Learning the mapping $\mathbf{x} \mapsto \sum_{i=1}^{d} x_i^2$: the cost of finding the needle in a haystack

A Preprint

Jiefu Zhang
Department of Mathematics,
University of California, Berkeley,
Berkeley, CA 94720.
jiefuzhang@berkeley.edu

Leonardo Zepeda-Núñez
Department of Mathematics,
University of Wisconsin-Madison,
Madison, WI 53706.
lzepeda@math.wisc.edu

Yuan Yao
Department of Mathematics,
Hong Kong University of Science and Technology,
Clear Water Bay, Kowloon, Hong Kong SAR.
yuany@ust.hk

Lin Lin
Department of Mathematics,
University of California, Berkeley,
Computational Research Division,
Lawrence Berkeley National Laboratory,
Berkeley, CA 94720.
linlin@math.berkeley.edu

February 26, 2020

Abstract

The task of using machine learning to approximate the mapping $\mathbf{x} \mapsto \sum_{i=1}^{d} x_i^2$ with $x_i \in [-1, 1]$ seems to be a trivial one. Given the knowledge of the separable structure of the function, one can design a sparse network to represent the function very accurately, or even exactly. When such structural information is not available, and we may only use a dense neural network, the optimization procedure to find the sparse network embedded in the dense network is similar to finding the needle in a haystack, using a given number of samples of the function. We demonstrate that the cost (measured by sample complexity) of finding the needle is directly related to the Barron norm of the function. While only a small number of samples is needed to train a sparse network, the dense network trained with the same number of samples exhibits large test loss and a large generalization gap. In order to control the size of the generalization gap, we find that the use of explicit regularization becomes increasingly more important as d increases. The numerically observed sample complexity with explicit regularization scales as $\mathcal{O}(d^{2.5})$, which is in fact better than the theoretically predicted sample complexity that scales as $\mathcal{O}(d^4)$. Without explicit regularization (also called implicit regularization), the numerically observed sample complexity is significantly higher and is close to $\mathcal{O}(d^{4.5})$.

1 Introduction

Machine learning and, in particular, deep learning methods have revolutionized numerous fields such as speech recognition [16], computer vision [22], drug discovery [27], genomics [24], etc. The foundation of deep learning is the universal approximation theorem [6, 17, 19, 28], which allows neural networks (NN) to approximate a large class of functions arbitrarily well, given a sufficient large number of degrees of freedom. In practice, however, the number of degrees of freedom is often limited by the computational power, thus the choice of the architecture to reduce the number of degrees of freedom is of paramount importance for the quality of the approximation [14, 15, 28]. Empirically, NN models have been shown to be surprisingly efficient in finding good local, and sometimes global, optima when using an overparameterized model, e.g. training a sparse teacher network is less efficient than training a dense, overparametrized student network [26]. It has been argued that the energy landscape of an overparameterized model may be benign, and in certain situations all local minima become indeed global minima [13, 18, 23, 30]. Furthermore, starting from an overparameterized model, observations such as the lottery ticket hypothesis [11, 12, 25] states that with proper initializations, it is possible to identify the "winning tickets", i.e. a sparse subnetwork with accuracy comparable to the original dense network.

We point out that many of the aforementioned studies focus on image classification problems using common data sets such as MNIST, CIFAR10 and ImageNet, with or without the presence of noise. However, in scientific computing, the setup of the problem can be very different: usually, we are interested in using NN models to parameterize a smooth, high-dimensional function accurately, and often without artificial noise. Within this context, the results mentioned above naturally raise the following questions:

(1) How important is it to select the optimal architecture? In other words, does it matter whether one uses an overparameterized model?

(2) If there is a sparse subnetwork that is as accurate as the overparameterized network, can the training procedure automatically identify the subnetwork? In other words, what is the cost of finding the needle (sparse subnetwork) in a haystack (overparameterized network)?

(3) If (2) is possible, how does the training procedure (such as the use of regularization) play a role?

This paper presents a case study of these questions in terms of a deceivingly simple task: given $\mathbf{x} \in [-1, 1]^d$ drawn from a certain probability distribution and a target accuracy ϵ, learn the square of its 2-norm, i.e. the function

$$\widetilde{f}^*(\mathbf{x}) := \sum_{i=1}^{d} x_i^2. \tag{1}$$

More specifically, for given a neural network model $f(\mathbf{x}, \theta)$, where θ denotes the parameters in the model, and for a given loss function, such as the quadratic loss $\ell(y, y') = \frac{1}{2}(y - y')^2$, our goal is to find θ such that the population loss

$$L(\theta) = \mathbb{E}_{\mathbf{x},y}[\ell(f(\mathbf{x}; \theta), \widetilde{f}^*(\mathbf{x}))] \leq \epsilon.$$

Note that $\widetilde{f}^*(\mathbf{x}) \sim \mathcal{O}(d)$, so we consider the scaled target function [1] in order to normalize the output

$$f^*(\mathbf{x}) = \frac{1}{d}\widetilde{f}^*(\mathbf{x}). \tag{2}$$

However, as in many scientific computing applications, the magnitude of the quantity of interest indeed grows with respect to the dimension, and our interest here is to approximate the original function $\widetilde{f}^*(\mathbf{x})$ to ϵ accuracy. Using a quadratic loss the population loss needed for approximating f^* becomes ϵ/d^2. In other words, if each component of $\mathbf{x}$ is chosen randomly, then by the law of large number $f^*(\mathbf{x})$ converges to a constant $\mathbb{E}[x^2]$ as $d \to \infty$. So the ϵ/d^2 target accuracy means that it is the deviation from such a mean value that we are interested in.

If we are allowed to use the mapping $x \mapsto x^2$ as an activation function, we would apply this function to each component and sum up the results, ensuing that the representation will be exact. Therefore, we exclude such an activation function, and only use standard activation functions such as ReLU or sigmoid, which requires only $\mathcal{O}(\log(1/\epsilon))$ neurons to reach accuracy ϵ [33].

On the one hand, we can build a network leveraging the separability of Eq. (1). In particular, we can use a small network to approximately represent the scalar mapping $x \mapsto x^2$, and sum up the results from all

[1] Correspondingly $\widetilde{f}^*$ will be called the original target function, or the unscaled target function.

components. The weights for the neural network of each component are shared, so the number of parameters is independent of d. The network will be called the local network (LN) below. On the other hand, if we do not have the *a priori* structural information that the target function is separable, we need to use a dense or fully connected neural network, which is referred to as the global network (GN). Fig. 1 sketches the structure of LN and GN. Note that LN can be naturally embedded into GN as a subnetwork by deleting certain edges. Therefore the *optimal* performance of GN should be at least as good as that of LN.

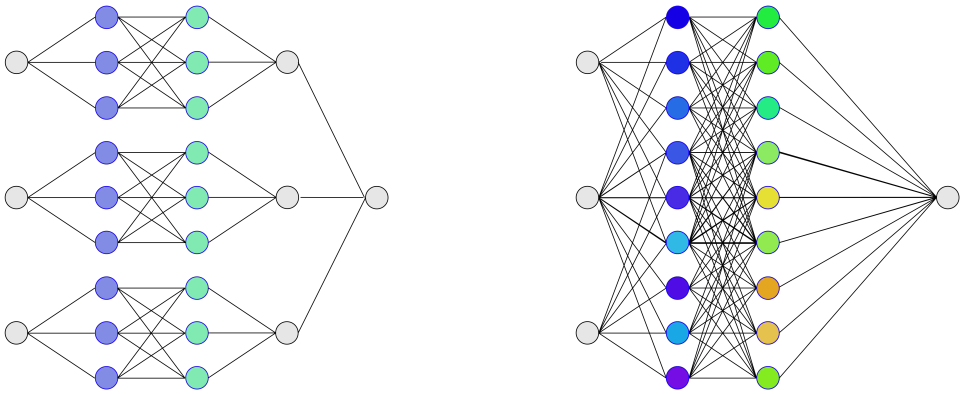


Figure 1: The architecture of the local network is on the left, and the architecture of the global network is on the right. Here $d = 3$ and the number of channels $\alpha = 3$.

Throughout the paper we assume that the number of neurons in the LN is large enough so that the scalar mapping $x \mapsto x^2$ is learned very accurately (with test error less than e.g. 10^{-5}), and the structure GN is then obtained by connecting all the remaining edges. In addition to the architectural bias, we are also concerned with the sample complexity about GN, i.e. the number of independent samples of $f(\mathbf{x})$ as the training data to reach a certain target accuracy (i.e. generalization error). Our results can be summarized as follows.

1. Using the same number of samples, LN can perform significantly better than GN. This shows that the global energy landscape of GN cannot be very simple, and the desired LN subnetwork cannot be easily identified.
2. If we embed a converged LN into a GN, add small perturbations to the weights of GN, and start the training procedure, LN can still outperform GN. This shows that the local energy landscape of GN may not be simple either.
3. In order to use GN to achieve performance that is comparable to LN, we need a significantly larger number of samples. The number of samples needed to reach certain target accuracy increases with respect to the dimension as $\mathcal{O}(d^\gamma)$ up to logarithmic factors. From *a priori* error analysis, we have $\gamma = 4$.
4. The numerical scaling of the sample complexity with respect to d depends on the regularizations. In particular, when proper regularization (ℓ^1, ℓ^2, or path norm regularization [9]) is used, the numerically observes sample complexity is around $\mathcal{O}(d^{2.5})$, which behaves better than the theoretically predicted worst case complexity, which scales as $\mathcal{O}(d^4)$. On the other hand, when implicit regularization (i.e. early stopping [32]) is used, we observe $n \sim \mathcal{O}(d^{4.5})$, i.e. the sample complexity using implicit regularization is significantly larger than that with explicit regularization. The early stopping criteria we use in this paper is that: after T (1000 in default) epochs, we find an optimal $t^* \leq T$ where the validation error is minimized. Furthermore, the trained weight matrix obtained with explicit regularization is approximately a sparse matrix, while the weight matrix obtained with early stopping is observed to be a dense matrix.

Related works:

In order to properly describe the sample complexity to reach certain target accuracy, we need to have *a priori* error estimate (a.k.a. worst-case error), and/or *a posteriori* error estimate (a.k.a. instance-based error) of the generalization error. For two-layer neural networks, such estimates have been recently established [3, 9] for a large class of functions called Barron functions [4, 21]. The estimates have also been recently extended

to deep networks based on the ResNet structure [15, 8]. We obtain our theoretical estimate of the sample complexity with respect to d by applying results of [9] to $f(\mathbf{x})$. For a sufficiently wide two-layer neural network, [2] studied the generalization error together with the gradient descent dynamics when using a polynomial activation function. The problem of "finding the needle in a haystack" by comparing the performance of fully-connected networks (FCN) and convolutional neural networks (CNN) was also recently considered by [7] for image recognition problems. It was found that there are rare basins in the space of fully-connected networks associated with very small generalization errors, which can be accessed only with prior information from CNN. This corroborates our finding for learning $f(\mathbf{x})$ here. Note that the separable structure in the target function can also be viewed from the perspective of permutation symmetry. Therefore our study also supports the argument for recognizing the importance of preserving symmetries when designing neural network architectures, which has been observed by numerous examples in physics based machine learning applications [34, 35]. Our result also corroborates the recent study [29], which questioned the prediction power of theoretical generalization error bound rates for overparameterized deep networks trained without explicit regularization.

1.1 Organization

In Section 2 we provide the theoretical foundations for the two-layer networks, including *a priori* and *a posteriori* bounds on the generalization gap. In Section 3 we use the theory developed in Section 2 to find the bounds for the generalization error for the squared norm. Section 4 provides the numerical experiments, followed by discussion in Section 5.

2 Generalization error of two-layer networks

In this section, we briefly describe the concept of the Barron norm, and the generalization error for two-layer neural networks. We refer readers to [9, 10] for more details. Let the domain of interest be $\Omega = [-1, 1]^d$. We assume the magnitude of the target function is already normalized to be $\mathcal{O}(1)$, e.g. the scaled function $f^*(\mathbf{x})$ in Eq. (2). Then for any $y' \sim \mathcal{O}(1)$, both the magnitude and the Lipschitz constant of the square loss function $\ell(\cdot, y')$ are of $\mathcal{O}(1)$.

2.1 Barron norm

We say that a function $f : \Omega \to \mathbb{R}$ can be represented by a two-layer NN if

$$f(\mathbf{x}; \theta) = \sum_{k=1}^{m} a_k \sigma\left(\mathbf{w}_k^\top \mathbf{x}\right). \tag{3}$$

Here $\mathbf{w}_k \in \mathbb{R}^d$, $\theta := \{(a_k, \mathbf{w}_k)\}_{k=0}^m$ represents all the parameters in the network, and $\sigma(\cdot)$ is an scale-invariant activation function such as ReLU. The scale invariance implies

$$\sigma(\mathbf{w}^\top \mathbf{x}) = \|\mathbf{w}\|_1 \sigma(\hat{\mathbf{w}}^\top \mathbf{x}), \quad \hat{\mathbf{w}} = \mathbf{w}/\|\mathbf{w}\|_1. \tag{4}$$

Therefore we may, without loss of generality, absorb the magnitude $\|\mathbf{w}\|_1$ into the scalar a, and assume $\|\mathbf{w}\|_1 = 1$.

The training set is composed of n i.i.d. samples $\mathcal{S} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^n$. To distinguish the indices, we use $\mathbf{x}^{(i)}$ to denote the i-th sample of the vector $(1 \le i \le n)$, and use $x_j^{(i)}$ to denote the j-th component of the vector $\mathbf{x}^{(i)}$ $(1 \le j \le d)$.

Our goal is to minimize the population loss

$$L(\theta) = \mathbb{E}_{\mathbf{x},y}[\ell(f(\mathbf{x}; \theta), y)],$$

through the minimization of the training loss

$$\hat{L}_n(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell\left(f\left(\mathbf{x}^{(i)}; \theta\right), y^{(i)}\right).$$

For a realization of parameters θ, the generalization gap is defined as $\left|L(\theta) - \hat{L}_n(\theta)\right|$.

A function f represented by a two-layer neural network is a special case of the Barron function, which admits the following integral representation

$$f(\mathbf{x}) = \int_{S^d} a(\mathbf{w})\sigma(\langle \mathbf{w}, \mathbf{x}\rangle)\,\mathrm{d}\pi(\mathbf{w}), \tag{5}$$

where π is a probability distribution over $S^d := \{\mathbf{w} | \, \|\mathbf{w}\|_1 = 1\}$, and $a(\cdot)$ is a scalar function. In particular, when we choose $\pi(\mathbf{w}) := \sum_{k=1}^{m} \delta(\mathbf{w} - \mathbf{w}_k)$ to be a discrete measure and define $a_k = a(\mathbf{w}_k)$, we recover the standard two-layer network in Eq. (3).

Definition 1 (Barron norm and Barron space)**.** *Let f be a Barron function. Denote by Θ_f all the possible representations of f, i.e.*

$$\Theta_f = \{(a, \pi) | f(\mathbf{x}) = \int_{S^d} a(\mathbf{w})\sigma(\langle \mathbf{w}, \mathbf{x}\rangle)\,\mathrm{d}\pi(\mathbf{w})\}.$$

Then the Barron-p norm is defined by

$$\gamma_p(f) := \inf_{(a,\pi)\in\Theta_f} \left(\int_{S^d} |a(\mathbf{w})|^p \,\mathrm{d}\pi(\mathbf{w}) \right)^{1/p}. \tag{6}$$

We may then define the Barron space as

$$\mathcal{B}_p(\Omega) = \{f : \gamma_p(f) < \infty\}. \tag{7}$$

Unlike the standard L^p norms, Proposition 2 shows a remarkable result, which is that all Barron norms are equivalent ([10, Proposition 2.1]).

Proposition 2 (Equivalence of Barron norms)**.** *For any function $f \in \mathcal{B}_1(\Omega)$,*

$$\gamma_1(f) = \gamma_p(f), \quad 1 \le p \le \infty. \tag{8}$$

To see why this can be the case, let us consider the two-layer NN in Eq. (3), and assume $\|\mathbf{w}_k\|_1 = 1$ for all k. By Hölder's inequality $\gamma_p(f) \le \gamma_q(f)$ when $1 \le p < q \le \infty$. In particular, $\gamma_1(f) \le \gamma_\infty(f)$, and we only need to show $\gamma_\infty(f) \le \gamma_1(f)$. Since π should be a discrete measure, let (a, π) be the minimizer for γ_1 as in Eq. (6). Then (recall $a_k := a(\mathbf{w}_k)$)

$$\gamma_1(f) = \sum_{k=1}^{m} |a_k|.$$

However, we may define a new distribution

$$\widetilde{\pi}(\mathbf{w}) = \sum_{k=1}^{m} \frac{1}{\gamma_1(f)} |a_k| \, \delta(\mathbf{w} - \mathbf{w}_k),$$

then

$$\int_{S^d} a(\mathbf{w})\,\mathrm{d}\pi(\mathbf{w}) = \sum_{k=1}^{m} |a_k| = \gamma_1(f) \sum_{k=1}^{m} \frac{1}{\gamma_1(f)} |a_k| = \gamma_1(f) \int_{S^d} \mathrm{d}\widetilde{\pi}(\mathbf{w}).$$

In other words, $(\widetilde{a}, \widetilde{\pi}) \in \Theta_f$, where $\widetilde{a}_k = \gamma_1(f)$ is a constant. By definition of Eq. (6),

$$\gamma_\infty(f) \le \gamma_1(f) \int_{S^d} \mathrm{d}\widetilde{\pi}(\mathbf{w}) = \gamma_1(f).$$

This proves $\gamma_\infty(f) = \gamma_1(f)$. The same principle can be generalized to prove Proposition 2 for general Barron functions [10].

Proposition 2 shows that the definition of the Barron space $\mathcal{B}_p(\Omega) = \mathcal{B}_1(\Omega)$ is independent of p. Therefore we may drop the subscript p and write $\mathcal{B}_p(\Omega)$ as $\mathcal{B}(\Omega)$. Similarly we denote by $\|f\|_{\mathcal{B}} := \gamma_1(f)$ as the Barron norm.

2.2 Error bound

For a given parameter set θ defining Eq. (3), the Barron norm is closely related to the path norm

$$\|\theta\|_{\mathcal{P}} := \sum_{k=1}^{m} |a_k| \,. \tag{9}$$

According to Eq. (6) we immediately have $\|f\|_{\mathcal{B}} \leq \|\theta\|_{\mathcal{P}}$. The path norm can be used to obtain the following *a posteriori* error estimate for the generalization gap for any choice of θ.

Theorem 3 (*A posteriori* error estimate [9])**.** *For any choice of parameter θ, for any $\delta > 0$, and with probability at least $1-\delta$ over the choice of the training set $\mathcal{S}$,*

$$\left| L(\theta) - \hat{L}_n(\theta) \right| \lesssim \sqrt{\frac{\ln(d)}{n}} \left(\|\theta\|_{\mathcal{P}} + 1 \right) + \sqrt{\frac{\ln((\|\theta\|_{\mathcal{P}}+1)^2/\delta)}{n}}.$$

For a given two-layer network, the path norm can be computed directly. Hence in order to reduce the generalization error to ϵ, the number of samples needed is approximately $\mathcal{O}(\|\theta\|_{\mathcal{P}}^2/\epsilon^2)$, which is a practically useful bound. However, in order to estimate the scaling of n with respect to the dimension d for a particular function, we need to replace the $\|\theta\|_{\mathcal{P}}$ by some measure of the complexity associated with f^* itself, such as the Barron norm. There indeed exists a two-layer network with a bounded path norm, so that the population loss is small. This is given in Proposition 4 ([9, Proposition 2.1]).

Proposition 4. *For any $f \in \mathcal{B}(\Omega)$, there exists a two layer neural network $f(\mathbf{x};\tilde{\theta})$ of width m with $\|\tilde{\theta}\|_{\mathcal{P}} \leq 2\|f^*\|_{\mathcal{B}}$, such that*

$$L(\widetilde{\theta}) \lesssim \frac{\|f^*\|_{\mathcal{B}}}{m}. \tag{10}$$

Eq. (10) characterizes the approximation error due to the use of a neural network of finite width, which decays as m^{-1}. Proposition 4 states that it is in principle possible to reduce the population loss while keeping the path norm being bounded. However, numerical results (both previous works and our own results here) indicate that when minimizing with respect to the training loss directly (when early stopping is used, this is also called the implicit regularization), the path norm associated with the optimizer can be very large. According to Theorem 3, the resulting generalization error bound can be large as well.

A key result connecting the *a priori* and *a posteriori* error analysis is to impose stronger requirements of the training procedure. Instead of minimizing with respect to the training loss $\hat{L}_n(\theta)$ directly, [9] proposes to minimize with respect to the following regularized loss function

$$J_\lambda(\theta) := \hat{L}_n(\theta) + \lambda \|\theta\|_{\mathcal{P}}, \tag{11}$$

where $\lambda > 0$ is a penalty parameter. The corresponding minimizer is defined as

$$\hat{\theta}_{n,\lambda} = \arg\min_{\theta} J_\lambda(\theta).$$

The benefit of minimizing with respect to Eq. (11) is that the path norm is penalized explicitly in the objective function, which allows us to control both the path norm and the generalization error.

Theorem 5 (*A priori* error estimate [9])**.** *Assume that the target function $f^* \in \mathcal{B}(\Omega)$, and $\lambda \geq \lambda_n = 4\sqrt{2\ln(2d)/n}$. Then for any $\delta > 0$ and with probability at least $1-\delta$ over the choice of the training set $\mathcal{S}$*

$$L(\hat{\theta}_{n,\lambda}) \lesssim \frac{\|f^*\|_{\mathcal{B}}^2}{m} + \lambda(\|f^*\|_{\mathcal{B}} + 1) + \frac{1}{\sqrt{n}} \left(\|f^*\|_{\mathcal{B}} + \sqrt{\ln(n/\delta)} \right). \tag{12}$$

The path norm of the parameter satisfies

$$\left\| \hat{\theta}_{n,\lambda} \right\|_{\mathcal{P}} \lesssim \frac{\|f^*\|_{\mathcal{B}}^2}{\lambda m} + \|f^*\|_{\mathcal{B}} + \sqrt{\ln(1/\delta)}. \tag{13}$$

For a fixed target function f^*, the contribution to the error in Eq. (12) can be interpreted as follows: the first term is the approximation error, determined by the width of the network. The third is determined by the sample complexity which is proportional to $n^{-\frac{1}{2}}$. The second term is present due to the need of balancing the loss and the path norm in the objective function (11). If λ is too large, then the regularized loss function is too far away from the training loss, and the error bound becomes large. On the other hand, Theorem 5 requires λ should be at least $\sim n^{-\frac{1}{2}}$. This can also be seen from Eq. (13) that if λ is too small, the path norm becomes unbounded. Therefore to balance the two factors we should choose the regularization parameter as $\lambda \sim n^{-\frac{1}{2}}$.

3 Generalization error for squared norm

In this section we apply the generalization error bound in Section 2 to study the scaling of the sample complexity with respect to the dimension d for the function in Eq. (1). The two-layer network in Eq. (3) can be viewed as a simplified model of GN (which can have multiple layers). According to Theorem 3 and Theorem 5, we need to estimate the Barron norm $\|f^*\|_{\mathcal{B}}$.

For a given function, the Barron norm is often difficult to compute due to the minimization with respect to all possible (a, π). Instead we may compute the spectral norm. For a given function $f \in C(\Omega)$, let F be an extension of f to $\mathbb{R}^d$, denoted by $F|_\Omega = f$. Define the Fourier transform

$$\hat{F}(\mathbf{k}) = \frac{1}{2\pi} \int_{\mathbb{R}^d} e^{-i\mathbf{k}\cdot\mathbf{x}} F(\mathbf{x}) \, \mathrm{d}\mathbf{x}.$$

Then spectral norm of f is defined as

$$\|f\|_s = \inf_{F|_\Omega = f} \int_{\mathbb{R}^d} \|\mathbf{k}\|_1^2 |\hat{F}(\mathbf{k})| \, \mathrm{d}\mathbf{k}. \tag{14}$$

Note that the infimum is taken over all possible extensions F. Then the Barron norm can be bounded by the spectral norm as [10, Theorem 2]

$$\|f\|_{\mathcal{B}} \leq 2 \|f\|_s + 2\|\nabla f(0)\|_1 + 2|f(0)|. \tag{15}$$

Therefore we may obtain an upper bound of the Barron norm via the spectral norm.

Let us now consider $f(\mathbf{x})$ in (2), which satisfies $\|\nabla f(0)\|_1 = |f(0)| = 0$. To evaluate the spectral norm, we consider the one-dimensional version $g(x) = x^2$. Consider any C^2 extension of g to $\mathbb{R}$, denoted by G, which satisfies $\int_{\mathbb{R}} k^2 \hat{G}(k) \, \mathrm{d}k < \infty$. Then by definition $\gamma_s(g) \leq \int_{\mathbb{R}} k^2 \hat{G}(k) \, \mathrm{d}k < \infty$.

Now consider the extension $F(\mathbf{x}) = \frac{1}{d} \sum_{i=1}^d G(x_i)$, and $\hat{F}(\mathbf{k}) = \frac{1}{d} \sum_{i=1}^d \hat{G}(k_i) \prod_{j \neq i} \delta(k_j)$. Here $\delta(\cdot)$ is the Dirac-δ function. Then

$$\int_{\mathbb{R}^d} \|\mathbf{k}\|_1^2 |\hat{F}(\mathbf{k})| \, \mathrm{d}\mathbf{k} = \frac{1}{d} \sum_{i=1}^d \int_{\mathbb{R}} k^2 \hat{G}(k) \, \mathrm{d}k = \int_{\mathbb{R}} k^2 \hat{G}(k) \, \mathrm{d}k,$$

which gives

$$\|f\|_s \leq \int_{\mathbb{R}} k^2 \hat{G}(k) \, \mathrm{d}k. \tag{16}$$

Combined with Eq. (15), we find that $\|f\|_{\mathcal{B}} \sim \mathcal{O}(1)$. Thus according to Proposition 4 we expect that the path norm of the regularized solution satisfies $\|\hat{\theta}\|_{\mathcal{P}} \sim \mathcal{O}(1)$. Hence the leading term of the generalization gap is

$$\|\theta\|_{\mathcal{P}} \sqrt{\frac{\ln(d)}{n}} \sim \sqrt{\frac{\ln(d)}{n}}. \tag{17}$$

This scaling seems quite favorable, as n only needs to grow as $\ln(d)$. However, recall the discussion in Section 1 that the target generalization error should be ϵ/d^2, this means that the required sample complexity (up to logarithmic factors) is

$$n \sim \mathcal{O}(d^4/\epsilon^2). \tag{18}$$

According to Theorem 5, the network also needs to be wide enough as $m \sim \mathcal{O}(d^2/\epsilon)$. $\lambda \sim \mathcal{O}(n^{-\frac{1}{2}}) = \mathcal{O}(\epsilon/d^2)$. In particular, the sample complexity increases very rapidly with respect to the dimension d in order to approximate the unscaled function $\widetilde{f}^*(\mathbf{x})$.

The analysis of the local network (LN) is essentially applying a two-layer network $g(x; \theta)$ to the scalar mapping $g(x) = x^2$. Following the discussion above, the sample complexity is simply $n \sim \mathcal{O}(1/\epsilon^2)$. Let $z_i = x_i^2 - g(x_i; \theta)$, and assume that the error from each component are of mean zero and independent, i.e.

$$\mathbb{E}(z_i z_j) = 0, \quad i \neq j. \tag{19}$$

Then the generalization error is simply

$$\begin{aligned} L(\hat{\theta}) =& \frac{1}{2}\mathbb{E}\left(\frac{1}{d}\sum_{i=1}^d x_i^2 - \frac{1}{d}\sum_{i=1}^d g(x_i;\theta)\right)^2, \\ =& \frac{1}{2d^2}\mathbb{E}\left(\sum_{i=1}^d z_i^2 + 2\sum_{i\neq j} z_i z_j\right), \\ \approx& \frac{1}{2d}\mathbb{E}\left(z_i^2\right) \sim \frac{1}{d}. \end{aligned}$$

Therefore the generalization error of LN can be $\mathcal{O}(d^{-1})$ smaller than that of GN. Note that the condition (19) is crucial for the d^{-1} factor. In fact if the errors from all components are correlated, there may be no gain at all in terms of the asymptotic scaling with respect to d!

However, our numerical results in Section 4 demonstrate that the performance of LN can be significant better than GN by a very large margin. Therefore there is still gap in terms of the theoretical understanding of the performance of LN.

4 Numerical results

In this section we describe in detail the numerical experiments along with the analysis of the empirical results.

Our first goal is to study the dependence of the generalization or test error with respect to the architectural bias. In section 4.2, we showcase the superior performance of the local network, which is built using structural information about the underlying problem, compared to the the global network, in which no information is used.

Our second goal is to study the impact of the explicit regularization (versus implicit regularization) when training the global network to approximate the unscaled target function. In section 4.3, we characterize the scaling of the test loss with respect to the input dimension, and the scaling of the test loss with respect to the number of samples with/without the explicit regularization. The explicit regularization improves both rates according to the numerical experiments.

4.1 Data generation and loss function

For all the numerical experiments, the dimension of the input denoted by d, ranges from 4 to 60 with step size 4. For each d, we generate 10^6 samples by first sampling $10^6 \times d$ numbers uniformly from the interval $[-1, 1]$ and then organizing the resulting data as a matrix of dimensions $10^6 \times d$. We then compute $y = \sum_{i=1}^d x_i^2$ for each row. With this setup, the domain of the target function is restricted to $\Omega = [-1, 1]^d$.

We also generate data for the sum of quartic and the cosine terms, namely $y = \sum_{i=1}^d x_i^4$ or $y = \sum_{i=1}^d \cos x_i$. These datasets are used in section 4.2 to showcase the importance of the architectural bias for target functions other than $y = \sum_{i=1}^d x_i^2$.

For the sake of reproducibility, all the experiments described below use the data generated with a fixed seed. In this section we use n to denote the number of the training samples and N_{sample} for the number of total samples (training, validation, test all combined). For experiments that involve $N_{\text{sample}} \leq 10^6$ samples, we extract the first N_{sample} rows from the total 10^6 rows of data.

To simplify the training procedure, we enforce the permutation symmetry of the inputs by including the sorting procedure while preprocessing the input data. We point out that this permutation symmetry can be directly embedded in the network [34].

On one hand, we have done our analysis in section 2 and 3 with respect to the scaled target function $f^*(\mathbf{x})$ in Eq. (2), so the neural networks we build in this section aim to learn the scaled target function. Thus, the mean squared error loss for training data when optimizing the networks are defined as

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^n \left(f_{NN}(\mathbf{x}^{(i)}, \theta) - f^*(\mathbf{x}^{(i)})\right)^2 \tag{20}$$

Here n denotes the number of training data, f_{NN} denotes the function represented by the network, and $\mathbf{x}^{(i)}$ represents ith row in the dataset. The test loss can be calculated in the same way using the test data instead of the training data. On the other hand, we are interested in the performance of the models in the original scale, so the reported training/test loss are scaled by d^2 to represent the mean squared error of approximating the original unscaled target function:

$$\text{MSE}_{\text{original}} = \frac{1}{n} \sum_{i=1}^{n} \left(d \cdot f_{NN}(\mathbf{x}^{(i)}, \theta) - \widetilde{f}^*(\mathbf{x}^{(i)}) \right)^2 \tag{21}$$

4.2 Architectural bias

In this section we showcase the empirical effects of the architectural bias on the test or generalization errors. To illustrate this effect, in a slightly more generality, we consider as target functions: $\widetilde{f}^*(\mathbf{x}) := \sum_{i=1}^{d} x_i^2$, $\sum_{i=1}^{d} x_i^4$, and $\sum_{i=1}^{d} \cos x_i$, respectively. We use three different network architectures to approximate each target function: global network, the local network, and the locally connected network (LCN), which is similar to the local network, but whose weights are not longer shared.

These three architectures are closely related. In the global (or dense) network, each layer is represented by a weight matrix that is an arbitrary dense matrix (see Figs. 1 and 2). In the locally connected network we have the same matrix but constrained to be block diagonal. Finally, in the local network we further constraint the weight matrix to be both block diagonal and block circulant, thus implying that the block in the diagonal are the same (usually called weight-sharing in the machine learning community). This observation allows us to embed a local network into a locally connected one, and then into a global one. This embedding is performed by simply copying the corresponding entries of the weights matrices at each layer and filling the rest with zeros as shown in Fig. 2.

Following [17] all three networks can approximate the target function to arbitrary accuracy with just one hidden layer: the result is straightforward for the global network by the universal approximation; for the locally connected and local networks, the result stems from applying the universal approximation to each coordinate, thus approximating the scalar component function $g : \mathbb{R} \to \mathbb{R}$ ($g(x) = x^2$ in the square case, x^4 in the quartic case and $\cos x$ in the cosine case). In practice, however, we find that with two hidden layers the networks are easier to train, all the hyperparameters being equal.

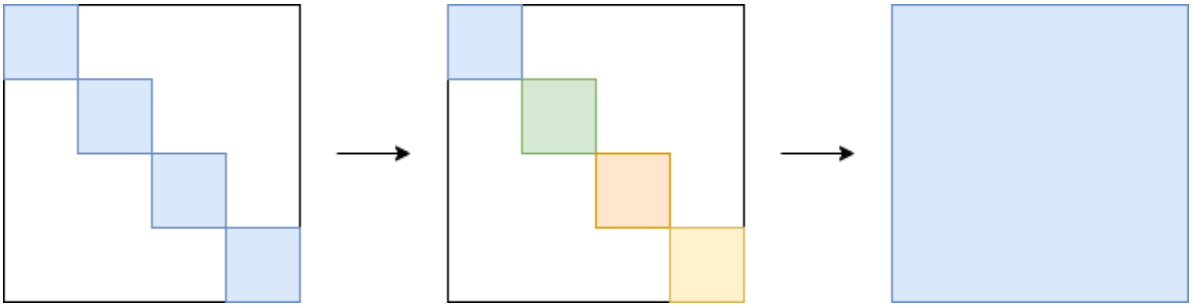


Figure 2: The relation of the weight matrices after embedding weight matrices for all three networks (between the first hidden layer and the second hidden layer with input dimension $d = 4$) into the weight matrix for the global network. The arrows mean that the matrices on the left hand side are subsets of the matrices on the right hand side. The blocks in the first weight matrix have the same color to illustrate the weight-sharing. Matrix elements in the white regions are zeros.

Following these considerations, for all the numerical experiments we fix the number of hidden layers to be two. In addition, we define α as the number of nodes per input node, which for the local networks coincides with the number of channels. Thus we have $d \cdot \alpha$ nodes in the hidden layers for the global network.

The networks are implemented[2] with Keras [5], using Tensorflow [1] as the back-end. Within the Keras framework, we use dense layers, locally connected layers and one-dimensional convolutional layers to implement the global network, the locally connected network, and the local network, respectively (see Algs. 1, 2, and 3). We add a lambda layer that divides the output by d at the end of the networks to learn the scaled function

[2]Detailed implementation can be found at `https://github.com/jfetsmas/NormSquareLearning.git`

$f^*(\mathbf{x})$ in Eq. (2). For both the locally connected and convolutional layers, we use stride and window size both equal to one, and α the number of channels. We point out that α can be considered as the number of nodes in the hidden layers for the block component that approximates the component function g as shown in Fig. 1. In this way, the function represented by the local network has the structure $\frac{1}{d}\sum_{i=1}^{d} g(x_i, \theta)$ enforced by the architecture, where $g(\cdot, \theta)$ is the neural network block that approximates component function g.

Algorithm 1: Code for the global network

```
layerInput       = Input(shape=(d,))
layerHidden1     = Dense(DenseNodes, activation='relu',
                         use_bias=True)(layerInput)
layerHidden2     = Dense(DenseNodes, activation='relu',
                         use_bias=True)(layerHidden1)
layerOutput_pre  = Dense(1, activation='linear', use_bias=False)(layerHidden2)
layerOutput      = Lambda(lambda x: x / d)(layerOutput_pre)
model            = Model(inputs=layerInput, outputs=layerOutput)
```

Algorithm 2: Code for the locally connected network

```
layerInput   = Input(shape=(d,1))
layerHidden1 = LocallyConnected1D(alpha, 1, strides=1,
               activation='relu', use_bias=True)(layerInput)
layerHidden2 = LocallyConnected1D(alpha, 1, strides=1,
               activation='relu', use_bias=True)(layerHidden1)
layerOutput  = LocallyConnected1D(1, 1, strides=1,
               activation='linear', use_bias=False)(layerHidden2)
Sum          = Lambda(lambda x: K.sum(x, axis=1), name='sum')
layerSum_pre = Sum(layerOutput)
layerSum     = Lambda(lambda x: x / d)(layerSum_pre)
model        = Model(inputs=layerInput, outputs=layerSum)
```

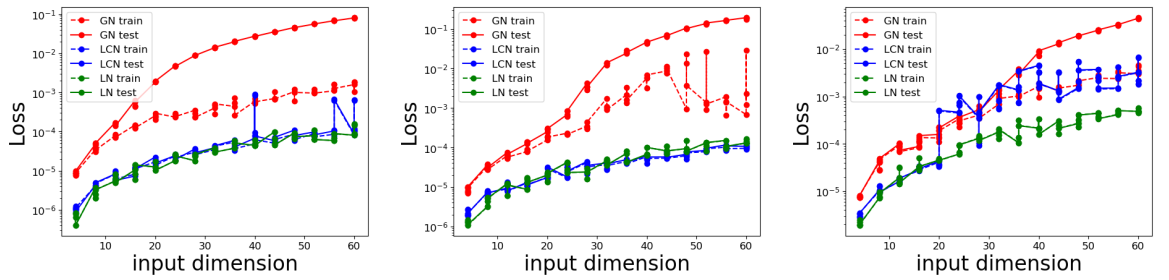
Algorithm 3: Code for the local network

```
layerInput   = Input(shape=(d,1))
layerHidden1 = Conv1D(alpha, 1, strides=1, activation='relu',
               use_bias=True)(layerInput)
layerHidden2 = Conv1D(alpha, 1, strides=1, activation='relu',
               use_bias=True)(layerHidden1)
layerOutput  = Conv1D(1, 1, strides=1, activation='linear',
               use_bias=False)(layerHidden2)
Sum          = Lambda(lambda x: K.sum(x, axis=1), name='sum')
layerSum_pre = Sum(layerOutput)
layerSum     = Lambda(lambda x: x / d)(layerSum_pre)
model        = Model(inputs=layerInput, outputs=layerSum)
```

In the experiments, we set the number of channels $\alpha = 50$ and we use ReLU as the nonlinear activation function. The models are permutation invariant due to a sorting procedure used to pre-process the data. In all experiments we use the default weight initializer (glorot uniform initializer) and the Adam optimizer with starting learning rate = 0.01 and other default parameters [20].

We split the data into training, validation, and test datasets. Among the data for a single numerical experiment, 64% is training data, 16% is validation data, and 20% is test data. We use batch size = (number of training and validation samples) / 100 for all experiments in this subsection, and thus each epoch contains 80 iterations. During the training process, we evaluate the loss on the validation set every epoch and we keep the model with the lowest validation loss, which is then reported.

For each value of the input dimension, we run four experiments with the same configuration, but with a different random seed for the optimizer. All experiments in this subsection use a dataset of size $N_{\text{sample}} = 10^5$ (training, validation, and test data all combined).



(a) target function $y = \sum_{i=1}^{d} x_i^2$ (b) target function $y = \sum_{i=1}^{d} x_i^4$ (c) target function $y = \sum_{i=1}^{d} \cos x_i$

Figure 3: Comparison of rescaled average mean square error for the three architectures. GN stands for global network, LCN for locally connected network, and LN for local network.

Fig. 3 depicts the behavior of the test and training losses for the three networks as the input dimension increases. The losses are computed using the mean squared error in the original scale as in Eq. (21). For all three target functions, the local network significantly outperforms the global network, especially for large input dimension. LN and LCN shows comparable performance for the quadratic and quartic functions. The training error of LCN is larger for the cosine function, but there is no noticeable generalization gap. We expect that the performance of LCN can be further improved through further hyperparameter tuning. In all three cases, the global network exhibits a large generalization gap, whereas this gap is almost non-existent for the local network. These results clearly indicates the influence of the architecture in the accuracy of the approximation and the generalization gap.

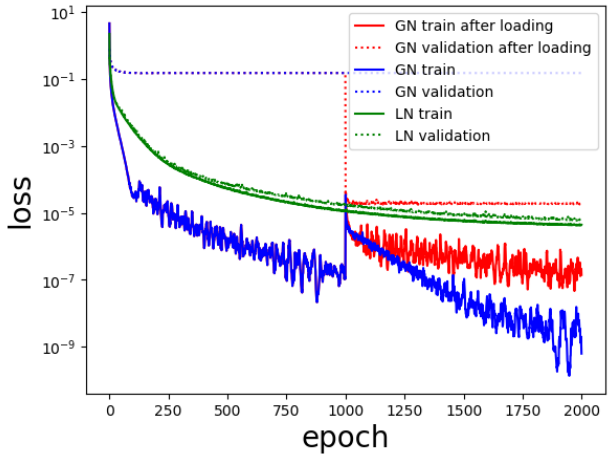


Figure 4: Evolution of the mean squared error with respect to the number of epochs for the different networks. Blue lines are the trajectories of the training and validation loss of the global network; green lines are for the local network; Red lines are the trajectories after loading the local network weights into the global network at epoch 1000.

As depicted in Fig. 2, we know that the local networks are indeed a subset of the global ones, and given the high-level of accuracy obtained by the local network we can infer that the global network should have enough representation power to approximate the target solution accurately. Following this logic, the large gap can be then attributed to the optimization procedure, which is unable to identify a suitable approximation to the local network ("needle") among all overparameterized dense networks ("haystack").

A natural question is: can the optimization procedure find a better solution starting from a good initial guess? To study this, we train a local network until a local minimum is achieved, and we transfer this minimum to its corresponding global network, and we use it as an initial guess for the training procedure. In particular, we fix $d = 20$, and the number of data at 1000. We train both the local and the global networks for 1000 epochs, do the weight transfer, and then resume training for another 1000 epochs. To obtain more stable training curves,

we add decay of 0.03 in the argument of the Adam optimizer. From Fig. 4 we can observe that by properly initializing the global network, we are able to significantly reduce the test loss, and to drastically bridge the generalization gap (red lines starting at epoch 1000). Although the training error of the global network is consistently lower than that of the local network (solid blue and red lines, compared with solid green line), the generalization gap (the gap between solid and dashed for the blue and red lines, compared with the gap for the green lines) of the global network remains noticeably larger compared with the local network.

4.3 The impact of the explicit regularization

In this subsection, we explore the relation between the test loss, the number of samples and the input dimension. The calculations in section 3 suggests that the leading term of the generalization gap should satisfy Eq. (17). For the global network, the generalization gap is sufficiently large[3] that we can regard the test loss to be approximately equal to the generalization error. Thus, we expect the test loss of the global network to be bounded by Eq. (17) (multiplied by d^2 for the unscaled function). We test the tightness of the bound by conducting the following two experiments:

1. we fix the sample size and investigate the relation between the test loss and the input dimension; and
2. we fix the input dimension d and obtain the rate of growth of the test loss with respect to the sample size.

On one hand, despite numerical errors and the fact that we only use the leading term in Theorem 3, we observe that the rate with respect to the input dimension suggested by Eq. (17) regarding the input dimension is close to optimal when using the explicit regularization. On the other hand, the rates with respect to the number of samples obtained in numerical experiments are around $\mathcal{O}(N_{\text{sample}}^{-0.8})$ without explicit regularization, and $\mathcal{O}(N_{\text{sample}}^{-1.0})$ with explicit regularization. Hence the convergence rate with respect to the number of samples is faster than the theoretical predicted worst case rate as $\mathcal{O}(N_{\text{sample}}^{-0.5})$.

For the first experiment we fix the sample size to 10^5. For each d ranging from 4 to 60, we repeat the training procedure four times with the same set of hyperparameters (the default values have been discussed in section 4.2). In Fig. 5(a), we display the training loss and test loss with respect to the input dimension in log-log scale. The losses are computed using the mean squared error in the original scale as in Eq. (21). The slope in Fig. 5(a) indicates that the generalization error of the approximation to the original target function $\widetilde{f}^*(\mathbf{x})$ as in Eq. (1) grows as $d^{3.6}$, and thus the generalization error of the approximation to the scaled $f^*(\mathbf{x})$ as in Eq. (2) grows as $d^{1.6}$. The empirical rate $d^{1.6}$ is a lot larger than logarithmic growth predicted in the bound in Eq. (17).

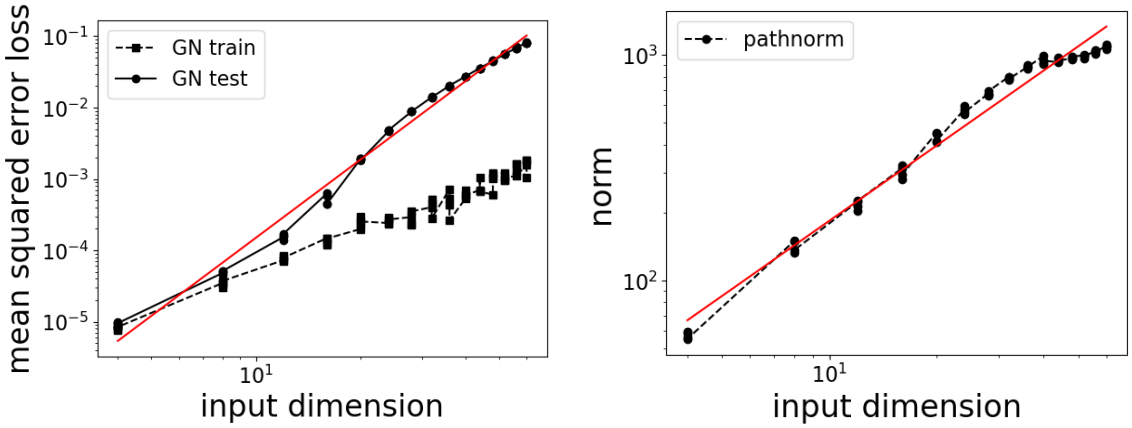
In particular, Eq. (17) indicates that in order to bound the generalization error, the path norm needs to be $\sim \mathcal{O}(1)$. However, the boundedness of the path norm is only proven in the context of explicit regularization. Without such an explicit regularization term (also called the "implicit regularization"), there is no *a priori* guarantee that the path norm can remain bounded as the input dimension increases.

We remark that to compute the path norm for our three-layer network, we need to modify the formula for two-layer network. Let us denote the weight matrices of the three layers to be $\mathbf{w}^1, \mathbf{w}^2, \mathbf{w}^3$. With the width of hidden layer for the global network being $d \cdot \alpha$ as shown in section 4.2, the size of the weight matrices are $d \cdot \alpha \times d, d \cdot \alpha \times d \cdot \alpha, 1 \times d \cdot \alpha$ respectively. In the three-layer case, the path norm can be calculated using the formula:

$$\|\theta\|_{\mathcal{P}} := \frac{1}{d} \sum_{i=1}^{d} \sum_{j,k=1}^{d\alpha} |\mathbf{w}_{ij}^1||\mathbf{w}_{jk}^2||\mathbf{w}_k^3| \tag{22}$$

The $\frac{1}{d}$ factor is due to the last layer in Alg. 1, which divides the output by d. Using the updated formula we plot the path norm with respect to the input dimension in Fig. 5(b). From the figure we can observe that the path norm grows as $\sim d^{1.1}$, thus violating the $\mathcal{O}(1)$ assumption for Eq. (17). Notice that in this scenario, although Eq. (17) does not apply, the bound in Theorem 3 provides a fairly good estimate of the growth of the generalization error. The leading term in the bound given by Theorem 3 grows as $\|\theta\|_{\mathcal{P}}\sqrt{\ln(d)}$. With the path norm $\sim d^{1.1}$, and $\sqrt{\ln(d)}$ empirically behaving like a fractional power of d for small d, the product has a rate similar to the rate of the test loss observed, which is $\sim d^{3.6}$ (after taking into account the d^2 scaling factor).

[3]Given that the training loss is usually one or more magnitude smaller than the test loss.



(a) the relationship between the generalization error and the input dimension. Slope of the best fitted line (red) is 3.638

(b) the relationship between the path norm and the input dimension. Slope of the best fitted line (red) is 1.107

Figure 5: Experiments without regularization

To demonstrate that explicit regularization indeed reduces the path norm, test loss and generalization error, we implement three types of regularization schemes, and study the growth of the errors with respect to the input dimension. Let the number of trainable parameters in the neural network be N_{par}. Denote the trainable parameters by $\{\theta_i\}_{i=1}^{N_{\text{par}}}$ and regularization constant by λ. We can summarize the implementation of the three regularization as

- L1 regularization, minimizing MSE+$\lambda \sum_{i=1}^{N_{\text{par}}} |\theta_i|$
- L2 regularization, minimizing MSE+$\lambda \sum_{i=1}^{N_{\text{par}}} \theta_i^2$
- path norm regularization, minimizing MSE+$\lambda \|\theta\|_{\mathcal{P}}$, where $\|\theta\|_{\mathcal{P}}$ is calculated by Eq. (22)

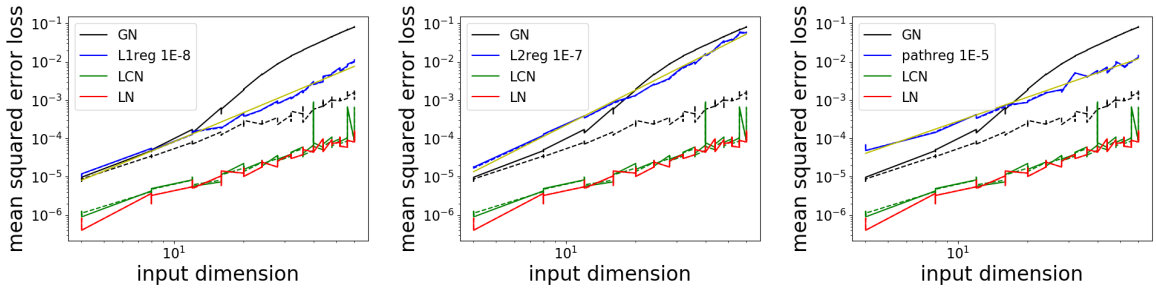
The MSE is computed as in Eq. (20). To implement the three regularization schemes, we use the kernel regularizer in Keras in tensorflow 1.7.0 for the L1 and the L2 regularization (the regularization is only on the weight matrices, but not on the bias), and we use tensorflow 2.0 to implement the path norm regularization. We choose the regularization constant λ (fixed with respect to d) empirically so that the global network achieves the best test loss. The regularization constants we select in Fig. 6 are 10^{-8} for L1 regularization, 10^{-7} for the L2 regularization, and 10^{-5} for the path norm regularization.

In addition to the test/train loss Vs. the input dimension for the regularization experiments, we also display our previous results for GN, LN, and LCN as a reference in Fig. 6

Recall that the growth rate of the test loss for the global network without regularization (shown in Fig. 5(a)) is around 3.638, all three regularization helps reduce down the rate in Fig. 6. Path norm regularization in Fig. 6(c) exhibits the best growth rate, despite that the test loss for small d is slightly larger. We plot the path norm Vs. the input dimension in Fig. 7 and we indeed observe that the path norm is $\mathcal{O}(1)$ as desired. As a matter of fact, the path norm even decreases slightly as d increases, and this is qualitatively different from the behavior of implicit regularization. In addition, with the explicit regularization and thus bounded path norm, Eq. (17) gives a tight estimate of the rate of growth.

Since the weight matrices for the LN embedded GN are block diagonal, and hence are sparse matrices. We will look at the first weight matrices of the models under the following scenario:

- GN without regularization,
- GN with L1 regularization,
- GN with L2 regularization,
- GN with path norm regularization, and
- GN with optimal LN weights embedded.



(a) Loss Vs. input dimension for GN with L1 regularization. Slope of the best fitted line (yellow): 2.527

(b) Loss Vs. input dimension for GN with L2 regularization. Slope of the best fitted line (yellow): 3.051

(c) Loss Vs. input dimension for GN with path norm regularization. Slope of the best fitted line (yellow): 2.098

Figure 6: Experiments with regularization, solid lines are test loss and dashed lines are training loss.

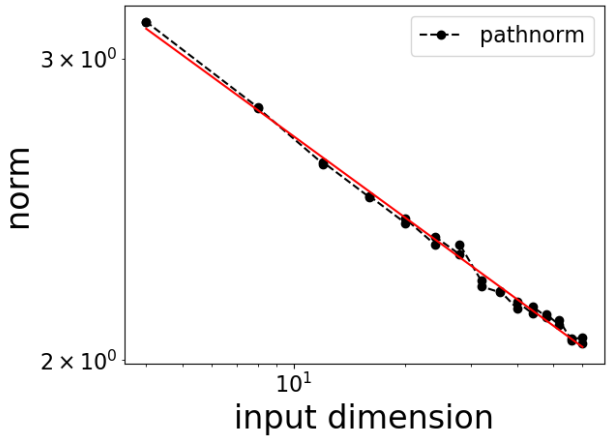


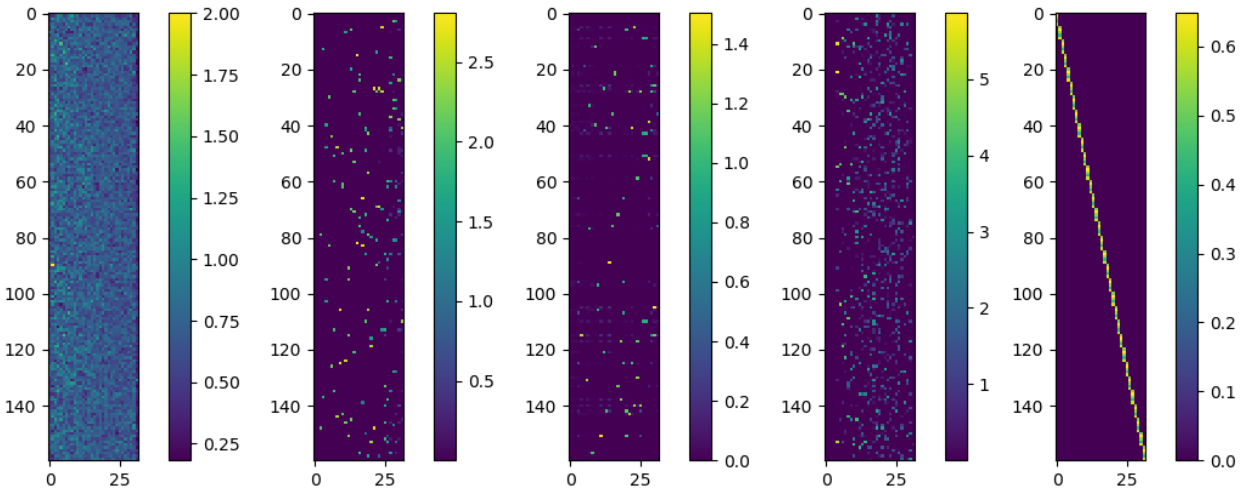
Figure 7: the relationship between the path norm and the input dimension for the path norm regularization experiment with $\lambda = 10^{-5}$. Slope of the best fitted line (red): -0.158

In Fig. 8, we display the first weight matrices for input dimension $d = 32$ for the five trained models. Since the weight matrix is from the input nodes to the first hidden layer, the dimension is 1600×32, where 1600 is obtained from input dimension (32) multiplied by number of nodes per input node (50). To improve visibility, we display the maximum absolute value among the 10 adjacent cells so that the first dimension is reduced by a factor of 10. We can see from the picture that L1 regularization, L2 regularization, and path norm regularization all lead to a much sparser weight matrix compared with the global network without regularization.

For the second experiment regarding the rate of growth of the test loss with respect to the sample size, we fix $d = 40$ (the choice is arbitrary), and we vary the sample size from 10^3 to 10^6. Since the sample size varies, we consider the following two choices of the batch size:

1. fix the ratio and let batch size be (number of training and validation samples) / 100, this choice ensures same number of iterations per epoch as the sample size increases.
2. fix the batch size to 80 as sample size varies. The number of iterations increases as the sample size increases.

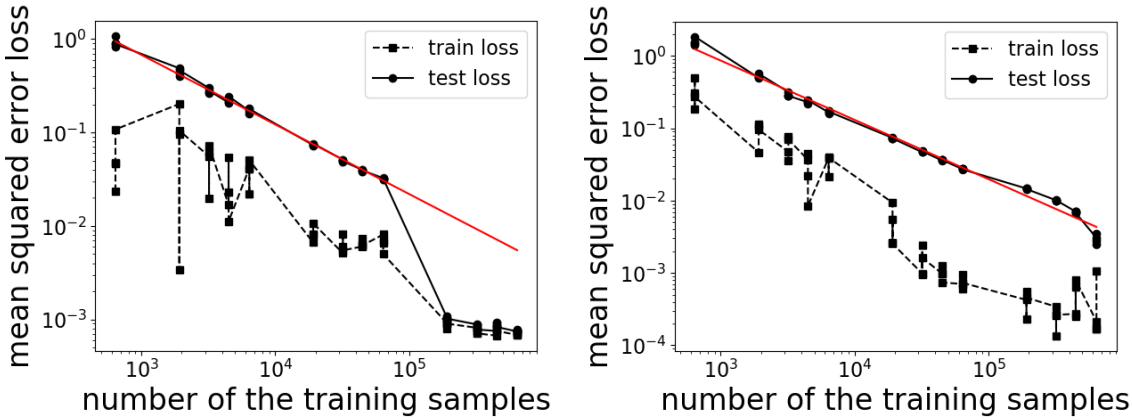
The other hyperparameters are the same as the setup in section 4.2. We run 4 training procedures for each number of samples and we report the resulting test loss (early stopping still applies) versus the number of samples, which are summarized in Fig. 9. In Fig. 9(a), with batch size fixed at 80, the performance changes when the number of training samples exceed 10^5. Since we focus on the generalization error in this paper (so we need the existence of generalization gap), and also the time cost of a training procedure



(a) GN w/o reg (b) GN L1 reg (c) GN L2 reg (d) GN path norm reg (e) loaded LN

Figure 8: The sparsity pattern of the weight matrices between the first and second layers after training.

drastically increases when the small batch size is applied to a large sample size (number of iterations per epoch, computed by training sample size / batch size, is large), we focus on the segment where the number of training samples is below 10^5 and the generalization gap is visible. Following Theorem 3, the generalization error decreases asymptotically as $\mathcal{O}(N_{\text{sample}}^{-0.5})$, but numerically we observe different exponents in both figures, which is around -0.8. The fixed batch size experiment in Fig. 9(a) returns a similar rate to the fixed ratio experiment in Fig. 9(b), therefore we may keep using the default setup (batch size with fixed ratio) for the rest of the studies in this subsection.



(a) Loss Vs. sample size for GN with fixed batch size. Slope of the best fitted line for test loss with an obvious generalization gap (red): -0.745

(b) Loss Vs. sample size for GN with batch size given by a fixed ratio. Slope of the best fitted line (red) is -0.820.

Figure 9: the relationship between the test loss and the sample size with two choices of batch size.

To confirm that the rate is consistent for other input dimension values, we repeat the same procedure (with batch size obtained by the fixed ratio) for $d = 60$. The trend and rate are reported in Fig. 10, and indeed the rate is still around -0.8, different from the theoretical rate -0.5.

We test the impact of explicit regularization on the rate of growth of the test loss with respect to the sample size by adding L1 regularization to all the weight matrices. The setup is the same as the global network with L1 regularization experiment displayed in Fig. 6(a), except that instead of fixing $N_{\text{sample}} = 10^5$ and letting d varies, we fix $d = 40$, $d = 60$ and let N_{sample} vary from 10^3 to 10^6. We fix the regularization constant at 10^{-8}

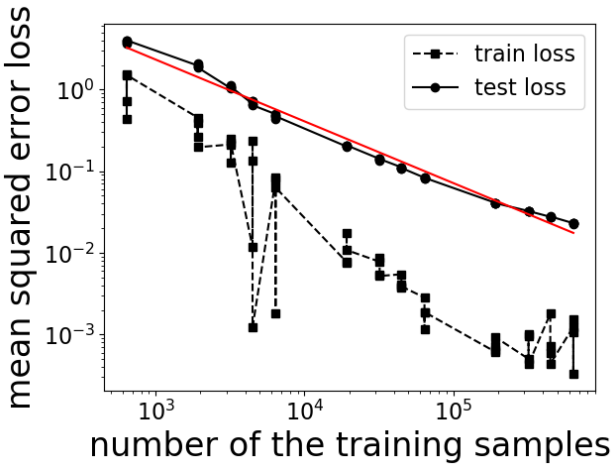
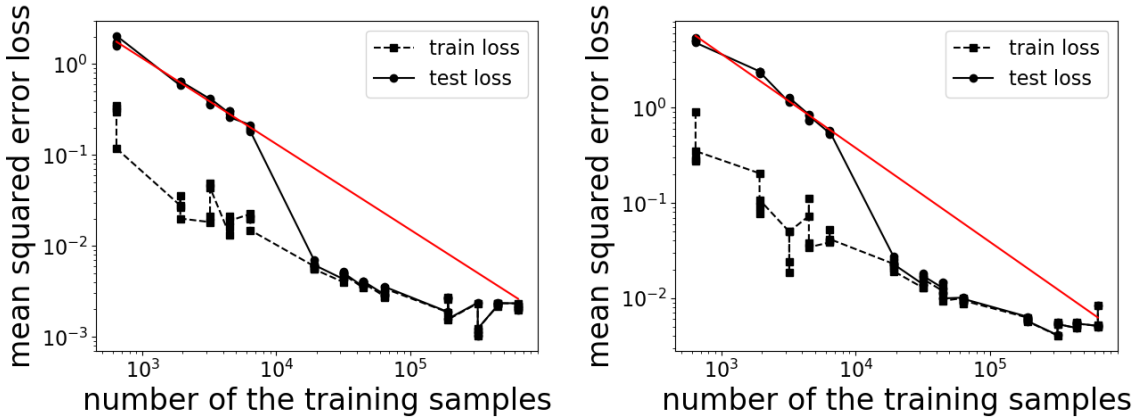


Figure 10: Loss Vs. sample size for GN for $d = 60$. Slope of the best fitted line (red) is -0.756

as in Fig. 6(a). The rates of the segment where there is a visible generalization gap are shown in Fig. 11. We observe that as N_{sample} increases to certain point (around 10^4), the fixed regularization constant makes the regularization term more significant in the loss, resulting in nearly vanishing generalization gap. Since our focus is on generalization error inspired by the theory in section 2 and 3, we report the rate in the segment with visible generalization gap. Notice that the rate in the L1 regularization is around -1.0, and hence the decay of the test loss with respect to the number of samples is faster than that without explicit regularization, which is around -0.8.



(a) Loss Vs. sample size for GN with L1 regularization for $d = 40$. Slope of the best fitted line for test loss with an obvious generalization gap (red): -0.943

(b) Loss Vs. sample size for GN with L1 regularization for $d = 60$. Slope of the best fitted line for test loss with an obvious generalization gap (red): -0.986

Figure 11: the relationship between the test loss and the sample size for the global network with L1 regularization.

Eq. (18) gives a theoretical prediction of the scaling of growth of number of samples with respect to the input dimension in order to maintain a predetermined level of test loss. We find this relation hard to verify directly because: 1) The input dimension, and the number of samples are usually chosen among finitely many values, so the grid may not be fine enough to find the (n, d) combination that gives the predetermined test loss; 2) The test loss varies even with same hyperparamter setting, so it is rather difficult to make sure that the test loss stays at a constant value especially given that the loss of the scaled function can be very small ($10^{-6} \sim 10^{-4}$). However, we can compute the numerical sample complexity if we assume that the generalization gap is in the function form $\frac{d^{\beta_1}}{n^{\beta_2}}$ as in Eq.(17). Recall that theoretical prediction gives $\beta_1 = 2$

because the loss for the original function needs to be scaled by d^2 as noted in Eq.(20), and $\beta_2 = 0.5$. Then the sample complexity with respect to d can be characterized by a rate γ as $n \sim \mathcal{O}(d^\gamma)$, where $\gamma = \frac{\beta_1}{\beta_2} = 4$.

The rate we obtained for the implicit regularization is $\beta_1 = 3.6$ as in Fig. 5(a), and $\beta_2 = 0.8$ as in Fig. 9(b), and thus the sample complexity rate is approximately $\gamma = 4.5$, i.e. $n \sim \mathcal{O}(d^{4.5})$. This is close to the theoretically predicted rate, but this largely benefited from the observation that $\beta_2 = 0.8 > 0.5$. The rate we obtained for the explicit regularization (L1 with regularization constant 10^{-8} to be precise) is $\beta_1 = 2.5$ as in Fig. 6(a), and $\beta_2 = 1.0$ as in Fig. 11, and thus the sample complexity rate is approximately $\gamma = 2.5$, i.e. $n \sim \mathcal{O}(d^{2.5})$, which is significantly better than the theoretically predicted rate. We do point out that this is a very rough estimate because of the assumed function form and the fact that the β_1 rate in the L1 regularization in Fig. 6(a) is not based on the generalization gap but the test loss (the generalization gap is very small). Overall, the explicit regularization helps improve the rate in both the test loss Vs. the input dimension, and the test loss Vs. the number of training samples, so the advantage of the explicit regularization is convincing.

5 Discussion

Despite the fact that an overparameterized neural network architecture can represent a large class of functions, such representation power can come at the cost of a large sample complexity. This is particularly relevant in many scientific machine learning applications, as the required accuracy (in the form of a regression problem) is high, and the training data can be difficult to obtain. Therefore a number of recent works have focused on domain-specific neural network architectures aiming at reducing the number of parameters, "retreating" from the overparameterized regime.

This paper gives an unambiguous, and minimal working example illustrating why this makes sense. Even for the seemingly simple task of computing the sum of squares of d numbers in a compact domain (i.e. learning the square of a 2-norm), a general purpose dense neural network struggles to find a highly accurate approximation of the function even for relatively low d (tens to hundreds). In particular, the sample complexity of an empirically optimally tuned and explicitly regularized dense network is $\mathcal{O}(d^{2.5})$. This behaves better than the sample complexity from *a priori* error bound, which is $\mathcal{O}(d^4)$. The origin of such improvement deserves study in the future. The sample complexity of an empirically optimally tuned and implicitly regularized dense network is close to $\mathcal{O}(d^{4.5})$, and hence can be prohibitively expensive as d becomes large.

When we choose a proper architecture, such as the local network or the locally connected network, the generalization error still grows with respect to d. However, we find that the generalization gap is nearly invisible with explicit or implicit regularization, and the test loss is orders of magnitude smaller than that of the global (dense) network. Given that the sample complexity can asymptotically scale as the square of the test loss (assuming training error is negligible), the practical savings due to the use of a local network is vital to the success of the neural network.

From a theoretical perspective, we remark that existing error analysis based on the Rademacher complexity-type cannot yet explain why the prefactor of the local network should be lower by orders of magnitude compared to the global (dense) network. Our results illustrate that implicit regularization as early stopping may not give the optimal generalization error rate in practice, although in theory it can be shown to achieve the same optimal rates as L^2 regularization in reproducing kernel Hilbert spaces [32, 31]. Given that implicit regularization is still a prevailing regularization method used in practical applications, better theoretical understanding of the behavior of early stopping regularization in neural networks, and methods to improve the performance of such an implicit regularization in practice are also needed.

Acknowledgments

This work was partially supported by the Department of Energy under Grant No. DE-SC0017867, the CAMERA program (L. L., J. Z., L. Z.-N.), and the Hong Kong Research Grant Council under Grant No. 16303817 (Y. Y.). We thank the Berkeley Research Computing (BRC) program at the University of California, Berkeley, and the Google Cloud Platform (GCP) for the computational resources. We thank Weinan E, Chao Ma, Lei Wu for pointing out the critical role of the path norm in understanding the numerical behavior of the generalization error, and thank Joan Bruna, Jiequn Han, Joonho Lee, Jianfeng Lu, Tengyu Ma, Lexing Ying for valuable discussions.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] S. Arora, S. S. Du, W. Hu, Z. Li, and R. Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks.

[3] F. Bach. Breaking the curse of dimensionality with convex neural networks. *J. Mach. Learn. Res.*, 18(1):629–681, 2017.

[4] A. R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Trans. Inform. Theory*, 39:930–945, 1993.

[5] F. Chollet et al. Keras. `https://keras.io`, 2015.

[6] N. Cohen, O. Sharir, and A. Shashua. On the expressive power of deep learning: A tensor analysis. In *Conference on Learning Theory*, pages 698–728, 2016.

[7] S. d'Ascoli, L. Sagun, J. Bruna, and G. Biroli. Finding the needle in the haystack with convolutions: on the benefits of architectural bias.

[8] W. E, C. Ma, and Q. Wang. A priori estimates of the population risk for residual networks. *arXiv:1903.02154*, 2019.

[9] W. E, C. Ma, and L. Wu. A Priori Estimates for Two-layer Neural Networks. *arXiv: 1810.06397*, pages 1–14, 2018.

[10] W. E, C. Ma, and L. Wu. Barron spaces and the compositional function spaces for neural network models. *arXiv:1906.08039*, 2019.

[11] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *International Conference on Learning Representations (ICLR)*, 2019. arXiv preprint arXiv:1803.03635.

[12] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin. The lottery ticket hypothesis at scale. *arXiv preprint arXiv:1903.01611*, 2019.

[13] C. D. Freeman and J. Bruna. Topology and geometry of half-rectified network optimization. In *ICLR*, 2017. arXiv preprint arXiv:1611.01540.

[14] K. He and J. Sun. Convolutional neural networks at constrained time cost. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5353–5360, 2015.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[16] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. r. 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 Processing Magazine*, 29(6):82–97, 2012.

[17] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

[18] K. Kawaguchi. Deep learning without poor local minima. In *arXiv preprint arXiv:1605.07110*, 2016.

[19] V. Khrulkov, A. Novikov, and I. Oseledets. Expressive power of recurrent neural networks. *arXiv:1711.00811*, 2017.

[20] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *The 3rd International Conference for Learning Representations (ICLR)*, 2015. arXiv:1412.6980v8.

[21] J. M. Klusowski and A. R. Barron. Risk bounds for high-dimensional ridge function combinations including neural networks.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[23] R. Kuditipudi, X. Wang, H. Lee, Y. Zhang, Z. Li, W. Hu, S. Arora, and R. Ge. Explaining landscape connectivity of low-cost solutions for multilayer net. In *NeurIPS*, 2019. arXiv preprint arXiv:161.

[24] M. K. K. Leung, H. Y. Xiong, L. J. Lee, and B. J. Frey. Deep learning of the tissue-regulated splicing code. *Bioinformatics*, 30(12):i121–i129, 2014.

[25] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. In *ICLR*, 2019. arXiv preprint arXiv:1810.05270.

[26] R. Livni, S. Shalev-Shwartz, and O. Shamir. On the computational efficiency of training neural networks. *Advances in Neural Information Processing Systems*, 2014. arXiv preprint arXiv:1410.1141.

[27] J. Ma, R. P. Sheridan, A. Liaw, G. E. Dahl, and V. Svetnik. Deep neural nets as a method for quantitative structure-activity relationships. *Journal of Chemical Information and Modeling*, 55(2):263–274, 2015.

[28] H. Mhaskar, Q. Liao, and T. Poggio. Learning functions: When is deep better than shallow. *arXiv preprint arXiv:1603.00988*, 2016.

[29] V. Nagarajan and J. Z. Kolter. Uniform convergence may be unable to explain generalization in deep learning. In *Advances in Neural Information Processing Systems*, pages 11611–11622, 2019.

[30] L. Venturi, A. S. Bandeira, and J. Bruna. Spurious valleys in two-layer neural network optimization landscapes. In *arXiv preprint arXiv:1802.06384*, 2018.

[31] Y. Wei, F. Yang, and M. J. Wainwright. Early stopping for kernel boosting algorithms: A general analysis with localized complexities. In *NIPS*, 2017. arXiv preprint arXiv:1707.01543.

[32] Y. Yao, L. Rosasco, and A. Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26:289–315, 08 2007.

[33] D. Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94:103–114, 2017.

[34] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. pages 3391–3401, 2017.

[35] L. Zhang, J. Han, H. Wang, R. Car, and W. E. DeePCG: constructing coarse-grained models via deep neural networks. *arXiv preprint arXiv:1802.08549*, 2018.