Tight Hardness Results for Training Depth-2 ReLU Networks*

Surbhi Goel
Microsoft Research, New York, NY, USA
goel.surbhi@microsoft.com

Adam Klivans
The University of Texas at Austin, TX, USA
klivans@cs.utexas.edu

Pasin Manurangsi
Google Research, Mountain View, CA, USA
pasin@google.com

Daniel Reichman
Worcester Polytechnic Institute, MA, USA
dreichman@wpi.edu

Abstract

We prove several hardness results for training depth-2 neural networks with the ReLU activation function; these networks are simply weighted sums (that may include negative coefficients) of ReLUs. Our goal is to output a depth-2 neural network that minimizes the square loss with respect to a given training set. We prove that this problem is NP-hard already for a network with a single ReLU. We also prove NP-hardness for outputting a weighted sum of k ReLUs minimizing the squared error (for $k > 1$) even in the realizable setting (i.e., when the labels are consistent with an unknown depth-2 ReLU network). We are also able to obtain lower bounds on the running time in terms of the desired additive error ϵ. To obtain our lower bounds, we use the Gap Exponential Time Hypothesis (Gap-ETH) as well as a new hypothesis regarding the hardness of approximating the well known Densest κ-Subgraph problem in subexponential time (these hypotheses are used separately in proving different lower bounds). For example, we prove that under reasonable hardness assumptions, any *proper* learning algorithm for finding the best fitting ReLU must run in time exponential in $1/\epsilon^2$. Together with a previous work regarding improperly learning a ReLU [21], this implies the first separation between proper and improper algorithms for learning a ReLU. We also study the problem of properly learning a depth-2 network of ReLUs with bounded weights giving new (worst-case) upper bounds on the running time needed to learn such networks both in the realizable and agnostic settings. Our upper bounds on the running time essentially matches our lower bounds in terms of the dependency on ϵ.

2012 ACM Subject Classification Theory of computation → Machine learning theory; Theory of computation → Problems, reductions and completeness

Keywords and phrases ReLU, Learning Algorithm, Running Time Lower Bound

Digital Object Identifier 10.4230/LIPIcs.ITCS.2021.22

Related Version A full version of the paper is available at `http://arxiv.org/abs/2011.13550`.

Funding *Surbhi Goel*: Work was done while the author was a PhD student at UT Austin and was supported by the JP Morgan AI Research PhD Fellowship.
Adam Klivans: Supported by NSF awards AF-1909204, AF-1717896, and the NSF AI Institute for Foundations of Machine Learning (IFML). Work done while visiting the Institute for Advanced Study, Princeton, NJ.

* This work subsumes our earlier manuscript [30].

Pasin Manurangsi: Part of this work was done while the author was at UC Berkeley and was partially supported by NSF under Grants No. CCF 1655215 and CCF 1815434.

Acknowledgements We would like to thank Amit Daniely, Amir Globerson, Meena Jagadeesan and Cameron Musco for interesting discussions.

1 Introduction

Neural networks have become popular in machine learning tasks arising in multiple applications such as computer vision, natural language processing, game playing and robotics [25]. One attractive feature of neural networks is being universal approximations: a network with a single hidden layer[1] with sufficiently many neurons can approximate arbitrary well any measurable real-valued function [23, 13]. These networks are typically trained on labeled data by setting the weights of the units to minimize the loss function (often the squared loss is used) over the training data. The challenge is to find a computationally efficient way to set the weights to achieve low error. While heuristics such as stochastic gradient descent (SGD) have been successful in practice, our theoretical understand about the amount of running-time needed to train neural networks is still lacking.

It has been known for decades [8, 31, 24] that finding a set of weights that minimizes the loss of the training set is NP-hard. These hardness results, however, only apply to classification problems and to settings where the neural networks involved use discrete, Boolean activations. Our focus here is on neural networks with real inputs whose neurons have the real-valued ReLU activation function. Specifically, we consider depth-2 networks of ReLUs, namely either a single ReLU or a weighted sum of ReLUs [2], and the optimization problem of training them giving labelled data points, which are defined below.

▶ **Definition 1.** *A* rectifier *is the real function* $[x]_+ := \max(0, x)$. *A* rectified linear unit (ReLU) *is a function* $f(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ *of the form* $f(\mathbf{x}) = [\langle \mathbf{w}, \mathbf{x} \rangle]_+$ *where* $\mathbf{w} \in \mathbb{R}^n$ *is fixed. A* depth-2 neural network with k ReLUs *(abbreviated as* k-ReLU*) is a function from* $\mathbb{R}^n$ *to* $\mathbb{R}$ *defined by*

$$\mathit{RELU}_{\mathbf{w}^1,\ldots,\mathbf{w}^k,\mathbf{a}}(\mathbf{z}) = \sum_{j=1}^{k} a_j [\langle \mathbf{w}^j, \mathbf{z} \rangle]_+.$$

Here $\mathbf{x} \in \mathbb{R}^n$ *is the input,* $\mathbf{a} = (a_1, \ldots, a_k) \in \{-1, 1\}^k$ *is a vector of "coefficients",* $\mathbf{w}^j = (w_1^j, \ldots, w_n^j) \in \mathbb{R}^n$ *is a weight vector associated with the* j*-th unit. When* $a_1 = \cdots = a_k = 1$*, we refer to* $\mathit{RELU}_{\mathbf{w}^1,\ldots,\mathbf{w}^k,\mathbf{a}}(\mathbf{z})$ *as the sum of* k *ReLUs, and we may omit* $\mathbf{a}$ *from the subscript.*

We note that the assumption that $a_1, \ldots, a_k \in \{+1, -1\}$ is without loss of generality (e.g., [32]): for any non-zero $a_1, \ldots, a_k \in \mathbb{R} \setminus \{0\}$ and $\mathbf{w}^1, \ldots, \mathbf{w}^k$, we may consider $\hat{a}_1 = \frac{a_1}{|a_1|}, \ldots, \hat{a}_k = \frac{a_k}{|a_k|}$ and $\hat{\mathbf{w}}^1 = |a_1|\mathbf{w}^1, \ldots, \hat{\mathbf{w}}^k = |a_k|\mathbf{w}^k$ instead, which represent the same depth-2 network of k ReLUs.

When training neural networks composed of ReLUs, a popular method is to find, given training data, a set of coefficients and weights for each gate minimizing the squared loss.

[1] We also refer to such networks as depth-2 networks or shallow networks.
[2] We also assume all the biases of the units are 0.

▶ **Definition 2.** *Given a set of m samples $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^n$ along with m labels $y_1, \ldots, y_m \in \mathbb{R}$, our goal is to find $\mathbf{w}^1, \ldots \mathbf{w}^k, \mathbf{a}$ which minimize the average squared training error of the sample, i.e.,*

$$\min_{\mathbf{w}^1,\ldots,\mathbf{w}^k,\mathbf{a}} \frac{1}{m} \sum_{i=1}^{m} (RELU_{\mathbf{w}^1,\ldots,\mathbf{w}^k,\mathbf{a}}(\mathbf{x}_i) - y_i)^2 \tag{1}$$

We refer to the optimization problem (1) as the k-ReLU training problem *(aka* k-ReLU regression*).*

When $\mathbf{w}^j = (w_1^j, \ldots, w_n^j)$ are assumed to have Euclidean norm at most 1 and y_i are assumed to be in $[-k, k]$, we refer to the optimization problem above as the bounded k-ReLU training problem*.*

Sometimes we assume that the "coefficient" vector $\mathbf{a}$ is fixed in advance (and known to the optimizer) and not part of the input to the training problem. We mention this explicitly when relevant. Also observe that in the optimization problem above we are looking for a **global minimum** rather than a local minimum. A multiset of samples $\{(\mathbf{x}_i, y_i)\}_{i\in[m]}$ is said to be *realizable* if there exist $\mathbf{w}^1, \cdots, \mathbf{w}^k, \mathbf{a}$ which result in zero training error.

Our goal is to pin down the computational complexity of the training problem for depth-2 networks of ReLUs, by answering the following question:

▶ **Question 3.** *What is the worst-case running time of training a k-ReLU?*

We focus on depth-2 networks which are rather involved and give rise to nontrivial algorithmic challenges [41, 6]. Understanding shallow networks seems to be a prerequisite for understanding the complexity of training networks of depth greater than 2.

1.1 Our results

We first consider arguably the simplest possible network: a single ReLU. We show that, already for such a network, the training problem is NP-hard. In fact, our result even rules out a large factor *multiplicative* approximation of the minimum squared error, as stated below.

▶ **Theorem 4** (Hardness of Training a single ReLU)**.** *The 1-ReLU training problem is NP-hard. Furthermore, given a sample of m data points of dimension n it is NP-hard to approximate the optimal squared error within a multiplicative factor of $(nm)^{1/poly\log\log(nm)}$.*

Given such a strong multiplicative inapproximability result, a natural question is whether one can get a good algorithm for *additive* approximation guarantee. Notice that we cannot hope for additive approximation in general, because scaling the samples and their labels can make the additive approximation gap arbitrarily large. Hence, we must consider the bounded 1-ReLU Training problem. For this, we give a simple $2^{O(1/\epsilon^2)}\text{poly}(n, m)$ time algorithm with additive approximation ϵ. Furthermore, it easily generalizes to the case of the bounded k-ReLU Training problem for $k > 1$, but we have to pay a factor of k^5 in the exponent:

▶ **Theorem 5** (Training Algorithm)**.** *There is a (randomized) algorithm that can solve the bounded k-ReLU training problem to within any additive error $\epsilon > 0$ in time $2^{O(k^5/\epsilon^2)} poly(n, m)$.*

Perhaps more surprisingly, we can prove a tight running time lower bound for the bounded 1-ReLU training problem, which shows that the term $1/\epsilon^2$ in the exponent is necessary. Our running time lower bound relies on the assumption that there is no subexponential

time algorithm for approximating the *Densest κ-Subgraph* problem within any constant (multiplicative) factor. Recall that, in the Densest κ-Subgraph (DκS) problem, we are given a graph $G = (V, E)$ and a positive integer κ. The goal is to select a subset $T \subseteq V$ of κ vertices that induces as many edges as possible. We use $\mathrm{den}_\kappa(G)$ to denote this optimum[3] and N to denote the number of vertices, $|V|$. Our hypothesis can be stated formally as follows.

▶ **Hypothesis 6.** *For every constant $C \geq 1$, there exist[4] $\delta = \delta(C) > 0$ and $d = d(C) \in \mathbb{N}$ such that the following holds. No $O(2^{\delta N})$-time algorithm can, given an instance (G, κ) of DκS where each vertex of G has degree at most d and an integer ℓ, distinguish between the following two cases:*

- *(Completeness)* $\mathrm{den}_\kappa(G) \geq \ell$.
- *(Soundness)* $\mathrm{den}_\kappa(G) < \ell/C$.

While this hypothesis is new (we are the first to introduce it), it seems fair to say that refuting it will require a breakthrough in current algorithms for the DκS problem. There are also other supporting evidences for the validity of this hypothesis. For example, it is known that $o(N)$-level of the Sum-of-Squares Hierarchies do not give constant factor approximation for DκS even for bounded degree graphs [7, 12, 28]. Furthermore, these Sum-of-Squares lower bounds are proved via reductions from a certain family of random CSPs, whose Sum-of-Squares lower bounds are shown in [35, 40]. This means that, if Hypothesis 6 is false, then one can refute this family of sparse random CSPs in subexponential time. This would constitute an arguably surprising development in the area of refuting random CSPs, which has been extensively studied for decades (see [1] and references therein).

As mentioned earlier, assuming Hypothesis 6, we can prove the tight running time lower bound for the bounded 1-ReLU Training problem:

▶ **Theorem 7** (Tight Running Time Lower Bound for 1-ReLU Training)**.** *Assuming Hypothesis 6, there is no algorithm that, for all given $\epsilon > 0$, can solve the bounded 1-ReLU training problem within an additive error ϵ in time $2^{o(1/\epsilon^2)}poly(n, m)$.*

We remark that, akin to standard conventions in the area of fine-grained and parameterized complexity, all lower bounds are stated against algorithms that work for *all* values of ϵ with the specified running time. Indeed, it is possible to significantly speed up the time bound $2^{O(1/\epsilon^2)}\mathrm{poly}(n, m)$ for extreme values of ϵ; for instance, enumerating all possible $\mathbf{w}$ over a $\Theta(\epsilon)$-net[5] of $\mathcal{B}^n$ gives an algorithm that runs in time $O(1/\epsilon)^{O(n)}\mathrm{poly}(m)$, which is asymptotically smaller than $2^{O(1/\epsilon^2)}\mathrm{poly}(n, m)$ when $\epsilon = o\left(\frac{1}{\sqrt{n \log n}}\right)$. Nonetheless, our lower bounds can be extended to include a large range of "reasonable" ϵ. Further discussion on such an extension is provided before Section 1.3.

An interesting consequence of Theorem 7 is that it gives a separation between proper and improper agnostic learning of 1-ReLU. Specifically, [21] shows that improper agnostic learning of 1-ReLU can be done in $2^{O(1/\epsilon)}\mathrm{poly}(n)$ time, while Theorem 7 rules out such a possibility for proper agnostic learning.

---

[3] Equivalently, $\mathrm{den}_\kappa(G) := \max_{T \subseteq V, |T| = \kappa} |E(T)|$.

[4] As C increases, δ and d decreases.

[5] Recall that an *δ-net* (also refer to as an *δ-cover*) of a set $S \subseteq \mathbb{R}^n$ is a set $T \subseteq \mathbb{R}^n$ such that, for every $x \in S$, there exists $y \in T$ where $\|x - y\|_2 \leq \delta$. It is well-known that, for any $\delta \in [0, 1]$, there is a δ-net of the unit ball $\mathcal{B}^n$ of size $(3/\delta)^n$ and that it can be found in $(3/\delta)^{O(n)}$ time.

Training k-ReLU: The Realizable Case

An important special case of the k-ReLU Training problem is the realizable case, where there is an unknown k-ReLU that labels every training sample correctly. When $k = 1$, it is straightforward to see that the realizable case of 1-ReLU Training can be phrased as a linear program and hence can be solved in polynomial time. On the other hand, we show that, once $k > 1$, the problem becomes NP-hard:

▶ **Theorem 8** (Hardness of Training k-ReLU in the Realizable Case). *For any constant $k \geq 2$, the k-ReLU training problem is NP-hard even in the realizable case.*

Our result is in fact slightly stronger than stated above: specifically, we show that, when the samples can be realizable by a (non-negative) sum of k ReLUs (i.e. k-ReLU when $\mathbf{a}$ is the all-one vector), it is still NP-hard to find a k-ReLU that realizes the samples even if negative coefficients in $\mathbf{a}$ are allowed. Furthermore, while we assume in this theorem that k is a constant independent of n, one can also prove an analogous hardness result, when k grows sufficiently slowly as a function of n. We refer the full version for more details.

Observe that Theorem 8 implies that efficient *multiplicative* approximation for the k-ReLU Training problem is impossible (assuming P$\neq$NP) for $k \geq 2$. As a result, we once again turn to additive approximation. On this front, we can improve the running time of the algorithm in Theorem 5 when we assume that the samples are realizable, as stated below.

▶ **Theorem 9** (Training Algorithm in the Realizable Case). *When the given samples are realizable by some k-ReLU, there is a (randomized) algorithm that can solve the bounded k-ReLU training problem to within any additive error $\epsilon > 0$ in time $2^{O((k^3/\epsilon)\log^3(k/\epsilon))}poly(n, m)$.*

Importantly, the dependency of ϵ in the exponent is $\tilde{O}(1/\epsilon)$, instead of $1/\epsilon^2$ that appeared in the non-realizable case (i.e. Theorems 5 and 7). We can also show that this dependency is tight (up to log factors), in the realizable case, under the Gap Exponential Time Hypothesis (Gap-ETH) [17, 29], a standard complexity theoretic assumption in parameterized complexity (see e.g. [11]). Gap-ETH states that there exists $\delta > 0$ such that no $2^{o(n)}$-time algorithm can, given a CNF formula with n Boolean variables, distinguish between (i) the case where the formula is satisfiable, and (ii) the case where any assignment violates at least δ fraction of the clauses. Our running time lower bound can be stated more formally as follows.

▶ **Theorem 10** (Tight Running Time Lower Bound for the Realizable Case). *Assuming Gap-ETH, for any constant $k \geq 2$, there is no algorithm that, for all given $\epsilon > 0$, can solve the bounded k-ReLU training problem within an additive error ϵ in time $2^{o(1/\epsilon)}poly(n, m)$ even when the input samples are realizable by some k-ReLU.*

Relation to Learning ReLUs

k-ReLU Training is closely related to the problem of *proper learning* of k-ReLU. In fact, an algorithm for the latter also solves the former. Hence, our hardness results immediately implies hardness of proper learning of k-ReLU as well. Furthermore, our algorithm also works for the learning problem. Please refer to the full version for more details.

Stronger Quantifier in Running Time Lower Bounds

As stated earlier, our running time lower bounds in Theorems 7 and 10 hold only against algorithms that work *for all* $\epsilon > 0$. A natural question is whether one can prove lower bounds against algorithms that work only *for some* "reasonable" values of ϵ. As explained in more detail below, we can quite easily also get a lower bound with this latter (stronger) quantifier, for any "reasonable" value of ϵ.

First, our lower bounds in Theorems 7 and 10 both apply in the regime where the lower bounds themselves are $2^{\Theta(n)}$; in other words, $\epsilon = \Theta(1/\sqrt{n})$ in Theorem 7 and $\epsilon = \Theta(1/n)$ in Theorem 10. These are essentially the smallest possible value of ϵ for which the lower bounds in Theorems 7 and 10 can hold, because the aforementioned algorithm that enumerates over an ϵ-net of $\mathcal{B}^n$ solves the problem in time $O(1/\epsilon)^{O(n)}\text{poly}(n)$. On the other hand, for smaller values of ϵ, we can get a running time lower bound easily by "padding" the dimension by "dummy" coordinates that are always zero. For instance, if we start with $\epsilon = \Theta(1/\sqrt{n})$, then we may pad the instance to say $n' = n^2$ dimensions, resulting in the relationship $\epsilon = \Theta(1/\sqrt[4]{n'})$. To summarize, this simple padding technique immediately gives the following stronger quantifier version of Theorem 7:

▶ **Theorem 11.** *For any non-increasing and efficiently computable*[6] *function* $\epsilon : \mathbb{N} \to \mathbb{R}^+$ *such that* $\omega(\sqrt{\log n}) \leq \frac{1}{\epsilon(n)} \leq o(\sqrt{n})$*, assuming Hypothesis 6, there is no algorithm that can solve the bounded 1-ReLU training problem within an additive error* $\epsilon(n)$ *in time* $2^{o(1/\epsilon(n)^2)}poly(n, m)$*.*

Notice that the constraint $\omega(\sqrt{\log n}) \leq \frac{1}{\epsilon(n)}$ is also essentially necessary, because for $\epsilon > \sqrt{\frac{\log\log n}{\log n}}$ our algorithm (Theorem 5) already runs in polynomial time. A strong quantifier version of Theorem 10 similar to above can be shown as well (but with $\omega(\log n) \leq \frac{1}{\epsilon(n)} \leq o(n)$). We omit the full (straightforward) proof via padding of Theorem 11; interested readers may refer to the proof of Lemma 3.4 of [16] which employs the same padding technique.

1.2 Independent and concurrent work

There have been several concurrent and independent works to ours that we mention here. We remark that the techniques in these works are markedly different than the ones in this paper. For a single ReLU, [14] proved that the 1-ReLU Training problem is NP-hard. With respect to two ReLUs, [6] showed that finding weights minimizing the squared error of a 2-ReLU is NP-hard, even in the realizable case. The work of [9] considered the problem of training a network with a slightly different architecture, in which there are two ReLUs in the first hidden layer and the final output gate is also a ReLU (instead of a sum gate as in our case); they showed that, for such networks with three ReLUs (two in the hidden layer, one in the output layer), the training problem is NP-hard even for the realizable case. As a result of having an output gate computing a ReLU, our NP-hardness result (regarding training a sum of two ReLUs) does not imply their result and their hardness result does not imply our hardness result for training a sum of 2 ReLUs.

1.3 Related work

The computational aspects of training and learning neural networks has been extensively studied. Due to this, we only focus on those directly related to our results.

We are not aware of a previous work showing that the general k-ReLU training problem is NP-hard for $k > 2$, nor are we aware of previous results regarding the hardness of *approximating* the squared error of a single ReLU. The $k = 2$ case and the $k > 2$ case seem to require different ideas and indeed our proof technique for Theorem 10 is different than those of [9, 6]. Moreover, the question of generalizing the NP-hardness result from $k = 2$ to $k > 2$ is mentioned explicitly in [9]. Finally, we remark that neither [14] nor [6] provides explicit

[6] That is, we assume that computing $\epsilon(n)$ can be done in time $\text{poly}(n)$ for any $n \in \mathbb{N}$.

running time lower bounds in terms of $1/\epsilon$ for the problem of training k ReLUs within an additive error of ϵ. To the best of our knowledge, our work is the first to obtain such lower bounds.

[42] has proven that finding weights minimizing the squared error of a k-ReLU is NP-hard when $\mathbf{a}$ is the all-one vector (or alternatively, when all the coefficients of the units are restricted to be positive) for every $k \geq 2$.

Some sources (e.g. [4, 5]) attribute (either implicitly or explicitly) the NP-hardness of the k-ReLU Training problem to [8], who consider training a neural network with *threshold units*. However, it is unclear (to us) how to derive the NP-hardness of training ReLUs from the hardness results of [8]. Several NP-hardness results for training neural networks with architectures differing from the fully connected architecture considered here are known. For example, in [10], the training problem is shown to be hard for a depth-2 *convolutional network* with (at least two) *non-overlapping* patches. To the best of our knowledge, these architectural differences render those previous results inapplicable for deriving the hardness results regarding the networks considered in this work.

Several papers have studied a slightly different setting of *improper learning* of neural networks. An example is [26] who show that improper learning of depth-2 networks of $\omega(1)$ ReLUs is hard, assuming certain average case assumptions. More recently, [21] show that even for a single ReLU, when $|\langle \mathbf{w}, \mathbf{x} \rangle|$ tends to infinity with n, learning $[\langle \mathbf{w}, \mathbf{x} \rangle]_+$ improperly in time $g(\epsilon) \cdot poly(n)$ is unlikely as it will result in an efficient algorithm for the problem of learning sparse parities with noise which is believed to be intractable. These hardness results for improper learning do imply hardness for the corresponding training problems. Nonetheless, it should be noted that the fact that these results have to rely on assumptions other than P $\neq$ NP is not a coincidence: it is known that basing hardness of improper learning on P $\neq$ NP alone will result in a collapse of the Polynomial Hierarchy [3].

On the algorithmic side, Arora et al. [4] provide a simple and elegant algorithm that exactly solves the ReLU training problem in polynomial time assuming the dimension n of the data points is an absolute constant; Arora et al.'s algorithm is for the networks we consider, and it has since been also extended to other types of networks [9]. Additionally, there have also been works on (agnostic) learning algorithms for ReLUs. Specifically, Goel et al. [21] consider the bounded norm setting where the inputs to the ReLUs as well as the weight vectors of the units have norms at most 1. For this setting, building on kernel methods and tools from approximation theory, they show how to *improperly* learn a single n-variable ReLU up to an additive error of ϵ in time $2^{O(1/\epsilon)} \cdot poly(n)$. Their result generalizes to depth-2 ReLUs with k units with running time of $2^{O(\sqrt{k}/\epsilon)} \cdot poly(n)$ assuming the coefficient vector $\mathbf{a}$ has norm at most 1. The algorithm they provide is quite general: it works for arbitrary distribution over input-output pairs, for ϵ that can be small as $1/\log n$ and also for the reliable setting.

A limitation of our hardness results is that they consider "pathologica" training data sets that are specifically constructed to encode intractable combinatorial optimization problems. Several works in literature have tried to overcome this issue by considering the training/learning problems on more "benign" data distributions, such as log-concave distributions or those with Gaussian marginals. On this front, both algorithms and lower bounds have been shown for depth-2 networks [38, 6, 22].

Using insights from the study of exponential time algorithms towards understanding the complexity of machine learning problems as is done in this work is receiving attention lately [36, 15, 37].

1.4 Organization of the Paper

In the remainder of the main body of this paper, we provide high-level overviews of our proofs (Section 2) and discuss several potential research directions (Section 3). Due to space constraints, all proofs are deferred to the full version.

2 Proof Overview

Below we provide the informal overviews of our proofs and intuition behind them.

NP-Hardness of Training 1-ReLU

Our reduction is from the (NP-hard) Set Cover problem, in which we are given subsets $T_1, \ldots, T_M$ of a universe U, and the goal is to select as few of these subsets as possible whose union covers the entire universe U. We reduce this to the problem of 1-ReLU Training, where the dimension n is equal to M. We think of each coordinate of $\mathbf{w}$ as an unknown (i.e. variable); specifically, the desired solution will have $w_i = -1$ iff T_i is picked and 0 otherwise. From this perspective, adding a labelled sample $(\mathbf{x}, y)$ is the same as adding a "constraint" $[\mathbf{w} \cdot \mathbf{x}]_+ = y$. There are two types of constraints we will add:

- (Element Constraint) For each $u \in U$, we add a constraint of the form $\left[1 + \sum_{T_i \ni u} w_i\right]_+ = 0$. The point is that such a constraint is satisfied when u is covered by the selected subsets.
- (Subset Constraint) For each $i \in [M]$, we add a constraint of the form $[\gamma + w_i]_+ = \gamma$ for some small $\gamma > 0$. This constraint will be violated for any selected subset.

By balancing the weights (i.e. number of copies) of each constraint carefully, we can ensure that the element constriants are never unsatisfied, and that the goal is ultimately to violate as few subset constraints as possible, which is equivalent to trying to pick as few subsets as possible that can fully cover U. This completes the high-level overview of our reduction.

We remark that there is a subtle point here because we cannot directly have a constant such that 1 or γ in the constraints themselves. Rather, we need to have "constraint coordinate" and adding the constants through this coordinate. This will also be done in the other reductions presented below, and we will not mention this again.

The outlined proof, together with the $\Theta(\log |U|)$ inapproximability of Set Cover [27, 20], already gives a hardness of approximation of a multiplicative factor of $\Theta(\log(nm))$ for the 1-ReLU Training problem. To further improve this inapproximability ratio to $(nm)^{1/\text{poly}\log\log(nm)}$, we reduce from the *Minimum Monotone Circuit Satisfiability (MMCS)* problem, which is a generalization of Set Cover. In MMCS, we are given a monotone circuit and the goal is to set as few input wires to true as possible under the condition that the circuit's output must be true. Strong inapproximability results for MMCS are known (e.g. [18]). Our reduction from MMCS proceeds in a similar manner as that of the Set Cover reduction above. Roughly speaking, the modification is that each unknown is now whether each wire is set/evaluated to true, whereas the constraints are now to ensure that the evaluation at each gate is correct and that the output is true.

Tight Running Time Hardness of 1-ReLU Training

We now move on to the proof overview of the tight running time lower bound for 1-ReLU Training. Recall that we will be reducing from the Densest κ-Subgraph (DκS) problem, in which we are given a graph $G = (V, E)$ and $\kappa \in \mathbb{N}$. The goal is to find a set of κ vertices that induces the maximum number of edges.

To motivate our construction, a simple combination of dimensionality reduction and δ-net can in fact find a ReLU that point-wise approximates the optimal ReLU to within an additive factor of δ in time $2^{\tilde{O}(1/\delta^2)}\text{poly}(n)$. That is, if the ReLU that achieves the optimal error has weight vector $\mathbf{w}^*$, then we can find a weight vector $\mathbf{w}$ such that $|[\mathbf{w}\cdot\mathbf{x}]_+ - [\mathbf{w}^*\cdot\mathbf{x}]_+| \le \delta$ for all input samples $(\mathbf{x}, y)$ in time[7] $2^{\tilde{O}(1/\delta^2)}\text{poly}(n)$.

Indeed, this is an explanation why, in the realizable case, we can get ϵ squared error in $2^{\tilde{O}(1/\delta)}\text{poly}(n)$ time by simply picking $\delta = \sqrt{\epsilon}$. Now, since we need our hardness here (for the non-realizable case) to hold with stronger running time lower bound of $2^{\Theta(1/\epsilon^2)}\text{poly}(n)$, we have to make sure that whenever $\delta \gg \epsilon$, the aforementioned point-wise approximation of δ is not sufficient to get an error of ϵ. Suppose that, for an input labelled sample $(\mathbf{x}, y)$, the optimal ReLU outputs y' and our approximation outputs y'' (where $|y'' - y'| \le \delta$). Notice that the difference in the square error between the two for this sample is only at most $O((y'-y)\delta) + \delta^2$. Now, if we want this quantity to be at least ϵ for any $\delta \ge \Omega(\epsilon)$, then it must be that $|y' - y| = \Omega(1)$. In other words, we have to make our samples so that even the optimal ReLU is "wrong" by $\Omega(1)$ additive factor (on average); this indeed means that, if the ReLU we find is "more wrong" by an additive factor of $\Theta(\epsilon)$, then the increase in the average squared error would be $\Omega(\epsilon)$ as desired.

With the observation in the previous paragraph in mind, we will now provide a rough description of our gadget. Given a DκS instance $(G = (V, E), \kappa)$, our samples will have $|V|$ dimensions, one corresponding to each vertex. In the YES case where there is $T \subseteq V$ of size κ that induces many edges, we aim to have our ReLU weight assigning $\frac{1}{\sqrt{\kappa}}$ to all coordinates corresponding to vertices in T, and zero to all other coordinates. To enforce this, we first add a sample for every vertex $v \in V$ that corresponds to the constraint

$$\left[\mathbf{w}_v - \frac{1}{2\sqrt{\kappa}}\right]_+ = 1.$$

We refer to these as the *cardinality constraints*. While this may look peculiar at first glance, the effect is that it ensures that roughly speaking $\mathbf{w}$ has κ coordinates that are "approximately" $\frac{1}{\sqrt{\kappa}}$ and the remaining coordinates are "small". To see that this is the case, observe that the average mean squared error here is $1 - \frac{2}{|V|}\sum_{v\in V}\left[\mathbf{w}_v - \frac{1}{2\sqrt{\kappa}}\right]_+ + \frac{1}{|V|}\sum_{v\in V}\left[\mathbf{w}_v - \frac{1}{2\sqrt{\kappa}}\right]_+^2$. The last term is small and may be neglected. Hence, we essentially have to maximize $\sum_{v\in V}\left[\mathbf{w}_v - \frac{1}{2\sqrt{\kappa}}\right]_+$. This term is indeed maximized when $\mathbf{w}$ has κ coordinates equal to $\frac{1}{\sqrt{\kappa}}$, and zeros in the remaining coordinates. Notice here that this also fits with our intuition from the previous paragraph: even in the optimal ReLU, the value out put by the value (which is either 0 or $\frac{1}{2\sqrt{\kappa}}$) is $\Omega(1)$ away from the input label of the sample (i.e. 1).

So far, the cardinality constraints have ensured that $\mathbf{w}$ "represents" a set $T \subseteq V$ of size roughly κ. However, we have not used the fact that T contains many edges at all. Thus, for every edge $e = \{u, v\} \in E$, we also add the example corresponding to the following constraint to our distribution:

$$\frac{1}{2}\left[\mathbf{w}_u + \mathbf{w}_v - \frac{1.75}{\sqrt{\kappa}}\right]_+ = 1.$$

We call these the *edge constraints*. The point here is that, if e is not an induced edge in T, then the output of the ReLU will be zero. On the other hand, if e is an edge in T, then the output of the ReLU will be $\frac{0.25}{\sqrt{\kappa}}$. Hence, the more edges T induces, the smaller the error.

[7] We assume throughout that $m = \text{poly}(1/\delta)$, which is w.l.o.g. due to standard generalization bounds.

By carefully selecting weights (i.e. number of copies) of each sample, one can indeed show that the average square error incurred in the completeness and soundness case of Hypothesis 6 differs by $\epsilon = \Omega\left(\frac{1}{\sqrt{|V|}}\right)$. Hence, if we can solve the 1-ReLU Training problem to within an additive error of ϵ in time $2^{o(1/\epsilon^2)}\text{poly}(n,m)$, we can also solve the problem in Hypothesis 6 in time $2^{o(|V|)}$, which breaks the hypothesis.

Hardness of Training k-ReLU in the Realizable Case

We next consider the problems of Training k-ReLU for $k \geq 2$ in the realizable case. Both the NP-hardness result (Theorem 8) and the tight running time lower bound (Theorem 10) employ similar reductions. These reductions proceed in two steps. First is to reduce from the NP-hard k-coloring problem to the problem of training *non-negative sum of k ReLUs*, in which we fix the coefficient vector $\mathbf{a}$ to be the all-one vector and only seeks to find $\mathbf{w}^1, \ldots, \mathbf{w}^k$ that minimizes the squared error. Then, in the second step, we reduce this to the original problem of k-ReLU Training (where the coefficient vector $\mathbf{a}$ can be negative).

Step I: From k-Coloring to Training Sum of k ReLUs. The NP-hardness of Sum of k ReLUs Training in fact follows directly from a reduction of [42]. We will now sketch Vu's reduction, since it will be helpful in our subsequent discussions below. Vu's reduction starts from the k-coloring problem, in which we are given a hypergraph $G = (V, E)$ and the goal is to determine whether there is a proper k-coloring[8] of the hypergraph. Given an instance $G = (V, E)$ of k-coloring, the number of dimensions in the training problem will be $n = |V|$ where we associate each dimension with a vertex. Notice that now we have k unknowns associated to each vertex v: $w_v^1, \ldots, w_v^k$. In the desired solution, these variables will tell us which color v is assigned to: specifically, $w_v^i > 0$ iff v is colored i and $w_v^i \leq 0$ otherwise.

Adding a labelled sample $(\mathbf{x}, y)$ is the same as adding a "constraint" $[\mathbf{w}^1 \cdot \mathbf{x}]_+ + \cdots + [\mathbf{w}^k \cdot \mathbf{x}]_+ = y$. There are two types of constraints we will add:

- (Vertex Constraint) For every vertex $v \in V$, we add a constraint[9] $[w_v^1]_+ + \cdots + [w_v^k]_+ = 1$. This constraint ensures that, for every $v \in V$, we must have $w_v^{i_v} > 0$ for at least one $i_v \in [k]$, meaning that the vertex v is assigned at least one color.
- (Hyperedge Constraint) For every hyperedge $e = \{v_1, \ldots, v_\ell\} \in E$, we add a constraint[10] $[w_{v_1}^1 + \cdots + w_{v_\ell}^1]_+ + \cdots + [w_{v_1}^k + \cdots + w_{v_\ell}^k]_+ = 0$. This ensures that the hyperedge e is not monochromatic. Otherwise, we have $i_{v_1} = \cdots = i_{v_\ell}$ meaning that $w_{v_1}^{i_{v_1}} + \cdots + w_{v_\ell}^{i_{v_1}} > 0$, which violates the hyperedge constraint.

This finishes our summary of Vu's reduction, which gives the NP-hardness of training a (non-negative) sum of k ReLUs.

Step II: Handling Negative Coefficients. The argument above, especially for the hyperedge constraints, relies on the fact that the coefficient vector $\mathbf{a}$ is the all-one vector. In other words, even if the input hypergraph is not k-coloring, it is still possible that there is a k-ReLU (possibly negative weight vector $\mathbf{a}$) that realizes the samples. Hence, the reduction above does not yet work for our original problem of k-ReLU Training. To handle this issue, we use an additional gadget which is simply a set of labelled samples with the following properties: these samples can be realized by a k-ReLU only when the weight vectors $\mathbf{a}$ is the

[8] A proper k-coloring is a mapping $\chi : V \to [k]$ such that no hyperedge is monochromatic, or equivalently $|\chi(e)| > 1$ for all $e \in E$.
[9] This constraint corresponds to $\mathbf{x}$ being the v-th vector in the standard basis and $y = 1$.
[10] This constraint corresponds to $\mathbf{x}$ being the indicator vector of e and $y = 0$.

all-one vector. Essentially speaking, by adding these samples also to our sample set, we have forced $\mathbf{a}$ to be the all-one vector, at which point we restrict ourselves back to the case of (non-negative) sum of k ReLUs and we can use the hard instance from the above reduction from k-coloring. These are the main ideas of the proof of Theorem 8.

Tight Running Time Lower Bound. As stated earlier, the tight running time lower bound for the bounded k-ReLU Training problem (Theorem 10) follows from a similar reduction, except that we now have to (1) carefully select the number of copies of each sample and (2) scale the labels y_i's down so that the norm of each of $\mathbf{w}^1, \ldots, \mathbf{w}^k$ is at most one. Roughly speaking, this means that the labels for the vertex constraints become $\Theta(1/\sqrt{|V|})$ instead of 1 as before. In other words, each violated constraint roughly contributes to $\Theta(1/|V|)$ squared error. Since it is known (assuming Gap-ETH) that distinguishing between a k-colorable hypergraph and a hypergraph for which every k-coloring violates a constant fraction of the edges takes $2^{\Omega(|V|)}$ time (e.g. [33]), we can arrive at the conclusion that solving the bounded k-ReLU Training problem to within an additive squared error of $\epsilon = \Theta(1/|V|)$ must take $2^{\Omega(1/|V|)} = 2^{\Omega(1/\epsilon)}$ time as desired.

We remark here that, interestingly, [42] used the reduction from k-coloring only for the case of $k = 2$ units and employed an additional gadget to handle the case $k > 2$. To the best of our knowledge, this approach seems to decrease the resulting error ϵ, which means that the running time lower bound is not of the form $2^{\Omega(1/\epsilon)}$. On the other hand, we argue the hardness directly from k-coloring for any constant $k \geq 2$. This, together with a careful selection of the number of copies of each sample, allows us to achieve the running time lower bound in Theorem 10.

Training and Learning Algorithms

Our k-ReLU training algorithm is based on the approach of [4]. The main idea behind the algorithm is to iterate over all possible sign patterns (whether each ReLU is active or not) of the inputs and subsequently solve the so formed convex optimization for each fixed pattern. The best hypothesis over all different sign patterns is chosen as the the final hypothesis. It is not hard to see that the run-time for such an algorithm would be $2^{(m+1)k}poly(n)$ since there are 2^{mk} different sign patterns.

Using standard generalization bounds, one can show that the number of samples m needed for the empirical loss to be ϵ close to the true loss is at most $O(k^4/\epsilon^2)$. Plugging this into the above algorithm gets us the desired running time ($2^{O(k^5/\epsilon^2)}poly(n, m)$ as in Theorem 5) for the agnostic setting. For the realizable setting, we use an improved generalization result of [39], which implies that $m = \tilde{O}(k^2/\epsilon)$ suffices; plugging this into the above algorithm yields us Theorem 9.

3 Conclusions and Open Questions

We have studied the computational complexity of training depth-2 networks with the ReLU activation function providing both NP-hardness results and algorithms for training ReLU's. Along the way we have introduced and used a new hypothesis regarding the hardness of approximating the Denset κ-Subgraph problem in subexponential time that may find applications in other settings. Our results provide a separation between proper and improper learning showing that for a single ReLU, proper learning is likely to be harder than improper learning. Our hardness results regarding properly learning shallow networks suggest that improperly learning such networks (for example, learning overparametrized networks whose number of units far exceeds the dimension of the labeled vectors [2, 19]) might be necessary to allow for tractable learning problems.

We stress here that our hardness results apply to minimizing the population loss[11] as well, since one may simply create an instance where the population is just the training data. Furthermore, the standard procedure for training neural networks is to perform ERM which is essentially minimizing the training loss. In fact, a bulk of theoretical work in the field focuses on generalization error assuming training error is small (often 0). Therefore, we believe it is a natural question to study the hardness of minimizing training loss.

Neural networks offer many choices (e.g., number of units, depth, choice of activation function, weight restrictions). Indicating which architectures are NP-hard to train can prove useful in guiding the search for a mathematical model of networks that can be trained efficiently. It should be remembered that our NP-hardness results are worst-case. Therefor they do not preclude efficient algorithms under additional distributional or structural assumptions [34]. Finally, as we focus on networks having significantly fewer units than data-points, the NP-hardness results reported here are not at odds with the ability to train neural networks in the overparmeterized regime where there are polynomial time algorithms that can fit the data with zero error [43].

While we restrict our attention to algorithms for training networks with bounded weights, our exponential dependency of the running time on k (the number of units) makes these algorithms impractical. It remains an interesting question whether the dependency of the running time on k can be improved, or alternatively whether strong running time lower bounds can be shown in terms of k (similar to what is done for ϵ in this work).

While we have focused on depth-2 networks, algorithms and lower bounds for deeper networks are of interest as well, especially given the multitude of their practical applications. It would be interesting to see whether the algorithms and hardness results extend to the setting of depth greater than 1. An interesting concrete question here is whether training/learning becomes harder as the network becomes deeper. For instance, is it possible to prove running time lower bounds that grow with the depth of the network?

References

1 Sarah R. Allen, Ryan O'Donnell, and David Witmer. How to refute a random CSP. In *FOCS*, pages 689–708, 2015. doi:10.1109/FOCS.2015.48.

2 Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. In *Advances in neural information processing systems*, pages 6155–6166, 2019.

3 Benny Applebaum, Boaz Barak, and David Xiao. On basing lower-bounds for learning on worst-case assumptions. In *FOCS*, pages 211–220, 2008. doi:10.1109/FOCS.2008.35.

4 Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units. In *International Conference on Learning Representations*, 2018. URL: https://openreview.net/forum?id=B1J_rgWRW.

5 Francis Bach. Breaking the curse of dimensionality with convex neural networks. *The Journal of Machine Learning Research*, 18(1):629–681, 2017.

6 Ainesh Bakshi, Rajesh Jayaram, and David P Woodruff. Learning two layer rectified neural networks in polynomial time. In *Conference on Learning Theory*, pages 195–268, 2019.

7 Aditya Bhaskara, Moses Charikar, Aravindan Vijayaraghavan, Venkatesan Guruswami, and Yuan Zhou. Polynomial integrality gaps for strong SDP relaxations of densest k-subgraph. In *SODA*, pages 388–405, 2012.

8 Avrim Blum and Ronald L Rivest. Training a 3-node neural network is NP-complete. In *Advances in neural information processing systems*, pages 494–501, 1989.

[11] The population loss is the expected square loss with respect to the distribution of data points.

9 Digvijay Boob, Santanu S Dey, and Guanghui Lan. Complexity of training relu neural network. *arXiv preprint*, 2018. arXiv:1809.10787.

10 Alon Brutzkus and Amir Globerson. Globally optimal gradient descent for a convnet with gaussian inputs. *arXiv preprint*, 2017. arXiv:1702.07966.

11 Parinya Chalermsook, Marek Cygan, Guy Kortsarz, Bundit Laekhanukit, Pasin Manurangsi, Danupon Nanongkai, and Luca Trevisan. From Gap-ETH to FPT-inapproximability: Clique, dominating set, and more. In *FOCS*, pages 743–754, 2017. doi:10.1109/FOCS.2017.74.

12 Eden Chlamtác, Pasin Manurangsi, Dana Moshkovitz, and Aravindan Vijayaraghavan. Approximation algorithms for label cover and the log-density threshold. In *SODA*, pages 900–919, 2017. doi:10.1137/1.9781611974782.57.

13 George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

14 Santanu S Dey, Guanyi Wang, and Yao Xie. An approximation algorithm for training one-node relu neural network. *arXiv preprint*, 2018. arXiv:1810.03592.

15 Ilias Diakonikolas, Daniel Kane, and Pasin Manurangsi. Nearly tight bounds for robust proper learning of halfspaces with a margin. In *Advances in Neural Information Processing Systems*, pages 10473–10484, 2019.

16 Ilias Diakonikolas, Daniel M. Kane, and Pasin Manurangsi. Nearly tight bounds for robust proper learning of halfspaces with a margin. *CoRR*, abs/1908.11335, 2019. arXiv:1908.11335.

17 Irit Dinur. Mildly exponential reduction from gap 3SAT to polynomial-gap label-cover. *Electronic Colloquium on Computational Complexity (ECCC)*, 23:128, 2016. URL: http://eccc.hpi-web.de/report/2016/128.

18 Irit Dinur, Prahladh Harsha, and Guy Kindler. Polynomially low error PCPs with polyloglog n queries via modular composition. In *STOC*, pages 267–276, 2015. doi:10.1145/2746539.2746630.

19 Simon S Du, Jason D Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. *arXiv preprint*, 2018. arXiv:1811.03804.

20 Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998. doi:10.1145/285055.285059.

21 Surbhi Goel, Varun Kanade, Adam R. Klivans, and Justin Thaler. Reliably learning the relu in polynomial time. In *COLT*, pages 1004–1042, 2017. URL: http://proceedings.mlr.press/v65/goel17a.html.

22 Surbhi Goel, Sushrut Karmalkar, and Adam Klivans. Time/accuracy tradeoffs for learning a relu with respect to gaussian marginals. In *Advances in Neural Information Processing Systems*, pages 8582–8591, 2019.

23 Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

24 Stephen Judd. On the complexity of loading shallow neural networks. *Journal of Complexity*, 4:177–192, 1988.

25 Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

26 Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. In *Advances in Neural Information Processing Systems*, pages 855–863, 2014.

27 Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, 1994. doi:10.1145/185675.306789.

28 Pasin Manurangsi. On approximating projection games. Master's thesis, Massachusetts Institute of Technology, January 2015.

29 Pasin Manurangsi and Prasad Raghavendra. A birthday repetition theorem and complexity of approximating dense csps. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 78:1–78:15, 2017. doi:10.4230/LIPIcs.ICALP.2017.78.

30 Pasin Manurangsi and Daniel Reichman. The computational complexity of training relu(s). *CoRR*, abs/1810.04207, 2018. arXiv:1810.04207.

31 Nimrod Megiddo. On the complexity of polyhedral separability. *Discrete & Computational Geometry*, 3(4):325–337, 1988.

32 Xingyuan Pan and Vivek Srikumar. Expressiveness of rectifier networks. In *International Conference on Machine Learning*, pages 2427–2435, 2016.

33 Erez Petrank. The hardness of approximation: Gap location. *Computational Complexity*, 4:133–157, 1994. doi:10.1007/BF01202286.

34 Tim Roughgarden. Beyond the worst-case analysis of algorithms, 2020.

35 Grant Schoenebeck. Linear level lasserre lower bounds for certain k-CSPs. In *FOCS*, pages 593–602, 2008. doi:10.1109/FOCS.2008.74.

36 Rocco A Servedio and Li-Yang Tan. What circuit classes can be learned with non-trivial savings? In *8th Innovations in Theoretical Computer Science Conference (ITCS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

37 Kirill Simonov, Fedor Fomin, Petr Golovach, and Fahad Panolan. Refined complexity of PCA with outliers. In *International Conference on Machine Learning*, pages 5818–5826, 2019.

38 Le Song, Santosh Vempala, John Wilmes, and Bo Xie. On the complexity of learning neural networks. In *Advances in Neural Information Processing Systems*, pages 5514–5522, 2017.

39 Nathan Srebro, Karthik Sridharan, and Ambuj Tewari. Smoothness, low noise and fast rates. In John D. Lafferty, Christopher K. I. Williams, John Shawe-Taylor, Richard S. Zemel, and Aron Culotta, editors, *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada*, pages 2199–2207. Curran Associates, Inc., 2010. URL: http://papers.nips.cc/paper/3894-smoothness-low-noise-and-fast-rates.

40 Madhur Tulsiani. CSP gaps and reductions in the lasserre hierarchy. In *STOC*, pages 303–312, 2009. doi:10.1145/1536414.1536457.

41 Santosh Vempala and John Wilmes. Polynomial convergence of gradient descent for training one-hidden-layer neural networks. In *Conference on Learning Theory*, pages 3115–3117, 2019.

42 Van H Vu. On the infeasibility of training neural networks with small mean-squared error. *IEEE Transactions on Information Theory*, 44(7):2892–2900, 1998.

43 Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint*, 2016. arXiv:1611.03530.