. Since the underlying optimization problem of deep learning is in general non-convex, it is conceivable that, while there theoretically exists a very good approximation of a function by a NN, finding it in practice is highly unlikely. Moreover, it is certainly possible that the sampling process does not contain sufficient information to guarantee that the optimization routine will identify the theoretically best approximation.
- *Asymptotic estimates* All approximation-theoretical results mentioned until here and almost all in the literature describe the capacity of NNs to represent functions approximately with accuracy ε for sufficiently large architectures only in a regime where ε tends to zero and the size of the architecture grows accordingly. The associated approximation rates may contain arbitrarily large implicit constants, and therefore it is entirely unclear if changes to the trade-off between the complexity of the data model and the size of the architecture have the theoretically predicted impact for moderately-sized practical learning problems.

We believe that, to understand the effect of approximation-theoretical capacities of NNs in practical learning scenarios, the learning problem associated with the parameter-to-solution map in a PPDE occupies a special role: It is in essence, a high-dimensional approximation problem of a function that has a very strong low-dimensional, but highly non-trivial structure. What is more is that one can, to a certain extent, control the complexity of the problem, as we have seen in [37]. In this context, we can ask ourselves the following questions: Do we observe a curse of dimensionality in the practical solution of the problem? If not, how does the difficulty in practice scale with the parameter dimension? On which characteristics of the problem does the hardness of the practical solution thereof depend?

If we study these questions numerically, then the answers can be compared with the predictions from approximation-theoretical considerations. If the predictions coincide with the observed behavior and other causes, such as artefacts from the optimization and sampling procedure, can be ruled out, then we can view these experiments as a strong support for the practical relevance of approximation-theoretical arguments.

Because of this, we study the aforementioned questions in an extensive numerical experiment that will be described in Sect. 1.2 below.

1.1.2 Feasibility of the Machine-Learning-Based Solution of Parametric PDEs

The method (as described in [37]) of learning the parameter-to-solution map has at least two major advantages over classical approaches to solve PPDEs: First of all, the setup is completely independent of the underlying PPDE. This versatility of NNs could be quite desirable in an environment where many substantially different PPDEs are treated. Second, because this approach is fully data-driven, we do not require any knowledge of the underlying PDE. Indeed, as long as sufficiently many data points are supplied, for example, from a physical experiment, the approach could be feasible under high uncertainty of the model.

The main drawback of the method is the lack of theoretical understanding thereof. Moreover, for the theoretical results that do exist, we lack any evaluation of how pertinent the theoretical observations are for practical behavior. Most importantly, we do not have an a priori assessment for the practical feasibility of certain problems.

In [37], we observed that the complexity of the solution manifold, i.e., the set of all solutions of the PDE, is a central quantity involved in upper bounding the hardness of approximating the parameter-to-solution map with a NN. In practice, it is unclear to what extent this notion is appropriate and if the complexity of the solution manifold influences the performance of the method at all.

In the numerical experiment described in the next section, we explore the performance of the learning approach for various test-cases with different intrinsic complexities and observe the sensitivity of the method to the different setups.

1.2 The Experiment

To analyze the approximation-theoretical effect of the architecture on the overall performance of the learning problem in practice, we train a fully connected NN as considered in [37] on a variety of datasets stemming from different parameter choices of the parametric diffusion equation. The design of the data sets is such that we vary the relationship between the capacity of the architecture and the complexity of the data and report the effect on the overall performance.

In designing such an experiment, we face *three fundamental challenges hindering the comparability between test-cases*:

- *Effect of the optimization procedure* Since the algorithms used to train neural networks are not guaranteed to converge to the optimal solution, there may be a significant effect of the chosen optimization procedure on the performance of the model. The success of the optimization routine may depend on the complexity of the data model this interplay may be a much stronger factor in the performance of the method than the capacity of the architecture to fit the data model. In addition, the optimization routine may be affected by random aspects of the set-up, such as the initialisation, which may obfuscate the effect of the complexity of the solution manifold.
- *Effect of the sampling procedure* We train our network based on a finite number of samples of the true solution. The number and choice of samples could have a non-negligible effect on the overall performance and potentially affect some test-cases more than others.
- *Quantification of the intrinsic complexity* While we have theoretically established that the complexity of the solution manifold is the main factor in upper-bounding the hardness of the problem in the approximation-theoretical framework, we cannot, in practice, quantify this complexity.

In view of the aforementioned obstacles, the design of a meaningful experiment is a significant challenge. To overcome the issues described above, we introduce the following measures:

- *Keeping the architecture fixed* An approximation-theoretical result on NNs is based on three ingredients. A function class $\mathcal{C}$, a worst-case accuracy $\varepsilon > 0$, and the size of the architecture.
  Whenever one of these hyper-parameters—the function class, the accuracy, or the architecture—is fixed, one can theoretically describe how changing a second parameter influences the last one. For example, for fixed $\mathcal{C}$, an approximation-theoretical statement yields an estimate of the necessary size of the architecture to achieve an accuracy of ε.
  Because of the potentially strong impact of the architecture on the optimization procedure, we expect that the most sensible point of view to test numerically is that where the architecture remains fixed while we vary the function class $\mathcal{C}$ and observe ε. This way, we can guarantee that the influence of the architecture on the optimization procedure is the same between test-cases.
- *Averaging over test-cases* We perform each experiment multiple times and compare the results. This way, we can see how sensitive the experiment is to random initialization of the model and if certain results are due to the optimization routine getting stuck in local minima.
- *Analyzing the convergence behavior a posteriori* We are not aware of any method to guarantee a priori that the choice of the data model would not influence the convergence behavior. We do, however, analyze the convergence after the experiment to see if there are fundamental differences between our test-cases. This analysis reveals no significant differences between all the setups and therefore indicates that the effect of the data model on the optimization procedure is very similar between test-cases.
- *Establishing independence of sample generation* We run the experiment multiple times for various numbers of training samples N chosen in the same way—uniformly at random—in every test-case. Between the choices of N, we observe a linear dependence of the achieved accuracy on N. This indicates that the influence of the number of N on the performance of the method is the same for all test-cases.
- *Design of semi-ordered test-cases* While we are not able to assess the intrinsic complexity exactly, it is straight-forward to construct series of test-cases with increasing complexity. In this sense, we can introduce a semi-ordering of test-cases according to their complexity and observe to what extent the performance of the method follows this ordering.
  In addition, we can measure the intrinsic complexity of the parametric problem via a heuristic. Here we quantify, via the singular value decomposition, how well a random point cloud of solutions of the problem can be represented by a linear model.

We present the construction of the test-cases in Sect. 4 and discuss the measures taken to remove effects caused by the optimization and sampling procedures in greater detail in "Appendix A". All of our test-cases consider the following parametric diffusion equation

$$-\nabla \cdot (a_y(\mathbf{x}) \cdot \nabla u_y(\mathbf{x})) = f(\mathbf{x}), \quad \text{on } \Omega = (0,1)^2, \quad u_y|_{\partial\Omega} = 0,$$

where $f \in L^2(\Omega)$ and $a_y \in L^\infty(\Omega)$, is a diffusion coefficient depending on a parameter $y \in \mathcal{Y}$. In our test-cases below, we learn a discretization of the map $\mathbb{R}^p \supset \mathcal{Y} \ni y \mapsto u_y$, where $p \in \mathbb{N}$, for various choices of parametrizations

$$\mathbb{R}^p \supset \mathcal{Y} \ni y \mapsto a_y. \tag{1.1}$$

Concretely, we vary the following characteristics of the parametrizations and observe the effect on the overall performance of the learning problem:

- *Type of parametrization* We choose test-cases which differ with respect to the following characteristics: First, we study parametrizations (1.1) of various degrees of smoothness. Second, we study test-cases where the parametrization (1.1) is affine-linear or non-linear. Third, we consider cases, where $a_y = \sum_{i=1}^{p} \widetilde{a}_{y_i}$ for $\widetilde{a}_{y_i} \in L^\infty(\Omega)$ and where the supports of $(\widetilde{a}_{y_i})_{i=1}^{p}$ overlap or have various degrees of separation.
- *Dimension of parameter space* The discretization of our solution space is done on the maximal computationally feasible grid (with respect to our workstation). We have chosen the dimensions p of the parameter spaces in such a way that the resolutions of the parametrized solutions are still meaningful with respect to the underlying discretization.
- *Complexity hyper-parameters* To generate comparable test-cases with increasing complexities, we include two types of hyper-parameters into the data-generation process. One that directly influences the ellipticity of the problem and another that introduces a weighting of the parameter values.

We expect that these tests yield answers to the following questions: How versatile is the approach? Does it perform well only for special types of parametrizations or is it generally applicable? Do we observe a curse of dimensionality and how much does the performance of the learning method depend on the dimension of the parameter space? How strongly does the performance of the learning method depend on the intrinsic complexity of the data?

1.3 Modern Architectures

The previously described experiment aims at understanding whether the theoretically predicted effect of Kutyniok et al. [37] persists in practice. In many practical applications, it is, however, common to use much more refined architectures than the fully-connected architecture studied in [37]. To understand if and to what extent our findings also affect more modern architectures, we perform a second set of experiments using deep convolutional neural networks [39].

Convolutional neural networks are build by alternating convolutional blocks with pooling blocks. A convolutional block can be interpreted as a regular fully-connected block, however, with the weight matrices required to be block-circular matrices. A pooling block acts as a form of subsampling. Depending on the pooling operation, we can consider convolutional neural networks as special fully-connected neural networks [53,77]. The restriction is, however, so strong that the estimates of Kutyniok et al. [37] do not extend to general convolutional neural networks.

The impact of this second experiment is expected to be twofold: First, if the results mirror the outcome of the first experiment, then we have grounds to believe that also the performance of modern architectures solving parametric problems is critically determined by the intrinsic dimension of the solution manifold. Second, if the effect of the intrinsic dimension is similar to the first experiment, then we have observed the same effect over multiple modalities. This serves as an additional validation that the effect observed is not based on the optimization procedure since this should affect substantially different architectures very differently.

1.4 Our Findings

In the numerical experiments, which we report in Sects. 4.3 and 5 and evaluate in Sects. 4.4 and 5.2, respectively, we find that the proposed method is very sensitive to the underlying type of test-case. Indeed, we observe qualitatively different scaling behaviors of the achieved error with the dimension p of the parameter space between different test-cases. Concretely, we observe the following asymptotic behavior of the errors in different test-cases: $\mathcal{O}(1)$, $\mathcal{O}(\log(p))$ and $\mathcal{O}(p^k)$ for $p \to \infty$ and $k > 0$, where k depends on one of the complexity hyper-parameters. Notably, we do not observe a scaling according to the curse of dimensionality, i.e., an error scaling exponentially with p, in any of the test-cases. We also observe that the achieved errors obey the semi-ordering of complexities of the test-cases and are compatible with a heuristic complexity measure. This shows that the method is very versatile and can be applied for various settings. Moreover, the complexity of the solution manifold appears to be a sensible predictor for the efficiency of the method.

In addition, we observe that the numerical results agree with the predictions that can be made via approximation-theoretical considerations. Due to the careful design of this experiment, we can exclude effects associated with the optimization and sampling procedures. In addition, we even observed the described effect in a very practical regime, where the theoretical analysis is not yet available. This indicates the practical relevance of approximation-theoretical results for this particular problem and for deep learning problems in general.

1.5 Related Works

The practical application of NNs in the context of PDEs dates back to the 1990s [38]. However, in recent years the topic again gained traction in the scientific community driven by the ever-increasing availability of computational power. Much of this research can be condensed into three main directions: Learning the solution of a single PDE, system identification, and goal-oriented approaches. The first of these directions uses NNs to directly model the solution of a (in some cases user-specified) single PDE [42,58,63,73,74], an SDE [8,72], or even the joint solution for multiple boundary conditions [68]. These methods mostly rely on the differential operator of the PDE to evaluate the loss, but other approaches do exist [27]. In system identification, one tries to discover an underlying physical law from data by reverse-engineering the PDE. This can be done by attempting to uncover a hidden parameter of a known equation [59], or modeling physical relations [11,57]. Conversely, goal-oriented approaches, try to infer a quantity of interest stemming from the solution of an underlying PDE. For example, NNs can be used as a surrogate model to directly learn the quantity of interest and thereby circumvent the necessity of explicitly solving the equation [35]. A practical example for this is given by the ground state energy of a molecule which is derived from the solution of the electronic Schrödinger equation. This task has been efficiently solved by graph NNs [22,43,66] or hybrid approaches [28]. Furthermore, building a surrogate model can be especially useful in uncertainty quantification [69]. NNs can also aid classical methods in solving goal-oriented tasks [15,44]. In addition to the aforementioned research directions, further work has been done on fusing NNs with classical numerical methods to assist, for example, in model-order reduction [40,61].

Our work focuses on PPDEs and more specifically we are interested in learning the mapping from the parameter to the coefficients of the high-fidelity solution. Related but different approaches were analyzed in [19,31,69], where the solution of the PPDE is learned

in an already precomputed reduced basis or at point evaluations in fixed spatial coordinates. The recent work [2] performs a comprehensive numerical study in the context of PPDEs that sheds light on the influence of the size of the training set. The work [64] employs NNs for model-order reduction in the context of transient flows.

On the theoretical side, the majority of works analyzing the power of NNs for the solution of (parametric) PDEs is concerned with an approximation-theoretical approach. Notable examples of such works include [8,9,12,21,24–26,33,34,72], in which it is shown that NNs can overcome the curse of dimensionality in the approximative solution of some specific single PDE. In the same framework, it was shown in [12] how estimates on the approximation error imply bounds on the generalization error. Concerning the theoretical analysis of PPDEs, we mention [13,29,37,47,51,65]. We will describe the results of the first two works in more detail in Sect. 3.2. The work [29] is concerned with an efficient approximation of a map that takes a noisy solution of a PDE as an input and returns a quantity of interest.

Additionally, we wish to mention that there exists a multitude of approaches (which are not necessarily directly RBM-or NN-related) that study the approximation of the parameter-to-solution map of PPDEs. These include methods based on sparse polynomials (see for instance [16,32] and the references therein), tensors (see for instance [6,20] and the references therein) and compressed sensing (see for instance [60,71] and the references therein).

Parametric PDEs also appear in the context of stochastic PDEs or PDEs with random coefficients (see for instance [55]) and have been theoretically examined under the perspective of *uncertainty quantification*. For the sake of brevity, we only mention [16] and the references therein.

Finally, we mention that a comprehensive numerical study analyzing to what extent the approximation theoretical findings of NNs (not in the context of PPDEs) are visible in practice has been carried out in [3]. Similarly, in [23], a numerical algorithm that reproduces certain approximation-theoretically established exponential convergence rates of NNs was studied. The approximation rates of [14] were also numerically reproduced in that paper.

1.6 Outline

We start by describing the parametric diffusion equation and how we discretize it in Sect. 2. Then, we provide a formal introduction to NNs and a review of the approximation-theoretical results of NNs for parameter-to-solution maps in Sect. 3. In Sect. 4, we describe our numerical experiment based on fully-connected NNs. We start by stating three hypotheses underlying the examples in Sect. 4.1, before describing the set-up of our experiments in Sect. 4.2. After that, we present the results of the experiments in Sect. 4.3 and evaluate the observations in Sect. 4.4. Finally, in Sect. 5 we augment our results by repeating the experiments with a more modern CNN architecture and compare its performance to the fully-connected NN in Sect. 5.2. In "Appendix A", we present additional measures taken to ensure comparability between test-cases and to remove all artefacts stemming from the optimization procedure.

2 The Parametric Diffusion Equation

In this section, we will introduce the abstract setup and necessary notation that we will consider throughout this paper. First of all, we will introduce the *parameter-dependent diffusion equation* in Sect. 2.1. Afterwards, in Sect. 2.2, we recapitulate some basic facts about *high-fidelity discretizations* and introduce the *discretized parameter-to-solution map*.

2.1 The Parametric Diffusion Equation

Throughout this paper, we will consider the *parameter-dependent diffusion equation* with homogeneous Dirichlet boundary conditions

$$-\nabla \cdot (a(\mathbf{x}) \cdot \nabla u_a(\mathbf{x})) = f(\mathbf{x}), \quad \text{on } \Omega = (0,1)^2, \quad u|_{\partial\Omega} = 0, \tag{2.1}$$

where $f \in L^2(\Omega)$ is the parameter-independent right-hand side, $a \in \mathcal{A} \subset L^\infty(\Omega)$, and $\mathcal{A}$ constitutes some compact set of *parametrized diffusion coefficients*. In the following, we will examine different varieties of parametrized diffusion coefficient sets $\mathcal{A}$. Following [16] (by restricting ourselves to the case of finite-dimensional parameter spaces), we will always describe the elements of $\mathcal{A}$ by elements in $\mathbb{R}^p$ for some $p \in \mathbb{N}$. To be more precise, we will assume that

$$\mathcal{A} = \left\{ a_y \colon y \in \mathcal{Y} \right\}, \tag{2.2}$$

where $\mathcal{Y} \subset \mathbb{R}^p$ is the *compact parameter space*.

A common assumption on the set $\mathcal{A}$, present in the first test-cases which we will describe below and especially convenient for the theoretical analysis of the problem, is given by *affine parametrizations* of the form

$$\mathcal{A} = \left\{ a_y = a_0 + \sum_{i=1}^{p} y_i a_i \colon y = (y_i)_{i=1}^p \in \mathcal{Y} \right\}, \tag{2.3}$$

where the functions $(a_i)_{i=0}^p \subset L^\infty(\Omega)$ are fixed.

After reparametrization, we consider the following problem, given in its variational formulation:

$$b_y\left(u_y, v\right) = \int_\Omega f(\mathbf{x}) v(\mathbf{x}) \, \mathrm{d}\mathbf{x}, \quad \text{for all } y \in \mathcal{Y},\ v \in \mathcal{H}, \tag{2.4}$$

where

$$b_y \colon \mathcal{H} \times \mathcal{H} \to \mathbb{R},\ (u, v) \mapsto \int_\Omega a_y(\mathbf{x}) \nabla u(\mathbf{x}) \nabla v(\mathbf{x}) \, \mathrm{d}\mathbf{x},$$

and $u_y \in \mathcal{H} := H_0^1(\Omega)$ is the solution.[1]

We will consider experiments in which the involved bilinear forms are *uniformly continuous* and *uniformly coercive* in the sense that there exist $C_{\mathrm{cont}}, C_{\mathrm{coer}} > 0$ with

$$\left|b_y(u, v)\right| \leq C_{\mathrm{cont}} \|u\|_{\mathcal{H}} \|v\|_{\mathcal{H}}, \quad \inf_{u \in \mathcal{H} \setminus \{0\}} \frac{b_y(u,u)}{\|u\|_{\mathcal{H}}^2} \geq C_{\mathrm{coer}}, \quad \text{for all } u, v \in \mathcal{H},\ y \in \mathcal{Y}.$$

By the Lax–Milgram lemma (see [56, Lemma 2.1]), the problem of (2.4) is *well-posed*, i.e., for every $y \in \mathcal{Y}$ there exists exactly one $u_y \in \mathcal{H}$ such that (2.4) is satisfied and u_y depends continuously on f.

[1] Throughout this paper, we denote by $\mathcal{H}$ the space $H_0^1(\Omega) := \{u \in H^1(\Omega) \colon u|_{\partial\Omega} = 0\}$, where $H^1(\Omega) := W^{1,2}(\Omega)$ is the *first-order Sobolev space* and where $\partial\Omega$ denotes the *boundary of* Ω. On this space, we consider the norm $\|u\|_{\mathcal{H}} = \|u\|_{H_0^1(\Omega)} := \|u\|_{H^1(\Omega)} = \left(\sum_{|\mathbf{a}| \leq 1} \|D^{\mathbf{a}} u\|_{L^2(\Omega)}^2\right)^{1/2}$.

2.2 High-Fidelity Discretizations

In practice, one cannot hope to solve (2.4) exactly for every $y \in \mathcal{Y}$. Instead, if we assume for the moment that y is fixed, a common approach towards the calculation of an approximate solution of (2.4) is given by the *Galerkin method*, which we will describe briefly below following [30, Appendix A] and [56, Chapter 2.4]. In this framework, instead of solving (2.4), one solves a discrete scheme of the form

$$b_y\left(u_y^{\mathrm{h}}, v\right) = \int_\Omega f(\mathbf{x}) v(\mathbf{x})\, \mathrm{d}\mathbf{x} \qquad \text{for all } v \in U^{\mathrm{h}}, \tag{2.5}$$

where $U^{\mathrm{h}} \subset \mathcal{H}$ is a subspace of $\mathcal{H}$ with $\dim\left(U^{\mathrm{h}}\right) < \infty$ and $u_y^{\mathrm{h}} \in U^{\mathrm{h}}$ is the solution of (2.5). Let us now assume that U^{h} is given. Moreover, let $D{:=}\dim\left(U^{\mathrm{h}}\right)$, and let $(\varphi_i)_{i=1}^D$ be a basis for U^{h}. Then the *stiffness matrix* $\mathbf{B}_y^{\mathrm{h}}{:=}(b_y(\varphi_j, \varphi_i))_{i,j=1}^D$ is non-singular and positive definite. The solution u_y^{h} of (2.5) satisfies

$$u_y^{\mathrm{h}} = \sum_{i=1}^D (\mathbf{u}_y^{\mathrm{h}})_i \varphi_i,$$

where $\mathbf{u}_y^{\mathrm{h}}{:=}(\mathbf{B}_y^{\mathrm{h}})^{-1}\mathbf{f}_y^{\mathrm{h}} \in \mathbb{R}^D$ and $\mathbf{f}_y^{\mathrm{h}}{:=}\left(\int_\Omega f(\mathbf{x})\varphi_i(\mathbf{x})\, \mathrm{d}\mathbf{x}\right)_{i=1}^D \in \mathbb{R}^D$. By Cea's Lemma (see [56, Lemma 2.2.]), u_y^{h} is, up to a universal constant, a best approximation of u_y in U^{h}.

In this framework, we can now define the central object of interest which is the map taking an element from the parameter space $\mathcal{Y}$ to the discretized solution $\mathbf{u}_y^{\mathrm{h}}$.

Definition 2.1 Let $\Omega = (0,1)^2$, $U^{\mathrm{h}} \subset \mathcal{H}$ be a finite dimensional space, $\mathcal{A} \subset L^\infty(\Omega)$ with $\mathcal{Y} \subset \mathbb{R}^p$ for $p \in \mathbb{N}$ be as in (2.2). Then we define the *discretized parameter-to-solution map (DPtSM)* by

$$\mathcal{P}\colon\ \mathcal{Y} \to \mathbb{R}^D, \qquad y \mapsto \mathcal{P}(y){:=}\mathbf{u}_y^{\mathrm{h}}.$$

Remark 2.2 The DPtSM $\mathcal{P}$ is a potentially nonlinear map from a p-dimensional set to a D-dimensional space. Therefore, without using the information that $\mathcal{P}$ has a very specific structure described through $\mathcal{A}$ and the PDE (2.1), a direct approximation of $\mathcal{P}$ as a high-dimensional smooth function will suffer from the curse of dimensionality [10,48].

Before we continue, let us introduce some crucial notation. Later, we need to compute the Sobolev norms of functions $v \in \mathcal{H}$. This will be done via a vector representation $\mathbf{v}$ of v with respect to the high-fidelity basis $(\varphi_i)_{i=1}^D$. We denote by $\mathbf{G}{:=}\left(\langle\varphi_i, \varphi_j\rangle_{\mathcal{H}}\right)_{i,j=1}^D \in \mathbb{R}^{D\times D}$ the symmetric, positive definite *Gram matrix* of the basis functions $(\varphi_i)_{i=1}^D$. Then, for any $v \in U^{\mathrm{h}}$ with coefficient vector $\mathbf{v}$ with respect to the basis $(\varphi_i)_{i=1}^D$ we have[2] (see [56, Equation 2.41]) $|\mathbf{v}|_{\mathbf{G}} := \left|\mathbf{G}^{1/2}\mathbf{v}\right| = \|v\|_{\mathcal{H}}$. In particular, $\left\|u_y^{\mathrm{h}}\right\|_{\mathcal{H}} = |\mathbf{u}_y^{\mathrm{h}}|_{\mathbf{G}}$, for all $y \in \mathcal{Y}$.

3 Approximation of the Discretized Parameter-to-Solution Map by Realizations of Neural Networks

In this section, we describe the approximation-theoretical motivation for the numerical study performed in this paper. We present a formal definition of NNs below. In Question 3.5,

[2] In this paper, $|\mathbf{x}|$ denotes the *Euclidean norm* of $\mathbf{x} \in \mathbb{R}^n$.

we present the underlying approximation-theoretical question of the considered learning problem. Thereafter, we recall the results of Kutyniok et al. [37] showing that one can upper bound the approximation rates that NNs obtain when approximating the DPtSM through an implicit notion of complexity of the DPtSM.

3.1 Neural Networks

NNs describe functions of compositional form that result from repeatedly applying affine linear maps and a so-called activation function. From an approximation-theoretical point of view, it is sensible to count the number of active parameters of a NN. To associate a meaningful and mathematically precise notion of the number of parameters to a NN, we differentiate here between *neural networks* which are sets of matrices and vectors, essentially describing the parameters of the NN, and *realizations of neural networks* which are the associated functions. Concretely, we make the following definition:

Definition 3.1 Let $n, L \in \mathbb{N}$. A *neural network* Φ *with input dimension n and L layers* is a sequence of matrix-vector tuples

$$\Phi = \big((\mathbf{A}_1, \mathbf{b}_1), (\mathbf{A}_2, \mathbf{b}_2), \ldots, (\mathbf{A}_L, \mathbf{b}_L)\big),$$

where $N_0 = n$ and $N_1, \ldots, N_L \in \mathbb{N}$, and where each $\mathbf{A}_\ell$ is an $N_\ell \times N_{\ell-1}$ matrix, and $\mathbf{b}_\ell \in \mathbb{R}^{N_\ell}$.

If Φ is a NN as above, $K \subset \mathbb{R}^n$, and if $\varrho\colon \mathbb{R} \to \mathbb{R}$ is arbitrary, then we define the associated *realization of* Φ *with activation function* ϱ *over* K (in short, the ϱ*-realization of* Φ *over* K) as the map $\mathrm{R}_\varrho^K(\Phi)\colon K \to \mathbb{R}^{N_L}$ such that $\mathrm{R}_\varrho^K(\Phi)(\mathbf{x}) = \mathbf{x}_L$, where $\mathbf{x}_L$ results from the following scheme:

$$\begin{aligned} &\mathbf{x}_0 := \mathbf{x}, \\ &\mathbf{x}_\ell := \varrho(\mathbf{A}_\ell\, \mathbf{x}_{\ell-1} + \mathbf{b}_\ell), \qquad \text{for } \ell = 1, \ldots, L-1, \\ &\mathbf{x}_L := \mathbf{A}_L\, \mathbf{x}_{L-1} + \mathbf{b}_L, \end{aligned}$$

and where ϱ acts componentwise, that is, $\varrho(\mathbf{v}) := (\varrho(v_1), \ldots, \varrho(v_m))$ for all $\mathbf{v} = (v_1, \ldots, v_s) \in \mathbb{R}^s$.

We call $N(\Phi) := n + \sum_{j=1}^{L} N_j$ the *number of neurons of the NN* Φ and L the *number of layers*. We call $M(\Phi) := \sum_{\ell=1}^{L} \|\mathbf{A}_\ell\|_0 + \|\mathbf{b}_\ell\|_0$ the *number of non-zero weights of* Φ. Moreover, we refer to N_L as *the output dimension* of Φ. Finally, we refer to $(N_0, \ldots, N_L)$ as the *architecture* of Φ.

We consider the following family of activation functions:

Definition 3.2 For $\alpha \in [0, 1)$, we define by $\varrho_\alpha(x) := \max\{x, \alpha x\}$ the *α-leaky rectified linear unit (α-LReLU)*. The activation function $\varrho_0 = \max\{x, 0\}$ is called the *rectified linear unit (ReLU)*.

Remark 3.3 For every $\alpha \in (0, 1)$ it holds that for all $x \in \mathbb{R}$

$$\varrho_0(x) = \frac{1}{1-\alpha^2}\,(\varrho_\alpha(x) + \alpha\varrho_\alpha(-x)) \text{ and } \varrho_\alpha(x) = \varrho_0(x) - \alpha\varrho_0(-x).$$

Hence, for every $\alpha \in (0, 1)$, we can represent the ReLU as the sum of two rescaled α-LReLUs and vice versa. If we define for $n \in \mathbb{N}$

$$\mathbf{P}_n(\mathbf{x}) := (x_1, -x_1, x_2, -x_2, \ldots, x_n, -x_n), \text{ for } \mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n,$$

$$\mathbf{Q}_{n,\alpha}(\mathbf{x}) := (x_1 - \alpha x_2, x_3 - \alpha x_4, \dots, x_{2n-1} - \alpha x_{2n}), \text{ for } \mathbf{x} = (x_1, \dots, x_{2n}) \in \mathbb{R}^{2n},$$
$$\mathbf{T}_{n,\alpha}(\mathbf{x}) := \frac{1}{1-\alpha^2}(x_1 + \alpha x_2, x_3 + \alpha x_4, \dots, x_{2n-1} + \alpha x_{2n}), \text{ for } \mathbf{x} = (x_1, \dots, x_{2n}) \in \mathbb{R}^{2n},$$

then, for NNs

$$\begin{aligned}
\Phi_1 &= \big((\mathbf{A}_1, \mathbf{b}_1), (\mathbf{A}_2, \mathbf{b}_2), \dots, (\mathbf{A}_L, \mathbf{b}_L)\big),\\
\Phi_2 &= \big((\mathbf{P}_{N_1}\mathbf{A}_1, \mathbf{P}_{N_1}\mathbf{b}_1), (\mathbf{P}_{N_2}\mathbf{A}_2\mathbf{Q}_{N_1,\alpha}, \mathbf{P}_{N_2}\mathbf{b}_2), \dots,\\
&\qquad (\mathbf{P}_{N_{L-1}}\mathbf{A}_{L-1}\mathbf{Q}_{N_{L-2},\alpha}, \mathbf{P}_{N_{L-1}}\mathbf{b}_{L-1}), (\mathbf{A}_L\mathbf{Q}_{N_{L-1},\alpha}, \mathbf{b}_L)\big),\\
\Phi_3 &= \big((\mathbf{P}_{N_1}\mathbf{A}_1, \mathbf{P}_{N_1}\mathbf{b}_1), (\mathbf{P}_{N_2}\mathbf{A}_2\mathbf{T}_{N_1,\alpha}, \mathbf{P}_{N_2}\mathbf{b}_2), \dots,\\
&\qquad (\mathbf{P}_{N_{L-1}}\mathbf{A}_{L-1}\mathbf{T}_{N_{L-2},\alpha}, \mathbf{P}_{N_{L-1}}\mathbf{b}_{L-1}), (\mathbf{A}_L\mathbf{T}_{N_{L-1},\alpha}, \mathbf{b}_L)\big),
\end{aligned}$$

we have that for $K \subset \mathbb{R}^n$ it holds that $\mathrm{R}^K_{\varrho_0}(\Phi_1) = \mathrm{R}^K_{\varrho_\alpha}(\Phi_3)$ and $\mathrm{R}^K_{\varrho_\alpha}(\Phi_1) = \mathrm{R}^K_{\varrho_0}(\Phi_2)$. Moreover, it is not hard to see that $M(\Phi_1) \leq M(\Phi_2)$, $M(\Phi_3)$ and $M(\Phi_2)$, $M(\Phi_3) \leq 4M(\Phi_1)$. Therefore, we have that for every $\alpha_1, \alpha_2 \in [0, 1)$ and every function $f : \mathbb{R}^n \to \mathbb{R}^{N_L}$ of a function space X such that

$$\left\| f - \mathrm{R}^K_{\varrho_{\alpha_1}}(\Phi) \right\|_X \leq \varepsilon$$

for a NN Φ implies that there exists another NN $\widetilde{\Phi}$ with $L(\widetilde{\Phi}) = L(\Phi)$ and $M(\widetilde{\Phi}) \leq 16M(\Phi)$ such that

$$\left\| f - \mathrm{R}^K_{\varrho_{\alpha_2}}(\Phi) \right\|_X \leq \varepsilon.$$

In other words, *up to a multiplicative constant the parameter α of the α-LReLU does not influence the approximation properties of realizations of NNs.*

Remark 3.4 While Remark 3.3 shows that all α-LReLUs yield, in principle, the same approximation behavior, these activation functions still display quite different behavior during the training phase of NNs, where a non-vanishing parameter α can help avoid the occurrence of dead neurons.

3.2 Approximation of the Discretized Parameter-to-Solution Map by Realizations of Neural Networks

We can quantify the capability of NNs to represent the DPtSM by answering the following question:

Question 3.5 *Let $p, D \in \mathbb{N}$, $\alpha \in [0, 1)$, $\Omega = (0, 1)^2$, $U^{\mathrm{h}} \subset \mathcal{H}$ be a D-dimensional space, $\mathcal{A} = \{a_y : y \in \mathcal{Y}\} \subset L^\infty(\Omega)$ be compact with $\mathcal{Y} \subset \mathbb{R}^p$ as in (2.2). We consider the following equivalent questions:*

- *For $\varepsilon > 0$, how large do $M_\varepsilon, L_\varepsilon \in \mathbb{N}$ need to be to guarantee, that there exists a NN Φ that satisfies*

  (1) $\sup_{y\in\mathcal{Y}} |\mathcal{P}(y) - \mathrm{R}^{\mathcal{Y}}_{\varrho_\alpha}(\Phi)(y)|_{\mathbf{G}} \leq \varepsilon$,
  (2) $M(\Phi), N(\Phi) \leq M_\varepsilon$ *and* $L(\Phi) \leq L_\varepsilon$*?*

- *For $M, L \in \mathbb{N}$, how small can $\varepsilon_{L,M} > 0$ be chosen so that there exists a NN Φ that satisfies*

  (1) $\sup_{y\in\mathcal{Y}} |\mathcal{P}(y) - \mathrm{R}^{\mathcal{Y}}_{\varrho_\alpha}(\Phi)(y)|_{\mathbf{G}} \leq \varepsilon_{L,M}$,

(2) $M(\Phi), N(\Phi) \leq M$ *and* $L(\Phi) \leq L$*?*

Remark 3.6 (i) Conditions (1) in both instances of Question 3.5 are trivially equivalent to

$$\sup_{y\in\mathcal{Y}} \left\| \sum_{i=1}^{D} (\mathcal{P}(y))_j \cdot \varphi_j - \left(\mathrm{R}^{\mathcal{Y}}_{\varrho_\alpha}(\Phi)(y)\right)_j \cdot \varphi_j \right\|_{\mathcal{H}} \leq \varepsilon \text{ or } \varepsilon_{L,M}.$$

(ii) The results to follow measure the necessary sizes of the NNs in terms of the numbers of non-zero weights $M(\Phi)$. However, from a practical point of view, we are also interested in the number of necessary neurons $N(\Phi)$. Invoking a variation of [52, Lemma G.1.] shows that similar rates to the ones below are valid for the number of neurons $N(\Phi)$.

If the regularity of $\mathcal{P}$ is known, then a straight-forward bound on M_ε and L_ε can be found in [75]. Indeed, if $\mathcal{P} \in C^s(\mathcal{Y}; \mathbb{R}^D)$ with $\|\mathcal{P}\|_{C^s} \leq 1$, then one can choose

$$M_\varepsilon \in \mathcal{O}(D\varepsilon^{-p/s}) \text{ and } L_\varepsilon \in \mathcal{O}(\log_2(1/\varepsilon)), \text{ for } \varepsilon \to 0. \tag{3.1}$$

In other situations, e.g., if L_ε is permitted to grow faster than $\log_2(1/\varepsilon)$, one can even replace s by $2s$ in (3.1), see [41,76].

This rate of (3.1) uses the smoothness of $\mathcal{P}$ only and does not take into account the underlying structure stemming from the PDE (2.1) and the choice of $\mathcal{A}$. As a result, we find this rate to be significantly suboptimal.

In [37], it was showed that $\mathcal{P}$ can be approximated in the sense of Question 3.5 with

$$\begin{aligned} M_\varepsilon &\in \mathcal{O}\left(d(\varepsilon)D + \left(d(\varepsilon)^3 \log_2(d(\varepsilon)) + pd(\varepsilon)^2\right) \mathrm{polylog}_2(1/\varepsilon)\right) \\ L_\varepsilon &\in \mathcal{O}\left(\mathrm{polylog}_2(1/\varepsilon)\right), \text{ for } \varepsilon \to 0, \end{aligned} \tag{3.2}$$

where $d(\varepsilon)$ is a certain *intrinsic dimension*[3] of the problem, essentially reflecting the size of a *reduced basis* required to sufficiently approximate $S(\mathcal{Y})$. In many cases, especially those discussed in this manuscript, one can theoretically establish the scaling behavior of $d(\varepsilon)$ for $\varepsilon \to 0$. For instance, if $\mathcal{A}$ is as in (2.3), then (see [5, Equation (3.17)])

$$d(\varepsilon) \in \mathcal{O}(\log_2(1/\varepsilon)^p), \text{ for } \varepsilon \to 0.$$

Applied to (3.2) this yields that

$$M_\varepsilon \in \mathcal{O}\left(D\log_2(1/\varepsilon)^p + p \cdot \log_2(1/\varepsilon)^{cp}\right), \text{ for } \varepsilon \to 0,$$

for some $c \geq 1$. We also mention a similar approximation result, not of the discretized parametric map $\mathcal{P}$ but of the parametrized solution $(y, \mathbf{x}) \mapsto u_{a_y}(\mathbf{x})$, where u_{a_y} is as in (2.1) for $a = a_y$. In this situation, and for specific parametrizations of $\mathcal{A}$, [65, Theorem 4.8] shows that this map can be approximated by the realization of a NN using the ReLU activation function up to an error of ε with a number of weights that essentially scales like ε^{-r} where r depends on the summability of the (in this case potentially infinite) sequence $(a_i)_{i=1}^\infty$ such that $a_y = a_0 + \sum_{i=1}^\infty y_i a_i$ for a coefficient vector $y = (y_i)_{i=1}^\infty$. Here r can be very small if $\|a_i\|_{L^\infty(\Omega)}$ decays quickly for $i \to \infty$. This leads to very efficient approximations.

While the aforementioned results all examine the approximation-theoretical properties of realizations of NNs with respect to the *uniform* approximation error, they trivially imply the same rates if we examine the *average* errors

$$\left(\int_{\mathcal{Y}} \left|\mathcal{P}(y) - \mathrm{R}^{\mathcal{Y}}_{\varrho_\alpha}(\Phi_\varepsilon)(y)\right|^p_{\mathbf{G}} \, \mathrm{d}\mu(y)\right)^{1/p},$$

[3] derived from bounds on the Kolmogorov N-width of $S(\mathcal{Y})$

which are often used in practice. Here, $1 \leq p < \infty$ and μ is an arbitrary probability measure on $\mathcal{Y}$. In this paper, we examine the discrete counterpart of the *mean relative error*

$$\int_{\mathcal{Y}} \frac{\left|\mathcal{P}(y) - \mathrm{R}_{\varrho_\alpha}^{\mathcal{Y}}(\Phi_\varepsilon)(y)\right|_{\mathbf{G}}}{|\mathcal{P}(y)|_{\mathbf{G}}} \, \mathrm{d}\mu(y),$$

where μ denotes the *uniform probability measure* on $\mathcal{Y}$.

In view of the aforementioned theoretical results, it is clear that a parameter that is not the dimension of the parameter space $\mathcal{Y}$ but a problem-specific notion of complexity determines the hardness of the approximation problem of Question 3.5. To what extent this theoretical observation influences the hardness of the practical learning problem will be analyzed in the numerical experiment presented in the next section.

4 Numerical Survey of Approximability of Discretized Parameter-to-Solution Maps

As outlined in Sect. 3, the theoretical hardness of the approximation problem of Question 3.5 is determined by an intrinsic notion of complexity that potentially differs substantially from the dimension of the parameter space.

To test how this intrinsic complexity affects the practical machine-learning based solution of (2.1), we perform a comprehensive study where we train NNs to approximate the DPtSM $\mathcal{P}$ for various choices of $\mathcal{A}$. Here, we are especially interested in the performance of the learned approximation of $\mathcal{P}$ for varying complexities of $\mathcal{A}$. In this context, we test the hypotheses listed in the following Sect. 4.1. The remainder of this section is structured as follows: in Sect. 4.2, we introduce the concrete setup of parametrized diffusion coefficient sets, NN architecture, and optimization procedure and explain how the choice of test-cases are related to our hypotheses. Afterwards, in Sect. 4.3, we report the results of our numerical experiments. Section 4.4 is devoted to an evaluation and interpretation of these results in view of the hypotheses of Sect. 4.1.

4.1 Hypotheses

[H1] *The performance of learning the DPtSM does not suffer from the curse of dimensionality*
The theoretical results of Kutyniok et al. [37] show that the dimension of the parameter space p is not the main factor in determining the hardness of the underlying approximation-theoretical problem. As already outlined in the introduction, it is by no means clear that this effect is visible in a practical learning problem.
We expect that after accounting for effects stemming from optimization and sampling to promote comparability between test-cases in a way described in "Appendix A", the performance of the learning method will scale only mildly with the dimension of the parameter space.

[H2] *The performance of learning the DPtSM is very sensitive to parametrization*
We expect that, within the framework of Question 3.5, there are still extreme differences of intrinsic complexities for different choices of parametrizations for the diffusion coefficient sets $\mathcal{A} \subset L^\infty(\Omega)$ as defined in (2.2). However, it is not clear to

what extent NNs are capable of resolving the low-dimensional sub-structures generated by various choices of $\mathcal{A} \subset L^\infty(\Omega)$.
Since realizations of NNs are a very versatile function class, we expect the degree to which the performance of a trained NN depends on the number of parameters to vary strongly over the choice of $(a_i)_{i=1}^p$.

[H3] *Learning the DPtSM is efficient also for non-affinely parametrized problems*
The analysis of PPDEs often relies on affine parametrizations as in (2.3) or smooth variations thereof.
We expect the overall theme that NNs perform according to an intrinsic complexity of the problem depending only weakly on the parameter dimension to hold in more general cases.

4.2 Setup of Experiments

To test the hypotheses **[H1]**, **[H2]**, and **[H3]** of Sect. 4.1, we consider the following setup.

4.2.1 Parameterized Diffusion Coefficient Sets

We perform training of NNs for different instances of the approximation problem of Question 3.5. Here, we always assume the right-hand side to be fixed as $f(\mathbf{x}) = 20 + 10x_1 - 5x_2$, for $\mathbf{x} = (x_1, x_2) \in \Omega$, and we vary the parametrized diffusion coefficient set $\mathcal{A}$.

We consider four different parametrized diffusion coefficient sets as described in the test-cases **[T1]**–**[T4]** (for a visualization of **[T3]** and **[T4]** see Fig. 1). **[T1]**, **[T2]** and **[T3-F]** are affinely parametrized whereas the remaining parametrizations are non-affine.

[T1] Trigonometric Polynomials In this case, the set $\mathcal{A}$ consists of trigonometric polynomials that are weighted according to a scaling coefficient σ. To be more precise, we consider

$$\mathcal{A}^{\mathrm{tp}}(p, \sigma) := \left\{ \mu + \sum_{i=1}^{p} y_i \cdot i^\sigma \cdot (1 + a_i) \colon y \in \mathcal{Y} = [0, 1]^p \right\},$$

for some fixed shift $\mu > 0$ and a scaling coefficient $\sigma \in \mathbb{R}$. Here $a_i(\mathbf{x}) = \sin\left(\left\lfloor \frac{i+2}{2} \right\rfloor \pi x_1\right) \sin\left(\left\lceil \frac{i+2}{2} \right\rceil \pi x_2\right)$, for $i = 1, \ldots, p$.
We analyze the cases $p = 2, 5, 10, 15, 20$ and, for each p, the scaling coefficients $\sigma = -1, 0, 1$. As a shift we always choose $\mu = 1$.

[T2] Chessboard Partition Here, we assume that $p = s^2$ for some $s \in \mathbb{N}$ and we consider[4]

$$\mathcal{A}^{\mathrm{cb}}(p, \mu) := \left\{ \mu + \sum_{i=1}^{p} y_i \mathcal{X}_{\Omega_i} \colon y \in \mathcal{Y} = [0, 1]^p \right\},$$

where $(\Omega_i)_{i=1}^p$ forms a $s \times s$ chessboard partition of $(0, 1)^2$ and $\mu > 0$ is a fixed shift.
We examine this test-case for the shifts $\mu = 10^{-1}, 10^{-2}, 10^{-3}$, and, for each μ we consider $s = 2, 3, 4, 5$ which yields $p = 4, 9, 16, 25$, respectively.

[T3] Cookies In this test-case we differentiate between two sub-cases:

[4] $\mathcal{X}_A$ denotes the *indicator function* of A.

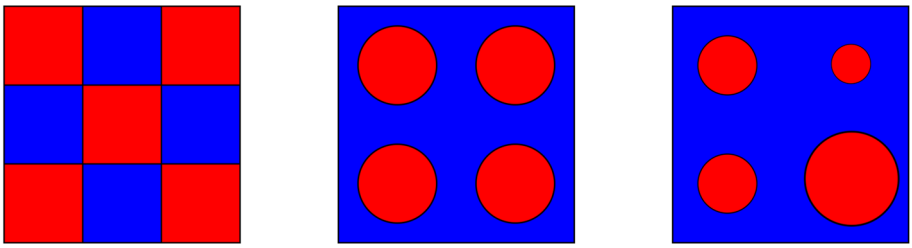

Fig. 1 Partition of Ω as in test-case **[T2]** (left) for $p = 9$, (the red and blue areas indicate the Ω_i), test-case **[T3-F]** (middle) for $p = 4$ (the red areas indicate the Ω_i) and test-case **[T3-V]** (right) for $p = 8$ (the red areas indicate the $\Omega_{i,y_{i+s^2}}$) (Color figure online)

[T3-F] Cookies with Fixed Radii In this setting, we assume that $p = s^2$ for some $s \in \mathbb{N}$ and we consider

$$\mathcal{A}^{\mathrm{cfr}}(p,\mu) := \left\{ \mu + \sum_{i=1}^{p} y_i \mathcal{X}_{\Omega_i} : y \in \mathcal{Y} = [0,1]^p \right\},$$

for some fixed shift $\mu > 0$ where the Ω_i are disks with centers $((2k+1)/(2s), (2\ell - 1)/(2s))$, where $i = ks + \ell$ for uniquely determined $k \in \{0, \dots s-1\}$ and $\ell \in \{1, \dots, s\}$. The radius is set to $r/(2s)$ for some fixed $r \in (0, 1]$.
We examine this test-case for fixed $\mu = 10^{-4}$, $r = 0.8$ and $s = 2, 3, 4, 5, 6$ which yields parameter dimensions $p = 4, 9, 16, 25, 36$, respectively.
[T3-V] Cookies with Variable Radii Here, we additionally assume that the radii of the involved disks are not fixed anymore. To be more precise, for $s \in \mathbb{N}$ and every $i = 1, \dots, s$, we are given disks $\Omega_{i,y_{i+s^2}}$ with center as before and radius $y_{i+s^2}/(2s)$ for $y_{i+s^2} \in [0.5, 0.9]$, so that $\mathcal{Y} = [0,1]^{s^2} \times [0.5, 0.9]^{s^2} \subset \mathbb{R}^p$ with $p = 2s^2$. We define

$$\mathcal{A}^{\mathrm{cvr}}(p,\mu) := \left\{ \mu + \sum_{i=1}^{p} y_i \mathcal{X}_{\Omega_{i,y_{i+s^2}}} : y \in \mathcal{Y} = [0,1]^p \times [0.5,0.9]^p \right\}.$$

Note that, $\mathcal{A}^{\mathrm{cvr}}(p,\mu)$ is *not* an affine parametrization.
We consider the shifts $\mu = 10^{-4}$ and $\mu = 10^{-1}$, and, for each μ, we consider the cases $s = 2, 3, 4, 5$ which yields the parameter dimensions $p = 8, 18, 32, 50$, respectively.

[T4] Clipped Polynomials Let

$$\mathcal{A}^{\mathrm{cp}}(p,\mu) := \left\{ \max\left\{ \mu, \sum_{i=1}^{p} y_i m_i \right\} : (y_i)_{i=1}^{p} \in \mathcal{Y} = [-1,1]^p \right\},$$

where $\mu > 0$ is the fixed clipping value and $(m_i)_{i=1}^{p}$ is the monomial basis of the space of all two-variate polynomials of degree $\leq k$. Therefore $p = \binom{2+k}{2}$.
We examine this test-case for fixed shift $\mu = 10^{-1}$ and for $k = 2, 3, 5, 8, 12$ which yields parameter dimensions $p = 6, 10, 21, 45, 91$, respectively.

4.2.2 Complexities of the Test-Cases

To understand the different complexity levels of the test-cases and to compare them with each other, we apply two methods:

- There is an intrinsic semi-ordering of complexities within the test-cases. In **[T1]** we have that $\mathcal{A}^{\text{tp}}(p,\sigma) \subset \mathcal{A}^{\text{tp}}(p,\sigma')$ if $\sigma < \sigma'$ which shows that the problem gets more complex with increasing σ. The influence of the shift μ is of a somewhat different type. *Higher values of μ make the underlying problem more elliptic*. This can be seen in Fig. 3: For a small value of μ, the impacts of the individual values on the chessboard-pieces on the solution appear to be almost completely decoupled. On the other hand, in the more elliptic case, the solution appears more smoothed out, and therefore each parameter value also influences the solution more globally. This implies a stronger coupling of the parameters and at least intuitively indicates a *reduced intrinsic dimensionality for higher values of μ*. The effect of μ describes the ordering of complexities exists within **[T2]** and **[T3]**.
- In addition to these non-quantitative comparisons of complexities, we offer the following heuristic measure of complexity: We solve the diffusion equation for 5000 i.i.d samples each on a 101×101 grid with the finite element method as described in detail in Sect. 4.2.3. Afterwards, we compute the singular values of the set of these solutions with respect to the $H_0^1(\Omega)$-norm on the finite element space. The resulting decay plots, shown in Fig. 2, serve as a heuristic to judge whether a problem can be efficiently solved by a *linear* reduced-order model (see [56, Sec. 6.7]).
  The different decay behavior between the test-cases suggests a significant variation in complexity between the corresponding parameter-to-solution maps. We observe that, independent of σ, **[T1]** exhibits the fastest decay among all cases, which indicates that this might be the easiest scenario. Even more interestingly, **[T2]** strongly differs from all other test-cases: First and foremost, the parameter dimension p has by far the largest impact for **[T2]**. Secondly, contrary to **[T3-V]**, the impact of the ellipticty μ varies between different parameter dimensions. Lastly, we note that **[T4]** has the smoothest decay among all test-cases and its decay rate appears to be independent of p.

4.2.3 Setup of Neural Networks and Training Procedure

Our experiments are implemented using Tensorflow [1] for the learning procedure and FEniCS [4] as FEM solver. The code used for dataset generation of all considered test-cases is made publicly available at www.github.com/MoGeist/diffusion_PPDE. To be able to compare different test-cases and remove all effects stemming from the optimization procedure, we train almost the same model for all parameter spaces. The only—to a certain extent inevitable—change that we allow between test-cases is that the input dimension of the NN changes to that of the parameter space. Concretely, we consider the following setup:

(1) The *finite element space* U^{h} resulting from a triangulation of $\Omega = [0,1]^2$ with $101 \times 101 = 10201$ equidistant grid points and first-order Lagrange finite elements. This space shall serve as a discretized version of the space $H^1(\Omega)$. We denote by $D = 10201$ its dimension and by $(\varphi_i)_{i=1}^D$ the corresponding finite element basis.
(2) The (fully-connected) *neural network architecture* $S = (p, 300, \ldots, 300, 10201)$ with $L = 11$ layers, where p is test-case-dependent and the weights and biases are initialized according to a normal distribution with mean 0 and standard deviation 0.1.
(3) The *activation function* is the 0.2-LReLU of Definition 3.2.

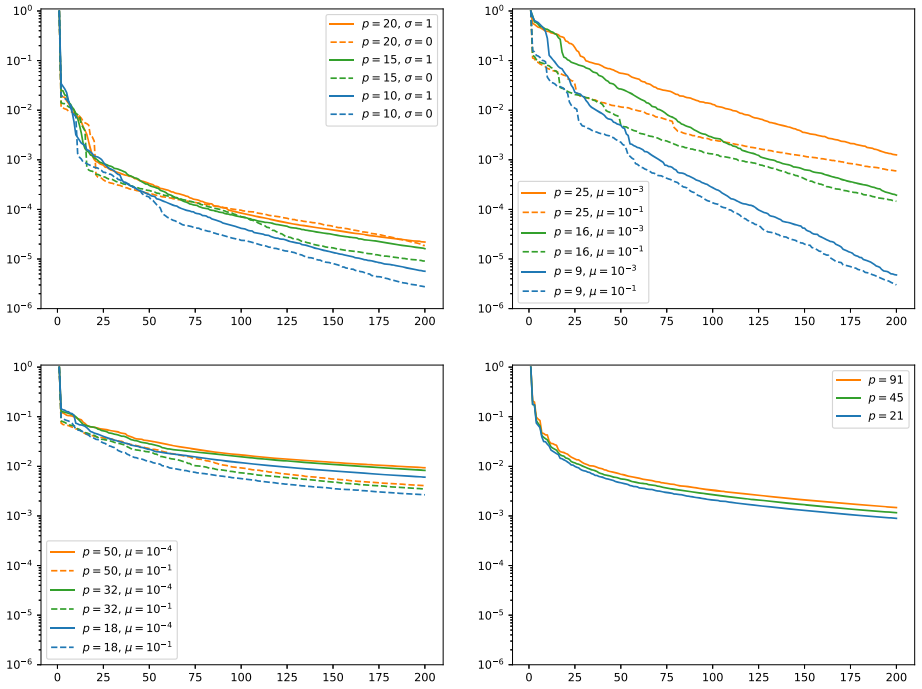

Fig. 2 Log-plot of the 200 largest normalized singular values for **[T1]** (top left), **[T2]** (top right), **[T3]-V** (bottom left) and **[T4]** (bottom right)

(4) The *loss function* is the relative error on the finite-element discretization of $\mathcal{H}$

$$\mathcal{L}\colon \mathbb{R}^D \times (\mathbb{R}^D \setminus \{0\}) \to \mathbb{R}, \quad (\mathbf{x}_1, \mathbf{x}_2) \mapsto \frac{|\mathbf{x}_1 - \mathbf{x}_2|_{\mathbf{G}}}{|\mathbf{x}_2|_{\mathbf{G}}}.$$

(5) The *training set* $(y^{i,\mathrm{tr}})_{i=1}^{N_{\mathrm{train}}} \subset \mathcal{Y}$ consists of $N_{\mathrm{train}} := 20000$ i.i.d. parameter samples, drawn with respect to the *uniform probability measure* on $\mathcal{Y}$.

(6) The *test set* $(y^{i,\mathrm{ts}})_{i=1}^{N_{\mathrm{test}}} \subset \mathcal{Y}$ consists of $N_{\mathrm{test}} := 5000$ i.i.d. parameter samples, drawn with respect to the *uniform probability measure* on $\mathcal{Y}$.

In our experiments, we aim at finding a NN Φ with architecture S such that the *mean relative training error*

$$\frac{1}{N_{\mathrm{train}}} \sum_{i=1}^{N_{\mathrm{train}}} \mathcal{L}\left(\mathrm{R}_{\varrho}^{\mathcal{Y}}(\Phi)(y^{i,\mathrm{tr}}), \mathbf{u}^{\mathrm{h}}_{y^{i,\mathrm{tr}}}\right) = \frac{1}{N_{\mathrm{train}}} \sum_{i=1}^{N_{\mathrm{train}}} \frac{\left\| \sum_{j=1}^{D} \left(\mathrm{R}_{\varrho}^{\mathcal{Y}}(\Phi)(y^{i,\mathrm{tr}})\right)_j \cdot \varphi_j - u^{\mathrm{h}}_{y^{i,\mathrm{tr}}} \right\|_{\mathcal{H}}}{\left\| u^{\mathrm{h}}_{y^{i,\mathrm{tr}}} \right\|_{\mathcal{H}}}$$

is minimized. We then test the accuracy of our NN by computing the *mean relative test error*

$$\frac{1}{N_{\mathrm{test}}} \sum_{i=1}^{N_{\mathrm{test}}} \mathcal{L}\left(\mathrm{R}_{\varrho}^{\mathcal{Y}}(\Phi)(y^{i,\mathrm{ts}}), \mathbf{u}^{\mathrm{h}}_{y^{i,\mathrm{ts}}}\right) = \frac{1}{N_{\mathrm{test}}} \sum_{i=1}^{N_{\mathrm{test}}} \frac{\left\| \sum_{j=1}^{D} \left(\mathrm{R}_{\varrho}^{\mathcal{Y}}(\Phi)(y^{i,\mathrm{ts}})\right)_j \cdot \varphi_j - u^{\mathrm{h}}_{y^{i,\mathrm{ts}}} \right\|_{\mathcal{H}}}{\left\| u^{\mathrm{h}}_{y^{i,\mathrm{ts}}} \right\|_{\mathcal{H}}}.$$

Here, we use the mean *relative* error instead of the mean absolute error in order to establish comparability of our results between different sets $\mathcal{A}$, allowing us to put our results into

context. In order to further limit the impact of the optimization procedure, each experiment is performed three times with varying initializations and test-train splits. We report the average error over these three runs. In "Impact of Initialization and Sample Selection" of appendix we analyze the variance of the three training runs in more detail to ensure that they do not influence our final results.

The optimization is done through *batch gradient descent*. To ensure further comparability between the different setups, the hyper-parameters in the optimization procedure are kept fixed: Training is conducted with batches of size 256 using the ADAM optimizer [36] with hyper-parameters $\alpha = 2.0 \times 10^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 1.0 \times 10^{-8}$ (with the exception of the learning rate α these are the default of the ADAM implementation in Tensorflow.). Training is stopped after reaching 40,000 epochs. Having trained the NN, for some new input $y \in \mathcal{Y}$, the computation of the approximate discretized solution $\mathrm{R}_{\varrho}^{\mathcal{Y}}(\Phi)(y)$ is done by a simple forward pass.

4.2.4 Relation to Hypotheses

The test-cases **[T1]**–**[T4]** are designed to test the hypotheses **[H1]**–**[H3]** in the following way:

Enabling comparability between test-cases We implement three measures to produce a uniform influence of the optimization and sampling procedure in all test-cases. These are that we only change the architecture in the minimally required way between test-cases to not alter the optimization behavior, we average our results over multiple initialisations, we analyze a posteriori the optimization behavior to see if there are qualitative differences between test-cases, and we choose the number of training samples in such a way that neither moderate further increasing or decreasing of the number of training samples affects the outcome of the experiments. We describe these measures in detail in "Appendix A".

Relation to Hypothesis **[H1]** To test if the learning method suffers from the curse of dimensionality or if the prediction of Kutyniok et al. [37] that its complexity is determined only by some intrinsic complexity of the function class holds, we run all test-cases **[T1]**–**[T4]** for various values of the dimension of the parameter space, and study the resulting scaling behavior.

Relation to Hypothesis **[H2]** To understand the extent to which the NN model is sufficiently versatile to adapt to various types of solution sets, we study four commonly considered parametrized diffusion coefficient sets which also include multiple subproblems described via the hyper-parameters σ and μ. The parametrized sets exhibit the following different characteristics:

- **[T1]** The parameter-dependence in this case is affine (i.e. the forward-map $y \mapsto b_y(u, v)$ depends affinely on y for all $u, v \in \mathcal{H}$) whereas the spatial regularity of the functions $(a_i)_{i=1}^p$ is analytic. To vary the difficulty of the problem at hand, we consider different instances of the scaling coefficient σ which put different emphasis on the high-frequency components of the functions $(a_i)_{i=1}^p$. In particular, if $\sigma > 0$, a higher weight is put on the high-frequency components than on the low-frequency ones whereas the opposite is true for $\sigma < 0$.
- **[T2]** The parameter-dependence in this case is affine again, whereas the spatial regularity of the $(\mathcal{X}_{\Omega_i})_{i=1}^p$ is very low. To vary the difficulty of the problem, we consider different instances of shifts μ. The higher the shift is, the more elliptic the problem becomes.

[T3] **[T3-F]** again exhibits affine parameter-dependence and the same regularity properties as test-case **[T2]**. However, this problem is considered to be easier than test-case **[T2]** since the $\overline{\Omega_i}$ do not intersect each other.
For test-case **[T3-V]**, the geometric properties of the domain partition are additionally encoded via a parameter thereby rendering the problem to be non-affine.

[T4] In this case, the parameter-dependence is non-affine and has low regularity due to the clipping procedure. Additionally, the spatial regularity of the functions a_y is comparatively low in general.

A visualization highlighting the versatility of our test-cases can be seen when comparing the FE solutions in Fig. 3 (test-case **[T2]**) with the FE solutions in Fig. 4 (test-case **[T4]**).
Relation to Hypothesis **[H3]**: The test-cases **[T3-V]** and **[T4]** are non-affinely parametrized.

4.3 Numerical Results

In this subsection, we report the results of the test-cases announced in the previous subsection.

[T1] Trigonometric Polynomials

We observe the following mean relative test errors for the sets $\mathcal{A}^{\mathrm{tp}}(p, \sigma)$.

[T2] Chessboard Partition

We observe the following mean relative test errors for the sets $\mathcal{A}^{\mathrm{cb}}(p, \mu)$.

In Fig. 3, we show samples from the test set for different values of μ. Here we always depict one average performing test-case and one with poor performance. These figures offer a potential explanation of why the scaling with p is qualitatively different for different values of μ. This seems to be because for lower μ the effect of the individual parameters on the solution seems to be much more local than for higher μ. This appears to lead to a higher intrinsic dimensionality of the problem.

[T3] Cookies with Fixed and Variable Radii

We start with one experiment where the radii of the cookies are fixed to $0.8/(2s)$.

Moreover, we find for the sets of cookies with variable radii $\mathcal{A}^{\mathrm{cvr}}(p, \mu)$ the following mean relative test errors.

[T4] Clipped Polynomials

For the set $\mathcal{A}^{\mathrm{cp}}(p, 10^{-1})$, we obtain the following mean relative test errors when varying p.

4.4 Evaluation and Interpretation of Experiments

We make the following observations about the numerical results of Sect. 4.3.

[O1] Our test-cases show that the error rate achieved by NN approximations for varying parameter sizes *differs strongly and qualitatively between different test-cases*. In Figs. 5, 6, 7, and 8 we depict the different scaling behaviors of the test-cases **[T1]**,

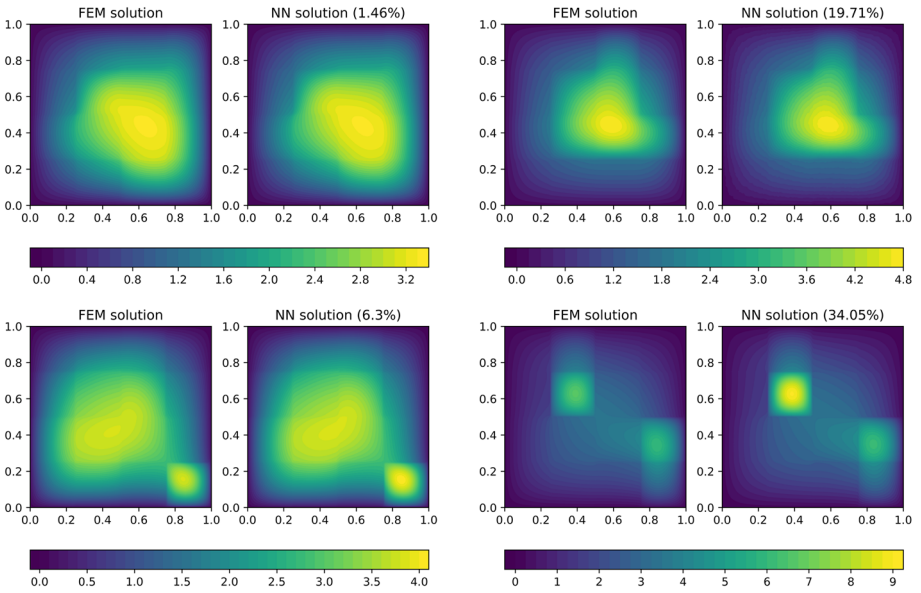

Fig. 3 Comparison of the ground truth solution and the one predicted by the NN for an average (left) and a poor performing (right) for $p = 16$ and $\mu = 10^{-1}$ (top) and $\mu = 10^{-3}$ (bottom) for test-case **[T2]**. The percentage in brackets represents the relative test error for this particular sample

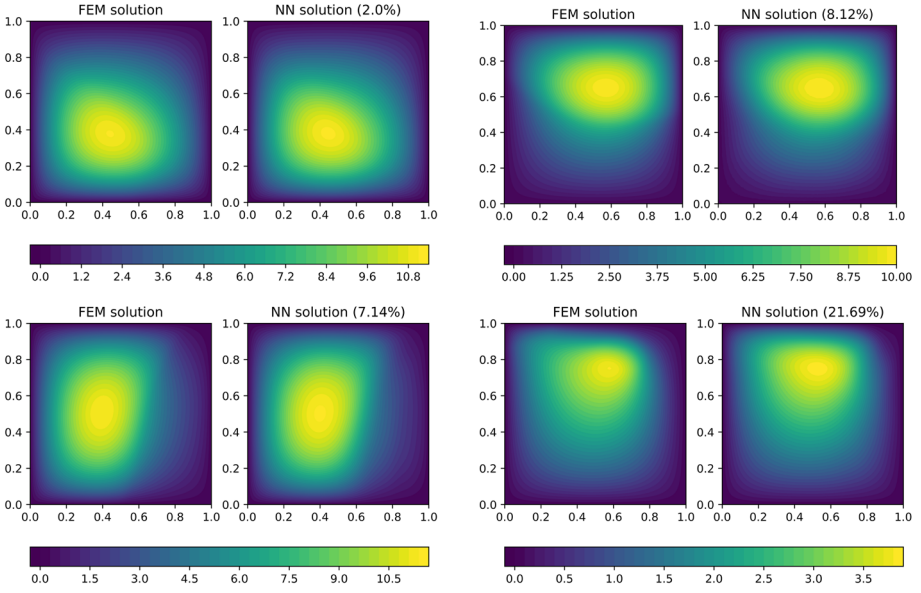

Fig. 4 Comparison of the ground truth solution and the one generated by the NN for an average (left) and a poor performing case (right) for $\mu = 10^{-1}$ and $p = 6$ (top) and $p = 91$ (bottom) for test-case **[T4]**. The percentage in brackets represents the relative test error for this particular sample

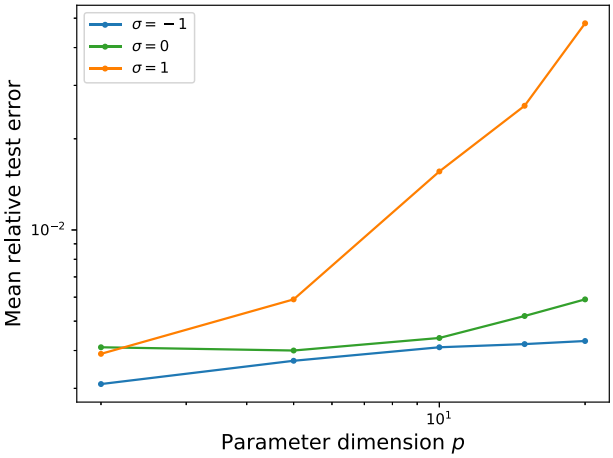

Fig. 5 Plot of the mean relative test error for the sets of test-case **[T1]** for different values σ. The horizontal axis follows the dimension of the parameter space p the mean relative test error is shown on the vertical axis. Both axes use a logarithmic scale

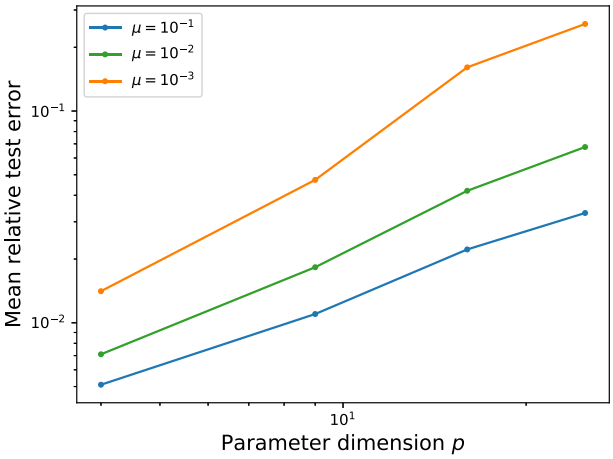

Fig. 6 Plot of the mean relative test error for the sets of test-case **[T2]** for different values of μ with p on the horizontal axis and the mean relative test error on the vertical axis. Both axes use a logarithmic scale

[T2], **[T3]**, and **[T4]**. For **[T1]** and $\sigma = -1$, *the error appears to be almost independent from* p for $p \to \infty$. In contrast to that, we observe for $\sigma = 1$ a linear scaling in the loglog plot implying a polynomial dependence of the error on p.

For test-case **[T2]**, we observe that the error scales linearly in the loglog scale of Fig. 6. We conclude that for $\mathcal{A}^{\text{cb}}(p, \mu)$, *the error scales polynomially with* p.

The errors of the test-cases associated with **[T3]** seem to scale linearly with p in the loglog scale depicted in Fig. 7. This implies that for **[T3]** *the error scales polynomially in* p *with the same exponent.*

The semilog plot of Fig. 8 shows that for test-case **[T4]** with the sets $\mathcal{A}^{\text{cp}}(p, 10^{-4})$, *the growth of the error is logarithmic in* p.

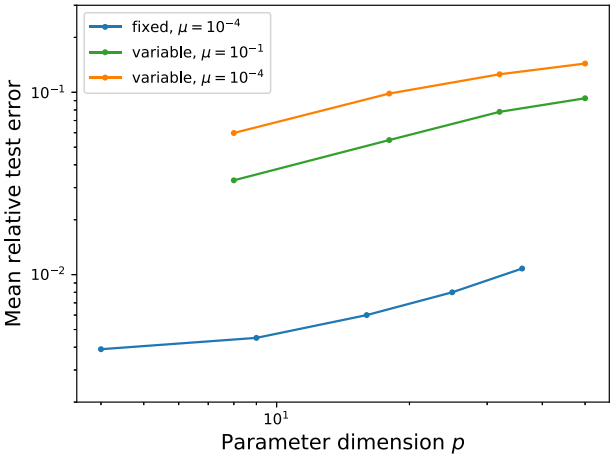

Fig. 7 Plot of the mean relative test error for the sets of test-case **[T3]** with p on the horizontal axis and the error on the vertical axis. Both axes use a logarithmic scale

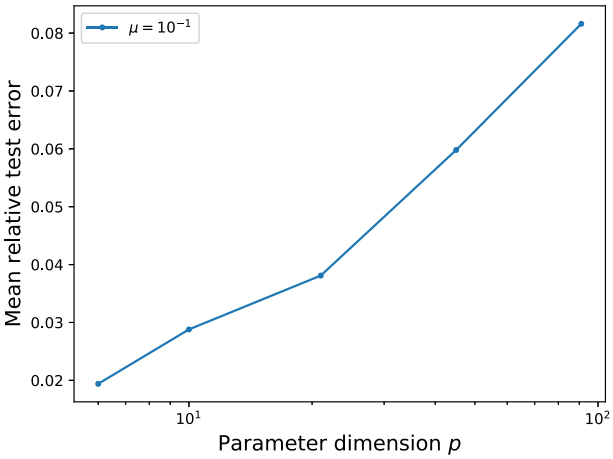

Fig. 8 Plot of the mean relative test error for the sets of test-case **[T4]** with p on the horizontal axis and the error on the vertical axis. Only the horizontal axis is scaled logarithmically

In total, we observed scaling behaviors of $\mathcal{O}(1)$, $\mathcal{O}(\log(p))$ and $\mathcal{O}(p^k)$ for $k > 0$ and for $p \to \infty$. Notably, none of the test-cases exhibited an exponential dependence of the error on p.

[O2] The choice of the hyper-parameters σ and μ in the test-cases **[T1]**, **[T2]**, **[T3]** *influences the scaling behavior according to its effect on the complexity of the parametrized diffusion coefficient set*.

The observed scaling behavior closely reflects the complexity of the solution manifolds as characterized by the singular value decay depict in Fig. 2. For **[T1]**, the plot suggests this to be the easiest case. Moreover, the impact of the scaling parameter σ was supposed to *simplify the parametric problem for smaller values of* σ. This is precisely, what we observe in Table 1 and Fig. 5.

Table 1 Mean relative test error averaged over three training runs for test-case **[T1]** and different parameter dimensions p, different scaling parameters σ and shift $\mu = 1$

Parameter dimension p	2	5	10	15	20
Mean relative test error $(\sigma = -1)$	0.31%	0.37%	0.41%	0.42%	0.43%
Mean relative test error $(\sigma = 0)$	0.41%	0.40%	0.44%	0.52%	0.59%
Mean relative test error $(\sigma = 1)$	0.39%	0.59%	1.56%	2.57%	4.81%

Table 2 Mean relative test error averaged over three training runs for test-case **[T2]** and parameter dimensions $p = s^2$

s	2	3	4	5
Parameter dimension p	4	9	16	25
Mean relative test error$(\mu = 10^{-1})$	0.51%	1.10%	2.22%	3.30%
Mean relative test error $(\mu = 10^{-2})$	0.71%	1.83%	4.20%	6.76%
Mean relative test error $(\mu = 10^{-3})$	1.41%	4.73%	16.08%	25.77%

Table 3 Mean relative test error averaged over three training runs for test-case **[T3-F]** and different parameter dimensions $p = s^2$ with shift $\mu = 10^{-4}$ and radius $0.8/(2s)$

s	2	3	4	5	6
Parameter dimension p	4	9	16	25	36
Mean relative test error	0.39%	0.45%	0.60%	0.80%	1.08%

Table 4 Mean relative test error averaged over three training runs for test-case **[T3-V]** and different parameter dimensions $p = 2s^2$

s	2	3	4	5
Parameter dimension p	8	18	32	50
Mean relative test error $(\mu = 10^{-1})$	3.29%	5.47%	7.81%	9.27%
Mean relative test error $(\mu = 10^{-4})$	5.98%	9.84%	12.55%	14.37%

Accordingly, we see for **[T2]** in Table 2 and Fig. 6 and for **[T3-V]** in Table 4 and Fig. 7 that the parameter μ influences the scaling behavior of the method with p. Both cases get harder by lowering μ and overall exhibit error scaling on the order $\mathcal{O}(p^k)$. However, for **[T2]** μ directly impacts the scaling exponent k while for **[T3-V]** it stays the same. This again resembles the singular value decay depicted in Fig. 2 (Table 3). Concluding, we can see that the approximation of the DPtSM by NNs appears to be very sensitive to these parameters, especially to the ellipticity μ, and that the scaling behavior is accurately predicted by the complexity of the solution manifold in regard to its singular value decay (Table 5).

[O3] We observe *no fundamentally worse scaling behavior for non-affinely parametrized test-cases* compared to test-cases with an affine parametrization. In test-case **[T3]**, we do observe that the non-linearly parametrized problem appears to be more challenging overall, while the scaling behavior is the same as for the affinely parametrized problem.

Table 5 Mean relative test error averaged over three training runs for test-case **[T4]** with clipping value $\mu = 10^{-1}$ and different parameter dimensions p

Polynomial degree k	2	3	5	8	12
Parameter dimension p	6	10	21	45	91
Mean relative test error	1.94%	2.88%	3.81%	5.98%	8.16%

In test-case **[T4]**, which is the test-case with the highest number of parameters p, we observe only a very mild (in fact logarithmic) dependence of the error on p.

From these observations we draw the following conclusions for our hypotheses:
Hypothesis [H1] In observation **[O1]**, we saw that over a wide variety of test-cases multiple types of scaling of the error with the dimension of the parameter space could be observed. None of them admit an exponential scaling. In fact, the behavior of the errors seems to be determined by an intrinsic complexity of the problems.
Hypothesis [H2] Comparing performance both between test-cases (observation **[O1]**) and within test-cases (observation **[O2]**), leads us to conclude that there exist strong differences in the performance of learning the DPtSM. For various test-cases, using NNs with precisely the same architecture, we observed (see **[O2]**) considerably different scaling behaviors of the test-cases **[T1]**–**[T4]** which have the error scale polynomially, logarithmically and being constant with changing parameter dimension p (described in **[O1]**). According to **[O2]**, the overall level of the errors and the type of scaling for increasing p follows the semi-ordering of complexities of test-cases in the sense that more complex parametrized sets yield higher errors whereas simpler sets or spaces with intuitively lower intrinsic dimensionality yield smaller errors (test-cases **[T1]** and **[T2]**).

Therefore, we conclude that the approximation theoretical intrinsic dimension of the parametric problem is a main factor in determining the hardness of learning the DPtSM.
Hypothesis [H3] In support of **[H3]**, we found no fundamental difference of the performance of the NN model for non-affinely parametrized problems (see **[O3]**).

In conclusion, we found support for all the hypotheses **[H1]**–**[H3]**. We consider this result a validation of the importance of approximation-theoretical results for practical learning problems, especially in the application of deep learning to problems of numerical analysis.

5 An Empirical Study Beyond the Theoretical Analysis

The design of our numerical study as presented in the previous section was chosen to closely resemble the approximation-theoretical framework of Kutyniok et al. [37]. Therefore, the underlying network model was chosen as a standard fully-connected neural network. In modern applications, typically more refined and performance driven architectures are used. To take a further step towards the practical relevance of approximation theory, we accompany the previous study by an experiment in a more practical and realistic setting.

Because our problem is set up on a rectangular domain, it is a natural choice to encode the network inputs, i.e. the parametrized diffusion coefficients, as an image resulting from discretization on a fixed grid. This allows us to incorporate convolutional layers into our network architecture. For a meaningful comparison, we chose a CNN of similar depth to the previously used fully-connected NN: We use six convolutional followed by four fully connected layers, already including the output. The first two convolutional layers contain

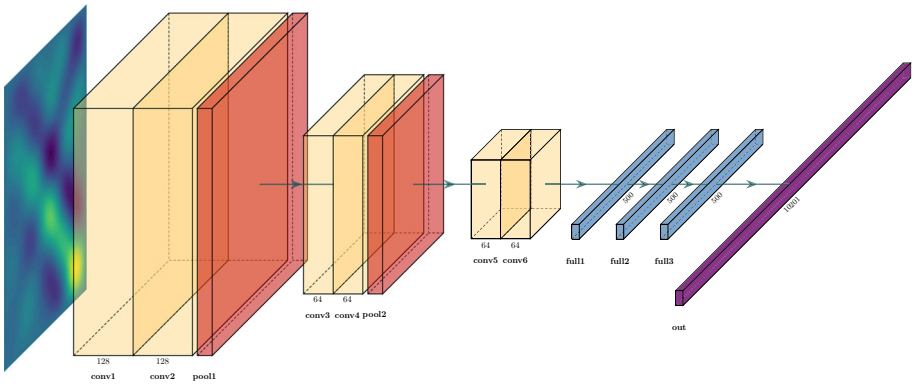

Fig. 9 CNN architecture used for experiments. Below each convolutional layer the number of filters is listed. The shrinking height and width of the blocks symbolize the lowered resolution after the red pooling layers. The visualization is created using the tools of www.github.com/HarisIqbal88/PlotNeuralNet

128 filters of size 2×2 while the other four only contain 64. They all utilize zero padding to preserve the input dimension. Moreover, after layer two and four 2×2 max pooling is performed to down-sample the input. The fully connected layers contain 500 neurons each. All layers except the output are equipped with the 0.2-LReLU activation function. The architecture is visualized in Fig. 9.

We again use the ADAM optimizer [36] with hyper-parameters $\alpha = 1.0 \times 10^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 1.0 \times 10^{-8}$ for training. However, the batch size is reduced to 32 and we employ early stopping based on a separate validation set of size $N_{\text{val}} := 1000$ to prevent overfitting. The maximum number of epochs is limited to 3000.

We noted earlier and described in detail in "Appendix A" that the impact of the number of samples seems to be independent of the scaling behavior. To emphasize this point, we cut the number of training samples in half to $N_{\text{train}} := 10{,}000$. In contrast to our earlier experiments, encoding the parameters as images now allows us to keep the architecture completely fixed throughout all test-cases, even the input layer. However, the resolution of the input becomes a new hyper-parameter. In general, increasing input resolution enhances the performance but comes at significant additional computational cost and the elevated risk of overfitting. To balance this trade-off, we settled at a resolution of 20×20 for all test-cases, except **[T3-V]** which required a resolution of 50×50 for an accurate representation. Note, that even for inputs of size 20×20 the diffusion coefficients are heavily over-parametrized which leads to a considerable increase in the required storage.

5.1 Numerical Results for CNNs

We tested the above network configuration for all cases besides **[T1]** and **[T3-F]**, which could already easily be learned by the fully-connected NN. This yielded the following results.

[T2] Chessboard Partition

We observe the following mean relative test errors for the sets $\mathcal{A}^{\text{cb}}(p, \mu)$ (Table 6).

Table 6 Mean relative test error of CNN for test-case **[T2]** and parameter dimensions $p = s^2$

s	2	3	4	5
Parameter dimension p	4	9	16	25
Mean relative test error ($\mu = 10^{-1}$)	0.72%	1.08%	1.90%	2.46%
Mean relative test error ($\mu = 10^{-2}$)	0.72 %	1.27%	2.16%	3.06%
Mean relative test error ($\mu = 10^{-3}$)	0.98%	1.54%	2.46%	3.75 %

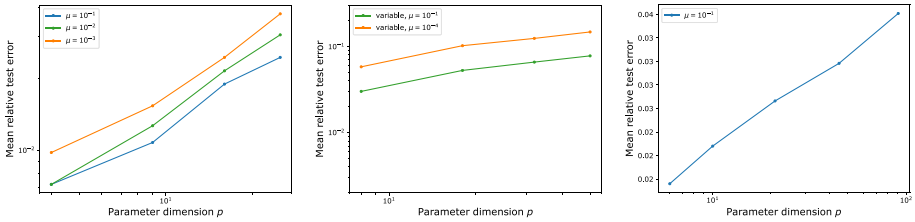

Fig. 10 Plot of the mean relative test errors achieved with our CNN architecture for the sets of test-case **[T2]** (left), **[T3-V]** (middle) and **[T4]** (right). All axes except the vertical axis for **[T4]** are scaled logarithmically

[T3-V] Cookies with Variable Radii

For the variable cookies $\mathcal{A}^{\text{cvr}}(p, \mu)$ we achieve the following mean relative test errors (Table 7).

[T4] Clipped Polynomials

For the set $\mathcal{A}^{\text{cp}}(p, 10^{-1})$, we obtain the following mean relative test errors (Table 8).

Table 7 Mean relative test error of CNN for test-case **[T3-V]** and different parameter dimensions $p = 2s^2$

s	2	3	4	5
Parameter dimension p	8	18	32	50
Mean relative test error ($\mu = 10^{-1}$)	3.01%	5.27%	6.58%	7.77%
Mean relative test error ($\mu = 10^{-4}$)	5.80%	10.20%	12.39%	14.72%

Table 8 Mean relative test error of CNN for test-case **[T4]** with clipping value $\mu = 10^{-1}$ and different parameter dimensions p

Polynomial degree k	2	3	5	8	12
Parameter dimension p	6	10	21	45	91
Mean relative test error	1.70%	2.07%	2.56 %	2.94%	3.48%

5.2 Evaluation and comparison to first experiment

Similar to the results obtained in the previous section, we examine the scaling behavior with regard to the parameter dimension p in Fig. 10. We notice that, although the CNN significantly outperforms the simple fully-connected NN, the scaling behavior remains nearly identical. The most notable difference between the two networks can be witnessed for **[T2]**. For this case, the error was reduced by an order of magnitude. This is especially noteworthy because, as mentioned earlier, only half the number of samples was used during training. Nonetheless, the similar scaling behavior (cf. Figs. 6, 7, 8 with Fig. 10) implies that the rates obtained previous section still accurately reflect the underlying hardness of the problem. This indicates that the relatively high relative error is a result of insufficient model capacity rather than an artifact of the optimization procedure.

Given that the experiments based on convolutional neural networks yielded lower overall errors with less training data, we identify two promising directions of future work. First, the fact that convolutional neural networks outperform their fully connected counterparts, requires a theoretical analysis and justification. Second, it appears worthwhile to search for additional architectural refinements which potentially allow cutting the overall error down even further.

Acknowledgements M. Geist and M. Raslan would like to thank Philipp Trunschke for fruitful discussions on the topic. This work was made possible by the computational resources provided by the Institute of Mathematics of the TU Berlin. G. Kutyniok acknowledges partial support by the Bundesministerium für Bildung und Forschung (BMBF) through the Berlin Institute for the Foundations of Learning and Data (BIFOLD), Project AP4, RTG DAEDALUS (RTG 2433), Projects P1, P3, and P8, RTG BIOQIC (RTG 2260), Projects P4 and P9, and by the Berlin Mathematics Research Center MATH+, Projects EF1-1 and EF1-4.

Funding Open access funding provided by University of Vienna.

A Elimination of Obfuscating Phenomena

Below we describe the measures taken to enable comparability between test-cases.

A.1 Fixing the Architecture

In all our experiments the network architecture was kept almost completely fixed, only varying the dimension of the input layer. Our choice of architecture was made on the basis of preliminary experiments with the goal of developing a network structure that performs well on all datasets and in particular displays good optimization behavior independent of the test-case as showcased in "A Posteriori Analysis of Convergence Behavior" of appendix. This was done to ensure comparability across all test-cases and parameter choices, allowing us to isolate the influence of the parametrization and the dimension of the parameter space. We emphasize

Table 9 Mean relative test error as well as the corresponding R^2 coefficient from a simple linear regression for varying sizes of the training set and all previously considered setups

Test-case	Size of training set					
	20,000 (%)	17,500 (%)	15,000 (%)	12,500 (%)	10,000 (%)	R^2
[T1] ($\mu = 1$, $\sigma = 0$, $p = 20$)	0.59	0.61	0.64	0.70	0.76	0.95
[T2] ($\mu = 10^{-1}$, $p = 9$)	1.06	1.29	1.49	1.81	2.18	0.98
[T2] ($\mu = 10^{-2}$, $p = 9$)	1.81	1.94	2.58	2.98	4.26	0.91
[T2] ($\mu = 10^{-3}$, $p = 9$)	4.47	5.31	6.23	7.78	9.24	0.98
[T3-F] ($\mu = 10^{-4}$, $p = 25$)	0.83	0.85	0.88	0.91	0.96	0.97
[T3-V] ($\mu = 10^{-1}$, $p = 18$)	5.44	5.60	5.83	6.16	6.56	0.97
[T3-V] ($\mu = 10^{-4}$, $p = 18$)	9.81	9.98	10.18	10.61	11.06	0.95
[T4] ($\mu = 10^{-1}$, $p = 21$)	3.86	4.17	5.06	5.50	6.46	0.98

that more sophisticated architectures and the usage of tools like weight regularization or learning rate decay in general enable better performance on individual datasets. However, in our case, they would only obfuscate the approximation-theoretical effect that we are seeking to identify.

A.2 Influence of the Size of the Training Set

Throughout this paper all training was conducted with a fixed number of 20,000 samples. Since it is clear that a larger training set will generally yield better results, this trend may affect different test-cases to various degrees. To guarantee that the effect of the choice of the number of training samples is uniform across cases, we chose the number of samples in the following way: We trained the same NN architecture as described in Sect. 4.2.3 for different parameter constellations with training sets ranging from 10,000 to 20,000 samples. The results are depicted in Table 9. The table also includes the coefficient of determination R^2 (see [70, p. 601]) for each individual dataset resulting from fitting a simple linear regression to the set of sample size and test error pairs.

This analysis shows that with R^2 values ranging from 0.91 to 0.98 the relation between the number of samples and the achieved accuracy is almost perfectly linear. Assuming this relation extrapolates to the other parameter dimension p, this implies that our results in Sect. 4 can be considered independent of the number of samples chosen. It should, however, be noted, that this linear dependence can only be observed in a reasonable range of training set sizes. In particular, the experiments revealed a lower bound on the number of samples needed to stably train our NN architecture. While in our case this bound can be observed in the range of 1000–5000 samples depending on the considered test-case, other NN setups may be able to effectively train with even lower sample counts.

A.3 Impact of Initialization and Sample Selection

The achieved accuracy of a NN is inevitably subject to stochastic variation. It is influenced by the initialization of the weights, the selection of training samples and the ordering of the dataset. Nonetheless, a stable optimization procedure should reproduce similar results independent of the aforementioned factors.

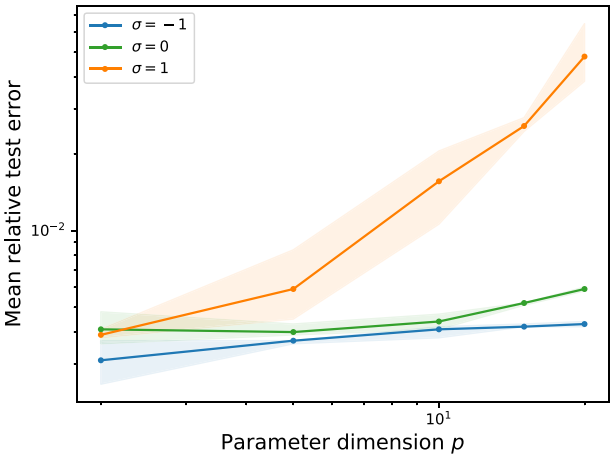

Fig. 11 Plot of the range of the mean relative test error over three runs for the sets of test-case **[T1]**. Both axes use a logarithmic scale

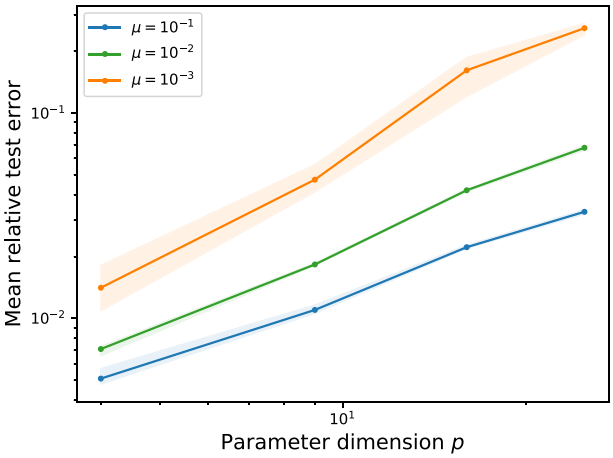

Fig. 12 Plot of the range of the mean relative test error over three runs for the sets of test-case **[T2]**. Both axes use a logarithmic scale

To show that our results were not impacted by the randomness of optimization, we performed all experiments three times for different initializations of the weights and permutations of the dataset. In Fig. 11 we show how the results varied between these three runs. The plots suggest that there is very little variance over different training runs. In particular, the observed scaling behavior stays the same over all runs and therefore is not an artefact of the stochastic nature of the optimization (Figs. 12, 13, 14).

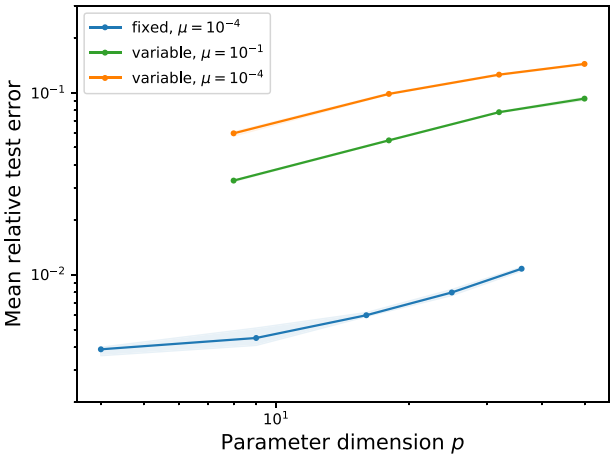

Fig. 13 Plot of the range of the mean relative test error over three runs for the sets of test-case **[T3]**. Both axes use a logarithmic scale

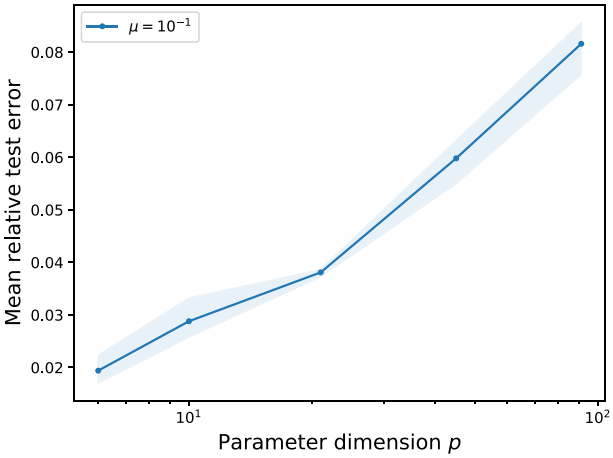

Fig. 14 Plot of the range of the mean relative test error over three runs for the sets of test-case **[T4]**. Only the horizontal axis is scaled logarithmically

A.4 A Posteriori Analysis of Convergence Behavior

Similarly to the architecture, the hyper-parameters of the optimization method were also kept fixed across all datasets and training runs. This measure, however, only eliminates the effect of the architecture on the optimization method and does not address any obfuscating effect that the choice of test-cases may have. To analyze if such an effect is present, we check the convergence on our two hardest test-cases **[T2]** and **[T3-V]** for the largest parameter dimension p considered. The results are depicted in Figs. 15 and 16, respectively. We see

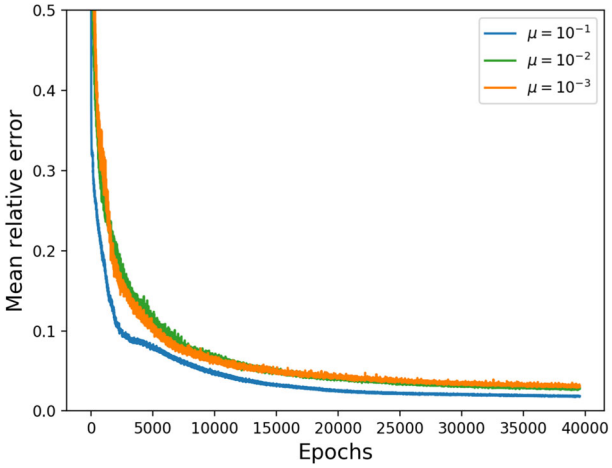

Fig. 15 Plot of the mean relative training error for **[T2]** with $p = 25$ and different shifts μ

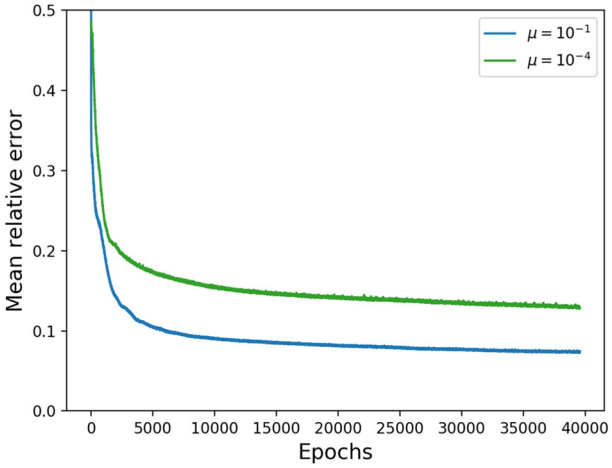

Fig. 16 Plot of the mean relative training error for **[T3-V]** with $p = 50$ and different shifts μ

that even for small shifts μ, i.e., the most difficult problem settings, the error on the training set converges smoothly. This behavior can also be witnessed on all other test-cases.

Another possible pitfall of our optimization procedure would be the occurrence of overfitting. In particular, this would render our attained accuracy levels invalid as we trained for a fixed number of epochs. However, this did not occur in any of our tests. We exemplarily showcase the convergence plot of the training and test error for the hardest parameter choices of **[T3-V]** and **[T4]** in Figs. 17 and 18 respectively. Similar behavior can also be observed on all other datasets.

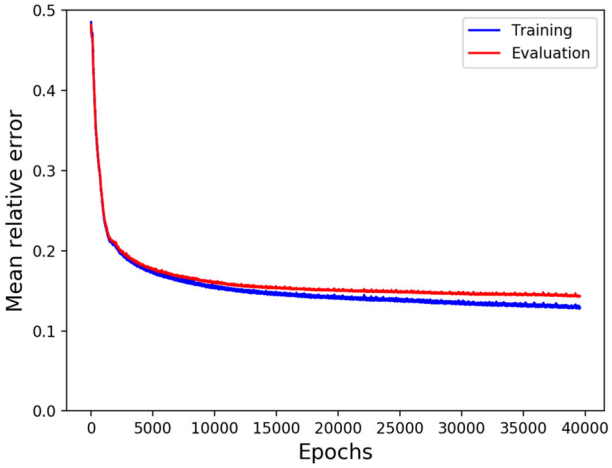

Fig. 17 Plot of the mean relative training and test error for **[T3-V]** with $p = 50$ and $\mu = 10^{-4}$

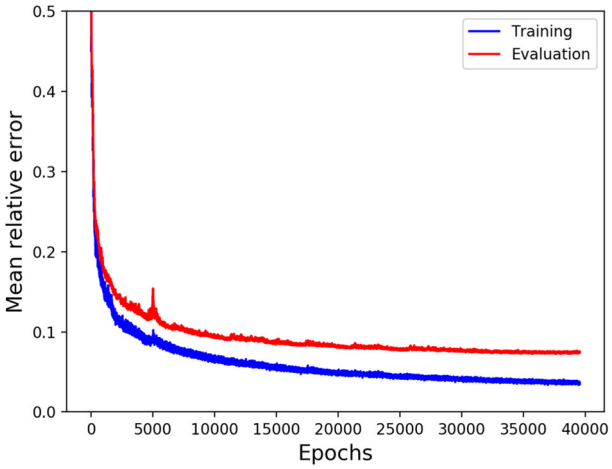

Fig. 18 Plot of the mean relative training and test error for **[T4]** with $p = 91$ and $\mu = 10^{-1}$

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems (2015). Software available from tensorflow.org
2. Adcock, B., Brugiapaglia, S., Dexter, N., Moraga, S.: Deep neural networks are effective at learning high-dimensional Hilbert-valued functions from limited data. arXiv preprint arXiv:2012.06081 (2020)
3. Adcock, B., Dexter, N.: The gap between theory and practice in function approximation with deep neural networks. arXiv preprint arXiv:2001.07523 (2020)
4. Alnæs, M.S., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., Ring, J., Rognes, M.E., Wells, G.N.: The FEniCS Project Version 1.5. Arch. Numer. Softw. **3**(100) (2015)

5. Bachmayr, M., Cohen, A.: Kolmogorov widths and low-rank approximations of parametric elliptic PDEs. Math. Comput. **86**(304), 701–724 (2017)
6. Bachmayr, M., Cohen, A., Dahmen, W.: Parametric PDEs: sparse or low-rank approximations? IMA J. Numer. Anal. **38**(4), 1661–1708 (2018)
7. Barron, A.: Universal approximation bounds for superpositions of a sigmoidal function. IEEE Trans. Inf. Theory **39**(3), 930–945 (1993)
8. Beck, C., Becker, S., Grohs, P., Jaafari, N., Jentzen, A.: Solving stochastic differential equations and Kolmogorov equations by means of deep learning. arXiv preprint arXiv:1806.00421 (2018)
9. Beck, C., Weinan, E., Jentzen, A.: Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations. J. Nonlinear Sci. **29**, 1563–1619 (2019)
10. Bellman, R.: On the theory of dynamic programming. Proc. Natl. Acad. Sci. U.S.A. **38**(8), 716 (1952)
11. Berg, J., Nyström, K.: Data-driven discovery of PDEs in complex datasets. J. Comput. Phys. **384**, 239–252 (2019)
12. Berner, J., Grohs, P., Jentzen, A.: Analysis of the generalization error: empirical risk minimization over deep artificial neural networks overcomes the curse of dimensionality in the numerical approximation of Black-Scholes partial differential equations. arXiv preprint arXiv:1809.03062 (2018)
13. Bhattacharya, K., Hosseini, B., Kovachki, N.B., Stuart, A.M.: Model reduction and neural networks for parametric PDEs. *arXiv preprint* arXiv:2005.03180 (2020)
14. Bölcskei, H., Grohs, P., Kutyniok, G., Petersen, P.C.: Optimal approximation with sparsely connected deep neural networks. SIAM J. Math. Data Sci. **1**, 8–45 (2019)
15. Brevis, I., Muga, I., van der Zee, K.G.: Data-driven finite elements methods: machine learning acceleration of goal-oriented computations. arXiv preprint arXiv:2003.04485 (2020)
16. Cohen, A., DeVore, R.: Approximation of high-dimensional parametric PDEs. Acta Numer. **24**, 1–159 (2015)
17. Cucker, F., Smale, S.: On the mathematical foundations of learning. Bull. Am. Math. Soc. **39**, 1–49 (2002)
18. Cucker, F., Zhou, D.-X.: Learning Theory: An Approximation Theory Viewpoint, Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press (2007)
19. Dal Santo, N., Deparis, S., Pegolotti, L.: Data driven approximation of parametrized PDEs by Reduced Basis and Neural Networks. arXiv preprint arXiv:1904.01514 (2019)
20. Eigel, M., Schneider, R., Trunschke, P., Wolf, S.: Variational Monte Carlo-bridging concepts of machine learning and high dimensional partial differential equations. Adv. Comput. Math. **45**, 2503–2532 (2019)
21. Elbrächter, D., Grohs, P., Jentzen, A., Schwab, C.: DNN expression rate analysis of high-dimensional PDEs: application to option pricing. arXiv preprint arXiv:1809.07669 (2018)
22. Faber, F.A., Hutchison, L., Huang, B., Gilmer, J., Schoenholz, S.S., Dahl, G.E., Vinyals, O., Kearnes, S., Riley, P.F., von Lilienfeld, O.A.: Prediction errors of molecular machine learning models lower than hybrid DFT error. J. Chem. Theory Comput. **13**(11), 5255–5264 (2017)
23. Fokina, D., Oseledets, I.: Growing axons: Greedy learning of neural networks with application to function approximation. arXiv preprint arXiv:1910.12686 (2019)
24. Grohs, P., Hornung, F., Jentzen, A., von Wurstemberger, P.: A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of Black–Scholes partial differential equations. arXiv preprint arXiv:1809.02362 (2018)
25. Han, J., Jentzen, A., Weinan, E.: Overcoming the curse of dimensionality: solving high-dimensional partial differential equations using deep learning. arXiv preprint arXiv:1707.02568 (2017)
26. Han, J., Jentzen, A., Weinan, E.: Solving high-dimensional partial differential equations using deep learning. Proc. Natl. Acad. Sci. USA **115**(34), 8505–8510 (2018)
27. Han, J., Nica, M., Stinchcombe, A.R.: A derivative-free method for solving elliptic partial differential equations with deep neural networks. arXiv preprint arXiv:2001.06145 (2020)
28. Hermann, J., Schätzle, Z., Noé, F.: Deep-neural-network solution of the electronic Schrödinger equation. Nat. Chem. **12**(10), 891–897 (2020)
29. Herrmann, L., Schwab, C., Zech, J.: Deep ReLU Neural Network Expression Rates for Data-to-QoI Maps in Bayesian PDE Inversion. Technical Report 2020-02, Seminar for Applied Mathematics, ETH Zürich (2020)
30. Hesthaven, J., Rozza, G., Stamm, B.: Certified Reduced Basis Methods for Parametrized Partial Differential Equations, Springer Briefs in Mathematics, 1st edn. Springer, Zurich (2015)
31. Hesthaven, J.S., Ubbiali, S.: Non-intrusive reduced order modeling of nonlinear problems using neural networks. J. Comput. Phys. **363**, 55–78 (2018)
32. Hoang, V.H., Schwab, C.: Analytic regularity and polynomial approximation of stochastic, parametric elliptic multiscale PDEs. Anal. Appl. (Singap.) **11**(1), 1350001 (2013)

33. Hutzenthaler, M., Jentzen, A., Kruse, T., Nguyen, T.: A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations. arXiv preprint arXiv:1901.10854 (2019)
34. Jentzen, A., Salimova, D., Welti, T.: A proof that deep artificial neural networks overcome the curse of dimensionality in the numerical approximation of Kolmogorov partial differential equations with constant diffusion and nonlinear drift coefficients. *arXiv preprint* arXiv:1809.07321 (2018)
35. Khoo, Y., Lu, J., Ying, L.: Solving parametric PDE problems with artificial neural networks. arXiv preprint arXiv:1707.03351 (2017)
36. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
37. Kutyniok, G., Petersen, P.C., Raslan, M., Schneider, R.: A Theoretical analysis of deep neural networks and parametric PDEs. In: Constructive Approximation (2020)
38. Lagaris, I.E., Likas, A., Fotiadis, D.I.: Artificial neural networks for solving ordinary and partial differential equations. IEEE Trans. Neural Netw. **9**(5), 987–1000 (1998)
39. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
40. Lee, K., Carlberg, K.T.: Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. J. Comput. Phys. **404**, 108973 (2020)
41. Lu, J., Shen, Z., Yang, H., Zhang, S.: Deep network approximation for smooth functions. arXiv preprint arXiv:2001.03040 (2020)
42. Lu, L., Meng, X., Mao, Z., Karniadakis, G.: DeepXDE: a deep learning library for solving differential equations. arXiv preprint arXiv:1907.04502 (2019)
43. Lubbers, N., Smith, J.S., Barros, K.: Hierarchical modeling of molecular energies using a deep neural network. J. Chem. Phys. **148**(24), 241715 (2018)
44. Lye, K., Mishra, S., Molinaro, R.: A Multi-level procedure for enhancing accuracy of machine learning algorithms. Technical Report 2019-54, Seminar for Applied Mathematics, ETH Zürich, Switzerland (2019)
45. Marcati, C., Opschoor, J.A., Petersen, P.C., Schwab, C.: Exponential relu neural network approximation rates for point and edge singularities. arXiv preprint arXiv:2010.12217 (2020)
46. Mhaskar, H.: Neural networks for optimal approximation of smooth and analytic functions. Neural Comput. **8**(1), 164–177 (1996)
47. Nelsen, N., Stuart, A.: The random feature model for input–output maps between Banach spaces. arXiv preprint arXiv:2005.10224 (2020)
48. Novak, E., Woźniakowski, H.: Approximation of infinitely differentiable multivariate functions is intractable. J. Complex. **25**(4), 398–404 (2009)
49. Ohlberger, M., Rave, S.: Reduced basis methods: success, limitations and future challenges. arXiv preprint arXiv:1511.02021v2 (2016)
50. Opschoor, J., Petersen, P.C., Schwab, C.: Deep ReLU networks and high-order finite element methods. Anal. Appl. **18**(5), 715–770 (2020)
51. Petersen, P., Laakmann, F.: Efficient approximation of solutions of parametric linear transport equations by ReLU DNNs. In: Advances in Computational Mathematics, vol. 47 (2021)
52. Petersen, P.C., Voigtlaender, F.: Optimal approximation of piecewise smooth functions using deep ReLU neural networks. Neural Netw. **180**, 296–330 (2018)
53. Petersen, P.C., Voigtlaender, F.: Equivalence of approximation by convolutional neural networks and fully-connected networks. Proc. Am. Math. Soc. **148**, 1567–1581 (2020)
54. Poggio, T., Mhaskar, H., Rosasco, L., Miranda, B., Liao, Q.: Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review. Int. J. Autom. Comput. **14**(5), 503–519 (2017)
55. Powell, C., Lord, G., Shardlow, T.: An Introduction to Computational Stochastic PDEs, 1 edn, bol. 8. Texts in Applied Mathematics. Cambridge University Press, London (2014)
56. Quarteroni, A., Manzoni, A., Negri, F.: Reduced basis methods for partial differential equations, volume 92 of Unitext. Springer, Cham (2016). An introduction, La Matematica per il 3+2
57. Raissi, M.: Deep hidden physics models: deep learning of nonlinear partial differential equations. arXiv preprint arXiv:1801.06637 (2018)
58. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics informed deep learning (part i): data-driven solutions of nonlinear partial differential equations. arXiv preprint arXiv:1711.10561 (2017)
59. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics informed deep learning (part ii): data-driven discovery of nonlinear partial differential equations. arxiv. arXiv preprint arXiv:1711.10561, (2017)
60. Rauhut, H., Schwab, C.: Compressive sensing Petrov–Galerkin approximation of high-dimensional parametric operator equations. Math. Comput. **86**, 661–700 (2014)
61. Regazzoni, F., Dedè, L., Quarteroni, A.: Machine learning for fast and reliable solution of time-dependent differential equations. J. Comput. Phys. **397**, 108852 (2019)

62. Rozza, G., Huynh, D.B.P., Patera, A.T.: Reduced basis approximation and a posteriori error estimation for affinely parametrized elliptic coercive partial differential equations: application to transport and continuum mechanics. Arch. Comput. Methods Eng. **15**(3), 229–275 (2008)
63. Samaniego, E., Anitescu, C., Goswami, S., Nguyen-Thanh, V.M., Guo, H., Hamdia, K., Rabczuk, T., Zhuang, X.: An energy approach to the solution of partial differential equations in computational mechanics via machine learning: concepts, implementation and applications. arXiv preprint arXiv:1908.10407 (2019)
64. San, O., Maulik, R., Ahmed, M.: An artificial neural network framework for reduced order modeling of transient flows. Commun. Nonlinear Sci. Numer. Simul. **77**, 271–287 (2019)
65. Schwab, C., Zech, J.: Deep learning in high dimension: Neural network expression rates for generalized polynomial chaos expansions in UQ. Anal. Appl. (Singap.) **17**(1), 19–55 (2019)
66. Schütt, K.T., Sauceda, H.E., Kindermans, P.-J., Tkatchenko, A., Müller, K.-R.: SchNet—a deep learning architecture for molecules and materials. J. Chem. Phys. **148**(24), 241722 (2018)
67. Shaham, U., Cloninger, A., Coifman, R.R.: Provable approximation properties for deep neural networks. Appl. Comput. Harmon. Anal. **44**(3), 537–557 (2018)
68. Sirignano, J., Spiliopoulos, K.: DGM: A deep learning algorithm for solving partial differential equations. J. Comput. Phys. **375**, 1339–1364 (2018)
69. Tripathy, R., Bilionis, I.: Deep UQ: learning deep neural network surrogate models for high dimensional uncertainty quantification. J. Comput. Phys. **375**, 02 (2018)
70. Wackerly, D., Mendenhall, W., Scheaffer, R.: Mathematical Statistics with Applications, 7th edn. Cengage Learning, Boston (2014)
71. Webster, C., Tran, H., Dexter, N.: A mixed ℓ_1 regularization approach for sparse simultaneous approximation of parameterized PDEs. ESAIM Math. Model. Numer. **53**, 2025–2045 (2019)
72. Weinan, E., Han, J., Jentzen, A.: Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. Commun. Math. Stat. **5**(4), 349–380 (2017)
73. Weinan, E., Yu, B.: The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. Commun. Math. Stat. **6**(1), 1–12 (2018)
74. Yang, Y., Perdikaris, P.: Physics-informed deep generative models. arXiv preprint arXiv:1812.03511 (2018)
75. Yarotsky, D.: Error bounds for approximations with deep ReLU networks. Neural Netw. **94**, 103–114 (2017)
76. Yarotsky, D.: Optimal approximation of continuous functions by very deep ReLU networks. arXiv preprint arXiv:1802.03620 (2018)
77. Zhou, D.-X.: Theory of deep convolutional neural networks: downsampling. Neural Netw. **124**, 319–327 (2020)

Affiliations

Moritz Geist[1] · Philipp Petersen[2] · Mones Raslan[1] · Reinhold Schneider[1] · Gitta Kutyniok[3,4]

✉ Philipp Petersen
philipp.petersen@univie.ac.at

Moritz Geist
geist@math.tu-berlin.de

Mones Raslan
raslan@math.tu-berlin.de

Reinhold Schneider
schneidr@math.tu-berlin.de

Gitta Kutyniok
kutyniok@math.lmu.de

[1] Institut für Mathematik, Technische Universität Berlin, Straße des 17. Juni 136, 10623 Berlin, Germany

[2] Faculty of Mathematics and Research Plattform Data Science @ Uni Vienna, University of Vienna, Oskar Morgenstern Platz 1, 1090 Vienna, Austria

[3] Mathematisches Institut, Ludwig-Maximilians-Universität München, Theresienstr. 39, 80333 Munich, Germany

[4] Department of Physics and Technology, University of Tromsø, Tromsø, Norway