When Expressivity Meets Trainability: Fewer than n Neurons Can Work

Jiawei Zhang[*]
Shenzhen Research Institute of Big Data
The Chinese University of Hong Kong,
Shenzhen, China
jiaweizhang2@link.cuhk.edu.cn

Yushun Zhang[*]
Shenzhen Research Institute of Big Data
The Chinese University of Hong Kong,
Shenzhen, China
yushunzhang@link.cuhk.edu.cn

Mingyi Hong
University of Minnesota - Twin Citie
MN, USA
mhong@umn.edu

Ruoyu Sun [†]
University of Illinois at Urbana-Champaign
IL, USA
ruoyus@illinois.edu

Zhi-Quan Luo
Shenzhen Research Institute of Big Data
The Chinese University of Hong Kong,
Shenzhen, China
luozq@cuhk.edu.cn

Abstract

Modern neural networks are often quite wide, causing large memory and computation costs. It is thus of great interest to train a narrower network. However, training narrow neural nets remains a challenging task. We ask two theoretical questions: Can narrow networks have as strong expressivity as wide ones? If so, does the loss function exhibit a benign optimization landscape? In this work, we provide partially affirmative answers to both questions for 1-hidden-layer networks with fewer than n (sample size) neurons. First, we prove that as long as the width $m \geq \frac{2n}{d}$ (where d is the input dimension), its expressivity is strong, i.e., there exists at least one global minimizer with zero training loss. Second, we identify a nice local region with no local-min or saddle points. Nevertheless, it is not clear whether gradient descent can stay in this nice region. Third, we consider a constrained optimization formulation where the feasible region is the local nice region, and prove that every KKT point is a nearly global minimizer. It is expected that projected gradient methods converge to KKT points under mild technical conditions, but we leave the rigorous convergence analysis to future work. Thorough numerical results show that projected gradient methods on this constrained formulation significantly outperform SGD for training narrow neural nets.

1 Introduction

Modern neural networks are huge (e.g. [8, 74]). Reducing the size of neural nets is appealing for many reasons: first, small networks are more suitable for embedded systems and portable devices; second, using smaller networks can reduce power consumption, contributing to "green computing".

[*]Equal contribution. These authors are listed in alphabetical order.
[†]Corresponding author: Ruoyu Sun.

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

There are many ways to reduce network size, such as quantization, sparcification and reducing the width (e.g. [20, 77]). In this work, we focus on reducing the width (training narrow nets).

Reducing the network width often leads to significantly worse performance [3]. What is the possible cause? From the theoretical perspective, there are three possible causes: worse generalization power, worse trainability (how effective a network can be optimized), and weaker expressivity (how complex the function a network can represent; see Definition 1). Our simulation shows that the training error deteriorates significantly as the width shrinks, which implies that the trainability and/or expressivity are important causes of the worse performance (see Section 5.1 for more evidence). We do not discuss generalization power for now, and leave it to future work.

So how is the training error related to expressivity and trainability? The training error is the sum of two parts (see, e.g., [64]): the expressive error (which is the best a given network can do; also the global minimal error) and the optimization error (which is the gap between training error and the global minimal error; occurs because the algorithm may not find global-min). The two errors are of different nature, and thus need to be discussed separately.

It is understandable that narrower networks might have weaker expressive power. What about optimization? There is also evidence that smaller width causes optimization difficulty. A number of recent works show that increasing the width of neural networks helps create a benign empirical loss landscape ([19, 35, 59]), while narrow networks (width $m <$ sample size n) suffer from bad landscape ([2, 60, 66, 72, 79]). Therefore, if we want to improve the performance of narrow networks, it is likely that both expressiveness and trainability need to be improved.

The above discussion leads to the following two questions:

(Q1) *Can a narrow network have as strong* **expressivity** *as a wide one?*

(Q2) *If so, can a local search method find a (near) globally optimal solution?*

The key challenges in answering these questions are listed below:

- It is not clear whether a narrow network has strong expressivity or not. Many existing works focus on verifying the relationship between zero-training-error solutions and stationary points, but they neglect the (non)existence of such solutions (e.g. [71], [63]). For narrow networks, the (non)-existence of zero-training-error-solution is not clear.
- Even if zero-training-error solutions do exist, it is still not clear how to reach those solutions because the landscape of a narrow neural network can be highly non-convex.
- Even assuming that we can identify a region that contains zero-training-error solutions and has a good landscape, it is potentially difficult to keep the iterates inside such a good region. One may think of imposing an explicit constraint, but this approach might introduce bad local minimizers on the boundary [6].

In this work, we (partially) answer **(Q1)** and **(Q2)** for a 1-hidden-layer nets with fewer than n neurons. Our main contributions are as follows:

- **Expressiveness and nice local landscape.** We prove that, as long as the width is larger than $m \geq \frac{2n}{d}$, then the expressivity of the 1-hidden-layer net is strong, i.e., w.p.1. there *exists* at least one global-min with zero empirical loss. In addition, such a solution is surrounded by a good local landscape with no local-min or saddles. Note that our results do not exclude the possibility that there are sub-optimal local minimizers on the *global* landscape.
- **Every KKT point is an approximated global minimizer.** For the original unconstrained optimization problem, the nice *local* landscape does not guarantee the *global* statement of "every stationary point is a global minimizer". We propose a constrained optimization problem that restricts the hidden weights to be close to the identified nice region. We show

[3] This can be verified on our empirical studies in Section 5. Another evidence is that structure pruning (reducing the number of channels in convolutional neural nets (CNN)) is known to achieve worse performance than unstructured pruning; this is an undesirable situation since many practitioners prefer structure pruning (due to hardware reasons).

that every Karush–Kuhn–Tucker (KKT) point is an approximated global minimizer of both the constrained problem and the unconstrained training problem [4].

- In real-data experiments, our proposed training regime can significantly outperforms SGD for training narrow networks. We also perform ablation studies to show that the new elements proposed in our method are useful.

2 Background and Related Works

The expressivity of neural networks has been a popular topic in machine learning for decades. There are two lines of works: One focuses on the *infinite-sample* expressivity, showing what functions of the entire domain can and cannot be represented by certain classes of neural networks (e.g. [4, 45]). Another line of works characterize the *finite-sample* expressivity, i.e. how many parameters are required to memorize finitely samples (e.g. [5, 13, 21, 25, 26, 42, 75]). The term "expressivity" in this work means the finite-sample expressivity; see Definition 1 in Section 4.1. A major research question regarding expressivity in the area is to show *deep* neural networks have much stronger expressivity than the *shallow* ones (e.g. [7, 17, 41, 48, 56, 57, 67, 70, 72]). However, all these works neglect the trainability.

In the finite-sample case, wide networks (width poly(n)) have both strong representation power (i.e. the globally minimal training error is zero) and strong trainability (e.g. for wide enough nets, Gradient Descent (GD) converges to global minima [1, 16, 28, 80]). While these wide networks are often called "over-parameterized", we notice that the number of parameters of a width-n network is actually at least nd, which is much larger than n. The transition from under-parameterization and over-parameterization for a one-hidden-layer fully-connected net (FCN) does not occur at width-n, but at width-n/d. In this work, we will analyze networks with width in the range $[n/d, n)$, which we call "narrow networks" (though rigorously speaking, we shall call them "narrow but still overparameterized networks").

There are a few works on the trainability of narrow nets (one-hidden-layer networks with $m \geq n/d$ neurons). Soudry and Carmon [63], Xie et al. [71] show that for such networks, stationary points with full-rank NTK (neural tangent kernel) are zero-loss global minima. However, it is not clear whether the NTK stays full rank during the training trajectory. In addition, these two works do not discuss whether a zero-loss global minimizer exists.

There are two interesting related works [9, 12] pointed out by the reviewers. Bubeck et al. [9] study how many neurons are required for memorizing a finite dataset by 1-hidden-layer networks. They prove the following results. Their first result is an "existence" result: there exists a network with width $m \geq \frac{4n}{d}$ which can memorize n input-label pairs (their Proposition 4). However, in this setting they did not provide an algorithm to find the zero-loss solution. Their second result is related to algorithms: they proposed a training algorithm that achieves accuracy up to error ϵ for a neural net with width $m \geq O\left(\frac{n}{d}\frac{\log(1/\epsilon)}{\epsilon}\right)$. This result requires width dependent on the precision ϵ; for instance, when the desired accuracy $\epsilon = 1/n$, the required width is at least $O\left(\frac{n^2}{d}\right)$. In contrast, in our work, the required number of neurons is just $2n/d$, which is independent of ϵ.

Daniely [12] also studies the expressivity and trainability of 1-hidden-layer networks. To memorize $n(1-\epsilon)$ random data points via SGD, their required width is $\tilde{O}(n/d)$. They assumed $n = d^c$ where $c > 0$ is a *fixed* constant (appeared in Sec. 3.3 of [12]), in which case the hidden factor in $\tilde{O}$ is $O(\log[d(\log d)]^c)$. In other words, if $n, d \to \infty$ with the scaling $n = d^c$ for a fixed constant c, then their bound is roughly $O(n/d)$ up to a log-factor. Nevertheless, for more general scaling of n, d, the exponent $c = (\log n)/(\log d)$ may not be a constant and the hidden factor may not be a log-factor (e.g. for fixed d and $n \to \infty$). We tracked their proof and find that the width bound for general n, d is $O\left((n/d)\left(\log(d\log n)\right)^{\log n/\log d}\right)$, which can be larger than $O\left(n^2/d\right)$ (see detailed computation and explanation in Appendix C). In contrast, our required width is $2n/d$ for arbitrary n and d. Our bound is always smaller than n when $d > 2$.

[4] This result describes the loss landscape of the constrained optimization problem, not directly related to algorithm convergence. Nevertheless, it is expected that first-order methods converge to KKT points and thus approximate global minimizers. A rigorous convergence analysis may require verifying extra technical conditions, which is left to future work.

Additionally, there is a major difference between Daniely [12] and our work: they analyze the original unconstrained problem and SGD; in contrast, we analyze a constrained problem and projected GD. This may be the reason why we can get a stronger bound on width. In the experiments in Section 5, we observe that SGD performs badly when the width is small (see the 1st column in Figure 4 (b)). Therefore, we suspect an algorithmic change is needed to train narrow nets with such width (due to the training difficulty), and we indeed propose a new method to train narrow nets.

Due to the space constraints, we defer more related works in Appendix B.

3 Challenges For Analyzing Narrow Nets

In this section, we discuss why it is challenging to achieve expressivity and trainability together for narrow nets. Consider a dataset $\{(x_i, y_i)\}_{i=1}^n \subset \mathbb{R}^d \times \mathbb{R}$ and a 1-hidden-layer neural network:

$$f(x;\theta) = \sum_{j=1}^{m} v_j \sigma\left(w_j^T x\right), \tag{1}$$

where $\sigma\left(w_j^T x\right)$ is the output of the j-th hidden nodes with hidden weights w_j, $\sigma(\cdot)$ is the activation function, and v_j is the corresponding outer weight (bias terms are ignored for simplicity). To learn such a neural network, we search for the optimal parameter $\theta = (w, v)$ by minimizing the following empirical (training) loss:

$$\min_{\theta} \ell(\theta) = \frac{1}{2} \sum_{i=1}^{n} \left(y_i - f\left(x_i;\theta\right)\right)^2, \tag{2}$$

The gradient of the above problem w.r.t. hidden weights $w = \{w_i\}_{i=1}^m$ is given by: $\nabla_w \ell(\theta) = J(w;v)^T (f(w;v) - y) \in \mathbb{R}^{md\times 1}$, where $J(w;v) \in \mathbb{R}^{n\times md}$ is the Jacobian matrix w.r.t w:

$$J(w;v) := \begin{bmatrix} \nabla_w f\left(w; x_1, v\right)^T \\ \vdots \\ \nabla_w f\left(w; x_n, v\right)^T \end{bmatrix} = \begin{bmatrix} v_1 \sigma'\left(w_1^T x_1\right) x_1^T & \cdots & v_m \sigma'\left(w_m^T x_1\right) x_1^T \\ & \vdots & \\ v_1 \sigma'\left(w_1^T x_n\right) x_n^T & \cdots & v_m \sigma'\left(w_m^T x_n\right) x_n^T \end{bmatrix} \in \mathbb{R}^{n\times md}. \tag{3}$$

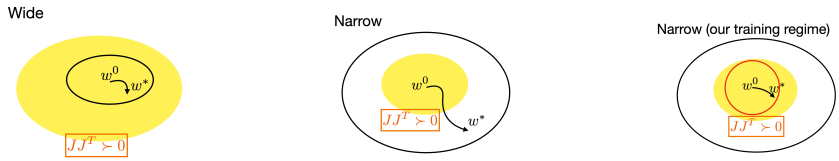
First order methods like GD converge to a stationary point $\theta^* = (w^*, v^*)$ (i.e. with zero gradient) under mild conditions [6]. For problem (2), it is easy to show that if (i) (w^*, v^*) is a stationary point, (ii) $J(w^*; v^*) \in \mathbb{R}^{n\times md}$ is full row rank and $n \le md$, then (w^*, v^*) is a global-min (this claim can be proved by setting the partial gradient of (2) over w to be zero). In other words, for training a network with width $m \ge n/d$, an important tool is to ensure the full-rankness of the Jacobian.

Recent works have shown that it is possible to guarantee the full rankness of the Jacobian matrix along the training trajectories, if the width is above $\Omega(\text{poly}(n))$. Roughly speaking, the proof sketch is the following: (i) with high probability, $J(w;v)$ is non-singular locally around the random initialization ([71], [63]), (ii) increasing the width can effectively bound the parameter movement from initialization, so the "nice" property of non-singualr $J(w, v)$ holds throughout the training, leading to a linear convergence rate [16]. Under this general framework, a number of convergence results are developed for wide networks with width $\Omega(\text{poly}(n))$ ([1, 11, 28, 34, 49, 53–55, 80, 81]). This idea is also illustrated in Figure 1 (a).

We notice that there is a huge gap between the necessary condition $m \ge n/d$ and the common condition $m \ge \Omega(\text{poly}(n))$. We suspect that it is possible to train a narrow net with width $\Theta(n/d)$ to small loss. To achieve this goal, we need to understand why existing arguments require a large width and cannot apply to a network with width $\Theta(n/d)$.

The first reason is about trainability. The above arguments no longer hold when the width is not large enough to control the movement of hidden weights. In this case, the iterates may easily travel far away from the initial point and get stuck at some singular-Jacobian critical points with high training loss (see Figure 1 (b). Also see Figure 4 (b) & (d) for more empirical evidence). In other words, GD may get stuck at sub-optimal stationary points for narrow nets.

The second reason, and also an easily ignored one, is the expressivity (a.k.a. the representation power, see Definition 1 for a formal statement). In above discussion, we implicitly assumed that there exists a zero-loss global minimizer, which is equivalent to "there exists a network configuration such that the network can memorize the data". For networks with width at least n, this assumption can be



(a) Wide networks (b) Narrow networks (c) Narrow networks (our regime)

Figure 1: The parameter movement under different regimes. The shaded area indicates the region where $J(w; v)$ is non-singular, the black circle denotes the region that GD iterates will explore, and the red circle is the constraint designed in our training regime, it will be discussed in Section 4.3.

justified in the following way. The feature matrix

$$\Phi(w) := \begin{bmatrix} \sigma\left(w_1^T x_1\right), \ldots, \sigma\left(w_m^T x_1\right) \\ \vdots \\ \sigma\left(w_1^T x_n\right), \ldots, \sigma\left(w_m^T x_n\right) \end{bmatrix} \in \mathbb{R}^{n \times m} \tag{4}$$

can span the whole space $\mathbb{R}^n$ when it is full rank and $m \geq n$, thus the network can perfectly fit any label $y \in \mathbb{R}^n$ even without training the hidden layer. It is important to note that when the width m is below the sample size n, full row-rankness does not ensure that the row space of the feature matrix is the whole space $\mathbb{R}^n$. In other words, it is not clear whether a global-min with zero loss exists.

In the next section, we will describe how we obtain strong expressive power with $\Theta(n/d)$ neurons, and how to avoid sub-optimal stationary points.

4 Main Results

4.1 Problem Settings and Preliminaries

We denote $\{(x_i, y_i)\}_{i=1}^n \subset \mathbb{R}^d \times \mathbb{R}$ as the training samples, where $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$. For theoretical analysis, we focus on 1-hidden-layer neural networks $f(x; \theta) = \sum_{j=1}^m v_j \sigma\left(w_j^T x\right) \in \mathbb{R}$, where $\sigma(\cdot)$ is the activation function, $w_j \in \mathbb{R}^d$ and $v_j \in \mathbb{R}$ are the parameters to be trained. Note that we only consider the case where $f(x; \theta) \in \mathbb{R}$ has 1-dimensional output for notation simplicity.

To learn such a neural network, we search for the optimal parameter $\theta = (w, v)$ by minimizing the empirical loss (2), and sometimes we also use $\ell(w; v)$ or $f(w; x, v)$ to emphasize the role of w. We use the following shorthanded notations: $x := (x_1^T; \ldots; x_n^T) \in \mathbb{R}^{n \times d}$, $y := (y_1, \ldots, y_n)^T \in \mathbb{R}^n$, $w := (w_1, \ldots, w_m)_{j=1}^m \in \mathbb{R}^{d \times m}$, $v := (v_1, \ldots, v_m)_{j=1}^m \in \mathbb{R}^m$, and $f(w; v) := (f(x_1; w, v), f(x_2; w, v), \ldots, f(x_n; w, v))^T \in \mathbb{R}^n$. We denote the Jacobian matrix of $f(w; v)$ w.r.t w as $J(w; v)$, which can be seen in (3). We define the feature matrix $\Phi(w)$ as in (4). We denote the operator ∇_w as "taking the gradient w.r.t. w", and the same goes for ∇_v. Throughout the paper, 'w.p.1' is the abbreviation for 'with probability one'; when we say 'in the neighborhood of initialization', it means 'w is in the neighborhood of the initialization w^0'.

Now, we formally define the term "expressivity". As discussed in Section 2, we focus on the finite-sample (as opposed to infinite-sample) expressivity, which is relevant in practical training.

Definition 1. *(Expressivity) We say a neural net function class $\mathcal{F} = \{f(x; \theta); \theta \in \Theta\}$ has strong (n-sample) expressivity if for any n input-output pairs $D = \{(x_i, y_i)\}_{i=1}^n \subset \mathbb{R}^d \times \mathbb{R}$ where x_i's are distinct, there exists a $\hat{\theta}(D) \in \Theta$ such that $f(x_i; \hat{\theta}(D)) = y_i$, $i = 1, \cdots, n$. Or equivalently, the optimal value of empirical loss* (2) *equals 0 for any D. Sometimes we may drop the word "strong" for brevity.*

Next, let us describe the *mirrored LeCun's* initialization in Algorithm 1. The idea is that through this initialization, the hidden outputs will cancel out with the outer weights, so that we get zero initial output for any input x; see Figure 2 for a simple illustration. Note that similar symmetric initialization strategies are also proposed in some recent works such as [11] and [12]. However, our purpose is different. More explanation can be seen in the final paragraph of Section 4.2.

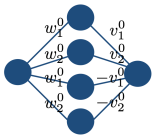


Figure 2: A simple example of the mirrored LeCun's initialization.

Algorithm 1 The mirrored LeCun's initialization

1: Initialize all the weights using LeCun's initialization: $w_{i,j}^0 \sim N(0, \frac{1}{d})$, $v_i^0 \sim N(0, \frac{1}{m})$, for $i = 1, \ldots, m/2$, $j = 1, \ldots, d$.
2: Set $(w_{\frac{m}{2}+1}^0, \ldots, w_m^0) \leftarrow (w_1^0, \ldots, w_{\frac{m}{2}}^0)$, and set $(v_{\frac{m}{2}+1}^0, \ldots, v_m^0) \leftarrow (-v_1^0, \ldots, -v_{\frac{m}{2}}^0)$

Throughout the paper, we will make the following assumptions.

Assumption 1. *For $f(x;\theta)$ in (1), we assume its width m is an even number, and $m \geq \frac{2n}{d}$.*

Assumption 2. *We assume the activation function $\sigma(\cdot) : \mathbb{R} \to \mathbb{R}$ is analytic and L-lipschitz continuous, its zero set only contains 0: $\{z|\sigma(z) = 0\} = \{0\}$. In addition, there are infinitely many non-zero coefficients in the Taylor expansion of $\sigma(\cdot)$.*

Assumption 3. *$x_1, \cdots, x_n$ are independently sampled from a continuous distribution in $\mathbb{R}^d$.*

When $d > 2$, Assumption 1 can be applied to narrow networks [5] with $m < n$. Assumption 2 covers many commonly used activation functions such as sigmoid, softplus, and Tanh, but it does not cover ReLU since it is nonsmooth.

4.2 Expressivity Analysis

In this section, we prove that narrow neural networks (which are still over-parameterized) have strong expressivity. Further, the zero-training-error solution is surrounded by a good landscape with no local-min or saddles, which motivates our trainability analysis in the following sections.

Theorem 1. *Suppose Assumption 1, 2, and 3 holds. If the neural network $f(x;\theta) = \sum_{j=1}^m v_j \sigma\left(w_j^T x\right)$ is initialized at the mirrored LeCun's initialization given in Algorithm 1, with $\theta^0 = (w^0, v^0)$, then there exists $\epsilon_0 > 0$ such that for any $\epsilon \leq \epsilon_0$, there exists a $w \in B_\epsilon(w^0) = \{w \mid \|w - w^0\|_F \leq \epsilon\}$ and a entry-wise non-zero v, such that with probabilty 1 of choosing $\{(x_i, y_i)\}_{i=1}^n$, the output of f will be exactly the groundtruth label:*

$$f(x_i;\theta) = \sum_{j=1}^m v_j \sigma(w_j^T x_i) = y_i, i = 1, \cdots, n. \tag{5}$$

In addition, every stationary point $\theta^* = (w^*, v^*)$ (i.e., the gradient is zero) is a global-min of (2) *with zero loss if it satisfies $w^* \in B_\epsilon(w^0)$ and v^* is entry-wise non-zero.*

Remark 1. Theorem 1 emphasizes the role of hidden weights of a neural network: it is a key ingredient for strong expressivity. When $m < n$, if we fix all the $w = w^0$, the range space of the feature matrix $\Phi(w^0)$ does not cover the whole $\mathbb{R}^n$ space, so there always exists a label y, such that no v^* can be found that perfectly maps the input to y. However, a small tolerance of the movement of w will let $f(x;\theta)$ perfectly fit any input-label pair, so the movement of w is vitally important. The free perturbation of w serves as an effective remedy against the limited expressivity.

Remark 2. We emphasize that Theorem 1 holds for "any small enough ϵ" instead of "any ϵ". Therefore, Theorem 1 only states "there is no spurious local-min" *locally*. It is still possible that on the *global* landscape results there "exist bad local-min" (e.g. Ding et al. [14]).

We comment a bit more on the maximum required size of ϵ. In our proof in Appendix E.1, it should not exceed the the radius of the region where the Jacobian $J(w;v)$ stays full-rank (the yellow-shaded area in Figure 1). To briefly summarize, the maximum radius is (linearly) proportional to the minimum singular value of the initial Jacobian $J(w^0;v^0)$. Technical details on the size of this radius can be seen in [18, Remark 4.1].

Proof sketch. Theorem 1 consists of two arguments: (i) there exists a global-min with zero loss, (ii) in the neighborhood of initialization, every stationary point is a global-min. A detailed proof is relegated to Appendix E. We outline the main idea below.

[5] When $d = 1, 2$, all our results still hold; nevertheless, the required width $m \geq n, 2n$, thus it does not belong to the "narrow" setting we defined earlier in Section 2 (which requires $m \in [n/d, n)$).

To argue (i), the key idea is to use the Inverse Function Theorem (IFT), which is stated in Appendix E.1. According to IFT, as long as an $n \times n$ submatrix of $J(w^0; v^0)$ is invertible, then for any $y \in \mathbb{R}^n$ and any small enough ϵ, there exists a $w^* \in B_\epsilon(w^0)$ whose prediction output $f(w^*; v^0) \propto y - f(w^0; v^0)$. Additionally, since $f(w^0; v^0) = 0$, we have $f(w^*; v^0) \propto y$. Once this is shown, then we just need to scale all the outer weight v_j uniformly and the output will be exactly y since $f(w^*; v)$ is linear in v.

To argue (ii), recall that all the stationary points $\theta^* = (w^*, v^*)$ satisfy $\nabla_w \ell(\theta^*) = J(w^*; v^*)^T (f(w^*; v^*) - y) = 0$. Therefore, if $J^T(w^*; v^*) \in \mathbb{R}^{md \times n}$ is of full column rank, the stationary point $\theta^* = (w^*, v^*)$ is a global minimizer with $\ell(\theta^*) = 0$. The desired full-rankness condition is true because: (i) $J(w^0; v^0)$ is full rank w.p.1 at initialization, (ii) w^* will not leave the small neighborhood $B_\epsilon(w^0)$, so the dynamics of w stays inside the manifold of full-rank Jacobian.

□

The proof of Theorem 1 relies on the full-rankness of the Jacobian matrix. We note that such full-rankness holds for *both* the mirrored and the regular LeCun's initialization (the case for the regular one can be proved using the same technique). So why do we insist on shifting the initial output to 0? Simply put, the "randomness of weights" contributes to the "full-rankness", while the 'shifting' allows us to gain local representation power by applying IFT properly. To be more specific, $f(w^0, v^0) = 0 \in \mathbb{R}^n$ is important because it is surrounded by all possible directions pointed from $0 \in \mathbb{R}^n$, so for any $y \in \mathbb{R}^n$, IFT claims that there exists at least one $w^* \in B_\epsilon(w^0)$, s.t. $f(w^*; v^0) \propto y - f(w^0; v^0) = y$, therefore, $f(w^*; v^*)$ can perfectly match y by scaling v^0 with some constant (Figure 3(a) illustrates this case when $n = 2$).

In contrast, if we use IFT around *regular* LeCun's initialization, the existence of $f(w^0; v^0)$ on the right hand side resists us from scaling v^0 like before (Figure 3(b) illustrates this case). As such, Theorem 1 does not hold for *any* small ϵ around the regular initialization. This is also revealed in our experiments in Section 5.1: training fails if we only search around the regular LeCun's initialization.

(a) The mirrored LeCun's initialization (b) Regular LeCun's initialization

Figure 3: Examples of using Inverse Function Theorem (IFT) under different initialization strategies. The illustration here is for $n = 2$. For (a), scaling v^0 will directly lead to zero loss, while it is not true for (b) due to the non-zero $f(w^0; v^0)$.

The idea of zero initial output is also used in other recent works in Table 1. Despite the similar design, they use such an initialization for different purposes. In NTK regime, zero initial output helps eliminate the bias term and simplify the proof ([3, 11, 23, 24, 34] and [12]). Nguyen [49] also uses zero initial output, but their initialization is very different in that all the hidden layers will have high values while the last layer is assigned to 0. In this way, they manage to limit the movement of hidden layers without increasing the width. To our knowledge, this is the first time that the zero initial output has been linked to Inverse Function Theorem, by which the strong expressivity of a narrow neural network can be identified.

Table 1: Comparison of recent works considering zero initial output.

Work	Width	Motivation
[3, 11, 23, 24, 34]	$m_{L-1} \to \infty$	To avoid handling the bias term in the NTK regime
[12]	$m = \tilde{O}(n/d)$	To avoid handling the bias term in the NTK regime
[49]	$m_{L-1} = O(n)$	To ensure linear convergence via imbalanced weight
Ours	$m = O\left(\frac{n}{d}\right)$	To achieve strong expressivity via Inverse Function Theorem

4.3 Trainability Analysis

Despite the expressivity and good local properties stated in Theorem 1, in practical training, the weights can easily escape the nice neighborhood, especially when the width is not sufficiently large. To keep the hidden weights inside this nice region, an intuitive idea is to impose an explicit constraint, but it may suffer from bad local-min on the boundary with a very large loss [6]. This is supported by our experiments in Section 5, Figure 4, (b): when we add the hidden-weight constraint directly to the regular training regime (i.e., let $\|w - w^0\|_F \leq \epsilon$), it fails to find a low-cost solution when ϵ and width become small.

In this sense, we need to design a constrained problem, such that all the KKT points will have small training loss, including those on the boundary. Fortunately, Theorem 1 suggests one such formulation. Recall in the proof of Theorem 1, we construct a zero-loss global-min by scaling v^0 , so v^* still follows the pairwise-opposite pattern. Inspired by this, we consider the following neural network (we abuse the notation of $f(x;\theta)$, $f(x;w,v)$ and $v = (v_1, \ldots, v_{\frac{m}{2}})$ here):

$$f(x;w,v) = \sum_{j=1}^{\frac{m}{2}} v_j \left(\sigma(w_j^T x) - \sigma(w_{j+\frac{m}{2}}^T x)\right). \tag{6}$$

Note that the optimization variable for the outer layer is only $v = (v_1, \ldots, v_{\frac{m}{2}})$, and the rest of the outer weights are automatically set to be $-v$. Despite the change of v, Theorem 1 still applies since it does not have any specific requirement on v. We then use (6) to formulate the following problem (7):

$$\min_{\theta} \ell(\theta) = \frac{1}{2}\sum_{i=1}^{n}(y_i - f(x_i;\theta))^2, \quad \text{s.t.} \qquad w \in B_\epsilon(w^0),\ v \in B_{\zeta,\kappa}(v) \tag{7}$$

where $f(x_i;\theta)$ is in the form of (6),

$B_\epsilon(w^0) := \{w \mid \|w - w^0\|_F \leq \epsilon\}$,

$B_{\zeta,\kappa}(v) := \{v \mid\ v \geq \zeta\mathbf{1}$ and $v_j/v_{j'} \leq \kappa, \forall\ (j,j') \in \{1,\cdots,m\}$, where $\zeta > 0,\ \kappa < \infty.\}$.

Here, $\zeta > 0$ is a small constant that keeps the entries of v away from zero, which is an essential requirement of Theorem 1. The requirement of $v_j/v_{j'} \leq \kappa < \infty$ allows all entries of v to be uniformly large, but it rules out the case when some entries are much larger than others. Instead of regarding all these requirements of $B(v)$ as prior assumptions, we formulate them into the constraints in the problem, so all the iterates in the practical training algorithm will strictly follow these requirements. In Theorem 2, we show that every KKT point of problem (7) implies the near-global optimality for the unconstrained training problem (2).

Theorem 2. *Suppose Assumption 1, 2, and 3 hold and assume $\theta^0 = (w^0, v^0)$ as given in Algorithm 1. Then every KKT point $\theta^* = (w^*, v^*)$ of (7) is an approximate global-min w.p.1., that is:*

$$\ell(w^*, v^*) = O(\epsilon^2). \tag{8}$$

The proof of Theorem 2 is based on the special structure of neural network $f(x;\theta)$, including the linear dependence of v and the mirrored pattern of parameters. To better illustrate our proof idea, we provide a user-friendly proof sketch in Appendix F.1. Detailed proof can be seen in Appendix F.2.

Theorem 2 motivates a training method to reach small loss. We highlight three new ingredients that is not used in regular neural net training: the mirrored initialization, the pairwise structure of v in (7), and the constrained parameter movement. Combining these elements with Projected Gradient Descent (PGD), we propose a new training regime in Algorithm 2 in Appendix H.1. Thorough numerical results are provided in the following sections to demonstrate the efficacy of Algorithm 2.

4.4 Discussion: Extension to Deep Networks

In the previous sections, we analyze the trainability and expressivity of narrow 1-hidden-layer networks. We find it possible to extend the previous analysis to deep nets, and we already have some preliminary results. Due to space constraints, more relevant discussions are deferred to Appendix G.

5 Experiments

In this section, we provide empirical validation for our theory. Specifically, we compare the performance of two training regimes [6]:

[6] We call it "training regime" instead of "training method" since we use a different formulation as well a different algorithm compared to standard SGD.

(1) Our training regime: we optimize a constrained problem (7) by using PGD (projected gradient descent), starting from the mirrored LeCun's initialization. (See Algorithm 2 in Appendix H.1.)

(2) Regular training regime: optimize an unconstrained problem (2) by using GD-based methods, starting from LeCun's initialization.

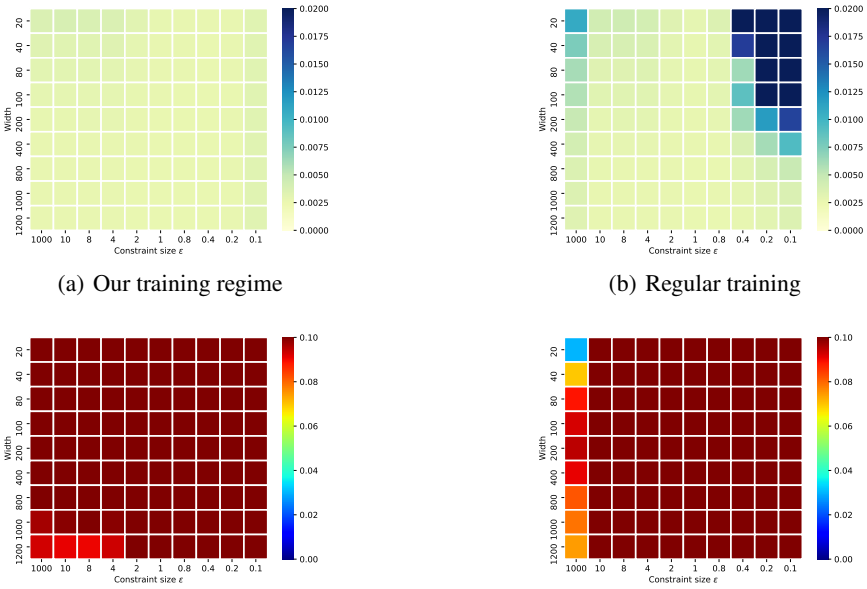
Main ingredients of our algorithm. As shown in Theorem 2, all KKT points in our training regime have small empirical loss. To reach such KKT points, we use Projected Gradient Descent (PGD) (see Bertsekas [6]). Even though we do not provide convergence analysis, PGD can, empirically, converge to a KKT point with proper choice of stepsize. We outline the proposed training regime in Algorithm 2 in Appendix H.1. To briefly summarize, there are three key ingredients in Algorithm 2: the mirrored initialization; the pairwise structure of v in (7); and the PGD algorithm. Each of these changes only involves a few lines of code changes based on the regular training. We demonstrate the `PyTorch` implementation of these changes in Appendix H.1.

Better training and test error. To evaluate our theory in terms of training error, we conduct experiments on synthetic dataset (shown in Section 5.1) and random-labeled CIFAR-10 [31] (shown in appendix H.6). We further observe the strong generalization power of Algorithm 2, even though it is not yet revealed in our theory. Our training regime brings higher or competitive test accuracy on (Restricted) ImageNet [58] (shown in Section 5.2), MNIST [33], CIFAR-10, CIFAR-100 [31] (shown in Appendix H). Detailed experimental setup are explained in Appendix H.2.

As a side note, for all the experiments in our training regime, we observe that v never touches the boundary of $B_{\zeta,\kappa}(v)$ when $\kappa = 1, \zeta = 0.001$. So we can regard problem (7) as an unconstrained problem for v, and PGD only projects the hidden weights w into $B_\epsilon(w^0)$. When $\epsilon = 1000$, problem (7) degenerates into an unconstrained problem (but still different from the regular training due to the changes in the structure of v and initialization).

5.1 Training Error on The Synthetic Dataset

On the synthetic regression dataset, we train 1-hidden-layer networks under different widths and different training settings, the final training errors are shown in Figure 4. We explain as follows.



(a) Our training regime

(b) Regular training

(c) $\lambda_{\min}(J(w^*; v^*))$in our training regime

(d) $\lambda_{\min}(J(w^*; v^*))$in regular training regime

Figure 4: Synthetic data: training losses and $\lambda_{\min}(J(w^*; v^*))$ in different width & hidden-weight constraint size ϵ.

As for our training regime, we try different hidden-weight constraint size ϵ, these results can be seen in Figure 4, (a). Accordingly, the performance of the regular training regime can be seen in the 1st column in Figure 4, (b). As argued above, $\epsilon = 1000$ degenerates PGD into unconstrained GD, so this column shows the results for regular training regime. As for the rest of the columns in the Figure 4,

(b), we try to investigate an ablation study: "What will happen if we directly add constraint $B_\epsilon(w^0)$ on w, and use PGD without any modification of the initialization & network structure?" In each block in Figure 4, a grid search of step-size is performed to ensure the convergence of algorithms. The key messages from this set of experiments are summarized as follows.

First, our training regime performs well regardless of width, yet regular unconstrained training fails when the network is narrow (1st column in Figure 4 (b)).

Second, it does not work when we directly impose the hidden-weight constraint of (7) on the regular training regime (and PGD is used accordingly), as it fails to find a low-cost solution when ϵ and width become small. There are two possible causes: perhaps there is no global-min inside the ball, or it converges to bad local-min on the boundary. In contrast, our training regime always finds a low-cost solution with any choice of ϵ. Therefore, we suggest *not* to directly add constraint and use PGD. Instead, when using PGD, it is better to utilize the mirrored initialization & the pairwise structure of v in (7) (see Figure 4 (a)).

Furthermore, Figure 4, (c) & (d) depict $\lambda_{\min}(J(w^*; v^*))$, i.e. the minimum singular value of the Jacobian at the stationary (or KKT) points. As expected in Section 1, when the width is small, regular training regime (when $\epsilon = 1,000$) has trouble controlling the parameter movement, and it is likely to get trapped at a stationary point with a singular Jacobian matrix, leading to a large loss despite the convergence. However, this is not an issue in our training regime.

5.2 Test Accuracy on R-ImageNet

We check the test accuracy (not just the training accuracy) of the proposed method on R(Restricted)-ImageNet ([58]). R-ImageNet is a subset of ImageNet with resolution 224×224, and it is widely used in various papers (e.g. [68], [27]). Detailed description can be found in Appendix H.2. In both training regimes, experiments are conducted on ResNet-18 [22], which contains 4 CNN blocks and 1 fully connected output layer. To compare the performance under different widths, we shrink the number of channels in the final CNN block gradually from 512 to 64 (note that we will call "the number of channels" as the "width" in CNN, this is a straightforward extension of the width in FCN). In our training regime, we apply all the constraints in problem (7) to the final CNN block & the output layer.

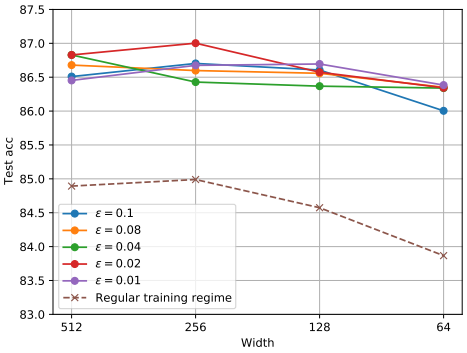


Figure 5: R-ImageNet: test accuracy under our training regime with different ϵ vs regular training regime. In x-axis, width stands for the number of channels in the final CNN block of ResNet-18. These results are averaged over 5 seeds.

There are two messages shown in Figure 5. First, our training regime outperforms regular training when using the standard ResNet-18 (i.e., width 512 in the final CNN block). Second, our training regime in the most narrow setting (width 64) performs quite close to the standard case (width 512). In comparison, regular SGD does not perform well in the narrow setting. More theoretical analysis on the generalization power will be considered as our future work.

6 Conclusion

In this work, we shed new light on both the expressivity and trainability of narrow networks. Despite the limited number of neurons, we prove that the network can memorize n samples, and it can be provably trained to approximately zero loss in our training regime. We notice some interesting questions by reviewers and colleagues. We provide further discussion on these questions in Appendix A, they may be intriguing for general readers.

Finally, there are several important future directions. First, we empirically observe that our training regime brings strong generalization power, more theoretical analysis will be interesting. Second, our current analysis is still limited to 1-hidden-layer networks, we are trying to extend it to deep ones. Third, the algorithmic convergence analysis to reach the KKT point is imperative.

Acknowledgments and Disclosure of Funding

We would like to thank Dawei Li for valuable and productive discussions. We want to thank the anonymous reviewers for their valuable suggestions and comments. We would also like to express our gratitude to Zeyu Qin, Jiancong Xiao and Congliang Chen for the support of R-ImageNet and CIFAR experiments. M. Hong is partially supported by an NSF grant CMMI-1727757, and an IBM Faculty Research Award. The work of Z.-Q. Luo is supported by the National Natural Science Foundation of China (No. 61731018) and the Guangdong Provincial Key Laboratory of Big Data Computation Theories and Methods.

References

[1] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *International Conference on Machine Learning*, pages 242–252. PMLR, 2019.

[2] Peter Auer, Mark Herbster, Manfred K Warmuth, et al. Exponentially many local minima for single neurons. *Advances in neural information processing systems*, pages 316–322, 1996.

[3] Yu Bai and Jason D Lee. Beyond linearization: On quadratic and higher-order approximation of wide neural networks. *arXiv preprint arXiv:1910.01619*, 2019.

[4] Andrew R Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, 1993.

[5] Eric B Baum. On the capabilities of multilayer perceptrons. *Journal of complexity*, 4(3):193–215, 1988.

[6] Dimitri P Bertsekas. Nonlinear programming. *Journal of the Operational Research Society*, 48(3):334–334, 1997.

[7] Monica Bianchini and Franco Scarselli. On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE transactions on neural networks and learning systems*, 25(8):1553–1565, 2014.

[8] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[9] Sébastien Bubeck, Ronen Eldan, Yin Tat Lee, and Dan Mikulincer. Network size and weights size for memorization with two-layers neural networks. *arXiv preprint arXiv:2006.02855*, 2020.

[10] Zixiang Chen, Yuan Cao, Difan Zou, and Quanquan Gu. How much over-parameterization is sufficient to learn deep relu networks? *arXiv preprint arXiv:1911.12360*, 2019.

[11] Lenaic Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. *arXiv preprint arXiv:1812.07956*, 2018.

[12] Amit Daniely. Neural networks learning and memorization with (almost) no over-parameterization. *arXiv preprint arXiv:1911.09873*, 2019.

[13] Olivier Delalleau and Yoshua Bengio. Shallow vs. deep sum-product networks. *Advances in neural information processing systems*, 24:666–674, 2011.

[14] Tian Ding, Dawei Li, and Ruoyu Sun. Sub-optimal local minima exist for almost all over-parameterized neural networks. *arXiv preprint arXiv:1911.01413*, 2019.

[15] Tian Ding, Dawei Li, and Ruoyu Sun. Sub-optimal local minima exist for neural networks with almost all non-linear activations. *arXiv preprint arXiv:1911.01413*, 2019.

[16] Simon Du, Jason Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. In *International Conference on Machine Learning*, pages 1675–1685. PMLR, 2019.

[17] Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference on learning theory*, pages 907–940. PMLR, 2016.

[18] Kai-Seng Chou et. al. Chapter 4: Inverse function theorem. `https://www.math.cuhk.edu.hk/course_builder/1415/math3060/Chapter%204.%20Inverse%20Function%20Theorem.pdf`. 2015.

[19] C Daniel Freeman and Joan Bruna. Topology and geometry of half-rectified network optimization. *arXiv preprint arXiv:1611.01540*, 2016.

[20] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[21] Boris Hanin and David Rolnick. Complexity of linear regions in deep networks. In *International Conference on Machine Learning*, pages 2596–2604. PMLR, 2019.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[23] Wei Hu, Zhiyuan Li, and Dingli Yu. Simple and effective regularization methods for training on noisily labeled data with generalization guarantee. *arXiv preprint arXiv:1905.11368*, 2019.

[24] Wei Hu, Lechao Xiao, Ben Adlam, and Jeffrey Pennington. The surprising simplicity of the early-time learning dynamics of neural networks. *arXiv preprint arXiv:2006.14599*, 2020.

[25] Guang-Bin Huang. Learning capability and storage capacity of two-hidden-layer feedforward networks. *IEEE transactions on neural networks*, 14(2):274–281, 2003.

[26] Guang-Bin Huang and Haroon A Babri. Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions. *IEEE transactions on neural networks*, 9(1):224–229, 1998.

[27] Nathan Inkawhich, Kevin J Liang, Binghui Wang, Matthew Inkawhich, Lawrence Carin, and Yiran Chen. Perturbing across the feature hierarchy to improve standard and strict blackbox attack transferability. *arXiv preprint arXiv:2004.14861*, 2020.

[28] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *arXiv preprint arXiv:1806.07572*, 2018.

[29] Ziwei Ji and Matus Telgarsky. Polylogarithmic width suffices for gradient descent to achieve arbitrarily small test error with shallow relu networks. *arXiv preprint arXiv:1909.12292*, 2019.

[30] Kenji Kawaguchi. Deep learning without poor local minima. In *Advances in neural information processing systems*, pages 586–594, 2016.

[31] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[32] Thomas Laurent and James Brecht. Deep linear networks with arbitrary loss: All local minima are global. In *International Conference on Machine Learning*, pages 2908–2913, 2018.

[33] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[34] Jaehoon Lee, Lechao Xiao, Samuel S Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. *arXiv preprint arXiv:1902.06720*, 2019.

[35] Dawei Li, Tian Ding, and Ruoyu Sun. On the benefit of width for neural networks: Disappearance of bad basins. *arXiv preprint arXiv:1812.11039*, 2018.

[36] Dawei Li, Tian Ding, and Ruoyu Sun. On the benefit of width for neural networks: Disappearance of bad basins, 2021.

[37] Shiyu Liang, Ruoyu Sun, Jason D Lee, and R Srikant. Adding one neuron can eliminate all bad local minima. In *Advances in Neural Information Processing Systems*, pages 4355–4365, 2018.

[38] SHIYU LIANG, Ruoyu Sun, Yixuan Li, and Rayadurgam Srikant. Understanding the loss surface of neural networks for binary classification. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2835–2843. PMLR, 10–15 Jul 2018.

[39] Shiyu Liang, Ruoyu Sun, and R. Srikant. Revisiting landscape analysis in deep neural networks: Eliminating decreasing paths to infinity, 2019.

[40] Shiyu Liang, Ruoyu Sun, and R. Srikant. Achieving small test error in mildly overparameterized neural networks. *CoRR*, abs/2104.11895, 2021.

[41] Henry W Lin, Max Tegmark, and David Rolnick. Why does deep and cheap learning work so well? *Journal of Statistical Physics*, 168(6):1223–1247, 2017.

[42] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. *arXiv preprint arXiv:1410.1141*, 2014.

[43] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[44] Haihao Lu and Kenji Kawaguchi. Depth creates no bad local minima. *arXiv preprint arXiv:1702.08580*, 2017.

[45] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6232–6240, 2017.

[46] George A Miller. *WordNet: An electronic lexical database*. MIT press, 1998.

[47] Boris Mityagin. The zero set of a real analytic function. *arXiv preprint arXiv:1512.07276*, 2015.

[48] Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. *arXiv preprint arXiv:1402.1869*, 2014.

[49] Quynh Nguyen. On the proof of global convergence of gradient descent for deep relu networks with linear widths. *arXiv preprint arXiv:2101.09612*, 2021.

[50] Quynh Nguyen, Mahesh Chandra Mukkamala, and Matthias Hein. On the loss landscape of a class of deep neural networks with no bad local valleys. *arXiv preprint arXiv:1809.10749*, 2018.

[51] Atsushi Nitanda, Geoffrey Chinot, and Taiji Suzuki. Gradient descent can learn less over-parameterized two-layer neural networks on classification problems. *arXiv preprint arXiv:1905.09870*, 2019.

[52] Maher Nouiehed and Meisam Razaviyayn. Learning deep models: Critical points and local openness. *arXiv preprint arXiv:1803.02968*, 2018.

[53] Asaf Noy, Yi Xu, Yonathan Aflalo, Lihi Zelnik-Manor, and Rong Jin. A convergence theory towards practical over-parameterized deep neural networks. *arXiv preprint arXiv:2101.04243*, 2021.

[54] Samet Oymak and Mahdi Soltanolkotabi. Overparameterized nonlinear learning: Gradient descent takes the shortest path? In *International Conference on Machine Learning*, pages 4951–4960. PMLR, 2019.

[55] Samet Oymak and Mahdi Soltanolkotabi. Toward moderate overparameterization: Global convergence guarantees for training shallow neural networks. *IEEE Journal on Selected Areas in Information Theory*, 1(1):84–105, 2020.

[56] Sejun Park, Jaeho Lee, Chulhee Yun, and Jinwoo Shin. Provable memorization via deep neural networks using sub-linear parameters. *arXiv preprint arXiv:2010.13363*, 2020.

[57] David Rolnick and Max Tegmark. The power of deeper networks for expressing natural functions. *arXiv preprint arXiv:1705.05502*, 2017.

[58] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[59] Itay Safran and Ohad Shamir. On the quality of the initial basin in overspecified neural networks. In *International Conference on Machine Learning*, pages 774–782. PMLR, 2016.

[60] Itay Safran and Ohad Shamir. Depth-width tradeoffs in approximating natural functions with neural networks. In *International Conference on Machine Learning*, pages 2979–2987. PMLR, 2017.

[61] Itay Safran and Ohad Shamir. Spurious local minima are common in two-layer relu neural networks. *arXiv preprint arXiv:1712.08968*, 2017.

[62] Mahdi Soltanolkotabi, Adel Javanmard, and Jason D Lee. Theoretical insights into the optimization landscape of over-parameterized shallow neural networks. *IEEE Transactions on Information Theory*, 65(2):742–769, 2019.

[63] Daniel Soudry and Yair Carmon. No bad local minima: Data independent training error guarantees for multilayer neural networks. *arXiv preprint arXiv:1605.08361*, 2016.

[64] Ruo-Yu Sun. Optimization for deep learning: An overview. *Journal of the Operations Research Society of China*, pages 1–46, 2020.

[65] Ruoyu Sun, Dawei Li, Shiyu Liang, Tian Ding, and Rayadurgam Srikant. The global landscape of neural networks: An overview. *IEEE Signal Processing Magazine*, 37(5):95–108, 2020.

[66] Grzegorz Swirszcz, Wojciech Marian Czarnecki, and Razvan Pascanu. Local minima in training of deep networks. 2016.

[67] Matus Telgarsky. Benefits of depth in neural networks. In *Conference on learning theory*, pages 1517–1539. PMLR, 2016.

[68] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. *arXiv preprint arXiv:1805.12152*, 2018.

[69] Luca Venturi, Afonso Bandeira, and Joan Bruna. Spurious valleys in two-layer neural network optimization landscapes. *arXiv preprint arXiv:1802.06384*, 2018.

[70] Roman Vershynin. Memory capacity of neural networks with threshold and rectified linear unit activations. *SIAM Journal on Mathematics of Data Science*, 2(4):1004–1033, 2020.

[71] Bo Xie, Yingyu Liang, and Le Song. Diverse neural network learns true target functions. In *Artificial Intelligence and Statistics*, pages 1216–1224. PMLR, 2017.

[72] Chulhee Yun, Suvrit Sra, and Ali Jadbabaie. Small nonlinearities in activation functions create bad local minima in neural networks. *arXiv preprint arXiv:1802.03487*, 2018.

[73] Chulhee Yun, Suvrit Sra, and Ali Jadbabaie. Small ReLU networks are powerful memorizers: a tight analysis of memorization capacity. *arXiv preprint arXiv:1810.07770*, 2018.

[74] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[75] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.

[76] Li Zhang. Depth creates no more spurious local minima. *arXiv preprint arXiv:1901.09827*, 2019.

[77] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*, 2017.

[78] Mo Zhou, Rong Ge, and Chi Jin. A local convergence theory for mildly over-parameterized two-layer neural network. *arXiv preprint arXiv:2102.02410*, 2021.

[79] Yi Zhou and Yingbin Liang. Critical points of neural networks: Analytical forms and landscape properties. *arXiv preprint arXiv:1710.11205*, 2017.

[80] Difan Zou, Yuan Cao, Dongruo Zhou, and Quanquan Gu. Stochastic gradient descent optimizes over-parameterized deep relu networks. arxiv e-prints, art. *arXiv preprint arXiv:1811.08888*, 2018.

[81] Difan Zou and Quanquan Gu. An improved analysis of training over-parameterized deep neural networks. *arXiv preprint arXiv:1906.04688*, 2019.

Appendix

Potential Negative Societal Impacts

In this paper, we discuss the expressivity and trainability of narrow neural networks. This paper provides a new understanding from the theoretical study, and such new insights will inspire a better training approach for neural networks with a small number of parameters. In industrial applications, several aspects of impact can be expected: training a huge neural network is at the expense of heavy computational burdens and large power consumption, which is a big challenge for embedded systems and small portable devices. In this sense, our work sheds new light on training narrow networks with much fewer parameters, so it will save energy for AI industries and companies. On the other hand, if everyone can afford to train powerful neural networks on their cell phone, then it will be a potential threat to most famous companies & institutes who are boast of their exclusive computational advantages. Additionally, there are chances that neural networks will be used for illegal usage.

Appendix Organization

The Appendix is organized as follows.

- Appendix A provides some interesting questions by reviewers and colleagues. We provide further discussion on these questions, they may be intriguing for general readers.
- Appendix B provides more discussions on the literature.
- Appendix C provides more discussions on the related work Daniely [12].
- Appendix D introduces all the notations that will occur in the proof.
- Appendix E and F provide detailed proof for Theorem 1 and Theorem 2, respectively.
- Appendix G discusses extending the current analysis to deep neural networks.
- Appendix H introduces the following contents. (i) the formal statement of our training regime, i.e., Algorithm 2. (ii) The `Pytorch` implementation for Algorithm 2. (iii) Experimental details and settings for all the experiments appear in the paper. (iv) More experiments.

A Some More Discussions

In this section, we organize some frequently asked questions by reviewers and colleagues. Many of these questions may also be intriguing for general readers. We are thankful for the valuable discussion and we would like to share these questions. Here are our answers from the authors' perspective.

Q1: *The paper merely focuses on the optimization aspect, that is, minimizing the training loss, and ignores the more important problem (from ML perspective) of generalization. It would have been helpful if the authors include a discussion on the implications of these results on the generalization error as well.*

A1: We agree that generalization is a very important issue for deep learning (and still largely mysterious). Much of recent effort is spent on explaining why a huge number of parameters can still lead to a small generalization gap. Despite this interesting line of research, for narrow networks, the risk of overfitting is much smaller (due to traditional wisdom that fewer parameters lead to a smaller generalization gap), thus generalization of narrow networks is probably less mysterious than wide networks. According to our experiments on various real datasets (in Section 5.2 & Appendix H), our training regime provides competitive or even better generalization performance than the regular training.

We think for the current stage, it may be more imperative to resolve expressiveness and optimization issues so as to improve practical performance. That being said, we agree that the theoretical study of the generalization gap is an interesting next step for research, and we will study it in future works.

Q2: *To which extent the result can be extended to non-smooth activations, such as ReLU or powers of ReLU?*

A2: We think it is possible, but definitely not easy. A main reason that we analyze smooth activation is that we need to prove the full-rankness of Jacobian $J(w; v)$ in Theorem 1. We believe that, for narrow nets, the full-rankness of Jacobian $J(w; v)$ with both smooth and non-smooth activation can be proved, but at the current stage, we lack suitable techniques for non-smooth ones (at least it is hard to extend the current technique to ReLU). We briefly summarize the technical difference below.

1. For ReLU, every entry of the NTK matrix ($J(w; v)J(w; v)^T$) has a closed-form solution when the width $m = \infty$, and the corresponding NTK matrix is full rank under mild data assumption. This nice property allows us to utilize concentration inequalities to keep the full-rankness of $J(w; v)J(w; v)^T$ under finite (but large enough) width. This idea is used in Du et al, [16] for neural nets with width $m = \text{Poly}(n)$. However, this "concentration-based" approach is sensitive to width, it may be difficult to extend to narrow cases.

2. As for smooth activation, we utilize an important property of analytic function (shown in Lemma E.2). Lemma E.2 is based the intrinsic property of *any* analytic function, instead of a "infinite-width" argument. Therefore, it casts a higher possibility to use it for narrow nets. This property is also used in Li et al. [35] to prove the full-rankness of $J(w; v)$ for neural nets with width $m = O(n)$. Further, we successfully extend this result to width $m < n$ (with more sophisticated analysis).

In summary, analyzing ReLU requires new techniques. It is hard to extend the current analysis to narrow nets with general non-smooth activation.

Q3: *The "trainability" results only allow the small movement of hidden weights. This, in essence, is not desirable as it does not allow learning representations which is crucial in deep learning. Also, this is conceptually similar to the requirements in the NTK / lazy training regime, with the difference that those results offer precise convergence rates.*

A3: We believe our analysis is different from "lazy training", based on the following reasons.

i) We would like to point out that "small movement" does not imply our method is similar to "lazy training". Chizat et al.[11] described"lazy training" as the situation where "these two paths remain close until the algorithm is stopped". Here, the two paths correspond to the trajectory of training the original model and the linearized model respectively. "Training in a neighborhood of initialization" is neither a sufficient nor a necessary condition of "lazy training". For wide nets, "moving in a neighborhood" and "lazy training" are also co-existent, not causal. Logically speaking, for narrow nets, the two paths can be rather different while the training appears in a neighborhood.

ii) We then argue for narrow nets, linearized trajectory and neural net trajectory have to be quite different. Note that the linearized model of narrow nets cannot fit arbitrary data. In fact, when width $m < n$, the feature matrix does not span the whole $\mathbb{R}^n$ space if we fix first-layer hidden weights $w = w^0$. Thus our training trajectory has to be rather different from the linearized model trajectory, so as to fit data. In contrast, for wide networks, random fixed features suffice to represent data, so staying close to the linearized trajectory (or even coincide) can lead to zero training error.

iii) In our setting, the movement of first-layer hidden weights is *necessary* (no matter small or big). So "feature learning" is critical for our algorithm to achieve small training error. For wide net analysis, the movement of first-layer hidden weights is NOT necessary for zero training error, so there is no need for "feature learning".

In summary, narrow networks are out of the scope of lazy training analysis and have extra difficulties, making our analysis rather different from lazy training analysis (despite some similarities such as using the full-rankness of Jacobian).

For completeness, we further explain the main differences between our work and the previous papers on wide nets, most of the following opinions are also expressed in Section 3.

1. expressivity is rarely an issue for wide-net papers. A wide enough network (width $m > n$), even linear, can always fit n arbitrary data samples (can be proved using simple linear algebra, which is shown in Section 3). However, when $m < n$, the expressivity is questionable. Fortunately, our Theorem 1 provides a clean positive answer.

2. When $m < n$, we are not clear whether the claim "GD converges to global-min in narrow nets" is true or not. In practice, GD easily got stuck at a large-loss stationary point in narrow-net training, but the wide nets are much easier to reach near-0 loss. So what we did

is NOT proving GD works in narrow nets, but designing an algorithm and showing it works (at least under certain conditions).

3. The empirical motivation is different. Existing NTK papers tried to "explain" why wide networks work well. We aim to "design" methods for training narrow networks, a topic of great interest for practitioners with limited computation resources and on-device AI.

B Additional Related Work

We provide discussions on other related work (in addition to those closely related ones mentioned in the main body).

Memorization of small-width networks. Yun et al. [73] studies how many neurons are required for a multi-layer ReLU network to memorize n samples. In particular, they proved that if there are at least three layers, then the number of neurons per layer needed to memorize the data can be $O(\sqrt{n})$. They also show that if initialized near a global-min of the empirical loss, then SGD quickly finds a nearby point with much smaller loss. However, they did not mention how to find such an initialization strategy.

Convergence analysis of $O(n)$-width networks. Zhou et al. [78] studies the local convergence theory of mildly over-parameterized 1-hidden-layer networks. They show that, as long as the initial loss is low, GD converges to a zero-loss solution. They further propose an initialization strategy that provably returns a small loss under a mild assumption on the width. However, their proposed initialization may be costly to find since it requires solving an additional optimization problem.

Convergence analysis of narrow networks. A few recent works [10, 29, 51] showed that under a "γ-margin neural tangent separability" condition, GD converges to global minimizers for training 2-layer ReLU net with width [7] $\text{poly}(\log n, \gamma)$. For certain special data distributions where $\gamma = \text{poly}(\log n)$, their width is $\text{poly}(\log n, \log \frac{1}{\epsilon})$. Nevertheless, for general data distribution, their width can be larger than n.

Global landscape analysis. There are many works on the global landscape analysis; see, e.g., [15, 30, 32, 36–40, 44, 50, 52, 61, 62, 69, 76] and the surveys [64, 65]. For networks with width less than n, the positive result on the landscape either requires special activation like quadratic activation [62], or special data distribution like linearly separable or two-subspace data [38]. For certain non-quadratic activations, it was shown that sub-optimal strict local minima can exist for networks with width less than n [36]. These results are different from ours in the scope, since they discuss *global landscape* of unconstrained problems, while we discuss local landscape.

C More Discussion on the Related Work: Daniely [12]

Daniely [12] studies the expressivity of 1-hidden-layer networks. They provide two results under different scenarios: to memorize $n(1-\epsilon)$ random input-binary-label pairs via SGD, the required width is either

(i) $\tilde{O}\left(\frac{n}{\epsilon^2}\right)$ ([12, Theorem 5]); or

(ii) $\tilde{O}(n/d)$ ([12, Theorem 7]), where $\tilde{O}$ hides a factor that is dependent on n and d.

In this section, we will explain our claim in Section 2 that their required width can be much larger than ours.

Major differences In our work, our required width is smaller than n (when $d > 2$). However, in either result (i) or (ii), their required width is often much larger than n. In fact, their width is actually at least $O\left(n^2\right)$ or larger for fixed d, if we consider the effect of ϵ (for their first bound) or the hidden factor in $\tilde{O}$ (for their second bound).

We will elaborate below.

[7] Their width also depends on desired accuracy ϵ, and we skip the dependence here.

For their result (i), similar to Bubeck et al. [9], their required width grows with ϵ. As the authors pointed out before their Theorem 7 on page 8, the required width is $O(n^3)$ if $\epsilon \leq 1/d$. This is why they add result (ii) in [12, Theorem 7].

For their result (ii), i.e., [12, Theorem 7] their required width is roughly $O(n/d\,(\log(d\log n))^{\log n})$. Compared to the desired bound $O(n/d)$, their actual bound contains an additional factor of roughly $(\log(d\log n))^{\log n}$. We would like to stress that the additional factor is *not* a constant factor or a log-factor, but more like a "super-polynomial-factor". As a result, the required width is actually at least $O(n^2/d)$ or larger. The detailed explanations are provided next, and briefly summarized below.

- In Appendix C.1, we will explain why there exists an extra "$(\log(d\log n))^{\log n}$" term in their required width bound. In [12, Theorem 7], their statement only mentioned $\tilde{O}(n/d)$ but not the exact expression.
- In Appendix C.2, we will explain why their bound is at least $O(n^2)$ or larger for fixed d.
- In Appendix C.3, we will summarize some other differences.

C.1 Identifying a More Precise Width Bound

To find a more precise width bound of [12, Theorem 7], we tracked the proof of [12] as follows (note that their width q is our m and their sample size m is our n, and we will use our notation below).

The desired result. Their goal is to memorize the $n(1-\epsilon)$ random input-label pairs via SGD.

Notation. They consider binary labels, i.e., $y_i = \{+1, -1\}$ for $i = 1, \cdots, n$. They consider a feature vector $\Psi_{\boldsymbol{\omega}}(\mathbf{x}_i) \in \mathbb{R}^{dm}$, where $\boldsymbol{\omega} \in \mathbb{R}^{d\times m}$. The predictor (classifier) is in the form of $f_{\boldsymbol{\omega},\mathbf{v}}(\mathbf{x}) := \mathbf{v}^T\Psi_{\boldsymbol{\omega}}(\mathbf{x}_i)$, where $\mathbf{v} \in \mathbb{R}^{dm}$.

Step 1 The desired result holds if there exists $\mathbf{v}$ such that a certain condition holds.

More specifically, they have shown that the desired result holds if the following claim holds: there exists certain $\mathbf{v}$, such that w.h.p. over the dataset and $\boldsymbol{\omega} \in \mathbb{R}^{d\times m}$,

$$\langle \mathbf{v}, \Psi_{\boldsymbol{\omega}}(\mathbf{x}_i)\rangle = y_i + o(1), \quad \forall i = 1, \cdots, n. \tag{C.1}$$

Thus, the goal becomes to find $\mathbf{v}$ such that (C.1) holds. This goal is stated in the first paragraph of Section 4.3, page 15. Though they did not define it explicitly, $o(1)$ in (C.1) means " a constant smaller than 1".

The relation between (C.1) and the desired result is independent of the calculation of the width bound. Anyhow, for completeness, we briefly explain why (C.1) leads to the desired result (most readers can skip the paragraph). First, (C.1) implies that f memorized the dataset for this $(\mathbf{v}, \Psi_{\boldsymbol{\omega}})$. In fact, if (C.1) holds, then $f_{\boldsymbol{\omega},\mathbf{v}}(\mathbf{x}_i)$ has the same sign as $y_i + o(1)$. Note that $y_i + o(1)$ shares the same sign as y_i because y_i is assumed to be either $+1$ or -1 and $o(1)$ is a constant smaller than 1. Together we conclude that $f_{\boldsymbol{\omega},\mathbf{v}}(\mathbf{x}_i)$ shares the same sign as y_i for any i, which means the predictions $\text{sign}(f_{\boldsymbol{\omega},\mathbf{v}}(\mathbf{x}_i)) = y_i$. Second, such a $\mathbf{v}$ can be reached approximately by SGD (see Algorithm 2 on page 10 of [12]). This requires an argument that we skip here.

Step 2: There exists $\mathbf{v}$ which satisfies a relation specified below.

More specifically, they proved the following result.

Theorem C.1 (Theorem 16, page 16, Daniely [12])**.** *W.p. $1-\delta-2^{-\Omega(d)}$ over the choice of dataset and $\boldsymbol{\omega}$, there exists a $\mathbf{v}$ such that the following for all i:*

$$\langle \mathbf{v}, \Psi_{\boldsymbol{\omega}}(\mathbf{x}_i)\rangle = f_{\boldsymbol{\omega}}(\mathbf{x}_i) = y_i + O\left(\frac{(\log(d/\delta))^{\frac{c'}{2}}}{d}\right) + O\left(\sqrt{\frac{d^{c-1}(\log(d/\delta))^{c'+2}}{m}}\right). \tag{C.2}$$

Note that in Theorem C.1, they did not explicitly write out the expression of c and c'. The definition of these two constants can be seen in Section 3.2 and Section 4.3 of C.1 respectively. We will discuss them in Step 3.

Step 3: **Identifying a condition on m so that the obtained bound** (C.2) **in Step 2 implies the desired bound** (C.1) **in Step 1.**

We demonstrate the detailed derivation next.

Step 3.0: To achieve the goal of (C.1), the 3rd term of (C.2) needs to be no more than 1. This term is the crucial part to derive the width bound on m. We provide detailed analysis as follows.

Step 3.1: Now we need to make sure the 3rd term of (C.2) is no more than 1, which is equivalent to (ignoring constant-factors)

$$m \geq d^{c-1}(\log(d/\delta))^{(c'+2)}, \tag{C.3}$$

where $c' \geq 4c + 2$. This definition of c' can be seen in the first paragraph of Section 4.3 on page 15; c, δ are specified in the next two steps.

Step 3.2: Identify $\delta = 1/\log n$. This is specified in the paragraph below Theorem 16, page 16. Plugging it into (C.3), we have

$$m \geq d^{c-1}(\log(d \log n))^{(c'+2)}. \tag{C.4}$$

Step 3.3: Identify $c = \log n/\log d$. The definition of c first appeared in the first paragraph of Section 3.2, where they assume the number of samples is $n = d^c$. This definition is used in the first paragraph of Section 4.3, with a slightly different form $n/d = d^{c-1}$. This definition implies $c = \log n/\log d$. Further, we have

$$c' \geq 4c + 2 \geq 4c + 2 = 4 \log n/\log d + 2 \tag{C.5}$$

Plugging (C.5) and $d^{c-1} = n/d$ into (C.4), we have:

$$m \geq \frac{n}{d}(\log(d \log n))^{4 \log n/\log d+2}. \tag{C.6}$$

Finally, ignoring the numerical constants, the bound becomes

$$m \geq O(\frac{n}{d}(\log(d \log n))^{\log n/\log d}). \tag{C.7}$$

C.2 Why The Bound is "Super-Polynomial"

Now we explain why their bound (C.7) is at least $O(n^2)$ or larger for fixed d. The bound (C.7) is a bit complicated as it depends on both d and n. We are more interested in its dependence on n, thus we fix d and analyze how it scales with n. With fixed d, the exponent $\log n/\log d$ can be simplified to $\log n$, thus the bound (C.7) can be simplified to

$$m \geq O(\frac{n}{d}(\log(d \log n))^{\log n}). \tag{C.8}$$

We will show that this bound (C.7) is at least $O(n^2)$ or larger for fixed d.

Define $B_1 = (\log d)^{\log n}$ and $B_2 = [\log(\log n)]^{\log n}$. From (C.8) we obtain

$$m \geq O(\frac{n}{d} \max\{B_1, B_2\}). \tag{C.9}$$

The extra factor $\max\{B_1, B_2\}$ is *not* a constant factor or a log-factor, but more like a "super-polynomial" factor on n, as explained below.

- For B_1, consider the fixed input dimension d for two cases.
  - For d satisfying $\log d > 2.7$ (i.e. $d > 15$), their required width is actually $O\left(\frac{n^2}{d}\right)$. This is an order of magnitude larger than our bound $O(n/d)$.
  - For d satisfying $\log d > 2.7^2$ (i.e. $d > 1395$), the required width is actually $O\left(\frac{n^3}{d}\right)$, two orders of magnitude larger than $O(n/d)$.
- For B_2, when $n > 2.7^{2.7^{27}} \approx 2.2 \times 10^6$, we have $\left(\log(\log(n))^{\log n} > n\right.$. As a result, the required width is at least $O\left(\frac{n^2}{d}\right)$.

Theoretically speaking, it is not hard to prove that their required width can be larger than n^k for any fixed integer k (which is why we say their bound is "super-polynomial"). Empirically speaking, a calculation of their bound for a real dataset can reveal how large it is. On CIFAR-10 dataset [31], $n = 50000, d = 3072$. Plugging these numbers into (C.7) (ignore $O(\cdot)$) we obtain $m \geq 58290499136 >> n$. In comparison, our required width is only $m \geq 2n/d \approx 33$, which is much smaller than $n = 50000$.

In summary, rigorously speaking, their required width $\tilde{O}(n/d)$ is $\mathrm{not}\, O(n/d)$, but can be larger than $O\left(\frac{n^2}{d}\right)$ or even $O\left(\frac{n^3}{d}\right)$.

C.3 Other Differences

Besides the above discussion, there are some other differences between Daniely [12] and our work.

First, they analyze SGD, and we analyze a constrained optimization problem and projected SGD. This may be the reason why we can get a stronger bound on width. In the experiments in Section 5, we observe that SGD performs badly when the width is small (see the first left column in (b), Figure 4). Therefore, we suspect an algorithmic change is needed to train narrow nets with such width (due to the training difficulty), and we indeed propose a new method to train narrow nets.

Second, they consider binary $\{+1, -1\}$ dataset, while our results apply to arbitrary labels. In addition, their proof seems to be highly dependent on the fact that the labels are $\{+1, -1\}$, and seems hard to generalize to general labels.

D Definition and Notations

Before going through the proof details, we restate some of the important notations that will repeatedly appear in the proof, the following notations are also introduced in Section 4.1.

We denote $\{(x_i, y_i)\}_{i=1}^n \subset \mathbb{R}^d \times \mathbb{R}$ as the training samples, where $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$. For theoretical analysis, we focus on 1-hidden-layer neural networks $f(x;\theta) = \sum_{j=1}^m v_j \sigma\left(w_j^T x\right) \in \mathbb{R}$, where $\sigma(\cdot)$ is the activation function, $w_j \in \mathbb{R}^d$ and $v_j \in \mathbb{R}$ are the parameters to be trained. To learn such a neural network, we search for the optimal parameter $\theta = (w, v)$ by minimizing the empirical loss:

$$\min_\theta \ell(\theta) = \frac{1}{2} \sum_{i=1}^n \left(y_i - f\left(x_i;\theta\right)\right)^2.$$

Sometimes we also use $\ell(w; v)$ or $f(w; x, v)$ to emphasize the role of w.

We use the following shorthanded notations:

- $x := (x_1^T; \ldots; x_n^T) \in \mathbb{R}^{n\times d}$, $y := (y_1, \ldots, y_n)^T \in \mathbb{R}^n$;
- $w := (w_1, \ldots, w_m)_{j=1}^m \in \mathbb{R}^{d\times m}$, $v := (v_1, \ldots, v_m)_{j=1}^m \in \mathbb{R}^m$;
- $w_{a,b}$ indicates the b-th component of $w_a \in \mathbb{R}^d$;
- $\theta^0 = (w^0, v^0)$ indicates the initial parameters. Unless otherwise stated, it means the parameters at *the mirrored* LeCun's initialization given in Algorithm 1;
- $f(w; v) := (f(x_1; w, v), f(x_2; w, v), \ldots, f(x_n; w, v))^T \in \mathbb{R}^n$, indicating the neural network output on the whole dataset x.
- Define $\ell \circ f = \frac{1}{2}\|y - f\|_2^2$.

We denote the Jacobian matrix of $f(w; v)$ w.r.t w as

$$J(w;v) := \begin{bmatrix} \nabla_w f\left(w; x_1, v\right)^T \\ \vdots \\ \nabla_w f\left(w; x_n, v\right)^T \end{bmatrix} = \begin{bmatrix} v_1\sigma'\left(w_1^T x_1\right) x_1^T & \cdots & v_m\sigma'\left(w_m^T x_1\right) x_1^T \\ & \vdots & \\ v_1\sigma'\left(w_1^T x_n\right) x_n^T & \cdots & v_m\sigma'\left(w_m^T x_n\right) x_n^T \end{bmatrix} \in \mathbb{R}^{n\times md}.$$

We define the feature matrix

$$\Phi(w) := \begin{bmatrix} \sigma\left(w_1^T x_1\right), \ldots, \sigma\left(w_m^T x_1\right) \\ \vdots \\ \sigma\left(w_1^T x_n\right), \ldots, \sigma\left(w_m^T x_n\right) \end{bmatrix} \in \mathbb{R}^{n\times m}.$$

E Proof of Theorem 1

The proof of Theorem 1 consists of two parts: when the hidden weights w is in the neighborhood of the mirrored LeCun's initialization, we have (i) There exists a global-min with 0 loss, (ii) every

stationary point is a global-min. We prove these two arguments respectively. The first part (argument (i)) can be seen in Appendix E.1, the second part (argument (ii)) can be seen in Appendix E.2.

E.1 Proof of The First Part of Theorem 1

To prove the first part of Theorem 1, we use the Inverse Function Theorem (IFT) [18] at the mirrored LeCun's initialization. IFT is stated below.

Theorem E.1 (Inverse function theorem (IFT))**.** *Let $\psi : U \to \mathbb{R}^n$ be a C^1 -map where U is open in $\mathbb{R}^n$ and $w \in U$. Suppose that the Jacobian $J(w)$ is invertible. There exist open sets W and F containing w and $\psi(w)$ respectively, such that the restriction of ψ on W is a bijection onto F with a C^1 -inverse.*

Our overall proof idea is as follows: we will use IFT to show that: then for any $y \in \mathbb{R}^n$ and any small enough ϵ, there exists a $w^* \in B_\epsilon(w^0)$ whose prediction output $f(w^*; v^0) \propto y - f(w^0; v^0)$. Additionally, since $f(w^0; v^0) = 0$, we have $f(w^*; v^0) \propto y$. Once this is shown, then we just need to scale all the outer weight v_j uniformly and the output will be exactly y since $f(w^*; v)$ is linear in v. More details can be seen as follows.

In our case, let $\psi = f(w; v^0)$ be the function of w, mapping from $\mathbb{R}^{md}$ to $\mathbb{R}^n$. It may appears that IFT cannot be directly applied since $md \geq 2n$ (cf. Assumption 1), so $f(w; v^0)$ is not dimension-preserved mapping. However, this issue can be alleviated by applying the IFT to *a subvector* of w, while fixing the rest of the variables.

More specifically, we denote $n = k_1 d + k_2$ with $k_1, k_2 \in \mathbb{N}$, and $w = (\tilde{w}^T, \tilde{w}'^T)^T$, where $\tilde{w} = (w_1^T, \cdots, w_{k_1}^T, w_{k_1+1,1}, \cdots, w_{k_1+1,k_2})^T \in \mathbb{R}^n$ and $\tilde{w}' = (w_{k_1+1,k_2+1}, \cdots, w_{k_1+1,d}, w_{k_1+2}^T, \cdots, w_m^T)^T \in \mathbb{R}^{md-n}$. Here, $w_{a,b}$ indicates the b-th component of $w_a \in \mathbb{R}^d$.

We now apply IFT to $f(\tilde{w}; v^0, \tilde{w}'^0) \in \mathbb{R}^n$ (this notation views $\tilde{w}$ as the variable and $v^0, \tilde{w}'^0$ are treated as parameters). Firstly, in Lemma E.1, we prove that w.p.1, the corresponding Jacobian matrix $J(\tilde{w}^0; v^0, \tilde{w}'^0) \in \mathbb{R}^{n \times n}$ is of full rank at the mirrored LeCun's initialization, so that the condition for IFT holds. Then, by IFT, there exist open sets W and F containing $\tilde{w}^0$ and $f(\tilde{w}; v^0, \tilde{w}'^0)$ respectively, such that the restriction of $f(\tilde{w}; v^0, \tilde{w}'^0)$ on W is a bijection onto F. Here, we denote ϵ and δ as the radius of W and F, respectively.

Now, since $f(\tilde{w}^0; v^0, \tilde{w}'^0) = f(w^0; v^0) = 0 \in \mathbb{R}^n$, set F contains all possible directions pointed from the origin. That is to say, for any label vector $y \in \mathbb{R}^n$, we can always scale it using δ, such that $\delta \frac{y}{\|y\|} \in F$, and then, by IFT , there exists a $\tilde{w}^* \in B_\epsilon(\tilde{w}^0) \subset W$ satisfying

$$f(\tilde{w}^*; v^0, \tilde{w}'^0) = \delta \frac{y}{\|y\|}. \tag{E.1}$$

Since $\tilde{w}$ is just the truncated version of w, (E.1) implies: there exists a $w^* \in B_\epsilon(w^0)$, s.t.

$$f(w^*; v^0) = \delta \frac{y}{\|y\|}. \tag{E.2}$$

Now, we scale the outer weight to $v^* = \frac{\|y\|}{\delta} v^0$ and the output will be exactly y, i.e.:

$$f(w^*; v^*) = y.$$

Therefore, the proof is concluded.

Remark: "there exists an ϵ" or "any small ϵ"? Readers may mention that IFT states "there exists a neighborhood with size ϵ", however, Theorem 1 claims for "any small enough ϵ". We would like to clarify that the statement of Theorem 1 is *not* a typo. Here is the reason: in the statement of IFT, "existence of a small neighborhood" will imply "IFT holds for any subset of this neighborhood", so actually, Theorem 1 holds for any small (enough) neighborhood with size ϵ.

Lemma E.1. *Under Assumption 1, 2 and 3, as a function of w, v and x, $J(\tilde{w}^0; v^0, \tilde{w}'^0) \in \mathbb{R}^{n \times n}$ is of full rank at the mirrored LeCun's initialization, w.p.1..*

Proof of Lemma E.1. Recall the Jacobian matrix of $f(\tilde{w}; v^0, \tilde{w}'^0)$ w.r.t. $\tilde{w}$:

$$J(\tilde{w}^0; v^0, \tilde{w}'^0) := \begin{bmatrix} \nabla_{\tilde{w}} f\left(\tilde{w}^0; x_1, v^0, \tilde{w}'^0\right)^T \\ \vdots \\ \nabla_{\tilde{w}} f\left(\tilde{w}^0; x_n, v^0, \tilde{w}'^0\right)^T \end{bmatrix} \in \mathbb{R}^{n\times n}.$$

We first consider a general case where $k_2 \neq 0$. Since $f\left(\tilde{w}; x, v, \tilde{w}'\right) = f(w; x, v) = \sum_{j=1}^m v_j \sigma\left(w_j^T x\right)$, taking derivative w.r.t. $\tilde{w}$ yields $J(\tilde{w}^0; v^0, \tilde{w}'^0)$ equals to:

$$\begin{pmatrix} v_1^0\sigma'\left(w_1^{0T}x_1\right)x_1^T & \cdots & v_{k_1}^0\sigma'\left(w_{k_1}^{0T}x_1\right)x_1^T & v_{k_1+1}^0\sigma'\left(w_{k_1+1}^{0T}x_1\right)x_{1,1} & \cdots & v_{k_1+1}^0\sigma'\left(w_{k_1+1}^{0T}x_1\right)x_{1,k_2} \\ & & \vdots & & & \\ v_1^0\sigma'\left(w_1^{0T}x_n\right)x_n^T & \cdots & v_{k_1}^0\sigma'\left(w_{k_1}^{0T}x_n\right)x_n^T & v_{k_1+1}^0\sigma'\left(w_{k_1+1}^{0T}x_n\right)x_{n,1} & \cdots & v_{k_1+1}^0\sigma'\left(w_{k_1+1}^{0T}x_n\right)x_{n,k_2} \end{pmatrix}. \tag{E.3}$$

To prove the full-rankness of $J(\tilde{w}^0; v^0, \tilde{w}'^0)$, we need to show that w.p.1, $\det(J(\tilde{w}^0; v^0, \tilde{w}'^0)) \neq 0$. Here, $\det(J(\tilde{w}^0; v^0, \tilde{w}'^0))$ is an analytic function since the activation function $\sigma(\cdot)$ is analytic (see Assumption 2). Therefore, we borrow an important result of [47, Proposition 0] which states that the zero set of an analytic function is either the whole domain or zero-measure. The result is formally stated as the following lemma under our notation.

Lemma E.2. *Suppose that: as a function of $\tilde{w}$, $\tilde{w}'$, v and x, $\det(J(\tilde{w}^0; v^0, \tilde{w}'^0)) : \mathbb{R}^{md+m+nd} \to \mathbb{R}$ is a real analytic function on $\mathbb{R}^{md+m+nd}$. If $\det(J(\tilde{w}^0; v^0, \tilde{w}'^0))$ is not identically zero, then its zero set $\Omega = \left\{\tilde{w}^0, v^0, \tilde{w}'^0, x \mid \det(J(\tilde{w}^0; v^0, \tilde{w}'^0)) = 0\right\}$ has zero measure.*

Based on Lemma E.2, in order to prove $\det(J(\tilde{w}^0; v^0, \tilde{w}'^0)) \neq 0$ w.p.1, we only need to prove it is not identically zero. To do so, we first transform $\det(J(\tilde{w}^0; v^0, \tilde{w}'^0))$ into its equivalent form:

$$(E.3) \quad = \quad \det\left([B_1, \cdots, B_{k_2}, C_{k_2+1}, \cdots, C_d]\right), \tag{E.4}$$

where

$$B_j = \begin{pmatrix} v_1^0\sigma'\left(w_1^{0T}x_1\right)x_{1,j} & \cdots & v_{k_1+1}^0\sigma'\left(w_{k_1+1}^{0T}x_1\right)x_{1,j} \\ & \vdots & \\ v_1^0\sigma'\left(w_1^{0T}x_n\right)x_{n,j} & \cdots & v_{k_1+1}^0\sigma'\left(w_{k_1+1}^{0T}x_n\right)x_{n,j} \end{pmatrix} \in \mathbb{R}^{n\times(k_1+1)}, \quad j = 1, \cdots, k_2,$$

$$C_j = \begin{pmatrix} v_1^0\sigma'\left(w_1^{0T}x_1\right)x_{1,j} & \cdots & v_{k_1}^0\sigma'\left(w_{k_1}^{0T}x_1\right)x_{1,j} \\ & \vdots & \\ v_1^0\sigma'\left(w_1^{0T}x_n\right)x_{n,j} & \cdots & v_{k_1}^0\sigma'\left(w_{k_1}^{0T}x_n\right)x_{n,j} \end{pmatrix} \in \mathbb{R}^{n\times k_1}, \quad j = k_2+1, \cdots, d.$$

In addition, B_j can be further rewritten as:

$$B_j = \begin{pmatrix} B_{1,j} \\ \vdots \\ B_{k_2,j} \\ D_j \end{pmatrix} \in \mathbb{R}^{n\times(k_1+1)},$$

where for $i = 1, \cdots k_2$:

$$\begin{aligned} B_{i,j} \quad &= \begin{pmatrix} v_1^0\sigma'\left(w_1^{0T}x_{(k_1+1)(i-1)+1}\right)x_{(k_1+1)(i-1)+1,j} & \cdots & v_{k_1+1}^0\sigma'\left(w_{k_1+1}^{0T}x_{(k_1+1)(i-1)+1}\right)x_{(k_1+1)(i-1)+1,j} \\ & \vdots & \\ v_1^0\sigma'\left(w_1^{0T}x_{(k_1+1)i}\right)x_{(k_1+1)i,j} & \cdots & v_{k_1+1}^0\sigma'\left(w_{k_1+1}^{0T}x_{(k_1+1)i}\right)x_{(k_1+1)i,j} \end{pmatrix} \\ &\in \quad \mathbb{R}^{(k_1+1)\times(k_1+1)}, \end{aligned}$$

and

$$\begin{aligned} D_j \quad &= \begin{pmatrix} v_1^0\sigma'\left(w_1^{0T}x_{(k_1+1)k_2+1}\right)x_{(k_1+1)k_2+1,j} & \cdots & v_{k_1+1}^0\sigma'\left(w_{k_1+1}^{0T}x_{(k_1+1)k_2+1}\right)x_{(k_1+1)k_2+1,j} \\ & \vdots & \\ v_1^0\sigma'\left(w_1^{0T}x_n\right)x_{n,j} & \cdots & v_{k_1+1}^0\sigma'\left(w_{k_1+1}^{0T}x_n\right)x_{n,j} \end{pmatrix} \\ &\in \quad \mathbb{R}^{(n-(k_1+1)k_2)\times(k_1+1)}. \end{aligned}$$

Similarly, C_j can be further rewritten as:

$$C_j = \begin{pmatrix} C_{1,j} \\ \vdots \\ C_{k_2,j} \\ E_j \end{pmatrix} \in \mathbb{R}^{n\times(k_1)},$$

where for $i = 1, \cdots k_2$:

$$\begin{aligned} C_{i,j} &= \begin{pmatrix} v_1^0\sigma'\left(w_1^{0T}x_{(k_1+1)(i-1)+1}\right)x_{(k_1+1)(i-1)+1,j} & \cdots & v_{k_1}^0\sigma'\left(w_{k_1}^{0T}x_{(k_1+1)(i-1)+1}\right)x_{(k_1+1)(i-1)+1,j} \\ & \vdots & \\ v_1^0\sigma'\left(w_1^{0T}x_{(k_1+1)i}\right)x_{(k_1+1)i,j} & \cdots & v_{k_1}^0\sigma'\left(w_{k_1}^{0T}x_{(k_1+1)i}\right)x_{(k_1+1)i,j} \end{pmatrix} \\ &\in \mathbb{R}^{(k_1+1)\times k_1} \end{aligned}$$

and

$$\begin{aligned} E_j &= \begin{pmatrix} v_1^0\sigma'\left(w_1^{0T}x_{(k_1+1)k_2+1}\right)x_{(k_1+1)k_2+1,j} & \cdots & v_{k_1}^0\sigma'\left(w_{k_1}^{0T}x_{(k_1+1)k_2+1}\right)x_{(k_1+1)k_2+1,j} \\ & \vdots & \\ v_1^0\sigma'\left(w_1^{0T}x_n\right)x_{n,j} & \cdots & v_{k_1}^0\sigma'\left(w_{k_1}^{0T}x_n\right)x_{n,j} \end{pmatrix} \\ &\in \mathbb{R}^{(n-(k_1+1)k_2)\times k_1}. \end{aligned}$$

Therefore, we can rewrite $\det(J(\tilde{w}^0; v^0, \tilde{w}'^0))$ as the following form:

$$\det(J(\tilde{w}^0; v^0, \tilde{w}'^0)) = \begin{pmatrix} B_{1,1} & \cdots & B_{1,k_2} & C_{1,k_2+1} & \cdots & C_{1,d} \\ & & \vdots & & & \\ B_{k_2,1} & \cdots & B_{k_2,k_2} & C_{k_2,k_2+1} & \cdots & C_{k_2,d} \\ D_1 & \cdots & D_{k_2} & E_{k_2+1} & \cdots & E_d \end{pmatrix}. \tag{E.5}$$

Based on Lemma E.2, in order to prove $\det(J(\tilde{w}^0; v^0, \tilde{w}'^0)) \neq 0$ w.p.1., we only need to prove that, as a function of $\tilde{w}$, $\tilde{w}'$, v and x, $\det(J(\tilde{w}^0; v^0, \tilde{w}'^0))$ is not identically zero. To do so, we just need to construct a dataset x such that $(E.5) \neq 0$. We construct such $x := (x_1, \ldots, x_n) \in \mathbb{R}^{n\times d}$ in the following way:

(1) For $i = 1, \cdots, (k_1+1)$: $x_{i,j} = \begin{cases} \delta_i, & j = 1 \\ 0, & \text{otherwise} \end{cases}$, where $\delta_i \neq \delta_{i'} \neq 0, \forall i, i'$.

(2) For $i = (k_1+1)+1, \cdots, 2(k_1+1)$: $x_{i,j} = \begin{cases} \delta_i, & j = 2 \\ 0, & \text{otherwise} \end{cases}$, where $\delta_i \neq \delta_{i'} \neq 0$, $\forall i, i'$.

(3) $\cdots$

(4) For $i = (k_2-1)(k_1+1)+1, \cdots, k_2(k_1+1)$: $x_{i,j} = \begin{cases} \delta_i, & j = k_2 \\ 0, & \text{otherwise} \end{cases}$, where $\delta_i \neq \delta_{i'} \neq 0, \forall i, i'$.

(5) For $i = k_2(k_1+1)+1, \cdots, n$: $x_{i,j} = \begin{cases} 1, & j = i - k_1k_2 \\ 0, & \text{otherwise} \end{cases}$.

Under such a construction, (E.5) becomes the determinant of a block-diagonal matrix:

$$\det(J(\tilde{w}^0; v^0, \tilde{w}'^0)) = \det \begin{pmatrix} B_{1,1} & \cdots & 0 & 0 \\ & & \vdots & \\ 0 & \cdots & B_{k_2,k_2} & 0 \\ 0 & \cdots & 0 & E \end{pmatrix}, \tag{E.6}$$

where $E = [E_{k_2+1} \cdots E_d]$ is a square matrix in $\mathbb{R}^{(n-(k_1+1)k_2)\times(n-(k_1+1)k_2)}$. To prove $(E.6) \neq 0$, we need to prove $B_{1,1}, \cdots, B_{k_2,k_2}$ and E are all full rank matrices.

As for the full-rankness of E, thanks to the construction (5), $E = [E_{k_2+1} \cdots E_d]$ now becomes:

$$E_{k_2+1} = \begin{pmatrix} v_1^0 \sigma'\left(w_1^{0T} x_{(k_1+1)k_2+1}\right) & \cdots & v_{k_1}^0 \sigma'\left(w_{k_1}^{0T} x_{(k_1+1)k_2+1}\right) \\ 0 & \cdots & 0 \\ & \vdots & \\ 0 & \cdots & 0 \end{pmatrix} \in \mathbb{R}^{(n-(k_1+1)k_2)\times k_1},$$

$$E_{k_2+2} = \begin{pmatrix} 0 & \cdots & 0 \\ v_1^0 \sigma'\left(w_1^{0T} x_{(k_1+1)k_2+2}\right) & \cdots & v_{k_1}^0 \sigma'\left(w_{k_1}^{0T} x_{(k_1+1)k_2+2}\right) \\ & \vdots & \\ 0 & \cdots & 0 \end{pmatrix} \in \mathbb{R}^{(n-(k_1+1)k_2)\times k_1},$$

and so on so for:

$$E_d = \begin{pmatrix} 0 & \cdots & 0 \\ & \vdots & \\ 0 & \cdots & 0 \\ v_1^0 \sigma'\left(w_1^{0T} x_n\right) & \cdots & v_{k_1}^0 \sigma'\left(w_{k_1}^{0T} x_n\right) \end{pmatrix} \in \mathbb{R}^{(n-(k_1+1)k_2)\times k_1}.$$

Since $md \geq 2n$ and $J(\tilde{w}^0; v^0, \tilde{w}'^0) \in \mathbb{R}^{n\times n}$ is the jacobian w.r.t. the first n components of $w \in \mathbb{R}^{md}$, it only involves $w_1, w_2, \cdots w_{k_1+1}$ and it will not reach beyond $w_{\frac{m}{2}} \in \mathbb{R}^d$. Recall in the mirrored LeCun's initialization, we only copy the 2nd half of w: $(w^0_{\frac{m}{2}+1}, \ldots, w^0_m) \leftarrow (-w^0_1, \ldots, -w^0_{\frac{m}{2}})$, that is to say, $w_1, w_2, \cdots w_{k_1+1}$ are independent Gaussian random variables, unaffected by the copying phase, similarly for $v_1, \cdots, v_{k_1+1}$.

In short, since Gaussian random variables take value 0 on a zero probability measure, and $\sigma(z) = 0$ only happens when $z = 0$ (see Assumption 2), we have $E = [E_{k_2+1} \cdots E_d]$ is full rank w.p.1.

As for the full-rankness of B_{kk}, $k = 1, \cdots, k_2$, we only need to prove B_{11} is invertible, the proof of the rest of B_{kk} are the same.

Under construction (1), we have:

$$B_{1,1} = \begin{pmatrix} v_1^0 \sigma'\left(w_1^{0T} x_1\right) \delta_1 & \cdots & v_{k_1+1}^0 \sigma'\left(w_{k_1+1}^{0T} x_1\right) \delta_1 \\ & \vdots & \\ v_1^0 \sigma'\left(w_1^{0T} x_{(k_1+1)}\right) \delta_{k_1+1} & \cdots & v_{k_1+1}^0 \sigma'\left(w_{k_1+1}^{0T} x_{(k_1+1)}\right) \delta_{k_1+1} \end{pmatrix} \in \mathbb{R}^{(k_1+1)\times(k_1+1)}.$$

Again, since Gaussian random variables take value 0 on a zero probability measure, and $\delta_i \neq 0$ for $i = 1, \cdots, k_1 + 1$, we only need to prove the following $\tilde{B}_{1,1}$ is full rank:

$$\tilde{B}_{1,1} = \begin{pmatrix} \sigma'\left(w_1^{0T} x_1\right) & \cdots & \sigma'\left(w_{k_1+1}^{0T} x_1\right) \\ & \vdots & \\ \sigma'\left(w_1^{0T} x_{(k_1+1)}\right) & \cdots & \sigma'\left(w_{k_1+1}^{0T} x_{(k_1+1)}\right) \end{pmatrix} \in \mathbb{R}^{(k_1+1)\times(k_1+1)}.$$

Next, we borrow the following lemma from [35, Proposition 1], which is the restatement under our notation (their original statement applies for deep neural network, here, we restate it for 1-hidden-layer case in Lemma E.3).

Lemma E.3. *Under Assumption 2, given an 1-hidden-layer neural network with width $m \geq n$, Let $\Omega = \{w \mid \operatorname{rank}(\Phi(w)) < \min\{m, n\}\}$, where $\Phi(w)$ is the hidden feature matrix*

$$\Phi(w) := \begin{bmatrix} \sigma\left(w_1^T x_1\right), \ldots, \sigma\left(w_m^T x_1\right) \\ \vdots \\ \sigma\left(w_1^T x_n\right), \ldots, \sigma\left(w_m^T x_n\right) \end{bmatrix} \in \mathbb{R}^{n\times m}.$$

Suppose there exists a dimension k such that $x_{i,k} \neq x_{i',k}, \forall i \neq i'$, then Ω is a zero-measure set.

To prove the full-rankness of $\tilde{B}_{1,1}$, we regard it as the hidden feature matrix of an 1-hidden-layer neural network equipped with width $m' = k_1 + 1$ and activation function $\sigma'(z)$, which satisfies Assumption 2. In addition, recall $\delta_i \neq \delta_{i'} \neq 0$ for $\forall i, i'$, so $(x_1, \cdots, x_{k_1+1})$ satisfies the condition of Lemma E.3 w.p.1, and the sample size equals to the width $m' = k_1 + 1$. Therefore, all the assumptions are satisfied and Lemma E.3 directly shows that $\tilde{B}_{1,1}$ is invertible w.p.1..

Similarly, with the same proof technique, it can be shown that the rest of $B_{k,k}$ are also invertible w.p.1.. In conclusion, we have constructed a dataset x, such that (E.6) is non-zero w.p.1., which implies $\Omega = \left\{\tilde{w}^0, v^0, \tilde{w}'^0, x \mid \det(J(\tilde{w}^0; v^0, \tilde{w}'^0)) = 0\right\}$ has zero measure by Lemma E.2. In other words, under the joint distribution of $\tilde{w}^0$, v^0, $\tilde{w}'^0$ and x, $J(\tilde{w}^0; v^0, \tilde{w}'^0)$ is invertible w.p.1.. Recall $\tilde{w}^0$, v^0, and $\tilde{w}'^0$ all follow continuous distribution, furthermore, x also follows a continuous distribution (Assumption 3), so $J(\tilde{w}^0; v^0, \tilde{w}'^0)$ is still invertible w.p.1. under the distribution of x, so the whole proof is completed.

When $n = k_1 d$, or equivalently, $k_2 = 0$, things become easier and we just need to change the size of B_{kk} to $\mathbb{R}^{k_1 \times k_1}$, and there is no need to consider $C_{i,j}$, D_j and E_j, the rest of the proof is the same, we omit it for brevity. □

E.2 Proof of The Second Part of Theorem 1

Suppose we have a 1-hidden-layer neural network $f(x; \theta) = \sum_{j=1}^{m} v_j \sigma\left(w_j^T x\right) \in \mathbb{R}$, and let $f(\theta) \in \mathbb{R}^n$ be output of $(x; \theta)$ on the dataset $x = (x_1, \cdots, x_n)$, let $J(w^*; v^*) \in \mathbb{R}^{n \times md}$ be its Jacobian matrix w.r.t. w at the stationary point $\theta^* = (w^*, v^*)$, we have:

$$\nabla_w \ell(\theta^*) = J(w^*; v^*)^T (f(\theta^*) - y) = 0. \tag{E.7}$$

Therefore, as long as we can prove that $J(w^*; v^*)^T \in \mathbb{R}^{md \times n}$ is full column rank, the stationary point θ^* will become a global minimizer with $\ell(\theta^*) = 0$, so the proof is completed.

Now we prove the full-rankness of $J(w^*; v^*)$. Recall in Lemma E.1, we have proved that, as a function of x, $\det(J(\tilde{w}^0; v^0, \tilde{w}'^0)) \neq 0$ w.p.1., where $J(\tilde{w}^0; v^0, \tilde{w}'^0)$ equals to

$$\begin{pmatrix} v_1^0 \sigma'\left(w_1^{0T} x_1\right) x_1^T & \cdots & v_{k_1}^0 \sigma'\left(w_{k_1}^{0T} x_1\right) x_1^T & v_{k_1+1}^0 \sigma'\left(w_{k_1+1}^{0T} x_1\right) x_{1,1} & \cdots & v_{k_1+1}^0 \sigma'\left(w_{k_1+1}^{0T} x_1\right) x_{1,k_2} \\ & & \vdots & & & \\ v_1^0 \sigma'\left(w_1^{0T} x_n\right) x_n^T & \cdots & v_{k_1}^0 \sigma'\left(w_{k_1}^{0T} x_n\right) x_n^T & v_{k_1+1}^0 \sigma'\left(w_{k_1+1}^{0T} x_n\right) x_{n,1} & \cdots & v_{k_1+1}^0 \sigma'\left(w_{k_1+1}^{0T} x_n\right) x_{n,k_2} \end{pmatrix}, \tag{E.8}$$

Since $\|w^* - w^0\|_F \leq \epsilon$ and the determinant is a continuous function of w, we have $\det(J(\tilde{w}^*; v^0, \tilde{w}'^*)) \neq 0$ (w.p.1.) when ϵ is small. In addition, as we can see from (E.8), $(v_1^0, \cdots, v_{k_1+1}^0)$ are just constant terms in the corresponding columns, so the full-rankness of (E.8) still holds if we change v_j^0 to $v_j^* \neq 0$. In summary, $\det(J(\tilde{w}^*; v^*, \tilde{w}'^*)) \neq 0$ when v^* is entry-wise non-zero, where $J(\tilde{w}^*; v^*, \tilde{w}'^*)$ equals to

$$\begin{pmatrix} v_1^* \sigma'\left(w_1^{*T} x_1\right) x_1^T & \cdots & v_{k_1}^* \sigma'\left(w_{k_1}^{*T} x_1\right) x_1^T & v_{k_1+1}^* \sigma'\left(w_{k_1+1}^{*T} x_1\right) x_{1,1} & \cdots & v_{k_1+1}^* \sigma'\left(w_{k_1+1}^{*T} x_1\right) x_{1,k_2} \\ & & \vdots & & & \\ v_1^* \sigma'\left(w_1^{*T} x_n\right) x_n^T & \cdots & v_{k_1}^* \sigma'\left(w_{k_1}^{*T} x_n\right) x_n^T & v_{k_1+1}^* \sigma'\left(w_{k_1+1}^{*T} x_n\right) x_{n,1} & \cdots & v_{k_1+1}^* \sigma'\left(w_{k_1+1}^{*T} x_n\right) x_{n,k_2} \end{pmatrix}, \tag{E.9}$$

Furthermore, $J(\tilde{w}^*; v^*, \tilde{w}'^*) \in \mathbb{R}^{n \times n}$ is nothing but a $n \times n$ submatrix of $J(w^*; v^*) \in \mathbb{R}^{n \times md}$. Now that $md \geq 2n > n$, $\det(J(\tilde{w}^*; v^*, \tilde{w}'^*)) \neq 0$ implies the full-row-rankness of $J(w^*; v^*)$ (w.p.1.). Thus the whole proof is completed.

F Proof of Theorem 2

In this section, we provide both proof sketch and the detailed proof of Theorem 2. They can be seen in Appendix F.1 and Appendix F.2, respectively. For general readers, reading proof sketch in Appendix F.1 will help grasp our main idea.

F.1 Proof Sketch of Theorem 2

Proof sketch. The proof is built on the special structure of neural network $f(x;\theta)$, including the linear dependence of v and the mirrored pattern of parameters. Here, we describe our high level idea and the analysis roadmap. The proof consists of proving the following claims:

(I) every KKT point θ^* satisfies $\|\nabla_w \ell(w^*;v^*)\| = O(\epsilon)$;

(II) the gradient of w always dominates the error term, i.e. $\|\nabla_w \ell(w;v)\|_2^2 = \Omega\left(\ell(w^*;v^*)\right)$, so we have $\ell(w^*;v^*) = O(\epsilon^2)$.

To prove claim (I), we only need to consider the case where w^* is on the boundary and $\|\nabla_w \ell(w^*;v^*)\|_2 \neq 0$ (otherwise Theorem 2 automatically holds based on Theorem 1). In this case, by the optimality condition, taking $\eta \in \mathbb{R}$ as a small step size, we have

$$-\eta \nabla_w \ell(w^*;v^*) = \eta \frac{\|\nabla_w \ell(w^*;v^*)\|}{\|w^* - w^0\|}(w^* - w^0) = \tilde{\eta}(w^* - w^0), \tag{F.1}$$

where $\tilde{\eta} = \eta \frac{\|\nabla_w \ell(w^*;v^*)\|}{\|w^* - w^0\|}$. Now, our key observation is that, after moving along (F.1) from w^*, the change of the loss is not significant due to the special local structure of $f(w;v^*)$, therefore, $\|\nabla_w \ell(w^*;v^*)\|_2$ can be bounded. To be more specific, we denote f^* as the neural network output at the KKT point θ^*; denote $\bar{f}$ as the neural network output after taking a small step η along the negative partial gradient direction of w; and denote f' as an rough estimate of $\bar{f}$:

$$f^* := f\left(w^*;v^*\right) = J(w^0;v^*)w^* + R^*, \tag{F.2}$$

$$\bar{f} := f\left(w^* - \eta \nabla_w \ell(w^*;v^*);v^*\right) = (1+\tilde{\eta})f^* - (1+\tilde{\eta})R^* + \bar{R}, \tag{F.3}$$

$$f' := \bar{f}_{\text{lin}} + (1+\tilde{\eta})R^* = (1+\tilde{\eta})f^*, \tag{F.4}$$

where $\bar{f}_{\text{lin}}$ is the first-order Taylor approximation of $\bar{f}$, R^* is the second-order Taylor residue of f^* (similarly for $\bar{R}$). Additionally, (F.2), (F.3), and (F.4) are due to the the symmetric property of w^0, v^0 and v^*, so the bias terms in the Taylor expansion will vanish. Now, we compare the value of the loss on each of f^*, $\bar{f}$ and f'. Define $\ell \circ f = \frac{1}{2}\|y - f\|_2^2$, we prove the following crucial relationship:

$$\eta\|\nabla_w \ell(w^*;v^*)\|_2^2 \overset{(a)}{\leq} \ell\circ f^* - \ell\circ \bar{f} \overset{(b)}{\leq} \ell\circ f' - \ell\circ \bar{f} = \frac{1}{2}\left\|f' - \bar{f}\right\|_2 \left\|f' + \bar{f} - 2y\right\|_2 \overset{(c)}{=} O(\tilde{\eta}\epsilon^2) \tag{F.5}$$

Eq. (F.5) plays a key role in our analysis. Here, (a) can be easily shown by applying Descent lemma in this local region, yet (b) and (c) are not that obvious. Recall in (F.2), (F.3), and (F.4), we know that: (i) although the location of $\bar{f}$ is unclear, f' points at the same direction as f^*. (ii) As an estimator of $\bar{f}$, f' is not far away from it, i.e. $\|\bar{f} - f'\|_2 = \|\bar{R} - (1+\tilde{\eta})R^*\|_2$ only involves the second-order Taylor residue terms. With this observation, (b) is proved in Lemma F.1 (stated below) by geometric properties, and (c) is calculated in Lemma F.2 (stated below), so the relationship (F.5) is proved. Therefore, we have θ^* satisfies $\|\nabla_w \ell(w^*;v^*)\| = O(\epsilon)$ by plugging in $\eta = \frac{\epsilon\tilde{\eta}}{\|\nabla_w \ell(w^*;v^*)\|_2}$, and claim (I) is proved.

Lemma F.1. *Under the conditions of Theorem 2, we have $\ell\circ f' \geq \ell\circ f^*$, i.e., $\|f' - y\|_2^2 \geq \|f^* - y\|_2^2$.*

Lemma F.2. *Under the conditions of Theorem 2, we have: $\left\|f' - \bar{f}\right\|_2 \left\|f' + \bar{f} - 2y\right\|_2 = O(\epsilon^2)$.*

As for claim (II), it is true as long as $J(w;v)$ is of full row rank, which has been shown in Theorem 1. A more detailed proof of Lemma F.1 and F.2, as well as the proof of the whole Theorem 2 are in Appendix F.2.

Proof sketch of Lemma F.1. Here, we provide a proof sketch of Lemma F.1, we need to discuss the following cases:

(1) When $f^{*T}y \geq 0$: we prove by contradiction. Since f^* is linear in v^*, $\|(1+\tilde{\eta})f^* - y\|_2^2 < \|f^* - y\|_2^2$ implies $\|f(w^*;(1+\tilde{\eta})v^*) - y\|_2^2 < \|f^* - y\|_2^2$, which means we can further reduce the loss by changing v^* to $(1+\tilde{\eta})v^*$, which is still feasible, we have a contradiction to the assumption that $(w,^* v^*)$ is a KKT point (see Figure 6, Middle).

(2) When $f^{*T}y < 0$: changing v^* to $-v^*$ will further reduce the distance to y (see Figure 6, Left), this is a contradiction to the fact that v^* is a KKT point, so case (2) will not happen.

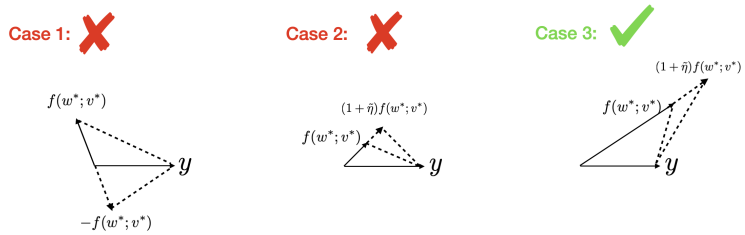


Figure 6: Geometrical illustration of three cases in Lemma F.1: comparing with f^*, $f' = (1+\tilde{\eta})f^*$ will not further reduce the distance to y.

In conclusion, we always have $\ell \circ f' \geq \ell \circ f^*$ (see Figure 6, Right). □

F.2 Detailed Proof of Theorem 2

The proof of Theorem 2 consists of proving the following claims: under the setting of Theorem 2,

(I) every KKT point θ^* satisfies $\|\nabla_w \ell(w^*; v^*)\| = O(\epsilon)$.

(II) the gradient of w always dominates the error term, i.e. $\|\nabla_w \ell(w; v)\|_2^2 = \Omega\left(l(w^*; v^*)\right)$, so we have $\ell(w^*; v^*) = O(\epsilon^2)$.

Note that the statement of claim (I) is not precise, there are chances that the constant terms will exponentially grow (will be discussed later). Nevertheless, it is just an intermediate result that helps provide a clearer big picture. our final result in claim (II) will be precise.

To prove claim (I), we only need to consider the case when w^* is on the boundary of the constraint $B_\epsilon(w^0)$ (Theorem 2 automatically holds when w^* is in the interior of $B_\epsilon(w^0)$).

Now, suppose w^* is a non-zero-gradient KKT point on the boundary, by the optimality condition, its negative gradient direction should be along the same direction as $w^* - w^0$, therefore, if we further take a small step η along the negative gradient direction, we have:

$$-\eta \nabla_w \ell(w^*; v^*) = \eta \frac{\|\nabla_w \ell(w^*; v^*)\|_2}{\|w^* - w^0\|_2}(w^* - w^0) = \tilde{\eta}(w^* - w^0), \tag{F.6}$$

where $\tilde{\eta} := \eta \frac{\|\nabla_w \ell(w^*; v^*)\|_2}{\|w^* - w^0\|_2} = \eta \frac{\|\nabla_w \ell(w^*; v^*)\|_2}{\epsilon}$. In this case, the loss function will decrease when we further move w along $-\nabla_w \ell(w^*; v^*)$ with a sufficiently small stepsize η. In other words, we can apply Descent lemma (details can be seen in Bertsekas et al. [6]) in this local region, i.e.

$$\ell\left(w^* - \eta \nabla_w \ell(w^*; v^*); v^*\right) - \ell(w^*; v^*) \quad \leq \quad -\eta \|\nabla_w \ell(w^*; v^*)\|_2^2. \tag{F.7}$$

As a matter of fact, after further taking a small GD step at w^*, $f\left(w^* - \eta \nabla_w \ell(w^*; v^*); v^*\right)$ will be closer to the groundtruth y, we will come back to this fact later, it will be used to bound $\|\nabla_w \ell(w^*; v^*)\|_2$.

Now, for any $w \in B_\epsilon(w^0)$, we take the Taylor expansion of $f(w; v^*)$ at w^0:

$$f(w; v^*) \quad = \quad f(w^0; v^*) + J(w^0; v^*)(w - w^0) + R(w) \tag{F.8}$$

$$\overset{\text{(a) \& (b)}}{=} \quad J(w^0; v^*)w + R(w), \tag{F.9}$$

$$f_{\text{lin}}(w; v^*) \quad := \quad f(w^0; v^*) + J(w^0; v^*)(w - w^0) \tag{F.10}$$

$$\overset{\text{(a) \& (b)}}{=} \quad J(w^0; v^*)w, \tag{F.11}$$

where $R(w) \in \mathbb{R}^n$ is the residue term of the Taylor expansion:

$$[R(w)]_i = \int_0^1 (w - w^0)^T H_i(w^0 + t(w - w^0))(w - w^0)(1 - t)dt, \quad i = 1, \cdots, n, \tag{F.12}$$

where $H_i(w)$ is the Hessian matrix of $f(w; v^*, x_i)$ at w (for simplicity, we drop the dependence of v^* and x_i in the notation), and $f_{\text{lin}}(w; v^*)$ is a linear approximation of $f(w; v^*)$. In addition, (a) & (b) is due to the fact that w_j^0 follows the mirrored LeCun's initialization with $(w_{\frac{m}{2}+1}^0, \ldots, w_m^0) = (w_1^0, \ldots, w_{\frac{m}{2}}^0)$, recall the construction of $f(w; v)$ in (6), the hidden output will cancel out with the outer weight, so we have (a):

$$f(w^0; v^*) \overset{(6)}{=} \sum_{j=1}^{\frac{m}{2}} v_j^* \left(\sigma(w_j^{0T} x) - \sigma(w_{j+\frac{m}{2}}^{0T} x)\right) = 0. \tag{F.13}$$

Similarly, we have (b):

$$J(w^0; v^*) w^0 = \nabla_w f\left(w^0; x, v^*\right)^T w^0 = \sum_{j=1}^{\frac{m}{2}} v_j^* \left(\sigma'(w_j^{0T} x) w_j^{0T} x - \sigma'(w_{j+\frac{m}{2}}^{0T} x) w_{j+\frac{m}{2}}^{0T} x\right) = 0. \tag{F.14}$$

Based on (F.9), we define the following quantities:

$$\bar{f} := f\left(w^* - \eta \nabla_w \ell(w^*; v^*); v^*\right) \overset{(F.9)}{=} J(w^0; v^*)(w^* - \eta \nabla_w \ell(w^*; v^*)) + \bar{R}, \tag{F.15}$$

$$f^* := f\left(w^*; v^*\right) \overset{(F.9)}{=} J(w^0; v^*) w^* + R^*, \tag{F.16}$$

where $\bar{R} = R\left(w^* - \eta \nabla_w \ell(w^*; v^*)\right)$, $R^* = R\left(w^*\right)$ as it is introduced in (F.12). Additionally, $\bar{f}$ can be re-written in the form of f^*:

$$\begin{aligned}
\bar{f} &\overset{(F.15)}{=} J(w^0; v^*)(w^* - \eta \nabla_w \ell(w^*; v^*)) + \bar{R} && \text{(F.17)}\\
&= J(w^0; v^*) w^* - \eta J(w^0; v^*) \nabla_w \ell(w^*; v^*) + \bar{R} && \text{(F.18)}\\
&\overset{(F.16)}{=} f^* - R^* - \eta J(w^0; v^*) \nabla_w \ell(w^*; v^*) + \bar{R} && \text{(F.19)}\\
&\overset{(F.6)}{=} f^* - R^* + \tilde{\eta} J(w^0; v^*)(w^* - w^0) + \bar{R} && \text{(F.20)}\\
&\overset{(F.14)}{=} f^* - R^* + \tilde{\eta} J(w^0; v^*)(w^*) + \bar{R} && \text{(F.21)}\\
&\overset{(F.16)}{=} f^* - R^* + \tilde{\eta}(f^* - R^*) + \bar{R} && \text{(F.22)}\\
&= (1 + \tilde{\eta}) f^* - (1 + \tilde{\eta}) R^* + \bar{R}. && \text{(F.23)}
\end{aligned}$$

Now, we construct a rough estimator of $\bar{f}$ by merely adding a residue term $(1 + \tilde{\eta}) R^*$ on $\bar{f}_{\text{lin}}$, which is f' defined as follows:

$$\begin{aligned}
f' &:= \bar{f}_{\text{lin}} + (1 + \tilde{\eta}) R^* && \text{(F.24)}\\
&= f_{\text{lin}}\left(w^* - \eta \nabla_w \ell(w^*; v^*); v^*\right) + (1 + \tilde{\eta}) R^* && \text{(F.25)}\\
&\overset{(F.11)}{=} J(w^0; v^*)(w^* - \eta \nabla_w \ell(w^*; v^*)) + (1 + \tilde{\eta}) R^* && \text{(F.26)}\\
&= J(w^0; v^*) w^* - \eta J(w^0; v^*) \nabla_w \ell(w^*; v^*) + (1 + \tilde{\eta}) R^* && \text{(F.27)}\\
&\overset{(F.16)}{=} f^* - R^* + \eta J(w^0; v^*) \nabla_w \ell(w^*; v^*) + (1 + \tilde{\eta}) R^* && \text{(F.28)}\\
&\overset{(F.6)}{=} f^* - R^* + \tilde{\eta} J(w^0; v^*)(w^* - w^0) + (1 + \tilde{\eta}) R^* && \text{(F.29)}\\
&\overset{(F.14)}{=} f^* - R^* + \tilde{\eta} J(w^0; v^*)(w^*) + (1 + \tilde{\eta}) R^* && \text{(F.30)}\\
&\overset{(F.16)}{=} f(w^*; v^*) - R^* + \tilde{\eta}(f^* - R^*) + (1 + \tilde{\eta}) R^* && \text{(F.31)}\\
&= (1 + \tilde{\eta}) f^*. && \text{(F.32)}
\end{aligned}$$

According to the descent property in (F.7), after taking a very small GD step at w^* , the new loss function $\ell \circ \bar{f}$ will be smaller than the old one $\ell \circ f^*$, where $\ell \circ f = \frac{1}{2}\|y - f\|_2^2$. In contrast, we

discuss the change of the loss function from f^* to that of the rough estimator f'. The following Lemma F.1 shows that $\ell \circ f' \geq \ell \circ f^*$, different from the fact that $\ell \circ \bar{f} \leq \ell \circ f^*$.

Lemma F.1. *[Corresponding to Lemma F.1 in the proof sketch.] Under the background of Theorem 2 and the definition of f' & f^* in (F.32) & (F.16), we have $\ell \circ f' \geq \ell \circ f^*$, i.e., $\|f' - y\|_2^2 \geq \|f^* - y\|_2^2$.*

The proof of Lemma F.1 can be seen in Appendix F.3.

Now, we have

$$\begin{aligned}
\eta \|\nabla_w \ell(w^*; v^*)\|_2^2 &\overset{(F.7)}{\leq} \ell(w^*; v^*) - \ell\left(w^* - \eta \nabla_w \ell(w^*; v^*); v^*\right) && \text{(F.33)}\\
&= \frac{1}{2}\|f^* - y\|_2^2 - \frac{1}{2}\left\|\bar{f} - y\right\|_2^2 && \text{(F.34)}\\
&\overset{\text{Lemma}F.1}{\leq} \frac{1}{2}\|f' - y\|_2^2 - \frac{1}{2}\left\|\bar{f} - y\right\|_2^2 && \text{(F.35)}\\
&= \frac{1}{2}\left\|f' - \bar{f}\right\|_2 \left\|f' + \bar{f} - 2y\right\|_2. && \text{(F.36)}
\end{aligned}$$

Next, we bound $\left\|f' - \bar{f}\right\|_2 \left\|f' + \bar{f} - 2y\right\|_2$ using the following Lemma F.2.

Lemma F.2. *[Corresponding to Lemma F.2 in the proof sketch.] Under the background of Theorem 2 and the definition of f' & $\bar{f}$ in (F.32) & (F.15), we have:*

$$\left\|f' - \bar{f}\right\|_2 \left\|f' + \bar{f} - 2y\right\|_2 \leq \left(\tilde{\eta}\epsilon^2 \lambda_{[0:2]}\sqrt{20n}\right)\left(\sqrt{n}C_y + 3\left(n\left(\frac{m}{2}L\zeta\epsilon\right)^2 + nC_y^2\right)^{\frac{1}{2}}\right), \tag{F.37}$$

where ζ is the constraint for v required in $B(v)$: $v \geq \zeta\mathbf{1}$; $\lambda_{[a:b]} := \max_i\{\lambda_{i[a:b]} | i = 1, \cdots, n\}$, and each $\lambda_{i[a:b]}$ is the maximum eigenvalue of the Hessian matrices $H_i(w^0 + t(w^* - w^0)) := \nabla_w^2 f(w^0 + t(w^* - w^0); v^*, x_i)$ in the interval $t \in [a, b]$.

The proof of Lemma F.2 can be seen in Appendix F.4. Now, with the help of Lemma F.2, we have:

$$\begin{aligned}
\eta \|\nabla_w \ell(w^*; v^*)\|_2^2 &\overset{(F.36)}{\leq} \frac{1}{2}\left\|f' - \bar{f}\right\|_2 \left\|f' + \bar{f} - 2y\right\|_2 && \text{(F.38)}\\
&\overset{(F.37)}{\leq} \frac{1}{2}\left(\tilde{\eta}\epsilon^2 \lambda_{[0:2]}\sqrt{20n}\right)\left(\sqrt{n}C_y + 3\left(n\left(\frac{m}{2}L\zeta\epsilon\right)^2 + nC_y^2\right)^{\frac{1}{2}}\right). && \text{(F.39)}
\end{aligned}$$

Recall $\eta = \frac{\epsilon}{\|\nabla_w \ell(w^*; v^*)\|_2}\tilde{\eta}$ and $\tilde{\eta}$ is sufficiently small, we have:

$$\|\nabla_w \ell(w^*; v^*)\|_2 \leq \frac{1}{2}\epsilon\left(\lambda_{[0:2]}\sqrt{20n}\right)\left(\sqrt{n}C_y + 3\left(n\left(\frac{m}{2}L\zeta\epsilon\right)^2 + nC_y^2\right)^{\frac{1}{2}}\right). \tag{F.40}$$

Since ζ can be chosen arbitrarily small, we can choose it to be smaller than $\frac{1}{\epsilon}$. That is to say, $\|\nabla_w \ell(w^*; v^*)\|_2 = O(\epsilon)$, so the claim (I) is proved. Note that to be precise, the constant on the right hand side of the above inequality depends on $\lambda_{[0:2]}$, which is a linear function of v, and the latter vector may not be upper bounded. Nevertheless, claim (I) is just an intermediate result, and our subsequent derivation will directly use the right hand side of (F.40), which is precise.

Now, we build the relationship between $\|\nabla_w \ell(w^*; v^*)\|_2$ and the loss function. Here, we need to eliminate the dependence of v in the final result: since there is no uniform upper bound for v, it can potentially make $\lambda_{[0:2]}$ grow exponentially. To alleviate this issue, we utilize the fact that $f(x; \theta)$ is linear in v, so $\lambda_{[0:2]}$ is also linear in v, and then we manage to remove the dependence of v in our final result. Specifically, let us define

$$v_{\text{min}}^* := \min_j\{v_j^* \mid j = 1, \cdots, m\}, \quad v_{\text{max}}^* := \max_j\{v_j^* \mid j = 1, \cdots, m\}.$$

So we have

$$\begin{aligned}
\|\nabla_w \ell(w^*; v^*)\|_2^2 &= \|J(w^*; v^*)^T(y - f^*)\|_2^2 \overset{(c)}{\geq} v_{\text{min}}^{*2} \cdot \|\tilde{J}(w^*; v^*)^T(y - f^*)\|_2^2 && \text{(F.41)}\\
&\geq v_{\text{min}}^{*2} \cdot \tilde{\lambda}_{\text{min}}^{*2} \cdot \|y - f^*\|_2^2, && \text{(F.42)}
\end{aligned}$$

where $\tilde{\lambda}^*_{\text{min}}$ is the smallest singular value of $\tilde{J}(w^*; v^*)$, and $\tilde{J}(w^*; v^*)$ is:

$$\tilde{J}(w^*; v^*) := \begin{pmatrix} \sigma'\left(w_1^{*T} x_1\right) x_1^T & \cdots & \sigma'\left(w_m^{*T} x_1\right) x_1^T \\ & \vdots & \\ \sigma'\left(w_1^{*T} x_n\right) x_n^T & \cdots & \sigma'\left(w_m^{*T} x_n\right) x_n^T \end{pmatrix} \in \mathbb{R}^{n \times md}.$$

Here, (c) is straightforward because $\tilde{J}(w^*; v^*)$ is just the simplified version of $J(w^*; v^*)$ by removing all the coefficient v_j^*; furthermore, $\tilde{J}(w^*; v^*)$ is full row rank because (i) $J(w^*; v^*)$ is proved to be full rank in the second part of Theorem 1 in Appendix E.2, (ii) v^* is entry-wise non-zero, so $\tilde{\lambda}^*_{\text{min}}$ is strictly positive.

Now, we need to remove the dependence of v in the right hand side of (F.40). Similarly as before, $\lambda_{[a:b]}$ can also be bounded by $v^*_{max}\tilde{\lambda}_{[a:b]}$, where $\tilde{\lambda}_{[a:b]}$ is equal to $\max_i\{\tilde{\lambda}_{i[a:b]} | i = 1, \cdots, n\}$, and each $\tilde{\lambda}_{i[a:b]}$ is the maximum eigenvalue of the Hessian matrices $\tilde{H}_i(w^0 + t(w^* - w^0)) := \nabla_w^2 \tilde{f}(w^0 + t(w^* - w^0); , x_i)$ in the interval $t \in [a, b]$, and $\tilde{f}(w; , x_i) := \sum_{j=1}^m \sigma(x_i^T w_j)$ is the simplified version of $f(w; v, x_i)$ by removing all the coefficient v_j. In conclusion, we have

$$\begin{aligned} v^*_{\text{min}}\tilde{\lambda}^*_{\text{min}}\|y - f^*\|_2 &\leq \|\nabla_w \ell(w^*; v^*)\|_2 && \text{(F.43)} \\ &\leq \frac{1}{2}\epsilon\left(\lambda_{[0:2]}\sqrt{20n}\right)\left(\sqrt{n}C_y + 3\left(n\left(\frac{m}{2}L\zeta\epsilon\right)^2 + nC_y^2\right)^{\frac{1}{2}}\right) && \text{(F.44)} \\ &\leq \frac{1}{2}\epsilon\left(v^*_{\text{max}}\tilde{\lambda}_{[0:2]}\sqrt{20n}\right)\left(\sqrt{n}C_y + 3\left(n\left(\frac{m}{2}L\zeta\epsilon\right)^2 + nC_y^2\right)^{\frac{1}{2}}\right). && \text{(F.45)} \end{aligned}$$

Rearrange and take the square on both sides, we get $\|y - f^*\|_2^2 = O\left(\kappa^2\epsilon^2\right)$, where κ is the finite constant in the constraint $B(v)$ of problem (7). So $\ell(\theta^*) = O(\epsilon^2)$, the proof of Theorem 2 is completed.

F.3 Proof of Lemma F.1

To prove Lemma F.1, we need to discuss the following cases:

(i) When $f^{*T}y \geq 0$, we prove Lemma F.1 by contradiction: when $\tilde{\eta}$ is sufficiently small, suppose $\|(1 + \tilde{\eta})f^* - y\|_2^2 < \|f^* - y\|_2^2$, then (w^*, v^*) is not a KKT point (this case corresponds to the Figure 6 (Middle)). Note that in problem (7), $f^* = \sum_{j=1}^{\frac{m}{2}} v_j^* \left(\sigma(w_j^{*T}x) - \sigma(w_{j+\frac{m}{2}}^{*T}x)\right)$ is linear in $(v_1^*, \cdots, v_{\frac{m}{2}}^*)$, so $(1 + \tilde{\eta})f^* = f(w^*; (1 + \tilde{\eta})v^*)$. That is to say, $\|(1 + \tilde{\eta})f^* - y\|_2^2 < \|f^* - y\|_2^2$ implies $\|f(w^*; (1 + \tilde{\eta})v^*) - y\|_2^2 < \|f^* - y\|_2^2$, which means we can further reduce the loss by changing $v^* \to (1 + \tilde{\eta})v^*$. Since $(1 + \tilde{\eta})v^*$ is still feasible if v^* is feasible, we have a contradiction to the assumption that $(w,^* v^*)$ is a KKT point.

Therefore, in case (i), moving from f^* to $f' = (1+\tilde{\eta})f^*$ will not further reduce the loss (this case corresponds to the Figure 6 (Right)). In other words, we always have $f^{*T}(f^* - y) \geq 0$, this property will also be used in Lemma F.2.

(ii) When $f^{*T}y < 0$, we have

$$\begin{aligned} \|f^* - y\|_2^2 &= \|\Phi(w^*)v^* - y\|_2^2 && \text{(F.46)} \\ &= \|\Phi(w^*)v^*\|_2^2 + \|y\|_2^2 - 2(\Phi(w^*)v)^T y && \text{(F.47)} \\ &> \|f^*\|_2^2 + \|y\|_2^2 - 2(-\Phi(w^*)v)^T y && \text{(F.48)} \\ &= \|\Phi(w^*)(-v^*) - y\|_2^2 && \text{(F.49)} \\ &= \|-f^* - y\|_2^2, && \text{(F.50)} \end{aligned}$$

where

$$\Phi(w) := \begin{bmatrix} \sigma\left(w_1^T x_1\right), \ldots, \sigma\left(w_m^T x_1\right) \\ \vdots \\ \sigma\left(w_1^T x_n\right), \ldots, \sigma\left(w_m^T x_n\right) \end{bmatrix} \in \mathbb{R}^{n \times m}.$$

Therefore, changing v^* to $-v^*$ will further reduce the loss function (see Figure 6 (Left)). Since $-v^*$ is feasible if v^* is feasible, this is a contradiction to the assumption that (w^*, v^*) is a KKT point. That is to say, we always have case (i): $f^{*T}y \geq 0$.

In conclusion, we always have $\ell \circ f' \geq \ell \circ f^*$, so the proof is completed.

F.4 Proof of Lemma F.2

Similarly with the residue term (F.12), we define $R \in \mathbb{R}^n$ with each component satisfying $[R]_i := \int_0^{1+\tilde{\eta}} (w - w^0)^T H_i(w^0 + t(w - w^0))(w - w^0)(1-t)dt$, we have:

$$
\begin{aligned}
\left\|f' - \bar{f}\right\|_2^2 &\overset{(F.32)\&(F.15)}{=} \left\|-(1+\tilde{\eta})R^* + \bar{R}\right\|_2^2 && \text{(F.51)}\\
&= \left\|(1+\tilde{\eta})R^* - \bar{R}\right\|_2^2 && \text{(F.52)}\\
&\overset{(*)}{=} \left\|(1+\tilde{\eta})R^* - (1+\tilde{\eta})^2 R\right\|_2^2 && \text{(F.53)}\\
&= \left\|(1+\tilde{\eta})(R^* - R) - (1+\tilde{\eta})\tilde{\eta}R\right\|_2^2 && \text{(F.54)}\\
&\leq (1+\tilde{\eta})^2 \sum_{i=1}^n \left(\int_1^{1+\tilde{\eta}} (w-w^0)^T H_i(w^0 + t(w-w^0))(w-w^0)(1-t)dt\right)^2 \\
&\quad + (1+\tilde{\eta})^2\tilde{\eta}^2 \sum_{i=1}^n \left(\int_0^{1+\tilde{\eta}} (w-w^0)^T H_i(w^0 + t(w-w^0))(w-w^0)(1-t)dt\right)^2 \\
&\leq (1+\tilde{\eta})^2 \sum_{i=1}^n \left(\lambda_{i[1:1+\tilde{\eta}]}\epsilon^2 \int_1^{1+\tilde{\eta}} dt\right)^2 + (1+\tilde{\eta})^2\tilde{\eta}^2 \sum_{i=1}^n \left(\lambda_{i[0:1+\tilde{\eta}]}\epsilon^2 \int_0^{1+\tilde{\eta}} dt\right)^2 \\
&= n(1+\tilde{\eta})^2\tilde{\eta}^2\lambda_{[1:1+\tilde{\eta}]}^2\epsilon^4 + n(1+\tilde{\eta})^4\tilde{\eta}^2\lambda_{[0:1+\tilde{\eta}]}^2\epsilon^4 && \text{(F.55)}\\
&= n\tilde{\eta}^2\epsilon^4\left((1+\tilde{\eta})^2\lambda_{[1:1+\tilde{\eta}]}^2 + (1+\tilde{\eta})^4\lambda_{[0:1+\tilde{\eta}]}^2\right) && \text{(F.56)}\\
&= n\tilde{\eta}^2\epsilon^4\left(4\lambda_{[1:1+\tilde{\eta}]}^2 + 16\lambda_{[0:1+\tilde{\eta}]}^2\right) && \text{(F.57)}\\
&= n\tilde{\eta}^2\epsilon^4\left(20\lambda_{[0:2]}^2\right), && \text{(F.58)}
\end{aligned}
$$

where the last two inequalities is because of the fact that $\tilde{\eta} \leq 1$ is sufficiently small and $\lambda_{[1:1+\tilde{\eta}]} \leq \lambda_{[0:1+\tilde{\eta}]} \leq \lambda_{[0:2]}$. (*) is due to: for $i = 1, \cdots, n$:

$$
\begin{aligned}
[\bar{R}]_i &= \int_0^1 \left(w - \eta\nabla_w \ell(w^*; v^*) - w^0\right)^T H_i\left(w^0 + t(w - \eta\nabla_w \ell(w^*; v^*) - w^0)\right)\left(w - \eta\nabla_w \ell(w^*; v^*) - w^0\right)(1-t)dt && \text{(F.59)}\\
&\overset{(F.6)}{=} (1+\tilde{\eta})^2 \int_0^1 \left(w - w^0\right)^T H_i\left(w^0 + (1+\tilde{\eta})t(w - w^0)\right)\left(w - w^0\right)(1-t)dt && \text{(F.60)}\\
&= (1+\tilde{\eta})^2 \int_0^{1+\tilde{\eta}} \left(w - w^0\right)^T H_i\left(w^0 + t(w - w^0)\right)\left(w - w^0\right)(1-t)dt && \text{(F.61)}\\
&= (1+\tilde{\eta})^2 [R]_i. && \text{(F.62)}
\end{aligned}
$$

Now, we bound $\left\|f' + \bar{f} - 2y\right\|_2$, since (w^*, v^*) is a KKT point, it is proved in Lemma F.1 in Appendix F.3 that $f^{*T}y \geq 0$, furthermore, at $w = w^*$, the loss function at (w^*, v^*) should be less or equal to all other feasible points (w^*, v), including $v = \zeta\mathbf{1}$, i.e.,

$$
\begin{aligned}
\|f^* - y\|_2^2 &= \|f(w^*; v^*) - y\|_2^2 && \text{(F.63)}\\
&\leq \|f(w^*; v^*)\|^2 + \|y\|_2^2 && \text{(F.64)}\\
&\leq \|f(w^*; \zeta\mathbf{1})\|^2 + \|y\|_2^2 && \text{(F.65)}\\
&\overset{(**)}{\leq} n\left(\frac{m}{2}L\zeta\epsilon\right)^2 + \|y\|_2^2 && \text{(F.66)}\\
&\leq n\left(\frac{m}{2}L\zeta\epsilon\right)^2 + nC_y^2, && \text{(F.67)}
\end{aligned}
$$

where the last inequality is because of Assumption 3 (each $y_i \leq C_y$), and (**): is due to

$$\|f(w;\zeta\mathbf{1})\|_2^2 \overset{(6)}{=} \sum_{i=1}^{n}\left(\sum_{j=1}^{\frac{m}{2}}\zeta\left(\sigma(w_j^T x_i)-\sigma(w_{j+\frac{m}{2}}^T x_i)\right)\right)^2 \tag{F.68}$$

$$= \zeta^2\sum_{i=1}^{n}\left(\sum_{j=1}^{\frac{m}{2}}\left(\sigma(w_j^T x_i)-\sigma(w_{j+\frac{m}{2}}^T x_i)\right)\right)^2 \tag{F.69}$$

$$\overset{\text{Assumption 2}}{\leq} \zeta^2 L^2\sum_{i=1}^{n}\left(\sum_{j=1}^{\frac{m}{2}}(w_j-w_{j+\frac{m}{2}})^T x_i\right)^2 \tag{F.70}$$

$$\leq \zeta^2 L^2\sum_{i=1}^{n}\left(\sum_{j=1}^{\frac{m}{2}}\|w_j-w_{j+\frac{m}{2}}\|_2\|x_i\|_2\right)^2 \tag{F.71}$$

$$\overset{(***)}{=} \zeta^2 L^2\sum_{i=1}^{n}\left(\sum_{j=1}^{\frac{m}{2}}\|w_j-w_{j+\frac{m}{2}}\|_2\right)^2 \tag{F.72}$$

$$\leq n\left(\frac{m}{2}L\zeta\epsilon\right)^2, \tag{F.73}$$

where $(***)$: we assume $\|x_i\|_2 \leq 1$ for $i = 1, \cdots, n$. For general $\|x_i\|_2$, the difference is up to a constant.

Recall $f' = (1+\tilde{\eta})f^*$, we have

$$\|f'-y\|_2 = \|(1+\tilde{\eta})f^*-y\|_2 \tag{F.74}$$

$$= \|f^*-y+\tilde{\eta}(f^*-y)+\tilde{\eta}y\|_2 \tag{F.75}$$

$$= \|f^*-y\|_2+\tilde{\eta}\|f^*-y\|_2+\tilde{\eta}\|y\|_2 \tag{F.76}$$

$$\overset{(F.67)}{\leq} (1+\tilde{\eta})\left(n\left(\frac{m}{2}L\zeta\epsilon\right)^2+nC_y^2\right)^{\frac{1}{2}}+\tilde{\eta}\|y\|_2 \tag{F.77}$$

$$\leq (1+\tilde{\eta})\left(n\left(\frac{m}{2}L\zeta\epsilon\right)^2+nC_y^2\right)^{\frac{1}{2}}+\tilde{\eta}\sqrt{n}C_y \tag{F.78}$$

$$\leq 2\left(n\left(\frac{m}{2}L\zeta\epsilon\right)^2+nC_y^2\right)^{\frac{1}{2}}+\sqrt{n}C_y, \tag{F.79}$$

where the last inequality is because $\tilde{\eta}$ is sufficiently small. Now, combining with the descent property $\|\bar{f}-y\|_2 \leq \|f^*-y\|_2$, we have

$$\|f'+\bar{f}-2y\|_2 \leq \|f'-y\|_2+\|\bar{f}-y\|_2 \tag{F.80}$$

$$\leq \|f'-y\|_2+\|f^*-y\|_2 \tag{F.81}$$

$$\leq 3\left(n\left(\frac{m}{2}L\zeta\epsilon\right)^2+nC_y^2\right)^{\frac{1}{2}}+\sqrt{n}C_y. \tag{F.82}$$

We conclude the proof of Lemma F.2 by combining (F.58) and (F.82).

G Extension To Deep Networks

In this section, we discuss how to extend our analysis to deep networks. To do so, we apply the mirrored LeCun's initialization and the constrained formulation (7) to the last two layers and treat the output of the $(L-2)$-th layer as the input features. In the proof of Theorem 1, the expressivity is guaranteed if the inputs $\{x_1, \cdots, x_n\}$ follow a *continuous joint distribution*, which is true under Assumption 3. Fortunately, for deep neural networks with $m_l \geq \frac{2n}{m_{l-1}}$ (where m_l is the width of the l-th layer), the outputs of the $(L-2)$-th layer still follow a *continuous joint distribution* under

Assumption 2 and 3, so the expressivity can be shown using the similar technique as Theorem 1. This result is formally stated and proved in the Lemma G.1 below.

Lemma G.1. *Given a deep fully-connected neural network with L layers:*

$$f(x;\theta) = w^{(L)}\sigma\left(w^{(L-1)}\dots\sigma\left(w^{(2)}\sigma\left(w^{(1)}x\right)\right)\right),$$

where $\sigma(\cdot):\mathbb{R}\to\mathbb{R}$ is the activation function, $w^{(l)}\in\mathbb{R}^{m_l\times m_{l-1}}$ are the weights, $l=1,\dots,L$. Under Assumption 2 and 3, suppose $m_l\geq m_{l+1}$, for $l\leq L-3$ and $m_{L-1}m_{L-2}\geq 2n$, then at the initialization θ^0 (for $l\leq L-2$, LeCun's initialization is used; for the last two layers, the mirrored LeCun's initialization is used), for the inputs $\{x_1,\cdots,x_n\}$, the outputs of the $(L-2)$-th layer follow a continuous joint distribution.

To prove Lemma G.1, we first prove the following lemma:

Lemma G.2. *Suppose that $\psi:\mathbb{R}^{k_1}\to\mathbb{R}^{k_2}$ is a analytic mapping and for almost every $u\in\mathbb{R}^{k_1}$, the Jacobian matrix $J(u)$ of ψ w.r.t. u is of full row rank. If $u\in\mathbb{R}^{k_1}$ follows a continuous distribution and $k_1\geq k_2$, then $\psi(u)$ also follows a continuous distribution.*

Proof. Let $Z_0\subseteq\mathbb{R}^{k_2}$ be a zero measure set in $\mathbb{R}^{k_2}$, We define $S_1(Z_0)=\{u\in\mathbb{R}^{k_1}\mid\psi(u)\in Z_0, J(u)\text{ is non-singular}\}$. By the definition of $S_1(Z_0)$, any $u\in S_1(Z_0)$ can be written as $u=(u_1^T,u_2^T)^T$, where $u_1\in\mathbb{R}^{k_2}$ and $J(u_1;u_2)$ is invertible ($J(u_1;u_2)$ is the $l\times l$ submatrix of $J(u)$, similarly as in Lemma E.1). Then by the Inverse Function Theorem, there exists some ball $\mathcal{B}_{\epsilon(u)}(u)\subseteq\mathbb{R}^{k_1}$ (centered at u with radius $\epsilon(u)$) such that for any $u'=((u')_1^T,(u')_2^T)^T\in\mathcal{B}_{\epsilon(u)}(u)\cap S_1(Z_0)$, $u_1'=\tau(u_2',z')$, where $z'=\psi(u')\in Z_0$ and τ is a smooth mapping in a neighborhood $\tilde{Z}_0$ of $(u_2,\psi(u))$. Then for any u, there exists a rational point $\bar{u}\in\mathbb{Q}^{k_1}$ and a rational number $\bar{\epsilon}(u)\in\mathbb{Q}$ such that $u\in N(u):=\mathcal{B}_{\bar{\epsilon}(u)}(\bar{u})\subseteq\mathcal{B}_{\epsilon(u)}(u)$. Since the collection of all open balls with a rational center and a rational radius is a countable set, we let $N_1,N_2,\cdots,N_n,\cdots$ be different $N(u)$ for $u\in S_1(V_0)$. Then $S_1(Z_0)=\cup_{i=1}^{\infty}(N_i\cap S_1(Z_0))$.

We then only need to prove that for any i, $N_i\cap S_1(Z_0)$ is of measure zero in $\mathbb{R}^k$. We define the mapping $\tilde{\tau}:\tilde{Z}_0$ as $\tilde{\tau}(u_2',z')=u'$ if $z'=\psi(u')$. Since Z_0 is of measure zero in $\mathbb{R}^{k_2}$, $\tilde{Z}_0$ is measure zero in $\mathbb{R}^{k_1}$. Then because $\tilde{\tau}$ is smooth, the image of $\tilde{\tau}$ of the set $\tilde{V}_0$ is of zero measure in $\mathbb{R}^{k_1}$ (The image of a zero mesure set under a smooth mapping is also measure zero). Notice that $\mathcal{B}_{\bar{\epsilon}(u)}(\bar{u})\cap S_1(Z_0)$ is contained in the image $\tilde{\tau}(\tilde{Z}_0)$, it is also of zero measure. This finishes the proof. □

Now we prove Lemma G.1. When $l\leq L-3$, let $u^{(l)}\in\mathbb{R}^{m_l}$ be the output vector of the l-th layer. Then $u^{(l+1)}=\psi(w^{(l+1)0},u^{(l)})$, where ψ is analytic and $w^{(l+1)0}$ is the initial parameter in the l-th layer. We now prove that $u^{(l)}$ follows a continuous distribution by induction. When $l=0$, it is true since $u^{(0)}=(x_1,\cdots,x_n)$ is just the input data, which follows a continuous distribution under Assumption 3. Now suppose $u^{(l)}$ still follows a continuous distribution, we have: (i) since $w^{(l+1)0}$ follows a continuous distribution at the mirrored initialization, $u^{(l)}$ and $w^{(l+1)0}$ follow a continuous joint distribution; (ii) Now, viewing $(w^{(l+1)0},u^{(l)})\in\mathbb{R}^{m_lm_{l+1}+m_l}$ as the input of ψ, when $m_l\geq m_{l+1}$, $J(w^{(l+1)0},u^{(l)})$ can be proved to be full row rank w.p.1. using the same technique as Theorem 1. In conclusion, we have $u^{(l+1)}$ follows a continuous distribution by Lemma G.2. Hence, we finish the proof of Lemma G.1.

We further comment a bit on extending the trainability analysis to deep nets. For this part, it requires more detailed analysis because the input feature of the penultimate layer is changing along the training (which is fixed in the shallow case), this topic will be considered as future work. Nevertheless, our idea motivates a better training regime for deep networks, and it is numerically verified in our experiments.

H Implementation Details & More Experiments

H.1 Guidance on PyTorch Implementation

In this section, we provide sample code to implement the our proposed method for narrow nets training, which can achieve small empirical loss as proved in Theorem 2. We formally state our training regime in Algorithm 2.

Algorithm 2 Our training regime

Set up hyperparameters:
Choose a constraint size ϵ, ζ, κ and a step size η.
Define $B_\epsilon\left(w^0\right) := \left\{w \mid \left\|w - w^0\right\|_F \leq \epsilon\right\}$
Define $B_{\zeta,\kappa}(v) = \{v | v \geq \zeta \mathbf{1}, \text{and for } \forall v_j, v_j', v_j/v_j' \leq \kappa\}$.
Set up the pairwise structure of v:
Consider $f(x;\theta) = \sum_{i=1}^{\frac{m}{2}} v_j(\sigma(w_j^T x) - \sigma(w_{j+\frac{m}{2}}^T x))$.
Initialization:
Initialize $\theta^0 = (w^0, v^0)$ by the mirrored LeCun's initialization, as shown in Algorithm 1
Training:
Update v via Projected Gradient Descent: $v^{t+1} \leftarrow \mathcal{P}_{B(v)}(v^t - \eta \nabla_v \ell(\theta^t))$.
Update w via Projected Gradient Descent: $w^{t+1} \leftarrow \mathcal{P}_{B_\epsilon(w^0)}(w^t - \eta \nabla_w \ell(\theta^t))$.
Until the final epoch $t = T$.

Algorithm 2 can be adopted to deep nets by viewing w as the hidden weights in the penultimate layer (or the final block of ResNet [22] in our computer vision experiments), and view x as the feature outputted by all the previous layers. As shown in Algorithm 2, there are several key ingredients: the pairwise structure of v in (7); the mirrored initialization; and the PGD algorithm. We now demonstrate their implementation in PyTorch. Each of them only involves several lines of code changes based on the regular training regime.

The pairwise structure of v & The Mirrored LeCun's initialization. .

```
import torch
import torch.optim as optim
import copy
class ShallowNet(nn.Module):
    def __init__(self, n_input, n_hidden):
        super(ShallowNet, self).__init__()

        self.fc1 = nn.Linear(n_input, n_hidden1,bias=False)
        self.tanh=nn.Tanh()
        self.n_hidden1=n_hidden1
        #Cut down half the width of the output layer
        self.fc2 = nn.Linear(int(n_hidden/2), 1, bias=False)

        #The mirrored initialization
        hidden_half = self.fc1.weight[0:int(n_hidden/2)]
        hidden_layer=torch.cat([hidden_half,hidden_half],dim=0)
        self.fc1.weight = torch.nn.Parameter(hidden_layer)


    def forward(self, x):


        x=self.fc1(x)
        h=self.tanh(x)

        #Keep the pairwise structure of v
        h1=h[:,0:int(self.n_hidden1/2)]
        h2=h[:,int(self.n_hidden1/2):self.n_hidden1]

        x_pred1=self.fc2(h1)
```

```
        x_pred2=self.fc2(h2)
        x_pred=x_pred1-x_pred2

        return x_pred
```

To extend the mirrored initialization to deeper nets such as ResNet, we just need to repeat the code line [12 - 14] for every hidden layer, including the BatchNorm layer and the CNNs in the shortcut layers in the Residue block.

Projected Gradient Descent. We now demonstrate how to implement PGD. First, we need to copy the parameters at the initialization, will be used for projection.

```
    # Copy the parameters at the initialization, will be used for projection
    model_initial = copy.deepcopy(model)
```

Then we do the projection after each gradient update.

```
def train(model, model_initial, epoch, x,y, optimizer):

    #standard code in regular training
    clf_criterion=nn.MSELoss()
    model.train()
    for i in range(epoch):
        optimizer.zero_grad()
        pred=model(x=x)
        loss = clf_criterion(pred,y)  # calculate current loss
        loss.backward() # calculate gradient
        optimizer.step() # update parameters

        # Projection
        for para,para0 in zip(model.parameters(), model_initial.parameters()):
            #project the hidden layer
            if para.data.size()[0]==model.n_hidden1:
                if torch.norm(para.data - para0.data) > eps:
                    para.data = para0.data + eps * (para.data - para0.data) / torch.norm(para.data - para0.data)

            #project the output layer
            if para.data.size()[0]==1:
                para.data=projectv(para.data)

def projectv(v):
    vmax=torch.max(v)
    argmax=torch.argmax(v)
    vmin=torch.min(v)
    argmin=torch.argmin(v)
    #print(vmax/vmin)
    #print('vmax',vmax)
    #print('vmin',vmin)
    if vmin <0.001:
        #print('projectv1')
        v[argmin]=0.001
    if vmax/vmin>1:
        v[argmax]=1*vmin
        #print('projectv2')
    return v
```

H.2 Details on Experimental Setup

Our empirical studies are based on the synthetic dataset, MNIST, CIFAR-10, CIFAR-100 and the R-ImageNet datasets. MNIST, CIFAR-10 and CIFAR-100 are licensed under MIT. Imagenet is

licensed under Custom (non-commercial). All the experiments are run on NVIDIA V100 GPU. Here, we introduce our settings on synthetic dataset and R-ImageNet.

- Synthetic datset: For $i = 1 \ldots, 1000$, we independently generate $x_i \in \mathbb{R}^{200}$ from standard independent Gaussian, and normalize it to $\|x_i\|_2 = 1$, and we set the ground truth as $y_i = (1^T x_i)^2$ for $i = 1, \cdots, 1000$. In short, sample size $n = 1000$, input dimension $d = 200$.
- R-ImageNet: This is a specifically constructed "restricted" version of ImageNet, with resolution 224×224.

  The vanilla ImageNet dataset spans 1000 object classes and contains 1,281,167 training images, 50,000 validation images and 100,000 test images. In our experiments, we use a subset of ImageNet, namely Restricted-ImageNet (R-ImageNet). Similar with [27], we leverage the WordNet [46] hierarchical structure of the dataset such that each class in the R-ImageNet is a superclass category composed of multiple ImageNet classes, noted in Table 2 as "components". For example, the "bird" class of R-ImageNet (both the train and validation parts) is the aggregation of ImageNet-1k classes: [10: 'brambling', 11: 'goldfinch', 12: 'house finch', 13: 'junco', 14: 'indigo bunting'], more details can be seen in Table 2. As a result, there are 20 super classes which contain a total of 190 vanilla ImageNet classes.

Table 2: Classes used in the R-ImageNet dataset. The class ranges are inclusive.

Class name	Corresponding ImageNet components
bird	$[10, 11, 12, 13, 14]$
turtle	$[33, 34, 35, 36, 37]$
lizard	$[42, 43, 44, 45, 46]$
snake	$[60, 61, 62, 63, 64]$
spider	$[72, 73, 74, 75, 76]$
crab	$[118, 119, 120, 121, 122]$
dog	$[205, 206, 207, 208, 209]$
cat	$[281, 282, 283, 284, 285]$
bigcat	$[289, 290, 291, 292, 293]$
beetle	$[302, 303, 304, 305, 306]$
butterfly	$[322, 323, 324, 325, 326]$
monkey	$[371, 372, 373, 374, 375]$
fish	$[393, 394, 395, 396, 397]$
fungus	$[992, 993, 994, 995, 996]$
musical-instrument	$[402, 420, 486, 546, 594]$
sportsball	$[429, 430, 768, 805, 890]$
car-truck	$[609, 656, 717, 734, 817]$
train	$[466, 547, 565, 820, 829]$
clothing	$[474, 617, 834, 841, 869]$
boat	$[403, 510, 554, 625, 628]$

In each dataset, the neural network architectures are chosen as follows, all of the following cases satisfy $m \geq \frac{2n}{d}$ or $m_{L-1} \geq \frac{2n}{m_{L-2}}$, where m_l is the width of the l-th layer.

- Synthetic dataset: we use 1-hidden-layer neural networks with Tanh activation (except for the last layer, where the output dimension equals 1 and no Tanh applied). We study different widths of the hidden layer among $m = 20, 40, 80, 100, 200, 400, 800, 100, 1200$. All of these cases satisfy $m \geq \frac{2n}{d}$.
- MNIST: we use 2-hidden-layer neural networks with ReLU activation (except for the last layer, where the output dimension equals the number of classes and no activation applied). The input dimension $d = 784$, the width of the 1st layer is fixed with $m_1 = 784$ and we study different widths of the 2nd hidden layer among $m_2 = 64, 128, 256, 512, 784, 1024$. All of these cases satisfy $m_{L-1} \geq \frac{2n}{m_{L-2}}$, where m_l is the width of the l-th layer.
- CIFAR-10, CIFAR-100 and R-ImageNet: we use ResNet-18 and we try different number of channels in the 4th block (the i.e., the final block) among $m = 64, 128, 256, 512$ (for

regular ResNet-18, the default number of channels in the 4st block should be 512). All of these cases satisfy $m_{L-1} \geq \frac{2n}{m_{L-2}}$, where m_l is the width of the l-th layer.

In each dataset, the setup for algorithms are as follows: as for training regime, we apply the mirrored LeCun's initialization for all the neural network structures mentioned above, and for regular training, we use the regular LeCun's initialization. We use square loss for the synthetic dataset, and multi-class cross entropy loss is used for the rest of the cases. During training, CIFAR-10, CIFAR-100 images are padded with 4 pixels of zeros on all sides, then randomly flipped (horizontally) and cropped. R-ImageNet images are randomly cropped during training and center-cropped during testing. Global mean and standard deviation are computed on all the training pixels and applied to normalize the inputs on each dataset. As it is required in problem (7), in our training regime, the optimization variable for and the output layer is cut off to half, i.e., $v = (v_1, \cdots, v_{\frac{m}{2}})$, the other half is always $-v$; as for the hyperparameters of $B(v)$, we set $\zeta = 0.001$ and $\kappa = 1$. After each iteration, relevant parameters will projected in to their feasible sets. In addition to the general setup above, more customized hyperparameters are listed as follows:

- Synthetic dataset: For both our training regime and the regular training regime, $B_\epsilon(w)$ constraint is added on the weights in the hidden layer with $\epsilon = 0.1, 0.2, 0.4, 0.8, 1, 2, 4, 8, 10, 1000$ ($\epsilon = 1000$ is equivalent to the unconstrained updates for w). Gradient Descent with 0.9 momentum is used, and we use different constant learning rates lr_1, lr_2 for hidden weights and outer weights, in all cases with different m and ϵ, we grid search learning rate lr_1=[1e-4,1e-3,5e-3,1e-2,5e-2,1e-1,5e-1], lr_2=[1e-4,1e-3,5e-3,1e-2,5e-2,1e-1,5e-1] and report the best results. The neural network is trained for 200000 iterations.
- MNIST: For our training regime, $B_\epsilon(w)$ constraint is added on the weights in the 2nd layer with $\epsilon = 0.1, 0.2, 0.4, 0.8, 1, 2, 4, 8, 10, 1000$ ($\epsilon = 1000$ is equivalent to the unconstrained updates for w). In each case, we either use Adam with 0.001 initial learning rate and 1e-4 weight decay, or Stochastic Gradient Descent (SGD) with 0.01 initial learning rate, 0.9 momentum and 5e-4 weight decay, and we report the best results. For both training regime and the regular training regime, we use cosine annealing learning rate scheduling [43] with $T_{\max}$=number of epochs , and the neural network is trained for 200 epochs and batch size of 64 is used.
- CIFAR-10, CIFAR-100: For our training regime , $B_\epsilon(w^0)$ constraint is added on the 4-th block with different constraint size among $\epsilon = 0.1, 0.2, 0.4, 0.8, 1, 2, 4, 8, 10, 1000$ ($\epsilon = 1000$ is equivalent to the unconstrained updates for w). For both our training regime and the regular training regime, SGD with 0.1 initial learning rate, 0.9 momentum and 5e-4 weight decay is used, and we use cosine annealing learning rate scheduling with T_{max}=number of epochs, and the neural network is trained for 600 epochs and batch size of 128 is used.
- R-ImageNet: For our training regime, $B_\epsilon(w^0)$ constraint is added on the 4th block (i.e., the final block) with different constraint size among $\epsilon = 0.01, 0.1, 1, 1000$ ($\epsilon = 1000$ is equivalent to the unconstrained updates for w), and the weights in the 4th block are projected into the constraint after each mini-batch iteration. For both our training regime and the regular training regime, SGD with 0.1 initial learning rate, 0.9 momentum and 5e-4 weight decay is used, we use a stage-wise constant learning rate scheduling with a multiplicative factor of 0.1 on epoch 30, 60 and 90. The neural network is trained for 90 epochs and batch size of 256 is used.

H.3 Test Accuracy on MNIST

Figure 7 shows the test accuracy in our training regime vs regular training regime in MNIST. With proper choice of ϵ, our training regime leads to higher test accuracy.

H.4 Test Accuracy on CIFAR-10 & CIFAR-100

In CIFAR-10 and CIFAR-100 dataset, our training regime and regular training regime have similar performance (see Figure 8). In several cases, our training regime leads to higher test accuracy. Here, regular training will not fail when we reduce the width of 4-th block, perhaps this is due to the

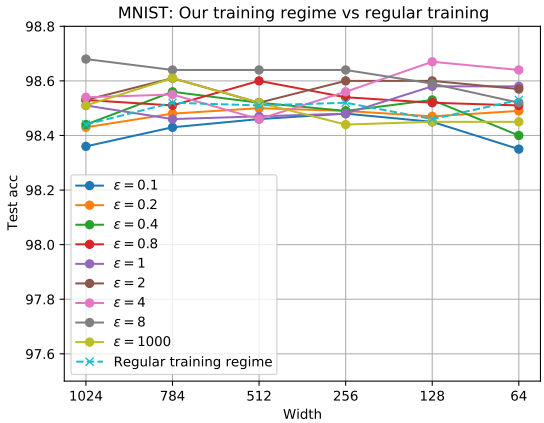
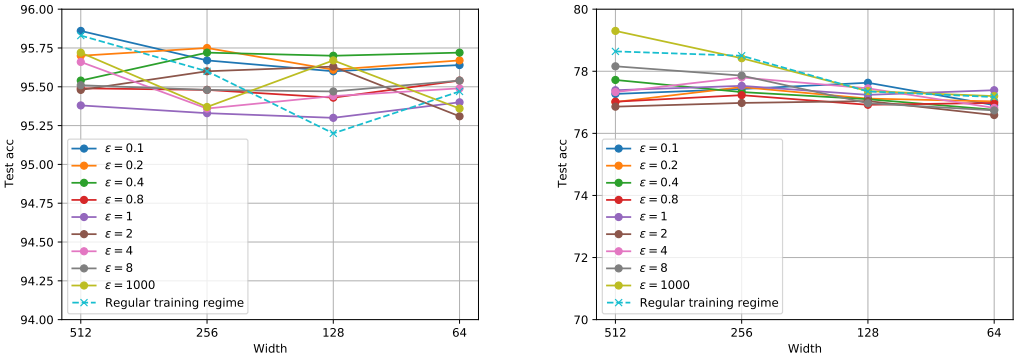


Figure 7: MNIST: test accuracy in our training regime with different ϵ vs regular training regime. In x-axis, width stands for width of 2nd layer (we use 2-hidden-layer neural nets here).

strong expressivity of ResNet-18. In comparison, on a more complicated dataset such as R-ImageNet, narrowing ResNet-18 will jeopardize the regular training (as illustrated in Section 5 and the following subsection).



(a) CIFAR-10 (b) CIFAR-100

Figure 8: CIFAR-10 & CIFAR-100: test accuracy in our training regime with different ϵ vs regular training regime. In x-axis, width stands for the number of channels in the final CNN block of ResNet-18.

H.5 Test Accuracy on R-ImageNet

In this subsection, Figure 9 is the same figure as Figure 5 in the full paper, but with 90% confidence error bars (based on 5 seeds). Besides, Figure 10 shows a selected result of Loss & Accuracy per epoch in our training regime with $\epsilon = 0.04$ & regular training regime (here, we present the early-stopped results). As a result, our training regime can reduce the number of parameters in the 4-th block of ResNet-18 by up to 94% while maintaining competitive test accuracy, especially when ϵ is small. In comparison, regular SGD does not perform well in narrow cases.

H.6 Results on Random-Labeled CIFAR-10

Since the main claim of our work is about memorization of *any* labels, we further explore the performance of our training regime even when the labels are not the correct ones. To do so, we further carry out experiments on the random-labeled CIFAR-10, where all the labels are randomly

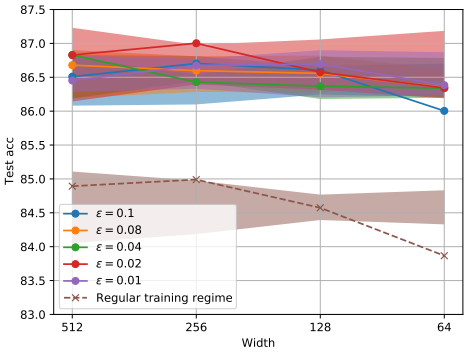
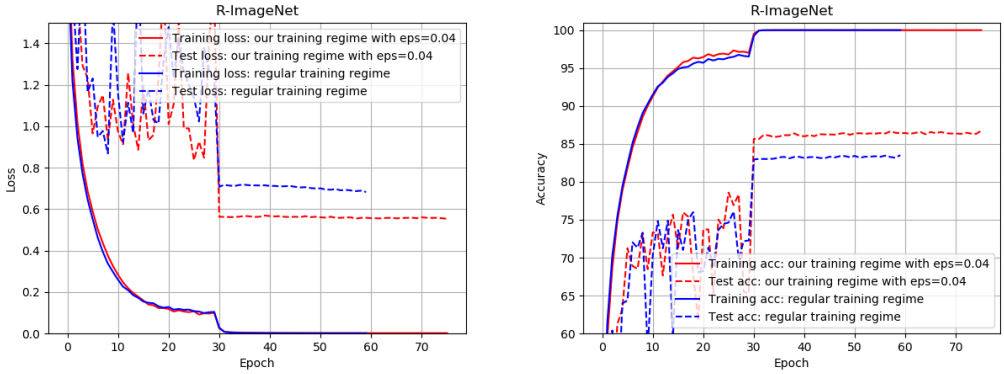


Figure 9: R-ImageNet: test accuracy under our training regime with different ϵ vs regular training regime. In x-axis, width stands for the number of channels in the final CNN block of ResNet-18. The solid & dotted lines are averaged results over 5 seeds, the shaded areas indicates the 90% confidence intervals.



(a) Loss (b) Accuracy

Figure 10: R-ImageNet (selected): loss & Accuracy per epoch in our training regime with $\epsilon = 0.04$ & regular training regime.

shuffled (same as in Zhang et al. [75]). We train a 1-hidden-layer network with width =1024, 2048, 4096 (smaller than n=50000) on the random-labeled CIFAR10 dataset. The hyperparameters in our constrained training regime (7) are $\epsilon = 10$, $\kappa = 1$ and initial learning rate =0.1. We use a stage-wise constant learning rate scheduling with a multiplicative factor of 0.1 on epoch 150, 225, 450. The result is shown in Table 3: after 1000 epochs, we can achieve more than 99% train accuracy, almost perfectly fit the random labels. Note that even though ReLU does not fall into our analysis framework, it works a bit better than Tanh. Extending our results to ReLU activation would be our intriguing future work.

Table 3: Results on the random-labeled CIFAR-10

Width	Epoch	Activation	Train acc	Test acc
1024	1000	ReLU	0.9931	0.1011
2048	1000	ReLU	0.9984	0.1022
4096	1000	ReLU	0.9998	0.0962
1024	1000	Tanh	0.9872	0.0991
2048	1000	Tanh	0.9927	0.1024
4096	1000	Tanh	0.9938	0.0962