SIAM J. MATH. DATA SCI.
Vol. 3, No. 2, pp. 624–655

The Gap between Theory and Practice in Function Approximation with Deep Neural Networks*

Ben Adcock† and Nick Dexter†

Abstract. Deep learning (DL) is transforming whole industries as complicated decision-making processes are being automated by *deep neural networks* (DNNs) trained on real-world data. Driven in part by a rapidly expanding literature on DNN approximation theory showing that DNNs can approximate a rich variety of functions, these tools are increasingly being considered for problems in scientific computing. Yet, unlike more traditional algorithms in this field, relatively little is known about DNNs in relation to the principles of numerical analysis, namely, stability, accuracy, computational efficiency, and sample complexity. In this paper we first introduce a computational framework for examining DNNs in practice, and then use it to study their empirical performance with regard to these issues. We examine the performance of DNNs of different widths and depths on a variety of test functions in various dimensions, including smooth and piecewise smooth functions. We also compare DL against best-in-class methods for smooth function approximation based on compressed sensing. Our main conclusion from these experiments is that there is a crucial gap between the approximation theory of DNNs and their practical performance, with trained DNNs performing relatively poorly on functions for which there are strong approximation results (e.g., smooth functions) yet performing well in comparison to best-in-class methods for other functions. To analyze this gap further, we then provide some theoretical insights. We establish a *practical existence theorem*, which asserts the existence of a DNN architecture and training procedure that offers the same performance as compressed sensing. This result establishes a key theoretical benchmark. It demonstrates that the gap can be closed, albeit via a DNN approximation strategy which is guaranteed to perform as well as, but no better than, current best-in-class schemes. Nevertheless, it demonstrates the promise of practical DNN approximation by highlighting the potential for developing better schemes through the careful design of DNN architectures and training strategies.

Key words. neural networks, deep learning, function approximation, compressed sensing, numerical analysis

AMS subject classifications. 41A25, 41A46, 42C05, 65D05, 65D15, 65Y20, 94A20

DOI. 10.1137/20M131309X

1. Introduction.

The past decade has seen an explosion of interest in the field of machine learning, largely due to the impressive results achieved with deep neural networks (DNNs). Breakthroughs have been obtained on large classes of historically challenging problems, including speech recognition [19, 47] and natural language processing [84], image classification [50, 71], game intelligence [70], and autonomous vehicles [32]. As DNNs have shown such promise in these real-world applications, a trend has developed in the scientific computing

*Received by the editors January 17, 2020; accepted for publication (in revised form) January 11, 2021; published electronically May 6, 2021.
https://doi.org/10.1137/20M131309X
Funding: This work was supported by the PIMS CRG "High-dimensional Data Analysis," by SFU's Big Data Initiative "Next Big Question" Fund, and by NSERC grant R611675. The work of the second author was also supported by the PIMS Postdoctoral Fellowship program.
†Department of Mathematics, Simon Fraser University, Burnaby, BC, V5A 1S6, Canada (ben_adcock@sfu.ca, nicholas_dexter@sfu.ca).

community towards applying them to problems in mathematical modeling and computational science. Recent studies have focused on applications in image reconstruction tasks in medical imaging [5], discovering underlying *partial differential equation* (PDE) dynamics [67] and approximating solutions of PDEs [31, 28], and complex mathematical modeling, prediction, and classification tasks in physics [12], biology [79, 72, 88], and engineering [55, 78].

Simultaneously, the broader applied mathematics community has taken interest in the approximation capabilities of neural networks (NNs) [85, 6, 65, 8]. The earliest results in this direction [17, 48] established that a fully connected NN with even a single hidden layer has the universal approximation capability: as long as the number of nodes in the hidden layer is allowed to grow unbounded, such architectures are able to approximate any Borel measurable function on a compact domain to arbitrary uniform accuracy. More recent works have studied the connection between expressiveness of DNNs and their depth [53, 56, 86], while others have established connections between DNNs and other methods of approximation, e.g., sparse grids [58], splines [81], polynomials [68, 20], and "h, p"-finite elements [63]. A plethora of results now exist concerning the approximation power of DNNs for different function spaces—e.g., Sobolev spaces [42], bandlimited functions [58], analytic functions [30, 64], Barron functions [29], cartoon-like functions [41], and Hölder spaces [69]—and for tasks in scientific computing, such as approximation of high-dimensional functions [68, 52] and PDEs [40, 11, 51], dimensionality reduction [87], and methods for differential equations (DEs) [57, 31]. Theoretically, these works establish best-in-class approximation properties of DNNs for many problems. Concurrently, some works have sought to address the construction of DNN approximations directly, although typically without theoretical guarantees on trainability. For example, [24, 34] employ ideas from greedy methods in order to do this, [25] derives a formula for integral representations of shallow *rectified linear unit* (ReLU) networks, and [21] constructs networks directly approximating the Legendre basis, using this as an initialization point for training. On the other hand, [18] and other works reinterpret DNN approximation as approximation in adaptive bases, which can be learned via training data.

1.1. Challenges. Yet despite the impressive empirical and theoretical results achieved in the broader DL community, there is concern that methods based on DNNs do not currently meet the usual rigorous standards for algorithms in computational science [7]. While the aforementioned theoretical results assert the *expressibility* of the class of DNNs—that is, the *existence* of a DNN of a given architecture that achieves a desired rate of convergence for a given problem—they say little about its practical performance when trained by modern approaches in DL. If such techniques are to achieve widespread adoption in scientific computing, it is vital they be understood through the lens of numerical analysis, namely, (i) stability, (ii) accuracy, (iii) sample complexity, (iv) the curse of dimensionality, and (v) computational cost.

(i) Stability. Recently, researchers have begun to question the stability properties of DNNs [59, 77, 33]. A series of works have demonstrated that DNNs trained on tasks such as image classification are vulnerable to misclassification when provided images with small "adversarial" perturbations [60] and can even completely fail on image reconstruction tasks in the presence of small structural changes in the data [4, 39]. As deep learning is increasingly being applied towards critical problems in healthcare, e.g., DeepMind's recent work on machine-assisted diagnostic imaging in retinal disease [22], many researchers have questioned

the ethics of applying such tools, because the effects of their stability properties on these problems are not fully understood.

(ii) Accuracy. Over the past five years, many works have been published on the classes of functions, e.g., analytic or piecewise continuous, that can be approximated by DNNs of a given size with a certain rate of convergence. These results are constructive, often showing the existence of a DNN emulating another approximation scheme, e.g., polynomials, for which convergence rates have already been established. While such results provide a useful benchmark for DNN expressivity, they do not suggest methods for training DNNs that reliably achieve the tolerances required in computational science applications.

(iii) Sample complexity. Areas in which DL has seen the greatest success include problems in supervised learning, such as image classification. In such settings, DNNs are trained on large sets of labeled images, yielding a model capable of predicting labels for unseen images. Popular datasets for DL competitions include the ImageNet database, which contains 14 million hand-annotated images of more than 20,000 categories of subjects [23]. In contrast, problems in computational science are often relatively *data-starved*, e.g., applications in *uncertainty quantification* (UQ) which involve computing a quantity of interest from sampled solutions of a parameterized PDE [43]. As each sample involves the discretization and solution of a PDE, which may require thousands of degrees of freedom to accurately resolve, there is great attention paid in such problems to minimizing the required number of samples [3, 27].

(iv) Curse of dimensionality. Many modern problems in scientific computing involve high dimensionality. High-dimensional PDEs occur in numerous applications, and parameterized PDEs in UQ applications often involve from tens to hundreds of variables. Recent works have shown that certain DNNs have the expressive capabilities to mitigate the curse of dimensionality to the same extent as current best-in-class schemes [58, 51, 30, 64, 68, 40, 11]. Yet, as noted, this does not assert that these rates can be achieved via training. Moreover, the curse of dimensionality is an important consideration in the sample complexity, as the cost of obtaining samples can often dominate the overall cost. A recent numerical study has shown that approximation quality degrades with increasing dimension for approximating solutions of high-dimensional parameterized PDEs [35]. However, the observed scaling is not exponential in the dimension d but dependent on the complexity of the underlying problem. Understanding this scaling with respect to the sample complexity is crucial to applying these methods in computational science.

(v) Computational cost. By far the largest barrier to entry for DL research is the cost of training. DNNs are typically trained on *graphics processing units* (GPUs), and a single GPU can cost thousands of U.S. dollars. In many industrial applications, models are trained on hundreds of these specialized cards. In addition, the training process itself is very energy-intensive and can produce a large amount of excess CO_2 emissions.[1] Even a small reduction in computational cost can yield large cost savings and greater access to resources for researchers.

In the near term, barring advances to algorithms to reduce the complexity of training, it seems likely that any DL implementation will pay a price in computational cost. Hence there

[1] A recent study estimated the cost of training a natural language processing model for 274,000 GPU hours at between $942,000 and $3,300,000 U.S. dollars, meanwhile producing in excess of 626,000 lbs of CO_2, or the equivalent of five cars' output over their expected lifespans [76].

needs to be a clear understanding of the benefits vis-à-vis properties (i)–(iv) above. The study of these concerns is the broad purpose of this paper.

1.2. Contributions. Our main objective is to examine practical DNN approximation on problems motivated by scientific computing. In many applications in computational science, the core task involves approximating a function $f : \mathcal{U} \to \mathbb{R}$, with domain $\mathcal{U} \subset \mathbb{R}^d$ where $d \geq 1$ (often $d \gg 1$). Hence our main aim is to examine the performance of DL on practical function approximation through the five considerations (i)–(v). Our main contributions are as follows:

1. We develop a computational framework for examining the practical capabilities of DNNs in scientific computing, based on the rigorous testing principles of numerical analysis. We provide clear practical guidance on training DNNs for function approximation problems.

2. We conduct perhaps the first comprehensive empirical study of the performance of training fully connected feedforward ReLU DNNs on standard function classes considered in numerical analysis, namely, (piecewise) smooth functions on bounded domains. We compare performances over a range of dimensions, examining the capability of DL for mitigating the curse of dimensionality. We examine the effect of network architecture (depth and width) on both ease of training and approximation performance of the trained DNNs. We also make a clear empirical comparison between DL and current best-in-class approximation schemes for smooth function approximation. The latter is based on polynomial approximation via *compressed sensing* (CS) [3], which (as we also show) achieves exponential rates of convergence for analytic functions in arbitrarily many dimensions; see section 5.1 for more details.

3. We present theoretical analysis that compares the performance of DL to that of CS for smooth function approximation. In particular, we establish a novel *practical existence theorem*, which asserts the existence of a DNN architecture and training procedure (based on minimizing a certain cost function) that attains the same exponential rate of convergence as CS for analytic function approximation with the same sample complexity.

1.3. Conclusions. The primary conclusion of this work is the following. While it is increasingly well understood that DNNs have substantial expressive power for problems relevant to scientific computing, there remains a large gap between expressivity and practical performance achieved with standard methods of training. Surprisingly, trained DNNs can perform very badly on functions for which there are strong expressivity results, such as smooth functions in high dimensions and piecewise smooth functions. Yet, on other examples, they are competitive with current best-in-class schemes based on CS. We also draw the following conclusions based on our experimental results training fully connected feedforward ReLU DNNs:

1. The accuracy of trained DNNs is limited by the precision used but is typically nowhere near machine epsilon despite training to such tolerances in the loss. In this work, we perform experiments in both single and double precision. Yet, in both cases, it is typically impossible to get beyond four digits of accuracy even when approximating extremely smooth functions. In contrast, a combination of Legendre polynomial approximation in the *hyperbolic cross* subspace with weighted ℓ_1-minimization and a specific choice of weights motivated by smooth function approximation can reliably achieve six or seven digits of accuracy on such problems. Since our theoretical contribution shows that DNNs should be capable of obtaining these results (albeit with a different initialization and training procedure), this fact suggests a limitation of standard training methods in obtaining more accurate results.

2. The training process itself is also highly sensitive to the parameterization of the solvers and initialization of the weights and biases of the DNNs. After extensive testing, we chose the `Adam` (**a**daptive **m**oments) optimizer with exponentially decaying learning rate over a variety of optimizers, including standard `SGD` (**s**tochastic **g**radient **d**escent), finding empirically that this strategy can help to mitigate some of the challenges of solving the nonconvex optimization problem of training, e.g., nonmonotonic decrease in the loss and slow convergence to minimizers. We also initialize our networks with symmetric uniform or normal distributions with small variance, finding larger variances can lead to failure in training or slow convergence. These choices combined lead to a training process that is largely stable and minimizes the *probability of failure* in training over a wide range of architectures. However, failures can still occur and are often a consequence of choosing a network that is either too shallow and narrow (where failure occurs immediately and the error stagnates) or too wide and deep (where the resulting network achieves order machine epsilon tolerance in the loss but massively overfits exhibiting numerical artifacts).

3. Generally speaking, deeper architectures (which are sufficiently wide) are both easier to train and have better performance than shallower architectures. However, this is clearly highly dependent on the regularity and dimensionality of the target function for smooth-function approximation problems. The width of the network also plays an important role; we find that networks with a width 5–10 times larger than the depth perform better. While this trend appears to be general, some of our results on less smooth and piecewise continuous functions indicate that it is not universal.

4. While much of the success of DL has been in the field of classification, surprisingly, performance of trained DNNs on simple piecewise constant functions is relatively poor in comparison to smooth functions and adversely affected by the curse of dimensionality. Yet, DNNs do approximate such functions to some accuracy, unlike polynomial-based CS techniques. This highlights the flexibility of the DL approach.

We also draw the following theoretical conclusions:

5. For analytic function approximation, a certain DNN strategy based on emulating polynomials via a suitable DNN can achieve the same guaranteed performance (up to constants) as best-in-class schemes based on CS.

6. However, such a scheme will not typically offer any superior performance. In particular, it is not *flexible*. It only performs well on the function classes on which CS performs well, and it performs correspondingly poorly on other, e.g., piecewise smooth, functions. This is in contrast to the experimental results in this paper, which show that standard feedforward architectures and training via the ℓ^2-loss can approximate both smooth and nonsmooth functions as well as expressibility results, which show that DNN architectures can approximate functions from a range of different function classes.

1.4. Outlook. This paper raises and seeks to answer the following question: Is DL a useful tool for problems in scientific computing? Some of the above conclusions may appear rather negative in this regard, certainly in comparison to the positive impression given by the plethora of expressivity results on DNNs. Let us raise several caveats. First, this study considers one particular setup, namely, fully connected, ReLU networks of constant hidden layer widths, in combination with the ℓ^2-loss function. There are almost countless variations

on this setup, some of which will undoubtedly perform better. These variations include different activation functions (e.g., sigmoid, hyperbolic tangent), different architectures (e.g., ResNets, sparsely connected layers, convolutional layers), and different loss functions (e.g., those incorporating regularization). We elected to use this setup based on standards in the literature; for instance, most expressibility results consider ReLU activations. Given the difficulties and intense computational resources required for training, it is beyond the scope of this paper to methodically compare all possible setups. Second, the practical existence theorem we prove shows that a careful choice of architecture and cost function can allow DNNs to offer similar performance to state-of-the-art techniques. While it comes at the price of flexibility, this result nonetheless provides a theoretical benchmark for DNN approximations. It also demonstrates the potential of DNN strategies for eventually outperforming such methods and indicates that theory-inspired architecture and training strategy design comprise a way to achieve such improvements. The extent to which this can be (provably) done while maintaining flexibility—i.e., the ability to approximate different function classes with the same DNN procedure—is an enticing challenge for future work.

1.5. Outline. The outline of the remainder of this paper is as follows. In section 2 we introduce the approximation problem, DNNs, DL, and CS. In section 3 we describe the experimental setup, including details of the training procedure used. Our numerical results are found in section 4. Finally, in section 5 we present our theoretical results. Additional information for this paper is contained in the supplementary materials file MLFA_supplement.pdf [local/web 2.30MB]. Code accompanying the computational framework is available at https://github.com/ndexter/MLFA.

2. Framework.

In this section, we first describe the function approximation problem, and then introduce DNNs, DL, polynomial approximation, and CS.

2.1. Problem formulation. Throughout this paper, we consider the unit cube in d dimensions, $\mathcal{U} = (-1,1)^d$, equipped with the uniform probability measure $\mathrm{d}\varrho = 2^{-d}\,\mathrm{d}\boldsymbol{x}$, where $\mathrm{d}\boldsymbol{x}$ is the Lebesgue measure and $\boldsymbol{x} = (x_1,\dots,x_d)$ is the d-dimensional variable. Let $L^2(\mathcal{U})$ denote the space of real-valued square-integrable functions on $\mathcal{U}$ with respect to ϱ. Our objective is to approximate an unknown function $f \in L^2(\mathcal{U})$ from samples. These samples are generated by simple Monte Carlo sampling: we draw $\boldsymbol{x}_1,\dots,\boldsymbol{x}_m$ randomly and independently from the measure ϱ. Hence the approximation problem we aim to solve is

(AP) Given the measurements $\{(\boldsymbol{x}_i, f(\boldsymbol{x}_i))\}_{i=1}^m$, approximate f.

We note in passing that much of what follows in this paper can be extended to more general domains, sampling strategies, and functions taking values in other vector spaces (for instance, complex-valued functions, vector-valued functions, or even Hilbert-valued functions, as arise commonly in UQ applications [27]). We assume the above setup for ease of presentation.

We require several further pieces of notation. We write $\|\cdot\|_{L^2}$ for the L^2-norm with respect to ϱ. The space of essentially bounded functions on $\mathcal{U}$ is denoted by $L^\infty(\mathcal{U})$ and its norm by $\|\cdot\|_{L^\infty}$. We use $\boldsymbol{\nu} = (\nu_1,\dots,\nu_d)$ to denote a (multi)index of length d. If $0 < p < \infty$ and $\mathcal{F} \subseteq \mathbb{N}_0^d$ is a finite or countable (multi)index set, we write $\ell^p(\mathcal{F})$ for the space of ℓ^p-summable

sequences $\boldsymbol{c} = (c_{\boldsymbol{\nu}})_{\boldsymbol{\nu}\in\mathcal{F}} \subset \mathbb{R}$, i.e., those satisfying $\|\boldsymbol{c}\|_p := \left(\sum_{\boldsymbol{\nu}\in\mathcal{F}} |c_{\boldsymbol{\nu}}|^p\right)^{1/p} < \infty$. When $p = \infty$, we define $\ell^\infty(\mathcal{F})$ and $\|\cdot\|_\infty$ in the usual way.

2.2. Deep learning. We now introduce DL. First, we recall the definition of a DNN.

Definition 2.1 (deep neural network). *Let $L \in \mathbb{N}_0$ and $N_0, \dots, N_{L+2} \in \mathbb{N}$. A map $\Phi : \mathbb{R}^{N_0} \to \mathbb{R}^{N_{L+2}}$ given by*

$$\Phi(x) = \begin{cases} \mathcal{A}_1(\rho(\mathcal{A}_0(\boldsymbol{x}))), & L = 0, \\ \mathcal{A}_{L+1}(\rho(\mathcal{A}_L(\rho(\cdots \rho(\mathcal{A}_0(\boldsymbol{x}))\cdots)))), & L \geq 1, \end{cases} \tag{2.1}$$

with affine linear maps $\mathcal{A}_l : \mathbb{R}^{N_l} \to \mathbb{R}^{N_{l+1}}$, $l = 0, \dots, L+1$, and the activation function ρ acting componentwise (i.e., $\rho(\boldsymbol{x}) := (\rho(x_1), \dots, \rho(x_d))$ for $\boldsymbol{x} = (x_1, \dots, x_d)$) is called a neural network (NN). The map $\mathcal{A}_l$ corresponding to layer l is given by $\mathcal{A}_l(\boldsymbol{x}) = \boldsymbol{W}_l\boldsymbol{x} + \boldsymbol{b}_l$, where $\boldsymbol{W}_l \in \mathbb{R}^{N_{l+1}\times N_l}$ is the lth weight matrix and $\boldsymbol{b}_l \in \mathbb{R}^{N_{l+1}}$ the lth bias vector. We refer to L as the depth of the network and to $\max_{1\leq l\leq L+1} N_l$ as its width.

Informally, we consider a DNN as any NN with $L \geq 1$ hidden layers. Definition 2.1 pertains to *feedforward* DNNs. We do not consider more exotic constructions such as recurrent networks or ResNets in this paper. We also consider so-called *fully connected* networks, meaning that the weights and biases can take arbitrary real values. The layers $l = 1, \dots, L$ are referred to as *hidden* layers. In our experiments later, we set their widths to be equal, $N_1 = \cdots = N_{L+1}$. Note that N_0 and N_{L+2} are specified by the problem. In our case, $N_0 = d$ and $N_{L+2} = 1$.

There are numerous choices for the activation function ρ, and moreover, one may also choose different activation functions in different layers. Since it is popular in both theory and practice, we use the *rectified linear unit* (ReLU), defined by $\rho(x) = \max\{0, x\}$. We will also refer to a DNN architecture having ReLU activation function and L hidden layers with N nodes per layer as a ReLU $L \times N$ DNN.

The *architecture* of a network is the specific choice of activation ρ and parameters L and $N_1, \dots, N_{L+1}$. We denote the set of NNs of a given architecture by $\mathcal{N}$. Note that this family is parameterized by the weight matrices and biases. Selecting the right architecture for a given problem is a significant challenge. We discuss this topic further in sections 3 and 4.

Given an unknown function $f \in L^2(\mathcal{U})$, we say that *training* is the process of computing an NN Φ that approximates f from the data $\{(\boldsymbol{x}_i, f(\boldsymbol{x}_i))\}_{i=1}^m$. This is normally achieved by minimizing a *loss function* $\mathcal{L} : \mathcal{N} \to \mathbb{R}$, i.e., we solve

$$\text{minimize}_{\Phi\in\mathcal{N}} \mathcal{L}(\Phi),$$

where $\mathcal{N}$ is the family of NNs of the chosen architecture. Note that this is equivalent to a minimization problem for the weights $\boldsymbol{W}_l$ and biases $\boldsymbol{b}_l$. A typical choice is the ℓ^2-loss (also known as *empirical risk*, *mean squared loss*),

$$\mathcal{L}(\Phi) := \frac{1}{m}\sum_{i=1}^m (\Phi(\boldsymbol{x}_i) - f(\boldsymbol{x}_i))^2 . \tag{2.2}$$

We primarily use this loss function in this paper. However, many other choices are possible. For instance, it is common to add a regularization term to the loss function, e.g.,

$$\mathcal{L}(\Phi) := \frac{1}{m}\sum_{i=1}^{m} (\Phi(\boldsymbol{x}_i) - f(\boldsymbol{x}_i))^2 + \mathcal{J}(\Phi).$$

Here $\mathcal{J} : \mathcal{N} \to \mathbb{R}$ is chosen to promote some desirable features of the network. For instance, $\mathcal{J}$ may be a norm of the weight matrices, and thus promotes small and/or sparse weights.

2.3. Polynomial approximation of smooth functions. We now introduce the polynomial approximation schemes against which we compare DL for function approximation. Polynomial approximation is a vast and classical topic. It has received renewed attention in the last several decades, motivated by applications in UQ where one seeks to approximate a smooth quantity of interest of a parametric PDE [16]. The particular scheme we consider is based on orthogonal expansions in orthonormal polynomials in $L^2(\mathcal{U})$, i.e., multivariate Legendre polynomials. The univariate, orthonormal Legendre polynomials on the $[-1, 1]$ are defined by

$$\psi_\nu(x) = \sqrt{2\nu + 1}P_\nu(x), \qquad \nu \in \mathbb{N}_0,$$

where P_ν is the classical Legendre polynomial with normalization $P_\nu(1) = 1$. The functions ψ_ν form an orthonormal basis of $L^2(-1, 1)$. When $d > 1$, we define the tensor orthonormal Legendre polynomials as

$$\Psi_{\boldsymbol{\nu}}(\boldsymbol{x}) = \prod_{i=1}^{d} \psi_{\nu_i}(x_i), \qquad \boldsymbol{\nu} = (\nu_1, \dots, \nu_d) \in \mathbb{N}_0^d,\ \boldsymbol{x} = (x_1, \dots, x_d) \in \mathcal{U}.$$

The set $\{\Psi_{\boldsymbol{\nu}}\}_{\boldsymbol{\nu}\in\mathbb{N}_0^d}$ forms an orthonormal basis of $L^2(\mathcal{U})$. Hence any function $f \in L^2(\mathcal{U})$ has a convergent expansion

$$f = \sum_{\boldsymbol{\nu}\in\mathbb{N}_0^d} c_{\boldsymbol{\nu}}\Psi_{\boldsymbol{\nu}}, \tag{2.3}$$

where $c_{\boldsymbol{\nu}} = \int_{\mathcal{U}} f(\boldsymbol{x})\Psi_{\boldsymbol{\nu}}(\boldsymbol{x})\,\mathrm{d}\varrho(\boldsymbol{x})$ is the coefficient of f with respect to $\Psi_{\boldsymbol{\nu}}$. Note that the sequence $\boldsymbol{c} = (c_{\boldsymbol{\nu}})_{\boldsymbol{\nu}\in\mathbb{N}_0^d}$ is an element of $\ell^2(\mathbb{N}_0^d)$, the space of square-summable sequences with indices in $\mathbb{N}_0^d$. By Parseval's identity, $\|f\|_{L^2(\mathcal{U})} = \|\boldsymbol{c}\|_2$.

When $d = 1$, approximating a smooth function in the Legendre basis is typically achieved by truncating the expansion (2.3) after its first s terms, then using, for instance, least-squares to approximately recover the coefficients $c_0, \dots, c_{s-1}$ from the measurements $\{(x_i, f(x_i))\}_{i=1}^m$. In $d \geq 2$ dimensions, the situation becomes more complicated, since there are many different choices of index set S of cardinality s one might employ to truncate the expansion (2.3): $f \approx f_S = \sum_{\boldsymbol{\nu}\in S} c_{\boldsymbol{\nu}}\Psi_{\boldsymbol{\nu}}$. By Parseval's identity, the error $\|f - f_S\| = \sqrt{\sum_{\boldsymbol{\nu}\in\mathbb{N}_0^d\setminus S} |c_{\boldsymbol{\nu}}|^2}$ depends on the coefficients outside S. Hence, an a priori choice of S may have limited effectiveness, since it may fail to capture any anisotropic behavior of f. This naturally motivates the concept of *best s-term approximation* [16]. In best s-term approximation, the index set S is chosen

so that it contains the multi-indices corresponding to the large s coefficients $c_{\boldsymbol{\nu}}$ in absolute value. This is a type of nonlinear approximation scheme [26]. If $\tilde{f}_s$ denotes the best s-term approximation, the error satisfies

$$\|f - \tilde{f}_s\|_{L^2} = \inf \left\{ \sqrt{\sum_{\boldsymbol{\nu} \notin S} |c_{\boldsymbol{\nu}}|^2} : S \subset \mathbb{N}_0^d,\ |S| \leq s \right\}.$$

Under appropriate conditions (e.g., f is analytic), this approximation converges exponentially fast in s (see Theorem 5.2). In high dimensions this is a significant improvement over any linear approximation scheme based on a fixed, isotropic choice of S. We discuss this further in section 5.1.

2.4. Polynomial approximation with compressed sensing. On the face of it, computing the best s-term approximation is a daunting task. In theory, it involves computing all infinitely many of the coefficients $\boldsymbol{c}$ and then selecting the largest s. This is of course intractable, but it is generally still computationally infeasible even if one limits oneself to computing a large but finite number of coefficients.

A solution is to use compressed sensing. Here one first selects a large but finite multi-index set $\Lambda \subset \mathbb{N}_0^d$. This set is generally assumed to contain the coefficients of some quasi-best s-term approximation, if not the coefficients of the true best s-term approximation itself. For reasons that will be made clear in section 5.2, a reasonable choice is $\Lambda = \Lambda_s^{\mathrm{HC}}$, where

$$\Lambda_s^{\mathrm{HC}} = \left\{ \boldsymbol{\nu} = (\nu_1, \ldots, \nu_d) \in \mathbb{N}_0^d : \prod_{j=1}^{d} (\nu_j + 1) \leq s + 1 \right\} \tag{2.4}$$

is the *hyperbolic cross* index set of degree s.

Having chosen Λ, we can now assume that the finite vector $\boldsymbol{c}_\Lambda = (c_{\boldsymbol{\nu}})_{\boldsymbol{\nu} \in \Lambda}$ is approximately sparse. Next, one formulates the normalized measurement matrix and vector of measurements

$$\boldsymbol{A} = \left(\frac{\Psi_\nu(\boldsymbol{x}_i)}{\sqrt{m}} \right)_{\substack{1 \leq i \leq m \\ \boldsymbol{\nu} \in \mathcal{J}}}, \qquad \boldsymbol{f} = \left(\frac{f(\boldsymbol{x}_i)}{\sqrt{m}} \right)_{1 \leq i \leq m}. \tag{2.5}$$

Then one searches for an approximately sparse solution of the linear system $\boldsymbol{A}\boldsymbol{z} = \boldsymbol{f}$. A standard means for doing this is to solve the *quadratically constrained basis pursuit* problem

$$\text{minimize}_{\boldsymbol{z} \in \mathbb{R}^N} \|\boldsymbol{z}\|_1 \quad \text{s.t.} \quad \|\boldsymbol{A}\boldsymbol{z} - \boldsymbol{f}\|_2 \leq \eta \tag{2.6}$$

for suitably chosen $\eta \geq 0$, or the *unconstrained LASSO* problem

$$\text{minimize}_{\boldsymbol{z} \in \mathbb{R}^n} \|\boldsymbol{z}\|_1 + \mu \|\boldsymbol{A}\boldsymbol{z} - \boldsymbol{f}\|_2^2, \tag{2.7}$$

for appropriately chosen $\mu > 0$. A solution $\hat{\boldsymbol{c}} = (\hat{c}_{\boldsymbol{\nu}})_{\boldsymbol{\nu} \in \Lambda}$ of either problem yields an approximation $\hat{f} = \sum_{\boldsymbol{\nu} \in \Lambda} \hat{c}_{\boldsymbol{\nu}} \Psi_{\boldsymbol{\nu}}$ of f.

Unfortunately, simply promoting the sparsity of the polynomial coefficients via the ℓ^1-norm is not sufficient to achieve favorable sample complexity bounds. Bounds on m for ℓ^1-norm based approaches can be exponential in the dimension d [3]. Fortunately, as considered

in [66, 1, 13], this issue can be overcome by replacing the ℓ^1-norm with a certain weighted ℓ^1-norm. For instance, instead of (2.6) one now solves

$$\text{minimize}_{\boldsymbol{z}\in\mathbb{R}^N} \|\boldsymbol{z}\|_{1,\boldsymbol{u}} \quad \text{s.t.} \quad \|\boldsymbol{A}\boldsymbol{z}-\boldsymbol{f}\|_2 \leq \eta. \tag{2.8}$$

Here $\boldsymbol{u} = (u_{\boldsymbol{\nu}})_{\boldsymbol{\nu}\in\Lambda}$ is a vector of weights, and $\|\boldsymbol{z}\|_{1,\boldsymbol{u}} = \sum_{\boldsymbol{\nu}\in\Lambda} u_{\boldsymbol{\nu}}|z_{\boldsymbol{\nu}}|$. As shown in [1], an appropriate choice of weights is

$$u_{\boldsymbol{\nu}} = \|\Psi_{\boldsymbol{\nu}}\|_{L^\infty} = \prod_{j=1}^{d} \sqrt{2\nu_j+1}, \qquad \boldsymbol{\nu} = (\nu_1,\dots,\nu_d). \tag{2.9}$$

For the remainder of this paper, we consider the CS polynomial approximation scheme $\hat{f} = \sum_{\boldsymbol{\nu}\in\Lambda} \hat{c}_{\boldsymbol{\nu}}\Psi_{\boldsymbol{\nu}}$, where $\hat{\boldsymbol{c}}$ is a solution of either (2.8) or, for the sake of comparison, (2.6).

3. Testing setup.

We now describe the testing setup for our experiments. Training DNNs requires careful choices of the optimization solver, initialization, and optimization parameters. This section describes the choices we made to deliver consistent performance across a range of DNN architectures. In summary, we find that the `Adam` optimizer with exponentially decaying learning rate and a specific random initialization delivers this performance, whereas other solvers, such as `SGD` and other learning rate schedules, perform less consistently.

3.1. Setup.

We first summarize the main methodology:

(i) Implementation. Our framework has been implemented in a package called `MLFA` (**M**achine **L**earning **F**unction **A**pproximation) in version 1.13 of Google's `TensorFlow` software library https://www.tensorflow.org/ and is available on GitHub at https://github.com/ndexter/MLFA. Details about the set of features supported by `MLFA` and data recorded by the code can also be found on the GitHub page.

(ii) Hardware. In the course of testing our DNN models, we observed improved accuracy on some of our test problems by initializing and training the networks in double precision. Modern GPUs often support half, single, and double precision arithmetic, though many commonly available GPUs are optimized to perform single precision computations much more quickly (see supplementary material section SM1.1). The majority of our computations were performed in single precision using the Tesla P100 GPUs on Compute Canada's Cedar compute cluster at Simon Fraser University,[2] though for some of our test problems we provide double precision results for a subset of the architectures considered for comparison.

(iii) Choice of architectures and initialization. We consider fully connected ReLU networks. This choice is inspired by the many theoretical existence results on such networks (see section 1). There is a vast literature suggesting various strategies for designing architectures and initializing neural networks. For an introduction to these topics see, e.g., [38, sections 5.2 and 8.4]. We recall the work [44] focusing on which DNN architectures result in exploding and vanishing gradients, a common problem in backpropagation which can result in failure during training. Our empirical results confirm that choosing DNN architectures with a fixed number of nodes per layer N and depth L such that the ratio $\beta := L/N$ is small, e.g.,

[2] See https://www.computecanada.ca/ and https://docs.computecanada.ca/wiki/Cedar.

$\beta \in (0.025, 0.5)$, is an effective choice for training. In section 3.3, we study several popular strategies for initializing DNNs. Our results show that initializing the weights and biases to be normal random variables with mean 0 and variance 0.01 is an effective choice for many of the architectures and problems considered herein. We note that for the range of architectures studied, this choice results in weights and biases with variance smaller than many other popular choices, e.g., the strategies from [37, 46], and is sufficiently small to avoid the failure modes analyzed in [45]. We also note that setting the variances smaller than 0.01 can result in networks which are more difficult to train, which can be attributed to the exponentially decaying length scales described in [45, Theorem 1]. The seed used for the random number generators can also have an important effect on the initialization. In this study, we initialize all of our networks from the same seed 0 for both `TensorFlow` and `NumPy` in order to reduce the complexity of our experiments. A more comprehensive study of the average performance of DL would require averaging over a large set of seeds used in initialization. We leave such a study to a future work.

(iv) Optimizers for training and parameterization. Section 3.4 compares the performance of a variety of solvers and learning rate schedules on the function approximation problem. In practice, we find that the `Adam` optimizer [49] with an exponentially decaying learning rate yields the most accurate results of the solvers tested in the least amount of training time. See supplementary material section SM1.2 for implementation details. In single precision, the DNNs are trained for 50,000 epochs or to a tolerance of $\varepsilon_{\text{tol}} = 5 \times 10^{-7}$, while in double precision DNNs are trained for 200,000 epochs or to a tolerance of $\varepsilon_{\text{tol}} = 5 \times 10^{-16}$. Due to the nonmonotonic convergence of minimizing the nonconvex loss (2.2) with respect to the weights and biases, we checkpoint our partially trained networks once the training loss has been reduced to one eighth of the previous checkpoint's training loss, saving the best result at the end of training. We then average our testing results over the final trained networks.

(v) Training data and design of experiments. To understand the average performance of DL on a variety of reconstruction tasks, we run 20 trials of solving (AP) with each of our DNN architectures and CS over a range of data sets of increasing size. We then average the testing error and run statistics over all trials for each data set in plotting. We define a trial as one complete run of training a DNN, initialized as above, or solving a CS problem on a set of training data consisting of the values $\{(\boldsymbol{x}_i, f(\boldsymbol{x}_i))\}_{i=1}^{m_k}$. To generate each data set of size m_k, with $0 < m_1 < m_2 < \cdots < m_{k_{\text{final}}}$, we sample 20 independent and identically distributed (i.i.d.) sets of points $\{\boldsymbol{x}_i\}_{i=1}^{m_k}$ from the uniform distribution on $(-1, 1)^d$ and evaluate our target function f at these points to form our training data. Since we are interested in the sample complexity of the DL problem, for most of our examples we choose $m_{k_{\text{final}}}$ to be relatively small, on the order of 1000 points.

(vi) Testing data and error metric. To study the generalization capabilities of DL, we use a common error metric for numerical analysis, the relative L^2 error

$$\varepsilon_{\text{rel}} = \|f - \tilde{f}\|_{L^2} / \|f\|_{L^2}, \tag{3.1}$$

where $\tilde{f}$ is an approximation obtained using either DL or CS. We purposefully choose this over other norms (e.g., the L^∞- or H^1-norms) so that we can use the same error metric across all function classes, including both smooth and nonsmooth functions. As we run multiple trials of

our experiments, we compute the average of (3.1) over all of our trials in testing. In contrast to the training data, we use deterministically generated points and function data for testing the relative L^2 errors. More specifically, we compute an approximation to the L^2 integrals using a high order isotropic Clenshaw–Curtis sparse grid quadrature [36] rule; see, e.g., [62] for more details. As opposed to the training data, we use a large set of testing points, e.g., ($d = 1$) 65,537, ($d = 2$) 311,297, ($d = 4$) 643,073, and ($d = 8$) 1,863,937 points, to ensure a good covering of our parameter space $\mathcal{U}$ in computing these statistics. For such moderate dimensional problems, an isotropic rule is sufficient for studying the generalization performance of both CS and DL. For higher-dimensional instances of (AP), the points and weights can be precomputed and reused in testing multiple functions. We rely on the `TASMANIAN` sparse grid toolkit [73, 74, 75] for the generation of these rules.

(vii) Compressed sensing. We solve the problems (2.6) and (2.8) (with weights (2.9)) using the MATLAB solver `SPGL1` [82, 83] in double precision. The parameter η is chosen as

$$\eta = \|\boldsymbol{A}\boldsymbol{c}_\Lambda - \boldsymbol{f}\|_2, \qquad \boldsymbol{c}_\Lambda = (c_{\boldsymbol{\nu}})_{\boldsymbol{\nu}\in\Lambda}. \tag{3.2}$$

To compute $\boldsymbol{c}_\Lambda$ we use the same sparse grid rule described above in evaluating each coefficient $c_{\boldsymbol{\nu}} = \int_{\mathcal{U}} f(\boldsymbol{x})\Psi_{\boldsymbol{\nu}}(\boldsymbol{x})\,\mathrm{d}\varrho(\boldsymbol{x})$. See supplementary material section SM2.1 for further discussion. We set Λ as in (2.4) to be the hyperbolic cross index set of degree s chosen so that $\#(\Lambda_s^{\mathrm{HC}}) \approx 3{,}000$.

3.2. Solvers.

The choice of solver and parameterization of the solver are important factors in training DNN models to a desired tolerance in a reasonable amount of time. A common choice in many computational science applications is the `Adam` optimizer, a variant of `SGD` incorporating moment estimates of the gradient. In the course of testing our implementation of the `MLFA` package, we studied the effect of the solvers on the generalization error and time to train given a fixed budget of 50,000 epochs in single precision. Figure 1 displays results for a variety of solvers and learning rate schedules. There we observe comparable performance for the `Adam` and `RMSProp` (root mean square propagation) algorithms in terms of accuracy, with `Adagrad` (adaptive gradient algorithm), `SGD`, and `PGD` (**p**roximal **g**radient **d**escent) performing the worst in both accuracy and computational cost. When comparing run times and accuracy for all methods tested, we found that the `Adam` optimizer achieves the best accuracy with the least computational cost. We also include results obtained with the `AdamW` (Adam with weight decay) optimizer [54], which implements a decoupled weight decay (a form of ℓ^2-regularization) on the weights and biases. In Figure 1 we observe that smaller values of the weight decay parameter λ allow the `AdamW` optimizer to achieve performance identical to that of `Adam` with minimal overhead but do not outperform standard `Adam`, while larger values of λ both decrease accuracy and increase run time.

For certain solvers, it is often the case that some of the trials of a given test may fail to achieve the desired loss tolerance before arriving at the final epoch. An example of this can be seen in the left plot of Figure 2, where none of the trials of `SGD` with a constant learning rate of 10^{-3} were able to achieve the desired tolerance in 50,000 epochs of training. The middle plot of Figure 2 displays the effect of using an exponentially decaying learning rate with `SGD`, though we also observe there that none of the trials trained with this learning rate schedule were able to achieve the tolerance in 50,000 epochs of training. On the other hand, the right

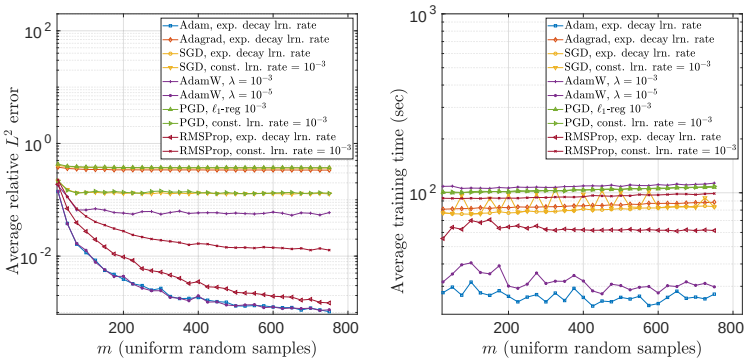


Figure 1. *Comparison of (left) average relative L^2 error and (right) average time in training a set of* 20 *ReLU* 10×100 *DNNs to approximate* $f(x) = \log(\sin(10x) + 2) + \sin(x)$.

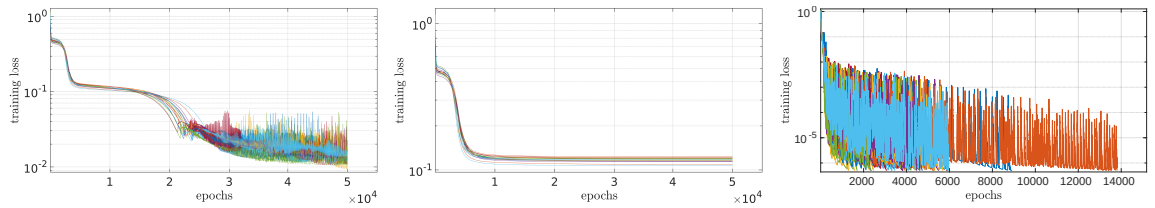


Figure 2. *Examples of typical convergence in training a set of* 20 *ReLU* 10×100 *DNNs on* $m = 750$ *data points of* $f(x) = \log(\sin(10x) + 2) + \sin(x)$ *with* `SGD` *with a constant learning rate* $\tau = 10^{-3}$ *(left),* `SGD` *with an exponentially decaying learning rate (middle), and* `Adam` *with an exponentially decaying learning rate (right).*

plot shows that, on the same problem, all 20 trials trained with the `Adam` optimizer with the exponentially decaying learning rate were able to converge to the 5×10^{-7} loss tolerance in under 14,000 epochs of training. We also compared learning rate schedules for the `Adam` optimizer but found the exponentially decaying learning rate to give consistently good results. See supplementary material section SM1.3.

Previous studies on training DNNs suggest that the batch size can affect the convergence of the algorithms. Due to the small data set sizes considered herein over those in, e.g., computer vision, we have found that setting the batch size too small can result in longer training times (due to the increased transfer time between the CPU and GPU) and only marginal performance benefit. See supplementary material section SM1.4. We leave a more detailed study of the effects of batch size on the performance of trained DNNs in higher-dimensional problems to a future work.

3.3. Initialization and precision. The strategy used to initialize the weights and biases of a DNN can have a large effect on the training process, with some initializations resulting in failure to train at all. To explore the impact of this choice, we tested three different initialization strategies, all of which are based on symmetric uniform or normal distributions with small variance. Empirically we find small variance to be an important factor in the success or failure of training. The strategies tested are (i) normal with mean 0 and variance 0.01, (ii) normal with mean 0 and variance $2/N$, and (iii) uniform on $(-2/N, 2/N)$.

Since we use constant layer widths in this work, i.e., $N_\ell = N$ for all hidden layers

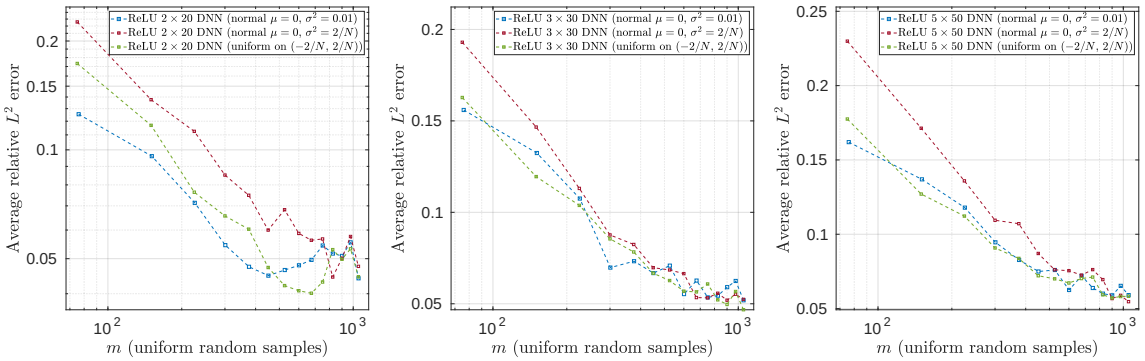


Figure 3. *Comparison of strategies for initializing ReLU DNNs with L hidden layers and N nodes per hidden layer with f (left) $L = 2$ and $N = 20$, (middle) $L = 3$ and $N = 30$, and (right) $L = 5$ and $N = 50$. Results were obtained by training with the* `Adam` *optimizer and an exponentially decaying learning rate schedule on function data generated from $f(x)$ from (4.4) with $d = 2$.*

$\ell = 1, \ldots, L + 1$, we can compare these strategies to the popular Xavier and He initializations (see [45]) as follows. The Xavier initialization sets the weights as mean zero normal random variables with variance $1/N_\ell$, where N_ℓ is the number of nodes on layer ℓ, while the He initialization modifies the variance to $2/N_\ell$. Therefore strategy (ii) is similar to the He initialization, with the exception of the input and output layers for which strategy (ii) prescribes much smaller variance for problems with input and output dimension in the ranges considered in this work. Also, for values of N_ℓ less than 100, the Xavier initialization results in variances larger than the variance of 0.01 used in strategy (i), and this holds similarly for values of N_ℓ less than 200 for the He initialization.

Figure 3 presents the results of these different initialization strategies for training three different architectures with `Adam` on the piecewise continuous function approximation problem (4.4). There we see that strategy (i) has close to the best performance for the smaller ReLU 2×20 and 3×30 architectures, and it performs about as well as strategy (ii) for the 5×50 architecture.

For the majority of our results, we focused on architectures with a small ratio $\beta = L/N$ (with L the number of layers and N the number of nodes per layer). However, we also note that setting β too small can result in DNNs which exhibit numerical instabilities after training (see Figure 5) despite achieving the training loss tolerance. We also remark that this can occur for networks with relatively small weights; see the right plot of Figure 14, which shows that the weights and biases for the 20×800 network remain, on average, bounded by 2 as we increase the samples.

We also observed that for some problems, initializing and training our DNN models in double precision can lead to improved accuracy in testing the generalization performance and that the improvement is more pronounced for deeper networks; see Figure 4. Due to the massive increase in computation time associated with training in double precision (the result of needing to train for more epochs to achieve the tolerance 5×10^{-16} and the increased time of double precision arithmetic; see item (ii) on hardware in section 3.1), we limit our comparisons of single vs. double precision results to a handful of problems and architectures.

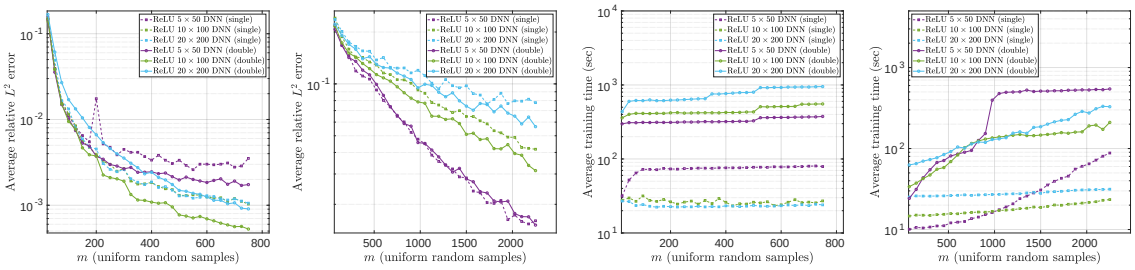


Figure 4. *Comparison of average relative L^2 error* (a) *and* (b) *and average training time* (c) *and* (d) *for DNNs initialized and trained in single vs. double precision on a one-dimensional smooth function $f(x) = \log(\sin(10x) + 2) + \sin(x)$* (a) *and* (c) *and eight-dimensional smooth function $f(x)$ from* (4.3) (b) *and* (d).

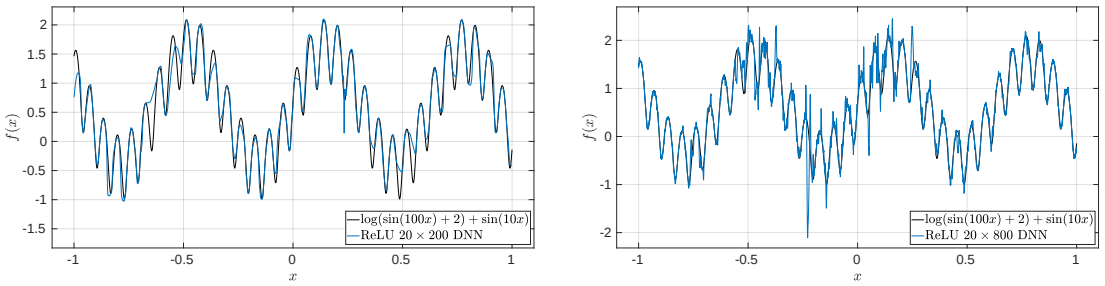


Figure 5. *Unstable ReLU networks with 20 hidden layers, trained in single precision to a loss tolerance of 5×10^{-7} on noiseless data. The left network has 200 nodes per layer and exhibits a numerical instability (a spike) near $x = 0.2334$, while the right network has 800 nodes per layer and exhibits multiple instabilities throughout the domain, providing poor pointwise estimation of the target function.*

3.4. Convergence of `Adam` on function approximation problems.

We now discuss the convergence of the `Adam` optimizer. Our stopping criteria for testing the convergence depend only on the ℓ_2-loss with respect to the training data and the current number of epochs of training. During the course of training, the process outlined above may result in networks which exhibit numerical instabilities or provide poor performance; see Figure 5. In section 3.2, we noted that some trials may not achieve the desired accuracy tolerance within our budget of 50,000 epochs in single precision and 200,000 epochs in double precision. This can also occur for some of the trials with the `Adam` optimizer; see Figure 6. Despite these issues, when viewing the average performance of the `Adam` optimizer on such problems we often find that the majority of networks are numerically stable and approximate the function well; see Figure 6, which displays boxplots of the average relative L^2 error of trained ReLU DNNs with 10 hidden layers and 100 nodes per layer on a discontinuous two-dimensional function. Note that the spread shown in the boxplots is somewhat large due to the discontinuity in the underlying function. On smoother functions (not shown), this spread is smaller. We also remark that the observed performance improves with depth; see Figure 7, which displays the outputs of all 20 trials of a variety of ReLU DNN architectures trained on a smooth, one-dimensional function.

4. Numerical experiments.

We now present our numerical experiments. We test on smooth functions of one or more variables as well as on piecewise continuous functions. The

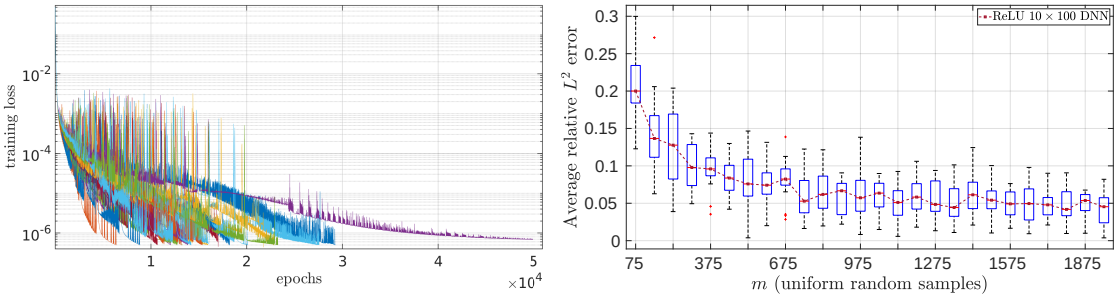


Figure 6. *Training a* 10 *layer DNN with* 100 *nodes per layer with* `Adam` *on the function* (4.4) *with* $d = 2$. *Left: An example where one out of* 20 *trials (each plotted with a different color) is unable to complete within* 50,000 *epochs. Right: Boxplot of the average relative* L^2 *errors of* 20 *trials. For the boxplot, the tops and bottoms of the boxes represent the* 25*th and* 75*th percentiles, with the whiskers covering the most extreme datapoints and outliers (red plusses) plotted individually; see* `boxplot` *from MATLAB for more details.*

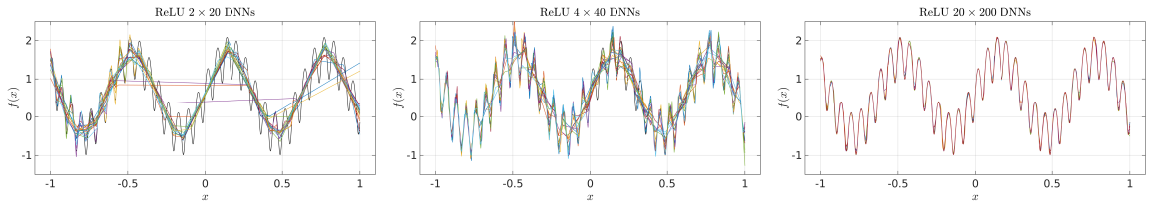


Figure 7. *Comparison of outputs of a variety of ReLU DNN architectures trained with* `Adam` *on* 750 *samples of the function* $f(x) = \log(\sin(100x) + 2) + \sin(10x)$ *(plotted in black).*

results presented in this section elaborate on the main conclusions in section 1.3.

4.1. Smooth one-dimensional functions. We first consider problems where the target function has analytic dependence on only one variable, with regularity controllable by a single parameter. Specifically, we consider

$$f(x) = \log(\sin(10Kx) + 2) + \sin(Kx) \tag{4.1}$$

for values of $K = 1$ and $K = 10$. When $K = 1$, f is smooth and has rapidly decaying Legendre coefficients, making it ideal for reconstruction with CS techniques. When $K = 10$, f oscillates rapidly and has slowly decaying coefficients, leading to less efficient approximation by polynomial-based methods. Figure 8 displays results in the case of $K = 1$, and the right plot compares the average relative L^2 errors of the CS and DL approaches with respect to the number of samples m used in training. We observe that shallower networks fail to achieve a good approximation, while increasing depth and width leads to networks which are competitive with the unweighted and weighted CS approaches. However, comparing the results obtained with the 10 and 20 hidden layer deep networks, we note diminishing returns in performance with increasing size.

Any number of factors can contribute to this phenomenon, including overfitting of the data, increased difficulty of training larger networks from our choice of random initialization, or development of numerical instabilities due to the accumulation of errors from standard sources, e.g., roundoff, overflow, and underflow. We remark that improved results have been

obtained on this function by training some of our networks in double precision; see Figure 4. However, we also observe that training the 20 hidden layer DNN in double precision did not improve, but actually decreased, its accuracy, suggesting room for improvement in training and initializing larger DNNs in double precision.

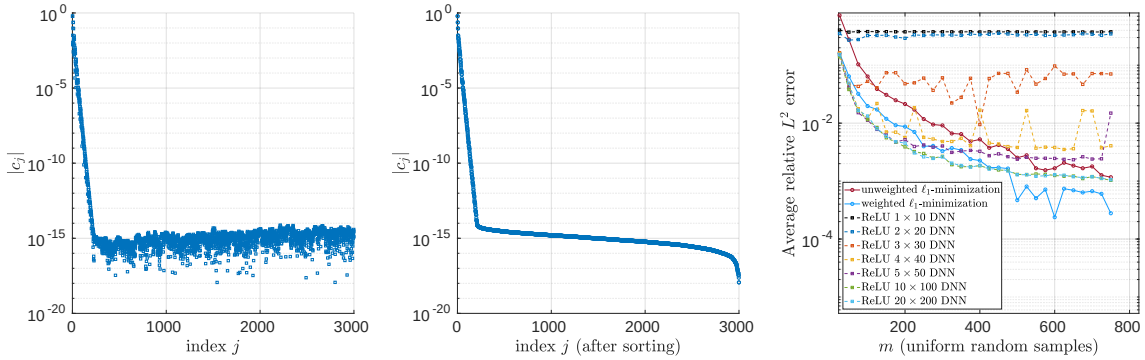


Figure 8. *Legendre coefficients of* $f(x) = \log(\sin(10x) + 2) + \sin(x)$ *sorted (left) lexicographically and (center) by decreasing magnitude. Right: Average relative* L^2 *error vs. number of samples used in training. CS approximations were computed with the Legendre basis of cardinality* $n = 3{,}000$.

Figure 9 displays the results of approximating f when $K = 10$, where the more rapid oscillation now leads to a degradation in performance for both methods. We again observe improved accuracy by increasing depth and width as in the previous example. We also again observe the diminishing returns increasing the size of the network architectures from 10×100 to 20×200, although the 20 hidden layer network provides the best accuracy of all tested DNNs after approximately 600 samples and achieves the same accuracy as the weighted CS approximations after 700 samples.

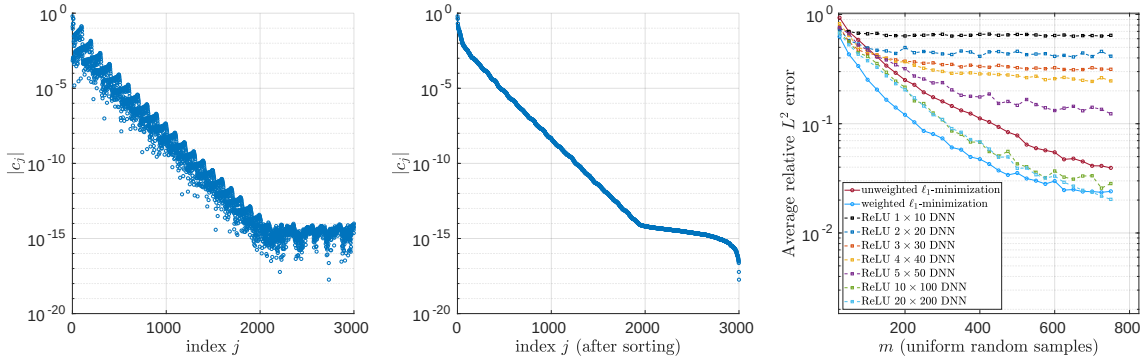


Figure 9. *Legendre coefficients of* $f(x) = \log(\sin(100x) + 2) + \sin(10x)$ *sorted (left) lexicographically and (center) by decreasing magnitude. Right: Average relative* L^2 *error vs. number of samples used in training. CS approximations were computed with the Legendre basis of cardinality* $n = 3{,}000$.

Figure 10 displays the effect of choosing wider networks in approximating f with $K = 10$. There we see that wider counterparts of the network architectures generally outperform narrower architectures with the same number of hidden layers. However, we also observe

diminishing returns going from 10 to 20 hidden layers, and in the right plot we see that the ReLU 20×800 DNNs diverge, with the resulting trained networks exhibiting numerical artifacts as in the right plot of Figure 5.

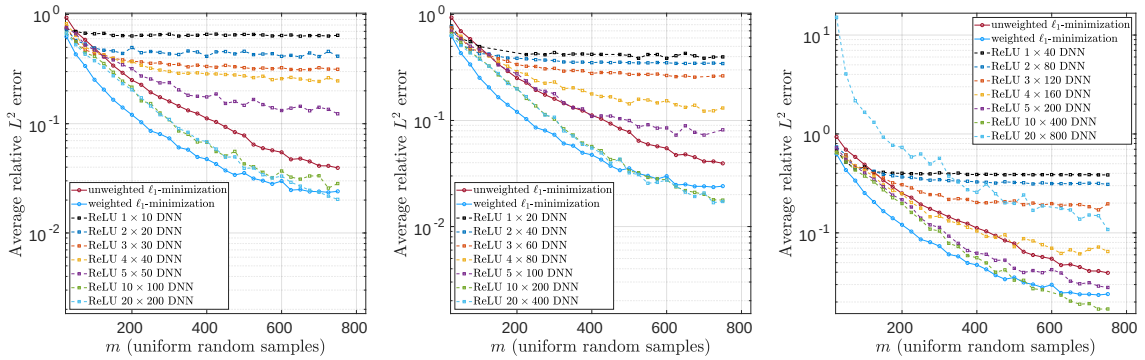


Figure 10. *Average relative L^2 error vs. number of samples of $f(x) = \log(\sin(100x) + 2) + \sin(10x)$ used in training DNNs with L hidden layers and N nodes per layer when $\beta = L/N$ is (left)* 0.1*, (middle)* 0.05*, and (right)* 0.025*. CS approximations were computed with the Legendre basis of cardinality $n = 3{,}000$.*

Figure 11 compares the average absolute maximum of the weights and biases for ReLU DNNs trained on (4.1). We observe in the case $K = 10$, corresponding to the more oscillatory version of this function, that on average the trained DNN architectures have larger weights and biases in magnitude when compared to those trained on the same function with $K = 1$. Also, when comparing the trained weights of the architectures when the ratios are $\beta = 0.1$ and $\beta = 0.025$ on the function with $K = 10$, we note the similarity in the magnitudes of the weights despite the presence of severe artifacts for the wider networks that are not present for their narrower counterparts; see Figure 5.

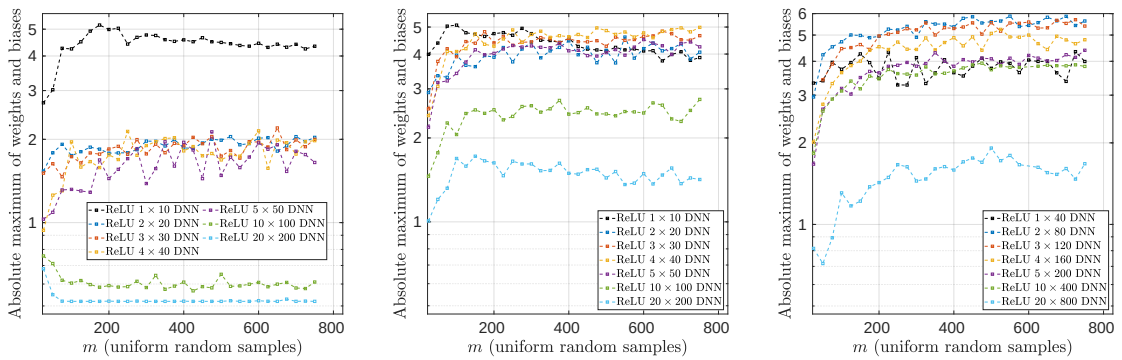


Figure 11. *Average absolute maximum of weights and biases vs. number of samples of $f(x) = \log(\sin(10Kx) + 2) + \sin(Kx)$ for (left) $K = 1$ and $\beta = 0.1$, (middle) $K = 10$ and $\beta = 0.1$, and (right) $K = 10$ and $\beta = 0.025$.*

4.2. Smooth higher-dimensional functions. In Figure 12, we present results for the function

$$f(x_1, \ldots, x_d) = \exp\left(-(\cos(x_1) + \cdots + \cos(x_d))/(8d)\right) \tag{4.2}$$

studied in [13]. This function has smooth, isotropic dependence on its parameters and rapidly decaying Legendre coefficients with increasing polynomial order, making it ideal for approximation with CS techniques. Despite the analyticity of f, the DNNs are unable to obtain an approximation accurate beyond three digits, while both CS approaches achieve nearly eight digits of accuracy. This result highlights the perceived gap between theory and practice in this work: theoretical results suggest DNNs are capable of achieving the same performance as CS on this problem, but in practice we never achieve these accuracies.

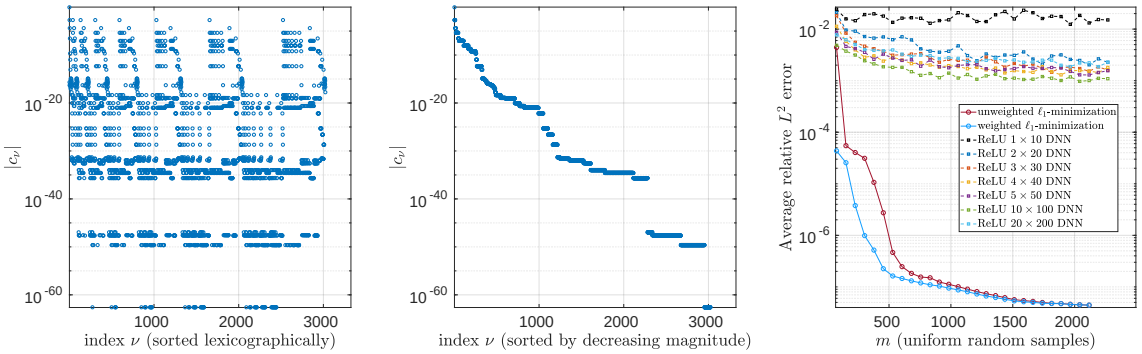
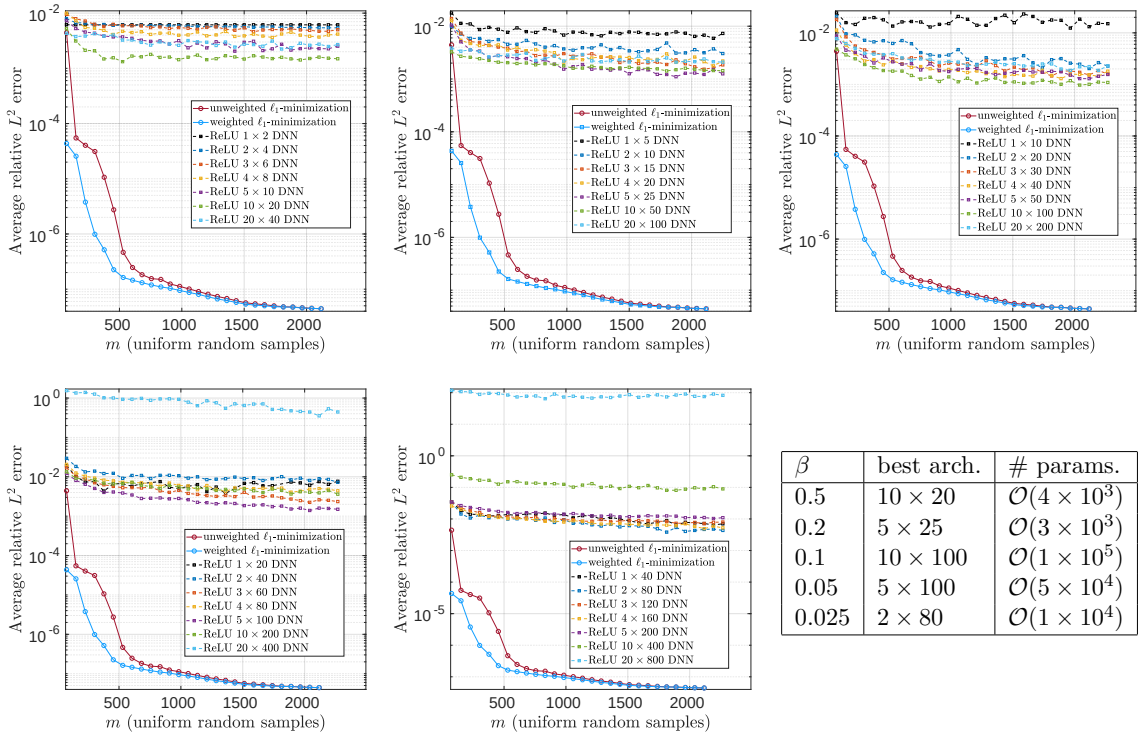


Figure 12. *Legendre coefficients of f from (4.2) with $d = 8$ sorted (left) lexicographically and (center) by decreasing magnitude. Right: Average relative L^2 error vs. number of samples used in training. CS approximations were computed with the Legendre basis of cardinality $n = 3{,}023$.*

Figure 13 displays the results of approximating f from (4.2) with both narrower and wider networks. For narrower DNN architectures, corresponding to $\beta \in \{0.1, 0.2, 0.5\}$, we observe that deeper networks perform better. For wider architectures corresponding to $\beta \in \{0.025, 0.05\}$, we observe that shallower networks perform better. In all cases, the best performing networks had between 10^3 and 10^5 total trainable parameters. Also, in contrast to the diminishing returns with increasing depth observed in the one-dimensional smooth function examples, on this smooth higher-dimensional problem we now observe divergence of wider and deeper networks. These observations suggest the existence of fundamental stability barriers in current methods of training DNNs.

Figure 14 displays the average absolute maximum of the weights and biases for each architecture. There we observe that the weights and biases of narrower network architectures, e.g., $\beta \in \{0.1, 0.2, 0.5\}$, remain relatively small as we increase the number of samples. However, as we increase the width of the networks to values corresponding to $\beta = 0.05$ and $\beta = 0.025$, we begin to see the weights growing with depth as we train on larger sample sets.

Figure SM4 in the supplementary material displays the average run time of the `Adam` optimizer in training our DNNs as we increase the number of training samples. For the narrower and shallower DNN architectures, we see that the transfer time bottleneck between



β	best arch.	# params.
0.5	10×20	$\mathcal{O}(4 \times 10^3)$
0.2	5×25	$\mathcal{O}(3 \times 10^3)$
0.1	10×100	$\mathcal{O}(1 \times 10^5)$
0.05	5×100	$\mathcal{O}(5 \times 10^4)$
0.025	2×80	$\mathcal{O}(1 \times 10^4)$

Figure 13. *Comparison of average relative L^2 errors with respect to number of samples of f from (4.2) with $d = 8$ used in training ReLU architectures parameterized with $\beta = L/N$ (hidden layers/nodes per hidden layer) for values (top left) $\beta = 0.5$, (top middle) $\beta = 0.2$, (top right) $\beta = 0.1$, (bottom left) $\beta = 0.05$, and (bottom middle) $\beta = 0.025$. Bottom right: Table of best-performing architectures for each choice of β and number of parameters. CS approximations were computed with the Legendre basis of cardinality $n = 3{,}023$.*

the CPU and GPU yields an effective linear scaling with respect to samples in the average training time. This effect is also present in the deeper and wider networks, though at a diminished amount due to the fast convergence of the `Adam` optimizer on larger networks.

Next we present results on the function

$$f(x) = \left(\frac{\prod_{k=1}^{\lceil d/2 \rceil} (1 + 4^k x_k^2)}{\prod_{k=\lceil d/2 \rceil + 1}^{d} (100 + 5x_k)} \right)^{1/d}, \tag{4.3}$$

which is also from [13]. This function has anisotropic dependence on its parameters and is less smooth than the previous example. Consequently its Legendre coefficients decay more slowly, as seen in the left and middle plots of Figure 15, and the right plot shows that the CS approximations only achieve an error of roughly 0.02. The best-performing ReLU DNNs achieve an error that is approximately five times smaller.

Figure 16 displays the effect of increasing the width of the DNNs as before. There, as in the previous example, we observe that best performance is achieved by networks that are both wide and deep, e.g., the ReLU 5×25 and 10×20 networks. We also again observe that for wider networks, architectures with fewer hidden layers perform the best. Nonetheless,

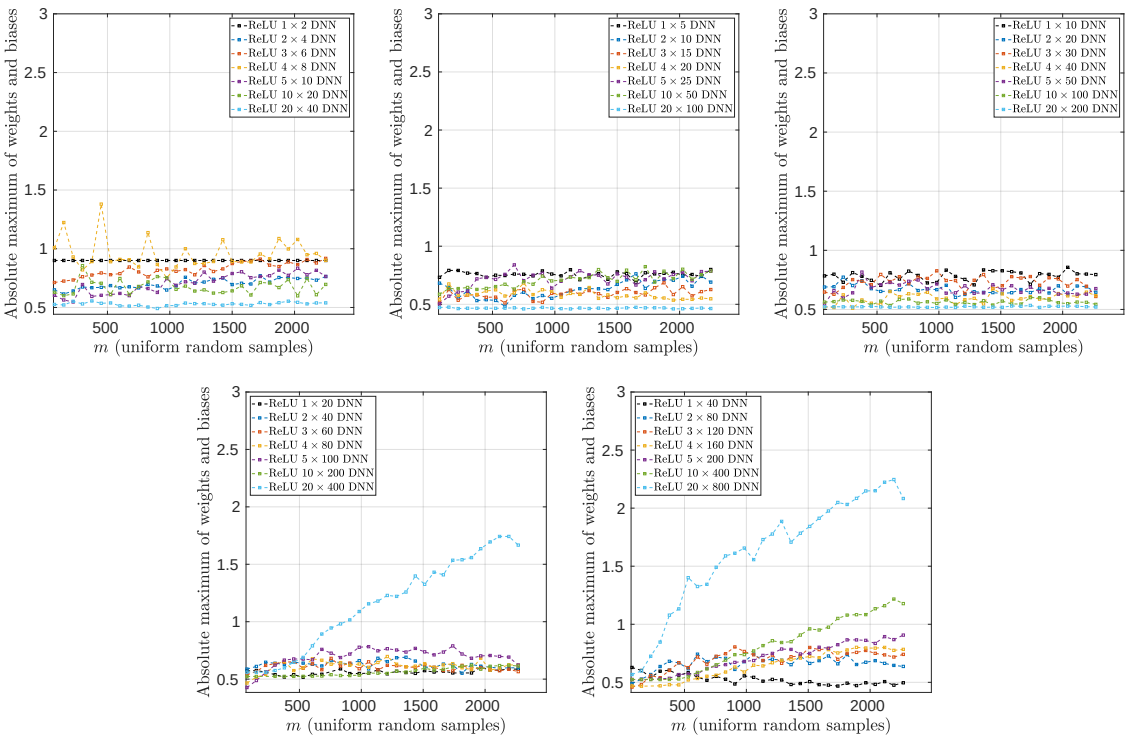


Figure 14. *Comparison of average absolute maximum of weights and biases with respect to number of samples of f from (4.2) with $d = 8$ used in training ReLU architectures parameterized with $\beta = L/N$ (hidden layers/nodes per hidden layer) for values (top left) $\beta = 0.5$, (top middle) $\beta = 0.2$, (top right) $\beta = 0.1$, (bottom left) $\beta = 0.05$, and (bottom right) $\beta = 0.025$.*

comparing the results between the ReLU 1×20 and 1×40 DNNs and those achieved by the 10×20, 5×25, and 2×20 DNNs, we see that far better performance is achieved with fewer samples by the deeper and narrower DNN architectures.

Figure 17 compares the average absolute maximum of the weights and biases of the trained DNNs. There, as in the one-dimensional examples, we observe that the weights and biases of DNNs trained on function data from the less smooth function (4.3) are on average larger than those obtained after training on the smoother function (4.2).

Figure SM5 in the supplementary material again displays the average run time of the `Adam` optimizer in training the DNNs as we increase the number of samples. There we observe patterns similar to the timing results obtained on function (4.2), e.g., linear scaling in the run times for narrower architectures due to the CPU-GPU transfer bottleneck. However, we also observe a longer training time in general in approximating the function (4.3) compared to Figure SM4 of the supplementary material, suggesting that the difficulty of approximating less smooth functions can impact training time.

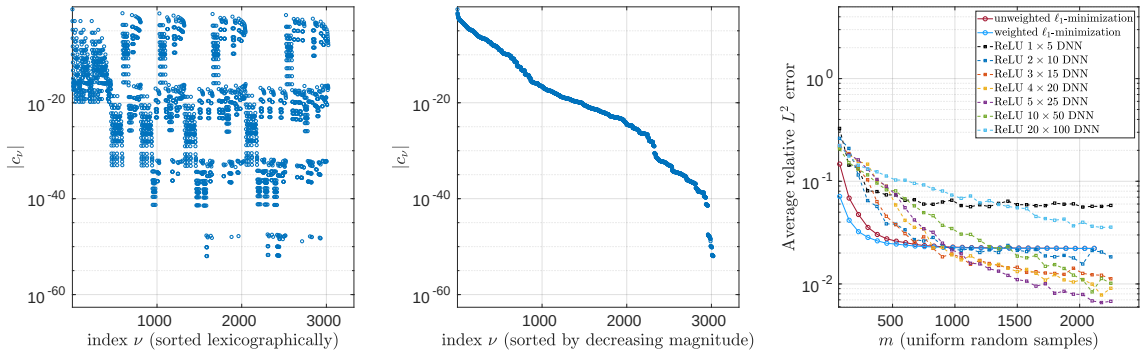


Figure 15. *Legendre coefficients of f from (4.3) with $d = 8$, sorted (left) lexicographically and (center) by decreasing magnitude. Right: Average relative L^2 error with respect to number of samples used in training. CS approximations were computed with the Legendre basis of cardinality $n = 3{,}023$.*

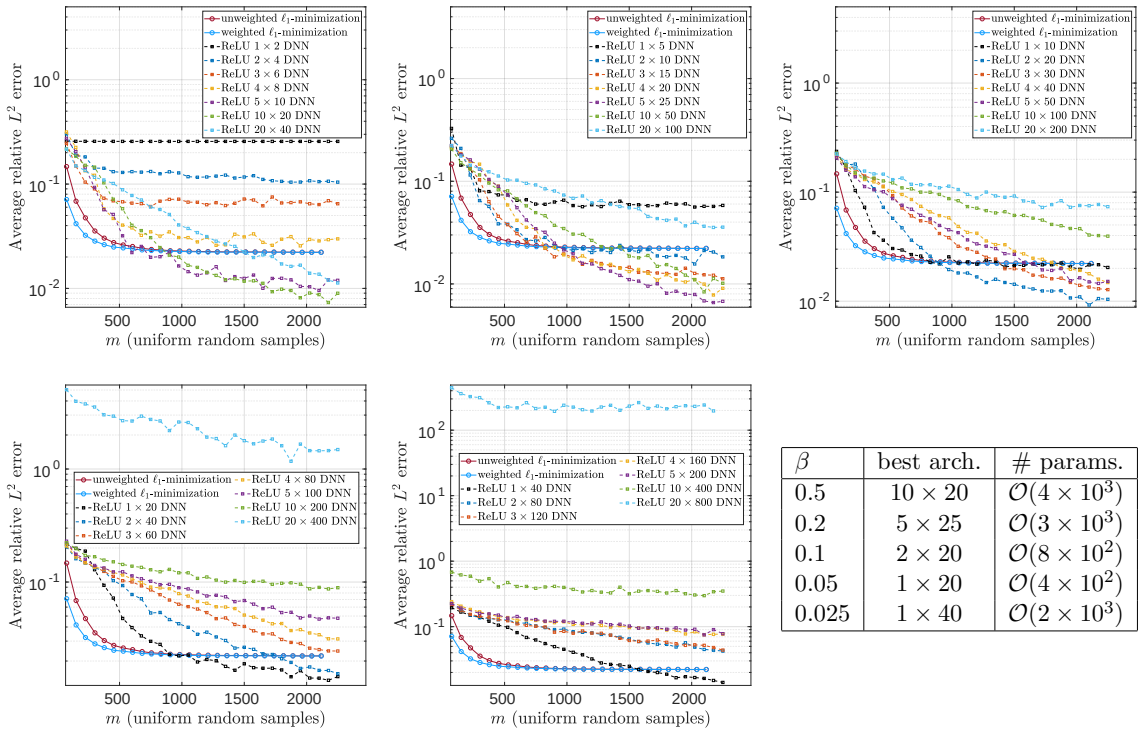
4.3. Piecewise continuous functions. In this section, we present results on approximating piecewise continuous functions. We consider the function

$$f(x_1, \dots, x_d) = \mathbb{1}_K(x_1, \dots, x_d) \quad \text{with } K = \{\boldsymbol{z} \in \mathbb{R}^d : z_1 + \cdots + z_d \geq 0\}, \tag{4.4}$$

where $\mathbb{1}_K(x_1, \dots, x_d) = 1$ if $(x_1, \dots, x_d) \in K$ and 0 otherwise. In one dimension, $K = \{x \in \mathbb{R} : x \geq 0\}$, so that $f(x) = \mathbb{1}_{\{x \geq 0\}}(x)$. We note that this simple discontinuous function equally splits the data between values in $\{0, 1\}$ in arbitrary dimension d, and that the separating hyperplane between the sets K and K^c is not aligned to any particular axis. In $d > 1$ dimensions, the CS approximation fails to converge since the coefficients are not sufficiently summable. On the other hand, the work [65] shows that the Heaviside function in d dimensions given by $H(x) = \mathbb{1}_{[0,\infty)\times\mathbb{R}^{d-1}}(x)$ can be approximated to L^2 accuracy $\varepsilon^{1/2}$ by a two-layer ReLU network with five nonzero weights taking value in $\{\varepsilon^{-1}, 1, -1\}$. As the function (4.4) is a rotation of the d-dimensional Heaviside function, the result holds in this case as well.

Figure 18 displays the results of approximating this function in 1, 2, and 4 dimensions. There we observe the lack of convergence of the CS approximations when $d > 1$. While the DNNs perform better than the CS approximations for this problem, none of the achieved results obtain more than 2 digits of accuracy. In the right plot of Figure 18, we see the *double descent* behavior that has been observed in other works; see, e.g., [61, section 7].

Figure 19 displays the average absolute maximum of the weights and biases in training the DNNs. There we observe that while on average the maximum weights are larger than those found for the smooth functions of the previous section, they remain bounded for the trained DNNs. Comparing to the aforementioned existence results, we note the weights never grow large enough in our experiments to obtain such high accuracy approximations in practice. Due to the initialization strategy chosen in this work to prevent failure during training, all of our networks start from an initial point corresponding to weights and biases close to 0. Combined with the training process, which uses an initial learning rate of 10^{-3} which decays exponentially with the number of epochs, the algorithms may never reach a minimizer corresponding to higher-accuracy approximations of the halfspace function with larger weights.



β	best arch.	# params.
0.5	10×20	$\mathcal{O}(4 \times 10^3)$
0.2	5×25	$\mathcal{O}(3 \times 10^3)$
0.1	2×20	$\mathcal{O}(8 \times 10^2)$
0.05	1×20	$\mathcal{O}(4 \times 10^2)$
0.025	1×40	$\mathcal{O}(2 \times 10^3)$

Figure 16. *Comparison of average relative L^2 errors with respect to number of samples of f from (4.3) with $d = 8$ used in training ReLU architectures parameterized with $\beta = L/N$ (hidden layers/nodes per hidden layer) for values (top left) $\beta = 0.5$, (top middle) $\beta = 0.2$, (top right) $\beta = 0.1$, (bottom left) $\beta = 0.05$, and (bottom middle) $\beta = 0.025$. Bottom right: Table of best-performing architectures for each choice of β and number of parameters. CS approximations were computed with the Legendre basis of cardinality $n = 3,023$.*

5. Theoretical insights.

We conclude this paper with some theoretical insights. Specifically, we first establish convergence rates for polynomial approximation via CS (Theorem 5.4) and then show a practical existence theorem for DL which demonstrates the existence of an architecture and training procedure that achieves the same rates (Theorem 5.5). Proofs for this section can be found in the supplementary material.

5.1. Exponential convergence of polynomial approximations.

In section 2.3 we introduced the best s-term Legendre polynomial approximation. We first establish exponential rates of convergence of such approximations. To this end, let $\mathcal{E}_\rho = \{(z+z^{-1})/2 : z \in \mathbb{C}, 1 \leq |z| \leq \rho\}$ be the Bernstein ellipse with parameter $\rho > 1$, and define the Bernstein polyellipse $\mathcal{B}_{\boldsymbol{\rho}} = \bigotimes_{j=1}^d \mathcal{E}_{\rho_j}$, where $\boldsymbol{\rho} = (\rho_1, \ldots, \rho_d) \geq \mathbf{1}$. This inequality is understood componentwise, i.e., $\boldsymbol{\rho} \geq \mathbf{1}$ if and only if $\rho_j \geq 1$ for all j. We now make the following assumption.

Assumption 5.1. *For some $\boldsymbol{\rho} > \mathbf{1}$, the function $f : \mathcal{U} \to \mathbb{R}$ admits a holomorphic extension to an open set $\mathcal{O}$ containing $\mathcal{E}_{\boldsymbol{\rho}}$.*

Note that this is a reasonable assumption in practice: it is known that functions that arise as quantities of interest for a range of parametric PDEs admit holomorphic extensions

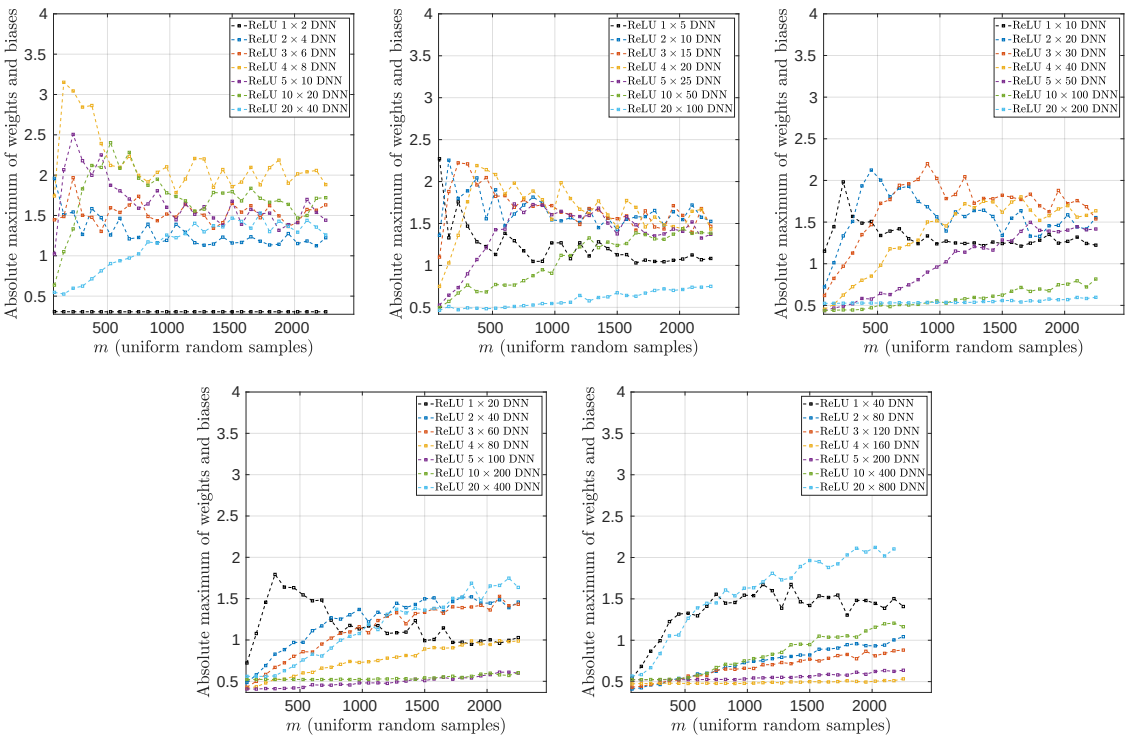


Figure 17. *Comparison of average absolute maximum of weights and biases with respect to number of samples of f from (4.3) with $d = 8$ used in training ReLU architectures parameterized with $\beta = L/N$ (hidden layers/nodes per hidden layer) for values (top left) $\beta = 0.5$, (top middle) $\beta = 0.2$, (top right) $\beta = 0.1$, (bottom left) $\beta = 0.05$, and (bottom right) $\beta = 0.025$.*

[15, 16]. Under Assumption 5.1, it is known that the Legendre polynomial coefficients satisfy

$$|c_{\boldsymbol{\nu}}| \leq \|f\|_{L^\infty(\mathcal{E}_{\boldsymbol{\rho}})} \boldsymbol{\rho}^{-\boldsymbol{\nu}} \prod_{j=1}^{d} (1 + 2\nu_j)^{1/2}, \tag{5.1}$$

where $\|u\|_{L^\infty(\mathcal{E}_{\boldsymbol{\rho}})} = \sup_{z \in \mathcal{E}_{\boldsymbol{\rho}}} |f(z)|$ (see, for instance, the proof of Theorem 3.5 in [64]). When $d = 1$, this is a classical result in polynomial approximation and guarantees convergence of the truncated expansion at an exponential rate ρ^{-s}. When $d > 1$, it also clarifies why best s-term approximation is a suitable strategy; namely, unless the parameter $\boldsymbol{\rho}$ is known, it is difficult to make an a priori choice of coefficients which lead to a fast rate of exponential convergence.

On the other hand, the following theorem shows favorable exponential rates of convergence for the best s-term approximation. It also reveals another important property, namely, that the prescribed rate can be achieved using an index set that is *lower*. We recall that a set Λ of multi-indices is lower if, for any $\boldsymbol{\nu} \in \Lambda$, one has $\boldsymbol{\mu} \in \Lambda$ whenever $\boldsymbol{\mu} \leq \boldsymbol{\nu}$ (this inequality is understood componentwise, i.e., $\mu_j \leq \nu_j$ for all j). As discussed later, the lower set property is crucial for obtaining approximations using CS that attain the same rates.

For convenience, from now on we consider the s-term approximation error in the L^∞-norm on $\mathcal{U}$. This is slightly more convenient for the subsequent analysis, although L^2-norm bounds

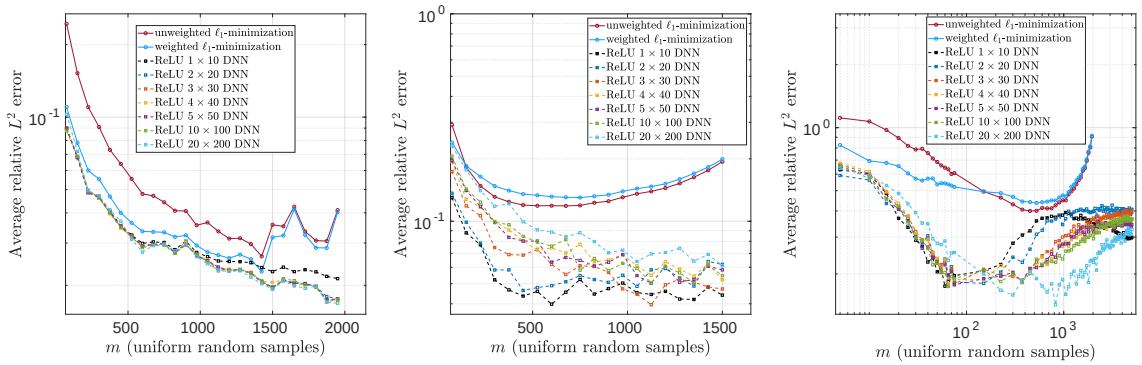


Figure 18. *Average relative L^2 error with respect to number of samples used in training ReLU DNNs and solving the CS problem with Legendre basis of cardinality n in d dimensions with (left) $d = 1$ and $n = 3{,}000$, (center) $d = 2$ and $n = 3{,}001$, and (right) $d = 4$ and $n = 3{,}079$.*

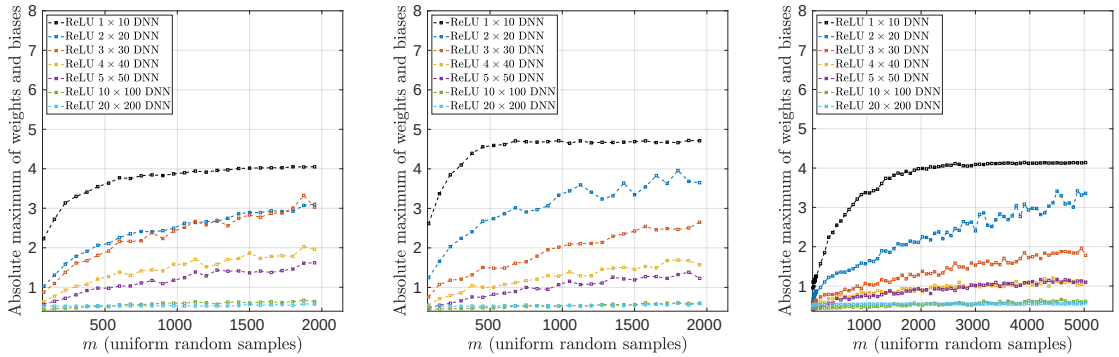


Figure 19. *Comparison of average absolute maximum of weights and biases vs. number of samples of the halfspace function from (4.4) used in training in (left) $d = 1$, (center) $d = 2$, and (right) $d = 4$ dimensions.*

could also be proved with some additional technicalities.

Theorem 5.2. *Let $s \geq 1$ and f satisfy Assumption 5.1 for some $\boldsymbol{\rho} > \mathbf{1}$. Then there exists a lower set $\Lambda \subset \mathbb{N}_0^d$ of size at most s for which*

$$\|f - p\|_{L^\infty(\mathcal{U})} \leq C \exp\left(-\gamma s^{1/d}\right),$$

where $p = \sum_{\boldsymbol{\nu} \in \Lambda} c_{\boldsymbol{\nu}} \Psi_{\boldsymbol{\nu}}$, and $C > 0$ depends on d, $\boldsymbol{\rho}$, γ, and f only, for any γ satisfying

$$0 < \gamma < (d+1)^{-1} \left(d! \prod_{j=1}^{d} \log(\rho_j) \right)^{1/d}. \tag{5.2}$$

This result asserts exponential convergence of the best s-term approximation at a rate depending on the product of $\log(\rho_j)$, as opposed to $\log(\rho_{\min})$, $\rho_{\min} = \min_j\{\rho_j\}$, which would arise if some fixed isotropic index set were used. Importantly, it also does not require any a priori knowledge of the parameters $(\rho_j)_{j=1}^d$, and it adapts to the underlying anisotropy of the

function. In the next subsection we show this rate can be achieved using CS with a sampling complexity m that scales favorably with s. We refer to CS as a best-in-class scheme since there are, currently, no other methods which provably achieve this property.

Remark 5.3. There are numerous results on the convergence rate of the best s-term polynomial approximation of holomorphic functions. Algebraic rates of convergence can be found in, for instance, [14, 15], for not only Legendre but also Chebyshev and Taylor expansions. Notably, these results also extend to the case $d = \infty$, which theoretically permits the approximation of functions of infinitely many variables. However, the constants in the error bounds may be large in practice [80]. The rates shown above, which are based on [16, sect. 3.9] and [64], possess the advantage of always being attained in lower sets. However, they may also exhibit a poor scaling with d in the constant C. For *quasi-optimal* error bounds and rates, see [9, 10, 80]. The results shown in [80] are asymptotically sharp as $s \to \infty$, and hence they provide the optimal rate in the finite-dimensional setting.

5.2. Exponential convergence via compressed sensing. Theorem 5.2 prescribes exponential rates of convergence for the best s-term approximation. We now show that these rates can be achieved via CS. To do so, following [2], we now suppose that the CS approximation $\hat{f} = \sum_{\boldsymbol{\nu}\in\Lambda} \hat{c}_{\boldsymbol{\nu}} \Psi_{\boldsymbol{\nu}}$ is computed by solving the so-called *weighted square-root LASSO* problem

$$\text{minimize}_{\boldsymbol{z}\in\mathbb{R}^n} \|\boldsymbol{z}\|_{1,\boldsymbol{u}} + \mu \|\boldsymbol{A}\boldsymbol{z} - \boldsymbol{f}\|_2, \tag{5.3}$$

rather than (2.8). The reasons for doing this are explained in supplementary material section SM2.1.

Theorem 5.4. *There exists a universal constant $c > 0$ such that the following holds. Suppose that $0 < \varepsilon < 1$, $m \geq C_1 L_{m,d,\varepsilon}$, where*

$$L_{m,d,\varepsilon} = \log(2m)\left(\log^2(2m)\log(2d) + \log(2\varepsilon^{-1}\log(2m))\right),$$

$\boldsymbol{x}_1, \dots, \boldsymbol{x}_m$ are drawn independently from the uniformly measure on $\mathcal{U}$,

$$1 \leq s \leq \sqrt{\frac{m}{cL_{m,d,\varepsilon}}}, \tag{5.4}$$

and $\Lambda = \Lambda_s^{\mathrm{HC}}$ is as in (2.4). Then the following holds with probability at least $1 - \varepsilon$. Let $f : \mathcal{U} \to \mathbb{R}$ satisfy Assumption 5.1 for some $\boldsymbol{\rho} > \mathbf{1}$, and let $\hat{f} = \sum_{\boldsymbol{\nu}\in\Lambda} \hat{c}_{\boldsymbol{\nu}} \Psi_{\boldsymbol{\nu}}$, where $\hat{\boldsymbol{c}} = (\hat{c}_{\boldsymbol{\nu}})_{\boldsymbol{\nu}\in\Lambda}$ is any minimizer of (5.3) with $\mu = \frac{12\sqrt{42}}{35} s$ and weights $\boldsymbol{u}$ given by (2.9). Then

$$\|f - \hat{f}\|_{L^\infty(\mathcal{U})} \leq C \cdot \exp(-\gamma s^{1/d})$$

for all γ satisfying (5.2), where $C > 0$ depends on d, $\boldsymbol{\rho}$, γ, and f only.

This theorem asserts that CS achieves the same rates as those guaranteed in Theorem 5.2, with a sample complexity that is (up to the logarithmic factor) quadratic in s. In particular, scaling implied by (5.4) depends only logarithmically on the dimension d. Hence, this estimate scales particularly well in higher dimensions.

It is important to note the key role the lower set property plays in this result. First, the union of all lower sets S of cardinality at most s is $\cup\{S : S \text{ lower}, |S| \leq s\} = \Lambda_s^{\mathrm{HC}}$, i.e., the hyperbolic cross set of degree s. This is the rationale for choosing Λ in this way. Second, much like how sparse sets are promoted by the ℓ^1-norm, sparse and lower sets are promoted by the weighted ℓ^1-norm, with the weights taken to be $\boldsymbol{u}$ (these weights penalize high indices). Had one considered the unweighted ℓ^1-norm in the above result, the sample complexity would have scaled with a higher algebraic power of s [3].

5.3. Existence of good training for DNN approximation. We now give our main result.

Theorem 5.5. *Let $0 < \varepsilon < 1$, $c > 0$, and $L_{m,d,\varepsilon}$ be as in Theorem 5.4, and suppose that $m \geq cL_{m,d,\varepsilon}$ and s satisfies (5.4). Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$ be drawn independently from the uniform measure on $\mathcal{U}$. Then the following holds with probability at least $1 - \varepsilon$. Let $f : \mathcal{U} \to \mathbb{R}$, $\|f\|_{L^\infty(\mathcal{U})} \leq 1$, satisfy Assumption 5.1 for some $\boldsymbol{\rho} > \mathbf{1}$. Then there exists a family of neural networks $\mathcal{N}$ with $n = |\Lambda_s^{\mathrm{HC}}|$ trainable parameters and of size and depth*

$$\begin{aligned} \mathrm{depth}(\Phi_{\Lambda,\delta}) &\leq c'(1 + d\log(d))(1 + \log(s))(s + \log(n) + \gamma s^{1/d}), \\ \mathrm{size}(\Phi_{\Lambda,\delta}) &\leq c'\left(d^2 s^2 + (ds + d^2 n)\left(1 + \log(s) + \log(n) + \gamma s^{1/d}\right)\right), \end{aligned} \qquad \Phi \in \mathcal{N},$$

where $c' > 0$ is a universal constant, and a regularization functional $\mathcal{J} : \mathcal{N} \to \mathbb{R}_+$ such that any minimizer $\hat{\Phi}$ of the regularized loss function $\mathcal{L}(\Phi) := \sqrt{\frac{1}{m}\sum_{i=1}^m |\Phi(\boldsymbol{y}_i) - f(\boldsymbol{y}_i)|^2} + \mathcal{J}(\Phi)$ satisfies

$$\|f - \hat{\Phi}\|_{L^\infty(\mathcal{U})} \leq C \cdot \exp(-\gamma s^{1/d})$$

for all γ satisfying (5.2), where $C > 0$ depends on d, $\boldsymbol{\rho}$, γ, and f only. The functional $\mathcal{J}$ is a weighted ℓ^1-norm penalty of the weights in the output layer.

Note that in this result the *size* of an architecture refers to the number of nonzero weights and biases. The *number of trainable parameters* in an architecture refers to the number of weights and biases that are trainable, as opposed to those that are fixed. The proof of this theorem uses a result of [64] (see Proposition SM3.6 in the supplementary material), which states that a finite set of Legendre polynomials can be uniformly approximated by a neural network of a given depth and size. Theorem 5.5 is obtained by rewriting the CS recovery using the weighted square-root LASSO as an NN training problem. See supplementary material section SM3 for the details.

Theorem 5.5 asserts the existence of a DNN architecture, with relatively few training parameters, and a training procedure for which the resulting DNN approximations are guaranteed to perform as well as CS, up to a constant, in terms of sample complexity and convergence rates. Of course, the specific procedure suggested by the proof would not be expected to lead to any superior performance over CS. We make no claim that this approach is either practical or numerically stable (indeed, its proof relies on monomials). This is analogous to how standard existence theorems in DNN approximation theory (see section 1), while constructive, do not lead to superior approximations over the classical approximations on which they are based. However, it does indicate that with sufficiently careful architecture design and training,

one may achieve superior performance with DNNs over CS. The extent to which this can be done is a largely open problem, requiring further theoretical and empirical investigation.

6. Conclusions. In this work we have presented results highlighting the key issues of accuracy, stability, sample complexity, and computational efficiency of practical function approximation with DNNs. Our theoretical contribution on the existence of a DL procedure which performs as well as CS suggests that DL can, in theory, enjoy the same accuracy and sample complexity properties as CS. However, our numerical results comparing standard DNN architectures and common methods of training with CS techniques suggest that current methods in DL are generally unable to achieve these theoretical convergence rates on smooth function approximation problems. On the other hand, while DNNs perform relatively poorly on highly smooth functions, their performance on more challenging problems, such as less smooth functions or functions with jump discontinuities, is rather more promising. Certainly the fact that the same DNN architectures can be used on quite different problems sets them apart from traditional methods in scientific computing, e.g., polynomial-based methods, which are usually tied to a specific class of function (e.g., smooth functions). Hence there is ample scope and need for future work along the lines of investigations initiated in this paper. This includes empirical investigations into architecture and cost function design, algorithms for training, and further novel theoretical insights into *practical existence theory*, that is, the existence of not only effective DNN architectures but also training procedure and sampling strategies which realize them efficiently. The hope is that, with these further efforts, DNNs may develop into effective tools for scientific computing that can consistently outperform current best-in-class approaches across a range of challenging problems.

Acknowledgments. The authors thank Clayton Webster for important discussions motivating our study of practical DNN approximation. We thank Simone Brugiapaglia for useful discussions related to various aspects of this work, and Sebastian Moraga for his assistance with several technical steps in the proofs. We also acknowledge and thank the anonymous referees for proofreading and providing valuable comments on the manuscript. Finally, we gratefully acknowledge Boris Hanin and Philipp Petersen for their helpful and inspiring comments on the approximation theory of DNNs and practical issues related to initialization and trainability.

REFERENCES

[1] B. Adcock, *Infinite-dimensional compressed sensing and function interpolation*, Found. Comput. Math., 18 (2018), pp. 661–701.

[2] B. Adcock, A. Bao, and S. Brugiapaglia, *Correcting for unknown errors in sparse high-dimensional function approximation*, Numer. Math., 142 (2019), pp. 667–711.

[3] B. Adcock, S. Brugiapaglia, and C. G. Webster, *Compressed sensing approaches for polynomial approximation of high-dimensional functions*, in Compressed Sensing and Its Applications, Birkhäuser, 2017, pp. 93–124.

[4] V. Antun, F. Renna, C. Poon, B. Adcock, and A. C. Hansen, *On Instabilities of Deep Learning in Image Reconstruction – Does AI Come at a Cost?*, preprint, https://arxiv.org/abs/1902.05300, 2019.

[5] S. Arridge, P. Maass, O. Öktem, and C.-B. Schönlieb, *Solving inverse problems using data-driven models*, Acta Numer., 28 (2019), pp. 1–174.

[6] F. Bach, *Breaking the curse of dimensionality with convex neural networks*, J. Mach. Learn. Res., 18 (2017), pp. 1–53.
[7] N. Baker, F. Alexander, T. Bremer, A. Hagberg, Y. Y. Kevrekidis, H. Najm, M. Parashar, A. Patra, J. Sethian, S. Wild, and K. Willcox, *Workshop Report on Basic Research Needs for Scientific Machine Learning: Core Technologies for Artificial Intelligence*, U.S. Department of Energy Advanced Scientific Computing Research, 2019.
[8] C. Beck, A. Jentzen, and B. Kuckuck, *Full Error Analysis for the Training of Deep Neural Networks*, preprint, https://arxiv.org/abs/1910.00121, 2019.
[9] J. Beck, F. Nobile, L. Tamellini, and R. Tempone, *Convergence of quasi-optimal stochastic Galerkin methods for a class of PDEs with random coefficients*, Comput. Math. Appl., 67 (2014), pp. 732–751.
[10] J. Beck, R. Tempone, F. Nobile, and L. Tamellni, *On the optimal polynomial approximation of stochastic PDEs by Galerkin and collocation methods*, Math. Models Methods Appl. Sci., 22 (2012), 1250023.
[11] J. Berner, P. Grohs, and A. Jentzen, *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*, preprint, https://arxiv.org/abs/1809.03062, 2018.
[12] J. Carrasquilla and R. Melko, *Machine learning phases of matter*, Nature Phys., 13 (2017), pp. 431–434.
[13] A. Chkifa, N. Dexter, H. Tran, and C. G. Webster, *Polynomial approximation via compressed sensing of high-dimensional functions on lower sets*, Math. Comp., 87 (2018), pp. 1415–1450.
[14] A. Cohen, R. DeVore, and C. Schwab, *Convergence rates of best N-term Galerkin approximations for a class of elliptic sPDEs*, Found. Comput. Math., 10 (2010), pp. 615–646.
[15] A. Cohen, R. DeVore, and C. Schwab, *Analytic regularity and polynomial approximation of parametric and stochastic elliptic PDEs*, Anal. Appl., 9 (2011), pp. 11–47.
[16] A. Cohen and R. A. DeVore, *Approximation of high-dimensional parametric PDEs*, Acta Numer., 24 (2015), pp. 1–159.
[17] G. Cybenko, *Approximation by superpositions of a sigmoidal function*, Math. Control Signals Systems, 2 (1989), pp. 303–314.
[18] E. C. Cyr, M. A. Gulian, R. G. Patel, M. Perego, and N. A. Trask, *Robust training and initialization of deep neural networks: An adaptive basis viewpoint*, in Proceedings of the First Mathematical and Scientific Machine Learning Conference (Princeton University, Princeton, NJ), PMLR 107, PMLR, 2020, pp. 512–536, http://proceedings.mlr.press/v107/cyr20a.html.
[19] G. E. Dahl, D. Yu, L. Deng, and A. Acero, *Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition*, IEEE Trans. Audio Speech Language Process., 20 (2012), pp. 30–42.
[20] J. Daws and C. Webster, *Analysis of Deep Neural Networks with Quasi-Optimal Polynomial Approximation Rates*, preprint, https://arxiv.org/abs/1912.02302, 2019.
[21] J. Daws and C. G. Webster, *A Polynomial-Based Approach for Architectural Design and Learning with Deep Neural Networks*, preprint, https://arxiv.org/abs/1905.10457, 2019.
[22] J. De Fauw, J. R. Ledsam, B. Romera-Paredes, S. Nikolov, N. Tomasev, S. Blackwell, H. Askham, X. Glorot, B. O'Donoghue, D. Visentin, G. van den Driessche, B. Lakshminarayanan, C. Meyer, F. Mackinder, S. Bouton, K. Ayoub, R. Chopra, D. King, A. Karthikesalingam, C. O. Hughes, R. Raine, J. Hughes, D. A. Sim, C. Egan, A. Tufail, H. Montgomery, D. Hassabis, G. Rees, T. Back, P. T. Khaw, M. Suleyman, J. Cornebise, P. A. Keane, and O. Ronneberger, *Clinically applicable deep learning for diagnosis and referral in retinal disease*, Nature Med., 24 (2018), pp. 1342–1350.
[23] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, *ImageNet: A large-scale hierarchical image database*, in Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 248–255.
[24] A. Dereventsov, A. Petrosyan, and C. Webster, *Greedy Shallow Networks: A New Approach for Constructing and Training Neural Networks*, preprint, https://arxiv.org/abs/1905.10409, 2019.
[25] A. Dereventsov, A. Petrosyan, and C. Webster, *Neural network integral representations with the ReLU activation function*, in Proceedings of the First Mathematical and Scientific Machine Learning

Conference, Proc. Mach. Learn. Res. 107, PMLR, 2020, pp. 128–143.
[26] R. A. DeVore, *Nonlinear approximation*, Acta Numer., 7 (1998), pp. 51–150.
[27] N. Dexter, H. Tran, and C. Webster, *A mixed ℓ_1 regularization approach for sparse simultaneous approximation of parameterized PDEs*, ESAIM Math. Model. Numer. Anal., 53 (2019), pp. 2025–2045.
[28] W. E, J. Han, and A. Jentzen, *Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations*, Commun. Math. Stat., 5 (2017), pp. 349–380.
[29] W. E, C. Ma, and L. Wu, *Barron Spaces and the Compositional Function Spaces for Neural Network Models*, preprint, https://arxiv.org/abs/1906.08039, 2019.
[30] W. E and Q. Wang, *Exponential Convergence of the Deep Neural Network Approximation for Analytic Functions*, preprint, https://arxiv.org/abs/1807.00297, 2018.
[31] W. E and B. Yu, *The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems*, Commun. Math. Stat., 6 (2018), pp. 1–12.
[32] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, *Scene Parsing with Multiscale Feature Learning, Purity Trees, and Optimal Covers*, preprint, https://arxiv.org/abs/1202.2160, 2012.
[33] A. Fawzi, S.-M. Moosavi-Dezfooli, and P. Frossard, *The robustness of deep networks: A geometrical perspective*, IEEE Signal Process. Mag., 34 (2017), pp. 50–62.
[34] D. Fokina and I. Oseledets, *Growing Axons: Greedy Learning of Neural Networks with Application to Function Approximation*, preprint, https://arxiv.org/abs/1910.12686, 2019.
[35] M. Geist, P. Petersen, M. Raslan, R. Schneider, and G. Kutyniok, *Numerical Solution of the Parametric Diffusion Equation by Deep Neural Networks*, preprint, https://arxiv.org/abs/2004.12131, 2020.
[36] T. Gerstner and M. Griebel, *Numerical integration using sparse grids*, Numer. Algorithms, 18 (1998), pp. 209–232.
[37] X. Glorot and Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, J. Mach. Learn. Res., 9 (2010), pp. 249–256.
[38] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.
[39] N. M. Gottschling, V. Antun, B. Adcock, and A. C. Hansen, *The Troublesome Kernel: Why Deep Learning for Inverse Problems Is Typically Unstable*, preprint, https://arxiv.org/abs/2001.01258, 2020.
[40] P. Grohs, F. Hornung, A. Jentzen, and P. Von Wurstemberger, *A Proof That Artificial Neural Networks Overcome the Curse of Dimensionality in the Numerical Approximation of Black-Scholes Partial Differential Equations*, preprint, https://arxiv.org/abs/1809.02362, 2018.
[41] P. Grohs, T. Wiatowski, and H. Bolcskei, *Deep convolutional neural networks on cartoon functions*, in Proceedings of the IEEE International Symposium on Information Theory, ISIT 2016, Barcelona, Spain, 2016, pp. 1163–1167.
[42] I. Gühring, G. Kutyniok, and P. Petersen, *Error Bounds for Approximations with Deep ReLU Neural Networks in $W^{s,p}$ Norms*, preprint, https://arxiv.org/abs/1902.07896, 2019.
[43] M. D. Gunzburger, C. G. Webster, and G. Zhang, *Stochastic finite element methods for partial differential equations with random input data*, Acta Numer., 23 (2014), pp. 521–650.
[44] B. Hanin, *Which neural net architectures give rise to exploding and vanishing gradients?*, in Advances in Neural Information Processing Systems, Curran Associates, Inc., 2018, pp. 582–591.
[45] B. Hanin and D. Rolnick, *How to start training: The effect of initialization and architecture*, in Advances in Neural Information Processing Systems, Curran Associates, Inc., 2018, pp. 571–581.
[46] K. He, X. Zhang, S. Ren, and J. Sun, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, in Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV 2015), 2015, pp. 1026–1034.
[47] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, *Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups*, IEEE Signal Process. Mag., 29 (2012), pp. 82–97.
[48] K. Hornik, M. Stinchcombe, and H. White, *Multilayer feedforward networks are universal approximators*, Neural Networks, 2 (1989), pp. 359–366.
[49] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, preprint, https://arxiv.org/

abs/1412.6980, 2014.
[50] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *ImageNet classification with deep convolutional neural networks*, in Advances in Neural Information Processing Systems, Curran Associates, Inc., 2012, pp. 1097–1105.
[51] G. Kutyniok, P. Petersen, M. Raslan, and R. Schneider, *A Theoretical Analysis of Deep Neural Networks and Parametric PDEs*, preprint, https://arxiv.org/abs/1904.00377, 2019.
[52] B. Li, S. Tang, and H. Yu, *Better Approximations of High Dimensional Smooth Functions by Deep Neural Networks with Rectified Power Units*, preprint, https://arxiv.org/abs/1903.05858, 2019.
[53] S. Liang and R. Srikant, *Why Deep Neural Networks for Function Approximation?*, preprint, https://arxiv.org/abs/1610.04161, 2016.
[54] I. Loshchilov and F. Hutter, *Decoupled weight decay regularization*, in Proceedings of the 7th International Conference on Learning Representations (ICLR), 2019, https://arxiv.org/abs/1711.05101.
[55] J.-L. Loyer, E. Henriques, M. Fontul, and S. Wiseall, *Comparison of machine learning methods applied to the estimation of manufacturing cost of jet engine components*, Int. J. Prod. Econ., 178 (2016), pp. 109–119.
[56] J. Lu, Z. Shen, H. Yang, and S. Zhang, *Deep Network Approximation for Smooth Functions*, preprint, https://arxiv.org/abs/2001.03040, 2021.
[57] Y. Lu, A. Zhong, Q. Li, and B. Dong, *Beyond Finite Layer Neural Networks: Bridging Deep Architectures and Numerical Differential Equations*, preprint, https://arxiv.org/abs/1710.10121, 2017.
[58] H. Montanelli, H. Yang, and Q. Du, *Deep ReLU Networks Overcome the Curse of Dimensionality for Bandlimited Functions*, preprint, https://arxiv.org/abs/1903.00735, 2019.
[59] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, *Universal Adversarial Perturbations*, preprint, https://arxiv.org/abs/1610.08401, 2016.
[60] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, *Deepfool: A Simple and Accurate Method to Fool Deep Neural Networks*, preprint, https://arxiv.org/abs/1511.04599, 2015.
[61] P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak, and I. Sutskever, *Deep Double Descent: Where Bigger Models and More Data Hurt*, preprint, https://arxiv.org/abs/1912.02292, 2019.
[62] F. Nobile, R. Tempone, and C. G. Webster, *A sparse grid stochastic collocation method for elliptic partial differential equations with random input data*, SIAM J. Numer. Anal., 46 (2008), pp. 2309–2345, https://doi.org/10.1137/060663660.
[63] J. A. A. Opschoor, P. C. Petersen, and C. Schwab, *Deep ReLU Networks and High-Order Finite Element Methods*, Tech. report, ETH Zürich, Zürich, 2019.
[64] J. A. A. Opschoor, C. Schwab, and J. Zech, *Exponential ReLU DNN Expression of Holomorphic Maps in High Dimension*, Tech. report, ETH Zürich, Zürich, 2019.
[65] P. Petersen and F. Voigtlaender, *Optimal approximation of piecewise smooth functions using deep ReLU neural networks*, Neural Netw., 108 (2018), pp. 296–330.
[66] H. Rauhut and R. Ward, *Interpolation via weighted ℓ_1-minimization*, Appl. Comput. Harmon. Anal., 40 (2016), pp. 321–351.
[67] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz, *Data-driven discovery of partial differential equations*, Sci. Adv., 3 (2017), e1602614.
[68] C. Schwab and J. Zech, *Deep learning in high dimension: Neural network expression rates for generalized polynomial chaos expansions in UQ*, Anal. Appl., 17 (2019), pp. 19–55.
[69] Z. Shen, H. Yang, and S. Zhang, *Deep network approximation characterized by number of neurons*, Commun. Comput. Phys., 28 (2020), pp. 1768–1811, https://global-sci.org/intro/article_detail/cicp/18396.html.
[70] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Van Den Driessche, T. Graepel, and D. Hassabis, *Mastering the game of Go without human knowledge*, Nature, 550 (2017), pp. 354–359.
[71] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, in Proceedings of the 3rd International Conference on Learning Representations (ICLR), San Diego, CA, 2015, https://arxiv.org/abs/1409.1556.
[72] C. Sommer and D. W. Gerlich, *Machine learning in cell biology - teaching computers to recognize phenotypes*, J. Cell Sci., 126 (2013), pp. 5529–5539.

[73] M. Stoyanov, *User Manual: Tasmanian Sparse Grids*, Tech. report ORNL/TM-2015/596, Oak Ridge National Laboratory, Oak Ridge, TN, 2015.
[74] M. Stoyanov, *Adaptive sparse grid construction in a context of local anisotropy and multiple hierarchical parents*, in Sparse Grids and Applications (Miami, 2016), Springer, 2018, pp. 175–199.
[75] M. K. Stoyanov and C. G. Webster, *A dynamically adaptive sparse grids method for quasi-optimal interpolation of multidimensional functions*, Comput. Math. Appl., 71 (2016), pp. 2449–2465.
[76] E. Strubell, A. Ganesh, and A. McCallum, *Energy and Policy Considerations for Deep Learning in NLP*, preprint, https://arxiv.org/abs/1906.02243, 2019.
[77] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, *Intriguing Properties of Neural Networks*, preprint, https://arxiv.org/abs/1312.6199, 2013.
[78] W. Z. Taffese and E. Sistonen, *Machine learning for durability and service-life assessment of reinforced concrete structures: Recent advances and future directions*, Automation in Construction, 77 (2017), pp. 1–14.
[79] A. L. Tarca, V. J. Carey, X.-w. Chen, R. Romero, and S. Drăghici, *Machine learning and its applications to biology*, PLoS Comput. Biol., 3 (2007), e116.
[80] H. Tran, C. G. Webster, and G. Zhang, *Analysis of quasi-optimal polynomial approximations for parameterized PDEs with deterministic and stochastic coefficients*, Numer. Math., 137 (2017), pp. 451–493.
[81] M. Unser, *A representer theorem for deep neural networks*, J. Mach. Learn. Res., 20 (2019), pp. 1–28.
[82] E. van den Berg and M. P. Friedlander, *SPGL1: A Solver for Sparse Least Squares*, 2020, https://friedlander.io/spgl1/.
[83] E. van den Berg and M. P. Friedlander, *Probing the Pareto frontier for basis pursuit solutions*, SIAM J. Sci. Comput., 31 (2008), pp. 890–912, https://doi.org/10.1137/080714488.
[84] C. Wu, P. Karanasou, M. J. Gales, and K. C. Sim, *Stimulated deep neural network for speech recognition*, in Proceedings of Interspeech 2016, San Francisco, 2016, pp. 400–404.
[85] D. Yarotsky, *Error bounds for approximations with deep ReLU networks*, Neural Netw., 94 (2017), pp. 103–114.
[86] D. Yarotsky, *Optimal Approximation of Continuous Functions by Very Deep ReLU Networks*, preprint, https://arxiv.org/abs/1802.03620, 2018.
[87] G. Zhang, J. Zhang, and J. Hinkle, *Learning nonlinear level sets for dimensionality reduction in function approximation*, in Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 13199–13208.
[88] B. Zieliński, A. Plichta, K. Misztal, P. Spurek, M. Brzychczy-Włoch, and D. Ochońska, *Deep learning approach to bacterial colony classification*, PLoS ONE, 12 (2017), e0184554.