TRAINABILITY OF ReLU NETWORKS AND DATA-DEPENDENT INITIALIZATION

Yeonjong Shin & George Em Karniadakis*

Division of Applied Mathematics, Brown University, Providence, Rhode Island 02912, USA

*Address all correspondence to: Yeonjong Shin, Division of Applied Mathematics, Brown University, Providence, Rhode Island 02912, USA; Tel.: +1 401 863 1320; Fax: +1 401 863 1355, E-mail: yeonjong_shin@brown.edu

Original Manuscript Submitted: 3/9/2020; Final Draft Received: 6/19/2020

In this paper we study the trainability of rectified linear unit (ReLU) networks at initialization. A ReLU neuron is said to be dead if it only outputs a constant for any input. Two death states of neurons are introduced—tentative and permanent death. A network is then said to be trainable if the number of permanently dead neurons is sufficiently small for a learning task. We refer to the probability of a randomly initialized network being trainable as trainability. We show that a network being trainable is a necessary condition for successful training, and the trainability serves as an upper bound of training success rates. In order to quantify the trainability, we study the probability distribution of the number of active neurons at initialization. In many applications, overspecified or overparameterized neural networks are successfully employed and shown to be trained effectively. With the notion of trainability, we show that overparameterization is both a necessary and a sufficient condition for achieving a zero training loss. Furthermore, we propose a data-dependent initialization method in an overparameterized setting. Numerical examples are provided to demonstrate the effectiveness of the method and our theoretical findings.

KEY WORDS: *ReLU networks, trainability, dying ReLU, overparameterization, overspecification, data-dependent initialization*

1. INTRODUCTION

Neural networks have been successfully used in various fields of applications. These include image classification in computer vision (Krizhevsky et al., 2012), speech recognition (Hinton et al., 2012), natural language translation (Wu et al., 2016), and superhuman performance in the game of Go (Silver et al., 2016). Modern neural networks are often severely overparameterized or overspecified. Overparameterization means that the number of parameters is much larger than the number of training data. Overspecification means that the number of neurons in a network is much larger than needed. It has been reported that the wider the neural networks, the easier it is to train (Livni et al., 2014; Nguyen and Hein, 2017; Safran and Shamir, 2016).

In general, neural networks are trained by first- or second-order gradient-based optimization methods from random initialization. Almost all gradient-based optimization methods stem from backpropagation (Rumelhart et al., 1985) and the stochastic gradient descent (SGD) method (Robbins and Monro, 1951). Many variants of vanilla SGD have been proposed; for example,

2689–3967/20/$35.00

AdaGrad (Duchi et al., 2011), RMSProp (Hinton, 2014), Adam (Kingma and Ba, 2015), AMSGrad (Reddi et al., 2019), and L-BFGS (Byrd et al., 1995), to name just a few. Different optimization methods have different convergence properties. It is still far from clear how different optimization methods affect the performance of trained neural networks. Nonetheless, how to start the optimization processes plays a crucial role for the success of training. Properly chosen weight initialization could drastically improve the training performance and allow the training of deep neural networks; for example, see LeCun et al. (1998), Glorot and Bengio (2010), Saxe et al. (2014), He et al. (2015), Mishkin and Matas (2016), and for more recent work see Lu et al. (2019). Among them, when it comes to the rectified linear unit (ReLU) neural networks, the "He initialization" (He et al., 2015) is one of the most commonly used initialization methods.

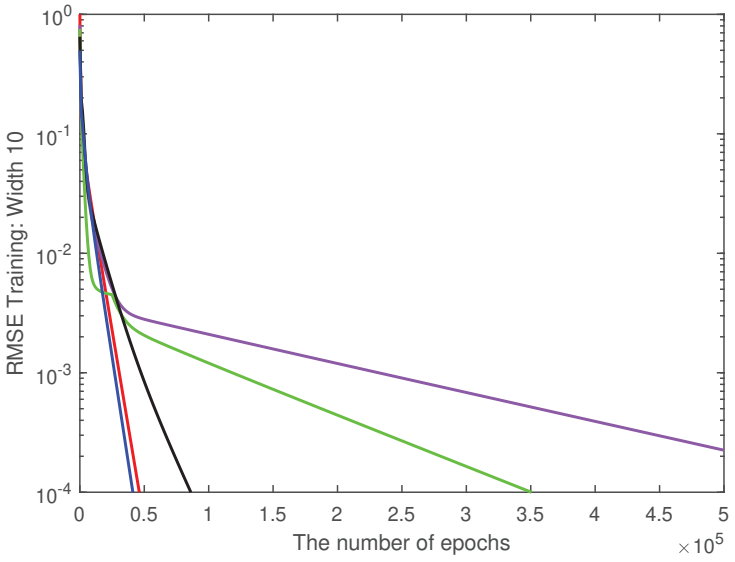
There are several theoretical works showing that under various assumptions, overparameterized neural networks can perfectly interpolate the training data. For the shallow (two-layer) neural network setting, see Oymak and Soltanolkotabi (2019), Soltanolkotabi et al. (2019), Du et al. (2018b), and Li and Liang (2018). For the deep (more than two layers) neural network setting, see Du et al. (2018a), Zou et al. (2018), and Allen-Zhu et al. (2018). Hence, overparameterization can be viewed as a sufficient condition for minimizing the training loss. In spite of the current theoretical progress, there still exists a huge gap between existing theories and empirical observations in terms of the level of overparameterization. To illustrate this gap, let us consider the problem of approximating $f(x) = |x|$. The same learning task was also used in Lu et al. (2019) but with a deep network. Here we consider a two-layer (shallow) rectified linear unit (ReLU) network. The training set consists of 10 random samples from the uniform distribution on $[-1, 1]$. To interpolate all 10 data points, the best existing theoretical condition requires the width of $\mathcal{O}(n^2)$ (Oymak and Soltanolkotabi, 2019). In this case, the width of 100 would be needed. Figure 1 shows the convergence of the root-mean-square errors (RMSE) on the training data with respect to the number of epochs for five independent simulations. On the left, the results of width 10 are shown. We observe that all five training losses converge to zero as the number of epochs increases. It would be an ongoing challenge to bridge the gap of the degree of overparameterization.

On the other hand, we know that $f(x) = |x|$ can be exactly represented by only two ReLU neurons as $|x| = \max\{x, 0\} + \max\{-x, 0\}$. Thus we show the results of width 2 on the right of Fig. 1. In contrast to the theoretical expressivity, we observe that only one out of five simulations shows the convergence. It turns out that there is a probability greater than 0.43 that the network of width 2 fails to be trained successfully (Theorem 1), see also Lu et al. (2019).

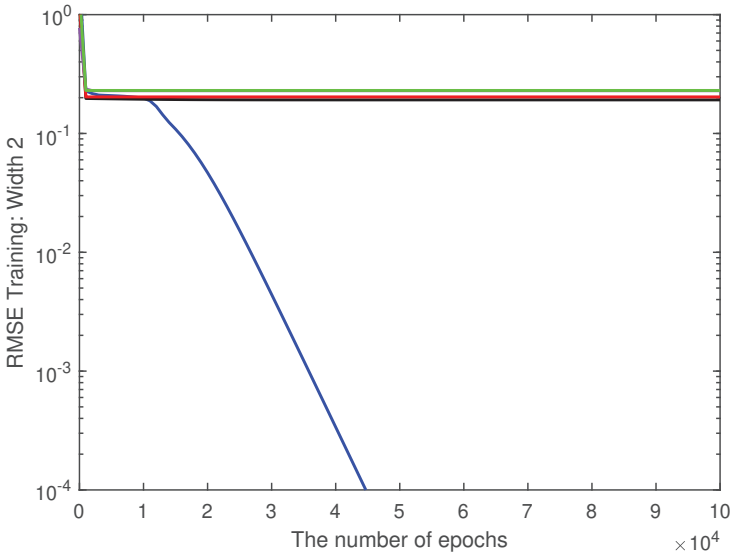
In this paper we study the trainability of ReLU networks, a *necessary* condition for successful training, and propose a data-dependent initialization for better training. Our specific contributions are summarized below:

- We classify a dead neuron into two states: tentatively dead and permanently dead. With the new classification, we introduce a notion of trainable networks (precise definition is given in Section 3). By combining it with Lemma 1, we conclude that a network being trainable is a necessary condition for successful training. That is, if an initialized ReLU network is not trainable, regardless of which gradient-based optimization method is selected, the training will not be successful.

- The probability of a randomly initialized network being trainable is referred to as *trainability* (trainable probability). We establish a general formulation of computing trainability and derive the trainabilities of ReLU networks of depths 2 and 3.



(a)



(b)

FIG. 1: The root-mean-square errors on the training data of five independent simulations with respect to the number of epochs. The standard L_2 loss is employed. (a) Width 10 and depth 2. (b) Width 2 and depth 2.

- With the computed trainability, we show that for shallow ReLU networks, overparameterization is both *a necessary and a sufficient condition* for minimizing the training loss, i.e., interpolating all training data.

- Motivated by our theoretical results, we propose a new data-dependent initialization scheme.

Taken together, our developments provide new insight into the training of ReLU neural networks that can help us design efficient network architectures and reduce the effort in optimizing the networks.

The rest of this paper is organized as follows. Upon presenting the mathematical setup in Section 2, we present the trainability of ReLU networks in Section 3. A new data-dependent initialization is introduced in Section 4. Numerical examples are provided in Section 5 before the conclusion in Section 6.

2. MATHEMATICAL SETUP

Let $\mathcal{N}^L : \mathbb{R}^{d_{\text{in}}} \mapsto \mathbb{R}^{d_{\text{out}}}$ be a feed-forward neural network with L layers and n_j neurons in the jth layer ($n_0 = d_{\text{in}} = d$, $n_L = d_{\text{out}}$). For $1 \le j \le L$, the weight matrix and the bias vector in the jth layer are denoted by $\boldsymbol{W}^j \in \mathbb{R}^{n_j \times n_{j-1}}$ and $\boldsymbol{b}^j \in \mathbb{R}^{n_j}$, respectively; n_j is called the width of the jth layer. We also denote the input by $\mathbf{x} \in \mathbb{R}^{d_{\text{in}}}$ and the output at the jth layer by $\mathcal{N}^j(\mathbf{x})$. Given an activation function ϕ which is applied element-wise, the feed-forward neural network is defined by

$$\mathcal{N}^j(\mathbf{x}) = \boldsymbol{W}^j \phi(\mathcal{N}^{j-1}(\mathbf{x})) + \boldsymbol{b}^j \in \mathbb{R}^{n_j}, \quad \text{for} \quad 2 \le j \le L,$$

and $\mathcal{N}^1(\mathbf{x}) = \boldsymbol{W}^1\mathbf{x} + \boldsymbol{b}^1$. Note that $\mathcal{N}^L(\mathbf{x})$ is called a $(L-1)$-hidden layer neural network or a L-layer neural network. Also, $\phi(\mathcal{N}_i^j(\mathbf{x}))$, $i = 1, \cdots, n_j$, is called a neuron or a unit in the jth hidden layer. We use $\boldsymbol{n} = (n_0, \cdots, n_L)$ to describe a network architecture. In this paper we refer to a two-layer network as a shallow network and a L-layer network as a deep network for $L > 2$.

Let θ be a collection of all weight matrices and bias vectors, i.e., $\theta = \{\boldsymbol{V}^j\}_{j=1}^L$ where $\boldsymbol{V}^j = [\boldsymbol{W}^j, \boldsymbol{b}^j]$. To emphasize the dependency on θ, we often denote the neural network by $\mathcal{N}^L(\mathbf{x}; \theta)$. In this paper, the ReLU is employed as an activation function, i.e.,

$$\phi(\mathbf{x}) = \text{ReLU}(\mathbf{x}) := (\max\{x_1, 0\}, \cdots, \max\{x_{d_{\text{in}}}, 0\})^T,$$

where $\mathbf{x} = (x_1, \cdots, x_{d_{\text{in}}})^T$.

In many machine learning applications, the goal is to train a neural network using a set of training data $\mathcal{T}_m$. Each datum is a pair of an input and an output, $(\mathbf{x}, \mathbf{y}) \in \mathcal{X} \times \mathcal{Y}$. Here $\mathcal{X} \subset \mathbb{R}^{d_{\text{in}}}$ is the input space and $\mathcal{Y} \subset \mathbb{R}^{d_{\text{out}}}$ is the output space. Thus we write $\mathcal{T}_m = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^m$. In order to measure the discrepancy between a prediction and an output, we introduce a loss metric $\ell(\cdot, \cdot) : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}$ to define a loss function $\mathcal{L}$:

$$\mathcal{L}(\theta) = \frac{1}{m} \sum_{i=1}^{m} \ell(\mathcal{N}^L(\mathbf{x}_i; \theta), \mathbf{y}_i). \tag{1}$$

For example, the squared loss $\ell(\hat{\mathbf{y}}, \mathbf{y}) = \|\hat{\mathbf{y}} - \mathbf{y}\|^2$, logistic $\ell(\hat{y}, y) = \log(1 + \exp(-y\hat{y}))$, hinge, or cross-entropy are commonly employed. We then seek to find θ^*, which minimizes the loss function $\mathcal{L}$. In general, a gradient-based optimization method is employed for the training. In its very basic form, given an initial value of $\theta^{(0)}$, the parameters are updated according to

$$\theta^{(k+1)} = \theta^{(k)} - \eta_k \frac{\partial \mathcal{L}(\theta)}{\partial \theta}\bigg|_{\theta=\theta^{(k)}},$$

where η_k is the learning rate of the kth iteration. There are many stochastic variants of gradient descent (Ruder, 2016) that are popularly employed in practice. Throughout this paper, such variants are referred to as gradient-based optimization. This includes minibatch stochastic gradient descent or its variants.

2.1 Weights and Biases Initialization and Data Normalization

Gradient-based optimization is a popular choice for training a neural network. It commences with the weight and bias initialization. How to initialize the network plays a crucial role in the success of the training. Typically, the weights are randomly initialized from probability distributions. However, the biases could be set to zeros initially or could be randomly initialized.

In this paper we consider the following weights and biases initialization schemes. One is the normal initialization. That is, all weights and/or biases in the $(t+1)$th layer are independently initialized from zero-mean normal distributions:

$$\begin{aligned} &\text{(“Normal” without bias)} \quad \boldsymbol{W}_j^{t+1} \sim N\left(0, \sigma_{t+1}^2 \boldsymbol{I}_{n_t}\right), \quad \boldsymbol{b}_j^{t+1} = 0, \\ &\text{(“Normal” with bias)} \quad \boldsymbol{W}_j^{t+1} \sim N\left(0, \sigma_{t+1}^2 \boldsymbol{I}_{n_t}\right), \quad \boldsymbol{b}_j^{t+1} \sim N(0, \sigma_{b,t+1}^2), \end{aligned} \tag{2}$$

where $\boldsymbol{I}_m$ is the identity matrix of size $m \times m$. When $\sigma_{t+1}^2 = 2/n_t$ and $\boldsymbol{b}_j^{t+1} = 0$, the initialization is known as the “He initialization” (He et al., 2015). The He initialization is one of the most popular initialization methods for ReLU networks. The other initialization is from the uniform distribution on the unit hypersphere. That is, each row of either $\boldsymbol{W}^{t+1}$ or $\boldsymbol{V}^{t+1} = [\boldsymbol{W}_j^{t+1}, \boldsymbol{b}_j^{t+1}]$ is independently initialized from its corresponding unit hypersphere uniform distribution.

$$\begin{aligned} &\text{(“Unit hypersphere” without bias)} \quad \boldsymbol{W}_j^{t+1} \sim \mathrm{Unif}(\mathbb{S}^{n_t - 1}), \quad \boldsymbol{b}_j^{t+1} = 0, \\ &\text{(“Unit hypersphere” with bias)} \quad \boldsymbol{V}_j^{t+1} = [\boldsymbol{W}_j^{t+1}, \boldsymbol{b}_j^{t+1}] \sim \mathrm{Unif}(\mathbb{S}^{n_t}). \end{aligned} \tag{3}$$

Throughout this paper we assume that the training input domain is the closed ball with radius $r > 0$, i.e., $B_r(0) = \{\mathbf{x} \in \mathbb{R}^{d_{\text{in}}} | \|\mathbf{x}\|_2 \leq r\}$. In many practical applications, such as image processing or classification, there is a natural bound on the magnitude of each datum. Also, in practice, the training data is often normalized to have mean zero and/or variance 1. Given a training data set $\mathcal{T}_m = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^m$, the normalization makes $\|\mathbf{x}_i\|_2^2 \leq 1$ for all $i = 1, \cdots, m$. Thus one may assume that the training input data domain is the unit closed ball. We note that this assumption is independent of the actual data domain. This is because one can normalize the given data set since we *always* have finitely many data. Many theoretical works (Allen-Zhu et al., 2018; Du et al., 2018a; Li and Liang, 2018; Soltanolkotabi et al., 2019; Zou et al., 2018) also assume a certain data normalization. Given a training data point $\mathbf{x}_i$, let $\tilde{\mathbf{x}}_i = [\mathbf{x}_i; \alpha_i]$ for some $\alpha_i > 0$. One can then normalize $\tilde{\mathbf{x}}_i$ to have a unit norm and let $\mathbf{z}_i = \tilde{\mathbf{x}}_i / \|\tilde{\mathbf{x}}_i\|$. Here $\|\cdot\|$ is the standard Euclidean vector norm. For example, if $\|\mathbf{x}_i\| = r$ and $\alpha_i = r\sqrt{k_i - 1}$ for any $k_i > 1$, we have $\mathbf{z}_i = [\mathbf{x}_i / \sqrt{k_i r^2}; \sqrt{k_i - 1}/\sqrt{k_i}]$ whose norm is 1. In Allen-Zhu et al. (2018), k_i was chosen to be 2 for all i. To this end, $\mathbf{x}_i$ is normalized to $\mathbf{x}_i / \sqrt{k_i r^2}$. In the later sections, we will see that the choice of k_i will affect the trainability of ReLU networks.

2.2 Dying ReLU and Born Dead Probability

Dying ReLU refers to the problem when ReLU neurons become inactive and only output a constant for any input. We say that a ReLU neuron in the tth hidden layer is dead on $B_r(0)$ if it

is a constant function on $B_r(0)$. That is, there exists a constant $c \in \mathbb{R}^+ \cup \{0\}$ such that

$$\phi(\boldsymbol{w}^T \phi(\mathcal{N}^{t-1}(\mathbf{x})) + b) = c, \quad \forall \mathbf{x} \in B_r(0).$$

Also, a ReLU neuron is said to be born dead (BD) if it is dead at the initialization. In contrast, a ReLU neuron is said to be active in $B_r(0)$ if it is not a constant function on $B_r(0)$. The notion of born death was introduced in Lu et al. (2019), where a ReLU network is said to be BD if there exists a layer where all neurons are BD. We refer to the probability that a ReLU neuron is BD as the born dead probability (BDP) of a ReLU neuron.

In the first hidden layer, once a ReLU neuron is dead it cannot be revived during the training. However, a dead neuron in the tth layer where $t > 1$ could be revived by other active neurons in the same layer. In the following we provide a condition on which a dead neuron cannot be revived. The lemma is based on Lemma 10 of Lu et al. (2019).

Lemma 1. *For a shallow ReLU network ($L = 2$), none of the dead neurons can be revived through gradient-based training. For a deep ReLU network ($L > 2$), suppose the weight matrices are initialized from probability distributions, which satisfy Pr $(\boldsymbol{W}_j^t \boldsymbol{z} = 0) = 0$ for any nonzero vector $\boldsymbol{z}$. If there exists a hidden layer whose neurons are all dead, with probability 1, none of the dead neurons can be revived through gradient-based training.*

Proof. The proof can be found in Appendix A. □

3. TRAINABILITY OF RELU NETWORKS

3.1 Shallow ReLU Networks

For pedagogical reasons, we first confine ourselves to shallow (one-hidden layer) ReLU networks. For shallow ReLU networks we define the trainability as follows:

Definition 1. For a learning task that requires at least m active neurons, a shallow ReLU network of width n is said to be trainable if the number of active neurons is greater than or equal to m. If the network parameters are randomly initialized, we refer to the probability of a network being trainable at the initialization as trainability.

We note that "trainable" is a state of neural networks. The definition of "trainable" is independent of how the network was trained or initialized. As training goes on, the state may change. Different random realizations of networks may have different states. In what follows we investigate this state *from the random initialization*.

From Lemma 1, dead neurons will never be revived during the training. Thus, given a learning task which requires at least m active neurons, in order for successful training, an initialized network should have at least m active neurons in the first place. If the number of active neurons is less than m, there is no hope to train the network successfully. Therefore *a network being trainable is a necessary condition for successful training*. We note that this condition is independent of the choice of loss metric $\ell(\cdot, \cdot)$ in (1), of the number of training data, and of the choice of gradient-based optimization methods.

We now present the trainability results for shallow ReLU networks.

Theorem 1. *Given a learning task which requires a shallow ReLU network having at least m active neurons, suppose the training input domain is $B_r(0)$ and a shallow network of width $n \geq m$ is employed.*

- *If either the 'normal' (2) or the "unit hypersphere" (3) initialization without bias is used in the first hidden layer, with probability 1, the network is trainable.*
- *If either the 'normal' (2) or the "unit hypersphere" (3) initialization with bias is used in the first hidden layer, with probability,*

$$\sum_{j=m}^{n} \binom{n}{j} (1-\hat{p}_{d_{in}}(r))^j (\hat{p}_{d_{in}}(r))^{n_1-j}, \quad \hat{p}_d(r) = \frac{1}{\sqrt{\pi}} \frac{\Gamma((d+1)/2)}{\Gamma(d/2)} \int_0^{\alpha_r} (\sin u)^{d-1} du,$$

where $\alpha_r = \tan^{-1}(1/r)$*, the network is trainable. Furthermore, on average, at least*

$$n \left[1 - \sqrt{\frac{d_{in}}{2\pi}} \alpha_r (\sin \alpha_r)^{d_{in}-1} \right]$$

neurons will be active at the initialization.

Proof. The proof can be found in Appendix D. □

Theorem 1 implies that if the biases are randomly initialized, overspecification is necessary for successful training. It also shows a degree of overspecification whenever one has a specific width in mind for a learning task. If it is known (either theoretically or empirically) that a shallow network of width m can achieve a good performance, one should use a network of width $m/(1-\hat{p}_{d_{\text{in}}}(r))$ to guarantee that the initialized network has m active neurons (on average) at the initialization. For example, when $d_{\text{in}} = 1$, $r = 1/\sqrt{3}$, $m = 200$, it is suggested to work on a network of width $n = 300$ in the first place. The example (Fig. 1) given in Section 1 can be understood in this manner. By Theorem 1, with probability at least 0.43, the network of width 2 fails to be trained successfully for any learning task that requires at least two active neurons. The trainability depends only on $\hat{p}_d(r)$, which evidently shows its dependency on the maximum magnitude r of training data. The smaller r is, the larger $\hat{p}_d(r)$ becomes. This indicates that how the data are normalized also affects the trainability.

On the other hand, if the biases are initialized to zero, overparameterization or overspecification is not needed from this perspective. However, the zero-bias initialization often finds a spurious local minimum or gets stuck on a flat plateau. In Section 4, we further investigate the bias initialization.

Next we provide two concrete learning tasks that require a certain number of active neurons. For this purpose, we introduce the minimal function class.

Definition 2. Let $\mathcal{F}_n(r)$ be a class of shallow ReLU neural networks of width n defined on $B_r(0)$:

$$\mathcal{F}_n(r) = \left\{ \sum_{i=1}^{n} c_i \phi(\boldsymbol{w}_i^T \mathbf{x} + b_i) + c_0 \,\middle|\, \forall i, \quad c_i, \quad b_i \in \mathbb{R}, \quad c_i \neq 0, \right.$$
$$\left. \boldsymbol{w}_i \in \mathbb{R}^{d_{\text{in}}}, \quad \text{and} \quad \phi(\boldsymbol{w}_i^T \mathbf{x} + b_i) \quad \text{is active in} \quad B_r(0) \right\}.$$

Given a continuous function f and $\epsilon > 0$, a function class $\mathcal{F}_{m_\epsilon}$ is said to be the ϵ-minimal function class for f if m_ϵ is the smallest number such that $\exists g \in \mathcal{F}_{m_\epsilon}(r)$ and $|g - f| < \epsilon$ in $B_r(0)$. If $\epsilon = 0$, we say $\mathcal{F}_{m_0}(r)$ is the minimal function class for f.

We note that $\mathcal{F}_j \cap \mathcal{F}_s = \emptyset$ for $j \neq s$, and a function $f \in \mathcal{F}_j(r)$ could allow different representations in other function classes $\mathcal{F}_s(r)$ for $s > j$ in $B_r(0)$. For example, $f(x) = x$ on $B_r(0) = [-r, r]$ can be expressed as either $g_1(x) = \phi(x + r) - r \in \mathcal{F}_1(r)$, or $g_2(x) = \phi(x) - \phi(-x) \in \mathcal{F}_2(r)$. However, it cannot be represented by $\mathcal{F}_0(r)$. Thus $\mathcal{F}_1(r)$ is the minimal function class for $f(x) = x$. We remark that g_1 and g_2 are not the same function in $\mathbb{R}$; however, they are the same on $B_r(0)$. Also, note that the existence of m_ϵ in Definition 2 is guaranteed by universal function approximation theorems for shallow neural networks (Cybenko, 1989; Hornik, 1991). Hence, approximating a function whose minimal function class is $\mathcal{F}_m(r)$ is a learning task that requires at least m active neurons. Also, we say any ReLU network of width greater than m_ϵ is overspecified for approximating f within ϵ.

A network is said to be overparameterized if the number of parameters is larger than the number of training data. In this paper we consider the overparameterization, where the size of the width is greater than or equal to the number of training data. Then overparameterization can be understood under the frame of overspecification by the following lemma.

Lemma 2. *For any non-degenerate $(m+1)$ training data, there exists a shallow ReLU network of width m which interpolates all the training data. Furthermore, there exists nondegenerate $(m+1)$ training data such that any shallow ReLU network of width less than m cannot interpolate all the training data. In this sense, m is the minimal width.*

Proof. The proof can be found in Appendix B. □

Lemma 2 shows that any network of width greater than m is overspecified for interpolating $(m+1)$ training data. Thus we could regard overparameterization as a kind of overspecification. Hence, interpolating any nondegenerate $(m+1)$ training data is also a learning task that requires at least m active neurons.

With the trainability obtained in Theorem 1, we show that overparameterization is both a necessary and a sufficient condition for minimizing the loss.

Theorem 2. *For shallow ReLU networks, suppose either the normal (2) or the unit hypersphere (3) initialization with bias is employed in the first hidden layer. Also, the training input domain is $B_r(0)$. For any nondegenerate $(m+1)$ training data, which requires a network to have at least m active neurons for the interpolation, suppose m and the input dimension d_{in} satisfy*

$$1 - (1 - \delta)^{1/m} < \frac{\exp(-C_r d_{in})}{\pi d_{in}}, \quad C_r = -\log(\sin(\tan^{-1}(1/r))), \tag{4}$$

where $0 < \delta < 1$. Then overparameterization is both a necessary and a sufficient condition for interpolating all the training data with probability at least $1 - \delta$ over the random initialization by the (stochastic) gradient-descent method.

Proof. The proof can be found in Appendix C. □

We remark that Theorem 2 assumes that the biases are randomly initialized. To the best of our knowledge, all existing theoretical results also assume the random bias initialization, e.g., Du et al. (2018b), Oymak and Soltanolkotabi (2019), and Li and Liang (2018).

3.2 Trainability of Deep ReLU Networks

We now extend the notion of trainability to deep ReLU networks. Unlike dead ReLU neurons in the first hidden layer, a dead neuron in the tth hidden layer ($t > 1$) could be revived during the training if two conditions are satisfied. One is that for all layers there exists at least one active neuron. This condition is directly obtained from Lemma 1. The other is that the dead neuron should be in the condition of *tentative death*, which will be introduced shortly. We remark that these two conditions are necessary conditions for the revival of a dead neuron. We now provide a precise meaning of the tentative death as follows.

Let us consider a neuron in the tth hidden layer:

$$\phi(\boldsymbol{w}^T \mathbf{x}^{t-1} + b), \quad \mathbf{x}^{t-1} = \phi(\mathcal{N}^{t-1}(\mathbf{x})).$$

Suppose the neuron is dead. For any changes in $\mathbf{x}^{t-1}$, but not in $\boldsymbol{w}$ and b, if the neuron is still dead we say a neuron is *permanently dead*. For example, if $\boldsymbol{w}_j$, $b \leq 0$, and $t > 1$, since $\mathbf{x}^{t-1} \geq 0$, regardless of how $\mathbf{x}^{t-1}$ changes, the neuron will never be active again. Hence, in this case there is no hope that the neuron can be revived during the gradient training; otherwise we say a neuron is *tentatively dead*. Therefore any neuron is always in one of three states: active, tentatively dead, and permanently dead.

We now define the trainability for deep ReLU networks.

Definition 3. For a learning task that requires a L-layer ReLU network having at least m_t active neurons in the tth layer, a L-layer ReLU network with $\boldsymbol{n} = (n_0, n_1, \cdots, n_L)$ architecture is said to be trainable if the number of permanently dead neurons in the tth layer is less than or equal to $n_t - m_t$ for all $1 \leq t < L$. We refer to the probability of a network being trainable at the initialization as trainability.

For $L = 2$, since there is no tentatively dead neuron, Definition 1 becomes a special case of Definition 3.

We now present the trainability results for ReLU networks of depth $L = 3$ at $d_{\text{in}} = 1$. Since each layer can be initialized in different ways, we consider some combinations of them.

Theorem 3. *Suppose the training input domain is $B_r(0)$ and $d_{in} = 1$. For a learning task that requires a three-layer ReLU network having at least m_t active neurons in the tth layer, a three-layer ReLU network with $\boldsymbol{n} = (1, n_1, n_2, n_3)$ architecture is initialized as follows, (here $n_1 \geq m_1$, $n_2 \geq m_2$ and $n_3 = m_3$):*

- *Suppose the "unit hypersphere" (3) initialization without bias is used in the first hidden layer.*
  1. *If the normal (2) initialization without bias is used in the second hidden layer, with probability at least*

$$\sum_{j=m_2}^{n_2} \binom{n_2}{j} \left[\left(1 - \frac{1}{2^{n_1-1}}\right) \frac{3^j}{4^{n_2}} + \frac{1}{2^{n_1+n_2-1}} \right] + Q,$$

where

$$Q = \sum_{j=1}^{m_2-1} \sum_{l=0}^{n_2-m_2} \binom{n_2}{n_2-j-l, j, l} \left[\frac{(1-2^{-n_1})^{n_2-j-l}}{2^{(l+1)n_1+n_2-1}} + \left(1 - \frac{1}{2^{n_1-1}}\right) \frac{3^j (3/4 - 2^{-n_1-1})^{n_2-j-l}}{2^{(n_1+1)l+2j}} \right],$$

the network is trainable.

2. *If the normal (2) initialization with bias is used in the second hidden layers, with probability at least*

$$(1-\hat{p}_d(r))^{n_1}\left[\sum_{j=m_2}^{n_2}\binom{n_2}{j}\mathbb{E}_s\left[(1-\mathfrak{p}_2(s))^j\mathfrak{p}_2(s)^{n_2-j}\right]+Q\right],$$

where $\hat{p}_d(r)$ is defined in Theorem 1, $s\sim B(n_1,1/2)$, $\alpha_s=\tan^{-1}(s/(n_1-s))$, $g(x)=\sin(\tan^{-1}(x))$, and

$$\mathfrak{p}_2(s)=\frac{1}{2}+\left\{\int_{\pi/2}^{\pi+\alpha_s}\frac{g(r\sqrt{s}\cos(\theta))}{4\pi}d\theta+\int_{\pi+\alpha_s}^{2\pi}\frac{g(r\sqrt{n_1-s}\sin(\theta))}{4\pi}d\theta\right\},$$

$$Q=\sum_{j=1}^{m_2-1}\sum_{l=0}^{n_2-m_2}\binom{n_2}{n_2-j-l,j,l}\times\mathbb{E}_s\left[(1-\mathfrak{p}_2(s))^j(\mathfrak{p}_2(s)-2^{-n_1-1})^{n_2-j-l}(2^{-n_1-1})^l\right],$$

the network is trainable.

- *Suppose the unit hypersphere (3) initialization with bias is used in the first hidden layer.*

1. *If the normal (2) initialization without bias is used in the second hidden layer and $n_1=m_1=1$, with probability at least*

$$2^{-n_2}\sum_{j=m_2}^{n_2}\binom{n_2}{j}+\sum_{j=1}^{m_2-1}\sum_{l=0}^{n_2-m_2}\binom{n_2}{n_2-j-l,j,l}2^{-2n_2+j},$$

the network is trainable.

2. *If the normal (2) initialization with bias is used in the second hidden layers and $n_1=m_1=1$, with probability at least*

$$(1-\hat{p}_d(r))^{n_1}\left[\sum_{j=m_2}^{n_2}\binom{n_2}{j}\mathbb{E}_\omega\left[(1-\mathfrak{p}_2(\omega))^j\mathfrak{p}_2(\omega)^{n_2-j}\right]+Q\right],$$

where $\hat{p}_d(r)$ is defined in Theorem 1, $\alpha_r=\tan^{-1}(r)$, $\omega\sim\mathrm{Unif}\,(0,\pi/2+\alpha_r)$, $g(x)=\tan^{-1}(1/[\sqrt{r^2+1}\cos(x)])$, and

$$\mathfrak{p}_2(\omega)=\begin{cases}\dfrac{1}{4}+\dfrac{g(\omega-\alpha_r)}{2\pi},\\ \quad \textit{if}\quad \omega\in\left[\dfrac{\pi}{2}-\alpha_r,\dfrac{\pi}{2}+\alpha_r\right),\\ \dfrac{1}{4}+\dfrac{g(\omega-\alpha_r)+\tan^{-1}(\sqrt{r^2+1}\cos(\omega+\alpha_r))}{2\pi},\\ \quad \textit{if}\quad \omega\in\left[0,\dfrac{\pi}{2}-\alpha_r\right),\end{cases}$$

$$Q = \sum_{j=1}^{m_2-1} \sum_{l=0}^{n_2-m_2} \binom{n_2}{n_2-j-l, j, l} \mathbb{E}_\omega \left[(1-\mathfrak{p}_2(\omega))^j (\mathfrak{p}_2(\omega) - 2^{-2})^{n_2-j-l} (2^{-2})^l \right],$$

the network is trainable.

Proof. The proof can be found in Appendix F. □

Theorem 3 suggests us to use a ReLU network with sufficiently large width at each layer to secure a high trainability. Also, it is clear that different initialization schemes result in different trainabilities. Our proof is built on the study of the probability distribution of the number of active neurons (see Lemma 6). In Fig. E1 of Appendix E, we illustrate the active neuron distributions by three different initialization schemes.

At last, we present an upper bound of the trainability when the biases are initialized to zeros.

Corollary 1. *For a learning task that requires a L-layer ReLU network having at least m_t active neurons in the tth layer, suppose that all weights are independently initialized from the normal (2) initialization without bias, and $d_{in} = 1$. Then the trainability of a L-hidden layer ReLU network having $n \geq m_t$ neurons at each layer is bounded above by*

$$\mathfrak{a}_1^{L-1} - \frac{(1-2^{-n+1})(1-2^{-n})}{1+(n-1)2^{-n}}(-\mathfrak{a}_1^{L-1} + \mathfrak{a}_2^{L-1}),$$

where $\mathfrak{a}_1 = 1-2^{-n}$ and $\mathfrak{a}_2 = 1-2^{-n+1}-(n-1)2^{-2n}$.

Proof. The proof can be found in Appendix G. □

Further characterization will be deferred to a future study, but a general formulation is established and can be found in Lemma 8 in Appendix F.

In principle, a single active neuron in the highest layer could potentially revive tentatively dead neurons through backpropagation (gradient). However, in practice it would be better for an initialized network to have at least m_t active neurons in the tth hidden layer for both faster training and robustness. Let A be the event that a ReLU network has at least m_t active neurons in the tth hidden layer for $t = 1, \cdots, L$. The probability of A is then a naive lower bound of trainability. Hence, having a high probability of A enforces a high trainability.

Remark 1. *A trainable network itself does not guarantee successful training. However, if a network is not trainable, there is no hope for the network to be trained successfully. Thus, a network being trainable is a necessary condition for successful training, and the trainability serves as an upper bound of the training success rate. The demonstration of trainability is given in Section 5.*

Remark 2. *Definition 3 (also Definition 1) requires the number of active neurons m_t needed for a learning task. Since neural networks are universal approximators (Cybenko, 1989; Hornik, 1991), the existence of m_t's is guaranteed; however, the exact determination of m_t's is challenging for a general learning task. This is also related to the design of network architecture. In practice, m_t's could be estimated based on trial and error or the practitioner's expertise.*

4. DATA-DEPENDENT BIAS INITIALIZATION: SHALLOW ReLU NETWORKS

In this section, we investigate the bias initialization in gradient-based training. In terms of trainability for shallow ReLU networks, Theorem 1 indicates that the zero-bias initialization would be preferred over the random-bias initialization. In practice, however, the zero-bias initialization often finds a spurious local minimum or gets stuck on a flat plateau. To illustrate this difficulty, we consider a problem of approximating a sum of two sine functions $f(x) = \sin(4\pi x) + \sin(6\pi x)$ on $[-1, 1]$. For this task, we use a shallow ReLU network of width 500 with the He initialization without bias. In order to reduce extra randomness in the experiment, 100 equidistant points on $[-1, 1]$ are used as the training data set. One of the most popular gradient-based optimization methods, `Adam` (Kingma and Ba, 2015), is employed with its default parameters. We use the full-batch size and set the maximum number of epochs to 15,000. The trained network is plotted in Fig. 2. It is clear that the trained network is stuck on a local minimum. A similar behavior is repeatedly observed in all of our multiple independent simulations.

This phenomenon could be understood as follows. Since the biases are zero, all initialized neurons are clustered at the origin. Consequently, it would take a long time for a gradient update to distribute neurons over the training domain to achieve a small training loss. In the worst case, along the way of distributing neurons it will find a spurious local minimum. We refer to this problem as the *clustered neuron problem*. Indeed, this is observed in Fig. 2. The trained network well approximates the target function on a small domain containing the origin, however, it loses its accuracy on the domain far from the origin.

On the other hand, if we randomly initialize the bias, as shown in Theorem 1, overspecification is inevitable to guarantee a certain number of active neurons. In this setting, at the initialization only 375 neurons will be active among 500 neurons on average. In Fig. 2, we also show the trained result by the He initialization with bias. Since neurons are now randomly distributed over the entire domain, the trained network approximates quite well the target function.

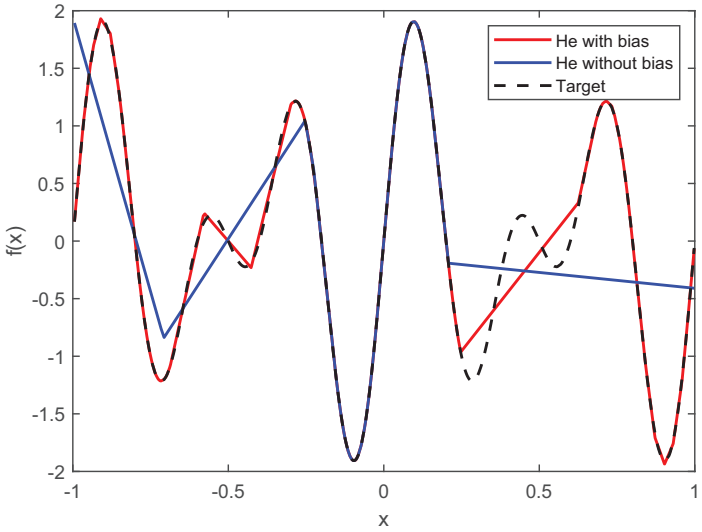


FIG. 2: The trained networks for approximating $f(x) = \sin(4\pi x) + \sin(6\pi x)$ by the He initialization without bias and with bias. A shallow ReLU network of width 500 is employed. The target function $f(x)$ is also plotted.

However, the randomness may locate some neurons in places that may lead to a spurious local minimum or a slow training. In the worst case, some neurons would never be activated. In this example the trained network by the random-bias initialization loses its accuracy at some parts of the domain, e.g., in the intervals containing ± 0.5.

In order to overcome such difficulties and accelerate the gradient-based training, we propose a new data-dependent initialization scheme. The scheme is for the overparameterized setting, where the size of width is greater than or equal to the number of training data. By adapting the trainability perspective, the method is designed to alleviate both the clustered neuron problem and the dying ReLU neuron problem at the same time. This is done by efficiently locating each neuron based on the training data.

Remark 3. *We aim to study the effect of bias initialization on gradient-based training. Interpolating all the training data results in a zero training loss. However, we do not simply attempt to interpolate the training data, which can be done by an explicit construction shown in Lemma 2. We remark that the idea of data-dependent initialization is not new; see Ioffe and Szegedy (2015), Krähenbühl et al. (2015), and Salimans and Kingma (2016). However, our method is specialized to the overparameterized setting.*

4.1 Data-Dependent Bias Initialization

Let m be the number of training data and n be the width of a shallow ReLU network. Suppose the network is overparameterized so that $n = hm$ for some positive number $h \geq 1$. We then propose to initialize the biases as follows:

$$\boldsymbol{b}_i = -\boldsymbol{w}_i^T \mathbf{x}_{j_i} + |\epsilon_i|, \quad \epsilon_i \sim N(0, \sigma_e^2),$$

where ϵ_i's are iid and $j_i - 1 = (i-1) \mod m$. We note that this mimics the explicit construction for the data interpolation in Lemma 2. By doing so, the ith neuron is initialized to be located near $\mathbf{x}_{j_i}$ as

$$\phi(\boldsymbol{w}_i^T(\mathbf{x} - \mathbf{x}_{j_i}) + |\epsilon_i|).$$

The precise value of σ_e^2 is determined as follows. Let $q(\mathbf{x})$ be the expectation of the normalized squared norm of the network, i.e., $q(\mathbf{x}) := \mathbb{E}[\|\mathcal{N}(\mathbf{x})\|_2^2]/d_{\text{out}}$, where the expectation is taken over weights and biases and $\mathcal{N}(\mathbf{x})$ is a shallow ReLU network having $\boldsymbol{n} = (d_{\text{in}}, n, d_{\text{out}})$ architecture. Given a set of training input data $\mathcal{X}_m = \{\mathbf{x}_i\}_{i=1}^m$, we define the average of $q(\mathbf{x})$ on $\mathcal{X}_m$ as

$$\mathbb{E}_{\mathcal{X}_m}[q(\mathbf{x})] := \frac{1}{m}\sum_{i=1}^m q(\mathbf{x}_i).$$

We then choose our parameters to match $\mathbb{E}_{\mathcal{X}_m}[q(\mathbf{x})]$ by our data-dependent initialization to the one by the standard initialization method. For example, when the normal (2) initialization without bias is used, we have

$$\mathbb{E}_{\mathcal{X}_m}[q(\mathbf{x})] := \frac{n\sigma_{\text{out}}^2\sigma_{\text{in}}^2}{2m}\|\boldsymbol{X}\|_F^2, \quad \boldsymbol{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_m],$$

where $\boldsymbol{W}_j^1 \sim N(0, \sigma_{\text{in}}^2 \boldsymbol{I}_{d_{\text{in}}})$ for $1 \leq j \leq n$, $\boldsymbol{W}_i^2 \sim N(0, \sigma_{\text{out}}^2 \boldsymbol{I}_n)$ for $1 \leq i \leq d_{\text{out}}$, and $\|\cdot\|_F$ is the Frobenius norm. When the "He initialization" without bias is used, i.e., $\sigma_{\text{in}}^2 = 2/d_{\text{in}}$ and $\sigma_{\text{out}}^2 = 2/n$, we have $\mathbb{E}_{\mathcal{X}_m}[q(\mathbf{x})] = 2\|\boldsymbol{X}\|_F^2/(d_{\text{in}}m)$.

Theorem 4. *For a shallow network of width n, suppose $n = hm$ for some positive number $h \geq 1$, where m is the number of training data. Let $\mathcal{X}_m = \{\boldsymbol{x}_i\}_{i=1}^m$ be the set of training input data. Suppose $\boldsymbol{W}_j^1 \sim N(0, \sigma_{in}^2 \boldsymbol{I}_{d_{in}})$ for $1 \leq j \leq n$, $\boldsymbol{W}_i^2 \sim N(0, \sigma_{out}^2 \boldsymbol{I}_n)$ for $1 \leq i \leq d_{out}$, $\boldsymbol{b}^2 = 0$, and $\boldsymbol{b}^1$ is initialized by the proposed method with $\sigma_e = s\sigma_{in}$. Then*

$$\mathbb{E}_{\mathcal{X}_m}[q(\boldsymbol{x})] = \frac{h\sigma_{out}^2\sigma_{in}^2}{m\pi} \sum_{k,i=1}^{m} \left[(s^2 + \Delta_{k,i}^2)h(s/\Delta_{k,i}) + s\Delta_{k,i}\right],$$

where $h(x) = \tan^{-1}(x) + \pi/2$, and $\Delta_{k,i} = \|\boldsymbol{x}_k - \boldsymbol{x}_i\|_2$.

Proof. The proof can be found in Appendix H. □

For example, if we set σ_{in}, σ_{out}, and σ_e to be

$$\sigma_{\text{in}}^2 = \frac{2}{d_{\text{in}}}, \quad \sigma_e^2 = 0, \quad \sigma_{\text{out}}^2 = \frac{1}{h} \cdot \frac{\sum_j \|\mathbf{x}_j\|^2}{\sum_{k<i} \|\mathbf{x}_k - \mathbf{x}_i\|^2}, \tag{5}$$

$\mathbb{E}_{\mathcal{X}_m}[q(\mathbf{x})]$ by the data-dependent initialization is equal to the one by the He initialization without bias.

The proposed initialization ensure that all neurons are equally distributed over the training data points. Also, it would ensure that at least one neuron will be activated at a training datum. By doing so, it would effectively avoid both the clustered neuron problem and the dying ReLU neuron problem. Furthermore, it locates all neurons in favor of the training data points with a hope that such a neuron configuration accelerates the training.

In Fig. 3, we demonstrate the performance of the proposed method in approximating the sum of two sine functions. On the left, the trained neural network is plotted, and on the right, the RMSE of the training loss are plotted with respect to the number of epochs by three different initialization methods. We remark that since the training set is deterministic and the fullbatch is used, the only randomness in the training process is from the weights and biases initialization. It can be seen that the proposed method not only results in the fastest convergence but also achieves the smallest approximation error among others. The number of dead neurons in the trained network is 127 (He with bias), 3 (He without bias), and 17 (data-dependent).

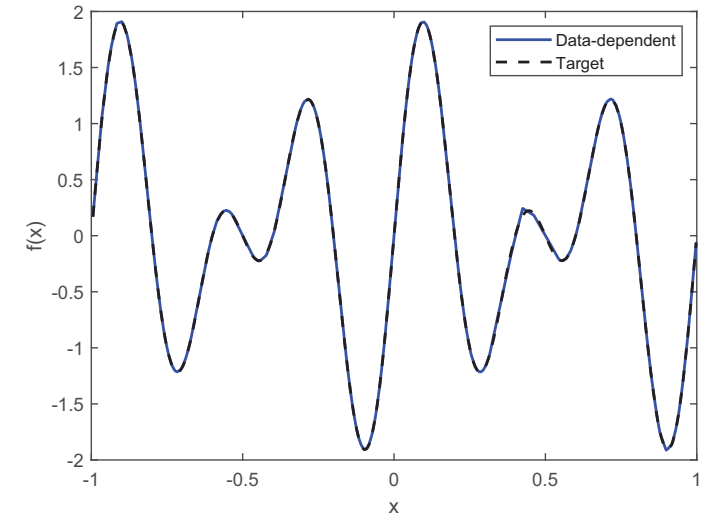
5. NUMERICAL EXAMPLES

We present numerical examples to demonstrate our theoretical findings and the effectiveness of the proposed data-dependent initialization method.

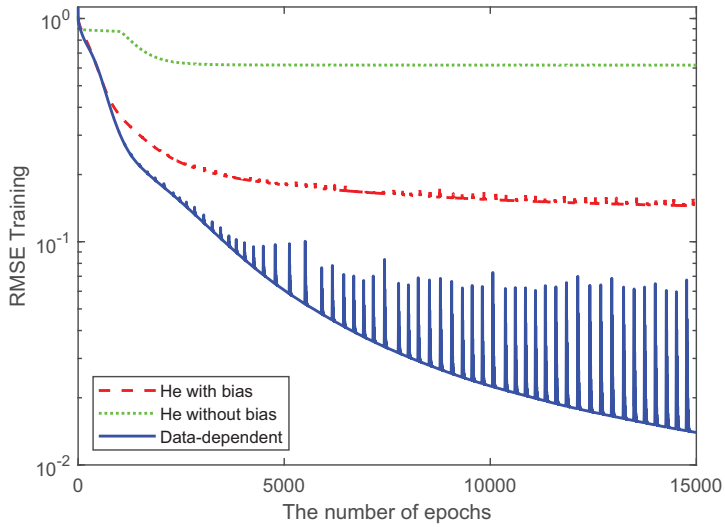
5.1 Trainability of Shallow ReLU Networks

We present two examples to demonstrate the trainability of a shallow ReLU neural network and justify our theoretical results. Here all the weights and biases are initialized according to the He initialization (2) with bias. We consider two univariate test target functions:

$$f_1(x) = |x| = \max\{x, 0\} + \max\{-x, 0\},$$
$$f_2(x) = |x| - \frac{\sqrt{3}}{\sqrt{3}-1}\max\{x-1, 0\} - \frac{\sqrt{3}}{\sqrt{3}-1}\max\{-x-1, 0\}.$$



(a)



(b)

FIG. 3: (a) The trained network for approximating $f(x) = \sin(4\pi x) + \sin(6\pi x)$ by the proposed data-dependent initialization. A shallow ReLU network of width 500 is employed. (b) The root-mean-square error of the training loss with respect to the number of epochs of `Adam` (Kingma and Ba, 2015).

We note that $\mathcal{F}_2$ is the minimal function class (see Definition 2) for $f_1(x)$, and $\mathcal{F}_4$ is the minimal function class for $f_2(x)$. That is, theoretically, f_1 and f_2 should be exactly recovered by a shallow ReLU network of width 2 and 4, respectively. For the training, we use a training set of 600 data points uniformly generated from $[-\sqrt{3}, \sqrt{3}]$ and a test set of 1,000 data points uniformly generated from $[-\sqrt{3}, \sqrt{3}]$. We employ the standard stochastic gradient descent with minibatch of size 128 and a constant learning rate of 10^{-3}. We set the maximum number of epochs to 10^6 and use the standard square loss.

In Fig. 4, we show the approximation results for approximating $f_1(x) = |x|$. On the left we plot the empirical probability of successful training with respect to the value of width. The empirical probabilities are obtained from 1,000 independent simulations, and a single simulation is regarded as a success if the test error is less than 10^{-2}. We also plot the trainability from Theorem 1. As expected, it provides an upper bound for the probability of successful training. It is clear that the more the network is overspecified, the higher trainability is obtained. Also, it can be seen that as the size of the width grows, the empirical training success rate increases. This suggests that a successful training could be achieved (with high probability) by having a very high trainability. However, since it is only a necessary condition, although an initialized network is in $\mathcal{F}_j$ for $j \geq 2$, i.e., trainable, the final trained result could be in either $\mathcal{F}_1$ or $\mathcal{F}_0$, as shown in the middle and right of Fig. 4, respectively.

Similar behavior is observed for approximating $f_2(x)$. In Fig. 5 we show the approximation results for $f_2(x)$. On the left, both the empirical probability of successful training and the trainability (Theorem 1) are plotted with respect to the size of width. Again, the trainability provides an upper bound for the probability of successful training. Also, it can be seen that the empirical training success rate increases as the width size grows. On the middle and right, we plot two of local minima which a trainable network could end up with. We remark that the choice of gradient-based optimization methods, well-tuned learning rate, and/or other tunable optimization parameters could affect the empirical training success probability. However, the maximum probability one can hope for is bounded by the trainability. In all of our simulations, we did not tune any optimization hyperparameters.

5.2 Data-Dependent Bias Initialization

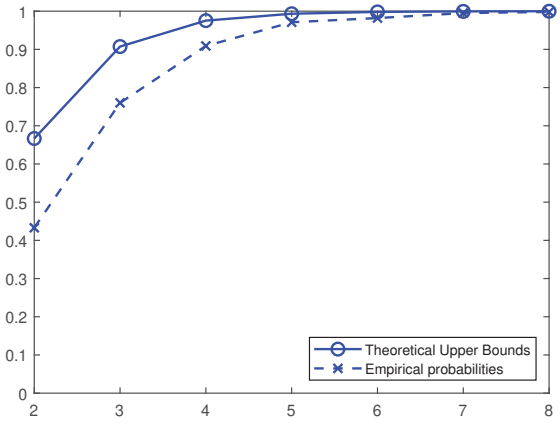
Next, we compare the training performance of three initialization methods. The first one is the He initialization (He et al., 2015) without bias. This corresponds to $\boldsymbol{W}^1 \sim N(0, 2/d_{\text{in}}), \boldsymbol{W}^2 \sim N(0, 2/n), \boldsymbol{b}^1 = 0, \boldsymbol{b}^2 = 0$. The second one is the He initialization with bias (2). This corresponds to $[\boldsymbol{W}^1, \boldsymbol{b}^1] \sim N(0, 2/(d_{\text{in}} + 1)), [\boldsymbol{W}^2, \boldsymbol{b}^2] \sim N(0, 2/(n + 1))$. Here n is the width of the first hidden layer. The last one is the proposed data-dependent initialization described in the previous section. We use the parameters from (5). All results are generated under the same conditions, except for the weights and biases initialization.

We consider the following $d_{\text{in}} = 2$ test functions on $[-1, 1]^2$:

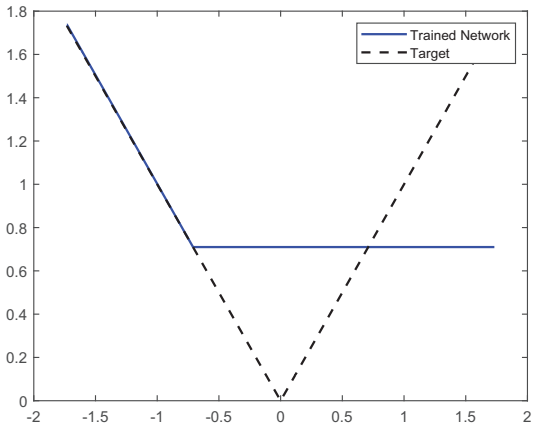
$$\begin{aligned} f_3(\mathbf{x}) &= \sin(\pi x_1)\cos(\pi x_2)e^{-x_1^2 - x_2^2}, \\ f_4(\mathbf{x}) &= \sin(\pi(x_1 - x_2))e^{x_1 + x_2}. \end{aligned} \tag{6}$$

In all tests, we employ a shallow ReLU network of width 100, and it is trained over 25 randomly uniformly drawn points from $[-1, 1]^2$. We employ the gradient-descent method with momentum with the square loss. The learning rate is a constant of 0.005, and the momentum term is 0.9.

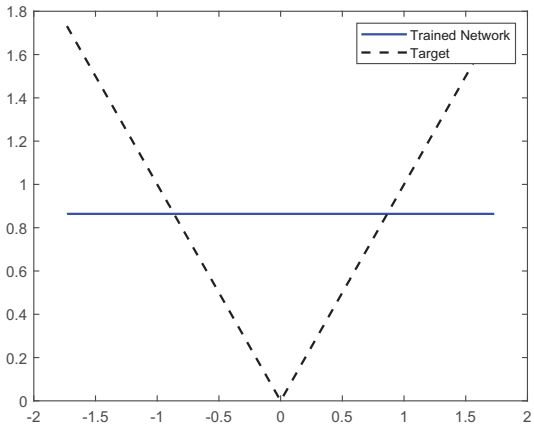
Figure 6 shows the mean of the RMSE on the training data from 10 independent simulations with respect to the number of epochs by three different initialization methods. The shaded area covers plus or minus one standard deviation from the mean. On the left and right, the results for approximating $f_3(x)$ and $f_4(x)$ are shown, respectively. We see that the data-dependent initialization not only results in the faster loss convergence but also achieves the smallest training loss. Also, the average number of dead neurons in the trained network is 11 (He with bias), 0 (He without bias), and 0 (data-dependent) for f_3, and 12 (He with bias), 0 (He without bias), and 0



(a)

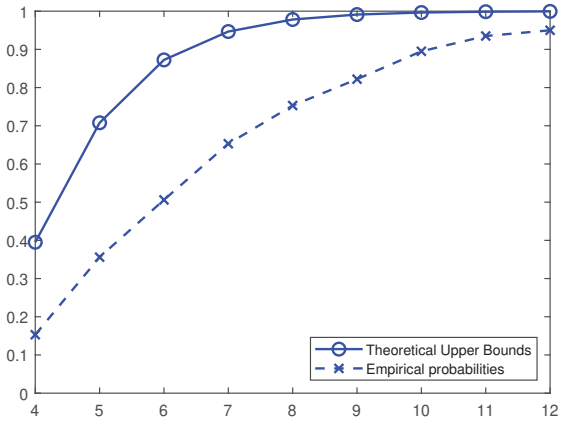


(b)

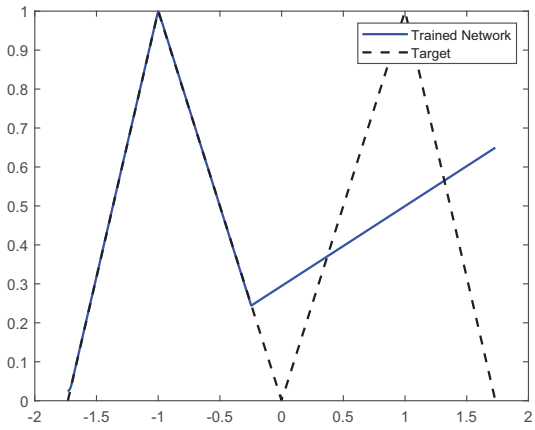


(c)

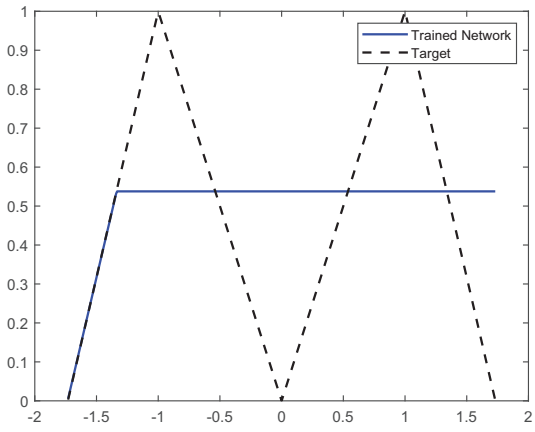
FIG. 4: (a) The empirical probability that a network approximates $f_1(x)$ successfully and the probability that a network is trainable (Theorem 1) with respect to the size of width n, and a trained network which falls in (b) $\mathcal{F}_1$ and (c) $\mathcal{F}_0$



(a)

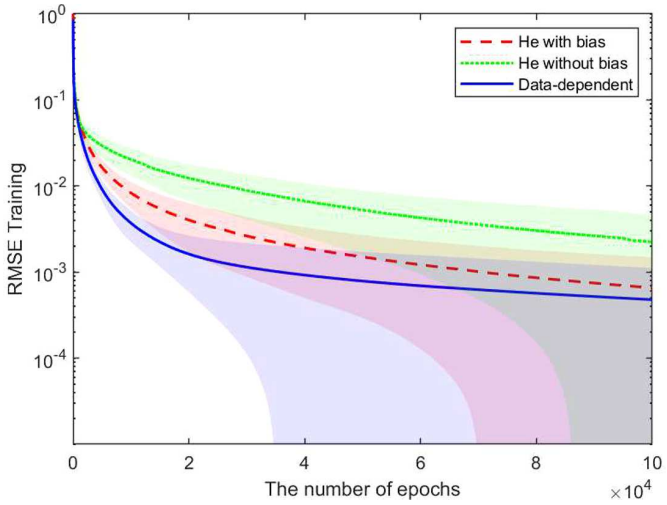


(b)

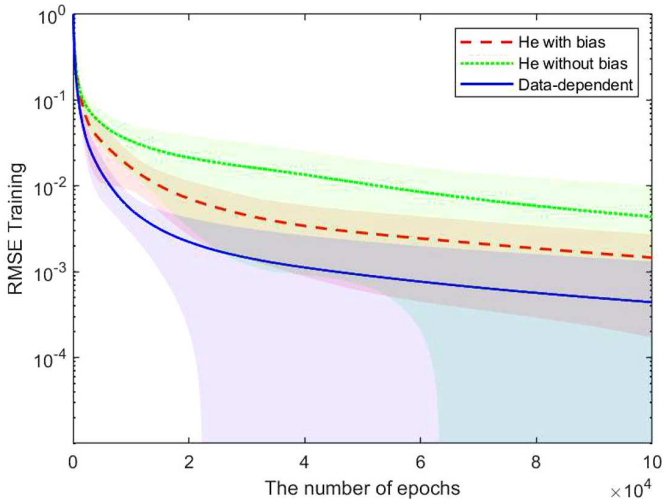


(c)

FIG. 5: (a) The empirical probability that a network approximates $f_2(x)$ successfully and the probability that a network is trainable (Theorem 1) with respect to the size of width n, and a trained network which falls in (b) $\mathcal{F}_3$ and (c) $\mathcal{F}_2$



(a)



(b)

FIG. 6: The convergence of the root-mean-square error on the training data for approximating (a) f_3 and (b) f_4 with respect to the number of epochs of the gradient descent with moment by three different initialization methods. A shallow (one-hidden layer) ReLU network of width 100 is employed. The shaded area covers plus or minus one standard deviation from the mean.

(data-dependent) for f_4. Together with the example in Section 4, all examples demonstrate the effectiveness of the proposed data-dependent initialization.

6. CONCLUSION

In this paper we establish the trainability of ReLU neural networks, a necessary condition for successful training, and propose a data-dependent initialization scheme for better training.

Upon introducing two states of dead neurons, tentatively dead and permanently dead, we define a trainable network. A network is trainable if it has sufficiently small permanently dead neurons. We show that a network being trainable is a necessary condition for the successful training. We refer to the probability of a randomly initialized network being trainable as *trainability*. The trainability serves as an upper bound of training success rates. We establish a general formulation for computing trainability and derive the trainabilities of some special cases. For shallow ReLU networks, by utilizing the computed trainability we show that overparameterization is both a neccessary and a sufficient condition for interpolating all training data, i.e., minimizing the loss.

Motivated by our theoretical results, we propose a data-dependent initialization scheme in the over-parameterized setting, where the size of width is greater than or equal to the number of training data. The proposed method is designed to avoid both the dying ReLU neuron problem and to efficiently locate all neurons at initialization for the faster training. Numerical examples are provided to demonstrate the performance of our method. We found that the data-dependent initialization method outperforms the He initialization, both with and without bias, in all of our tests.

ACKNOWLEDGMENTS

This work is supported by the DOE PhILMs project (No. de-sc0019453), the DARPA AIRA (Grant No. HR00111990025), and the AFOSR (Grant No. FA9550-17-1-0013).

REFERENCES

Allen-Zhu, Z., Li, Y., and Song, Z., A Convergence Theory for Deep Learning via Over-Parameterization, arXiv preprint, 2018. arXiv: 1811.03962

Byrd, R.H., Lu, P., Nocedal, J., and Zhu, C., A Limited Memory Algorithm for Bound Constrained Optimization, *SIAM J. Sci. Comput.*, vol. **16**, no. 5, pp. 1190–1208, 1995.

Cybenko, G., Approximation by Superpositions of a Sigmoidal Function, *Math. Control, Signals Sys.*, vol. **2**, no. 4, pp. 303–314, 1989.

Du, S.S., Lee, J.D., Li, H., Wang, L., and Zhai, X., Gradient Descent Finds Global Minima of Deep Neural Networks, arXiv preprint, 2018a. arXiv: 1811.03804

Du, S.S., Zhai, X., Poczos, B., and Singh, A., Gradient Descent Provably Optimizes Over-Parameterized Neural Networks, arXiv preprint, 2018b. arXiv: 1810.02054

Duchi, J., Hazan, E., and Singer, Y., Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, *J. Machine Learning Res.*, vol. **12**, no. Jul, pp. 2121–2159, 2011.

Glorot, X. and Bengio, Y., Understanding the Difficulty of Training Deep Feedforward Neural Networks, *Int. Conf. on Artificial Intelligence and Statistics*, pp. 249–256, Sardinia, Italy, May 13–15, 2010.

He, K., Zhang, X., Ren, S., and Sun, J., Delving Deep into Rectifiers: Surpassing Human-Level Performance on Imagenet Classification, *IEEE Int. Conf. on Computer Vision*, pp. 1026–1034, Santiago, Chile, December 13–16, 2015.

Hinton, G., Overview of Mini-Batch Gradient Descent, accessed from http://www.cs.toronto.edu/∼tijmen/csc321/slides/lecture_slides_lec6.pdf, 2014.

Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., and Kingsbury, B., Deep Neural Networks for Acoustic Modeling in Speech Recognition, *IEEE Signal Process. Mag.*, vol. **29**, no. 6, pp. 82–97, 2012.

Hornik, K., Approximation Capabilities of Multilayer Feedforward Networks, *Neural Networks*, vol. **4**, no. 2, pp. 251–257, 1991.

Ioffe, S. and Szegedy, C., Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, *Proc. of the 32nd Int. Conf. on Machine Learning*, vol. **37**, pp. 448–456, 2015.

Kingma, D.P. and Ba, J., Adam: A Method for Stochastic Optimization, *Int. Conf. on Learning Representations*, San Diego, CA, USA, May 7–9, 2015.

Krähenbühl, P., Doersch, C., Donahue, J., and Darrell, T., Data-Dependent Initializations of Convolutional Neural Networks, arXiv preprint, 2015. arXiv: 1511.06856

Krizhevsky, A., Sutskever, I., and Hinton, G., Imagenet Classification with Deep Convolutional Neural Networks, *Adv. Neural Inf. Proc. Sys.*, vol. **25**, pp. 1097–1105, 2012.

LeCun, Y., Bottou, L., Orr, G.B., and Müller, K.R., Efficient Backprop, *Neural Networks: Tricks of the Trade*, Berlin–Heidelberg, Germany: Springer, pp. 9–48, 2012

Leopardi, P.C., Distributing Points on the Sphere: Partitions, Separation, Quadrature and Energy, PhD, University of New South Wales, Sydney, Australia, 2007.

Li, Y. and Liang, Y., Learning Overparameterized Neural Networks via Stochastic Gradient Descent on Structured Data, *Adv. Neural Inf. Proc. Sys.*, vol. **31**, pp. 8157–8166, 2018.

Livni, R., Shalev-Shwartz, S., and Shamir, O., On the Computational Efficiency of Training Neural Networks, *Adv. Neural Inf. Proc. Sys.*, vol. **27**, pp. 855–863, 2014.

Lu, L., Shin, Y., Su, Y., and Karniadakis, G.E., Dying ReLU and Initialization: Theory and Numerical Examples, arXiv preprint, 2019. arXiv: 1903.06733

Mishkin, D. and Matas, J., All You Need Is a Good Init, *Int. Conf. on Learning Representations*, San Juan, Puerto Rico, USA, May 2–4, 2016.

Nguyen, Q. and Hein, M., The Loss Surface of Deep and Wide Neural Networks, *Proc. of the 34nd Int. Conf. on Machine Learning*, vol. **70**, pp. 2603–2612, 2017.

Oymak, S. and Soltanolkotabi, M., Towards Moderate Overparameterization: Global Convergence Guarantees for Training Shallow Neural Networks, arXiv preprint, 2019. arXiv: 1902.04674

Reddi, S.J., Kale, S., and Kumar, S., On the Convergence of Adam and Beyond, arXiv preprint, 2019. arXiv: 1904.09237

Robbins, H. and Monro, S., A Stochastic Approximation Method, *Annals Math. Stat.*, vol. **22**, no. 3, pp. 400–407, 1951.

Ruder, S., An Overview of Gradient Descent Optimization Algorithms, arXiv preprint, 2016. arXiv: 1609.04747

Rumelhart, D.E., Hinton, G.E., and Williams, R.J., Learning Internal Representations by Error Propagation, Tech. Rep., California University San Diego, La Jolla Institute for Cognitive Science, 1985.

Safran, I. and Shamir, O., On the Quality of the Initial Basin in Overspecified Neural Networks, *Proc. of the 33rd Int. Conf. on Machine Learning*, vol. **48**, pp. 774–782, 2016.

Salimans, T. and Kingma, D.P., Weight normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks, *Adv. Neural Inf. Proc. Sys.*, vol. **29**, pp. 901–909, 2016.

Saxe, A.M., McClelland, J.L., and Ganguli, S., Exact Solutions to the Nonlinear Dynamics of Learning in Deep Linear Neural Networks, *Int. Conf. Learning Representations*, Banff, Canada, April 14–16, 2014.

Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Driessche, G.V.D., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., and Lanctot, M., Mastering the Game of Go with Deep Neural Networks and Tree Search, *Nature*, vol. **529**, no. 7587, p. 484, 2016.

Soltanolkotabi, M., Javanmard, A., and Lee, J.D., Theoretical Insights into the Optimization Landscape of

Over-Parameterized Shallow Neural Networks, *IEEE Transact. Inf. Theor.*, vol. **65**, no. 2, pp. 742–769, 2019.

Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al., Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, arXiv preprint, 2016. arXiv: 1609.08144

Zou, D., Cao, Y., Zhou, D., and Gu, Q., Stochastic Gradient Descent Optimizes Over-Parameterized Deep RelU Networks, arXiv preprint, 2018. arXiv: 1811.08888

APPENDIX A. PROOF OF LEMMA 1

Proof. Suppose a ReLU neural network $\mathcal{N}(x)$ of width N is initialized to be

$$\mathcal{N}(x;\theta) = \sum_{i=1}^{n} c_i \phi(\boldsymbol{w}_i^T \mathbf{x} + b_i) + \sum_{i=n+1}^{N} c_i \phi(\boldsymbol{w}_i^T \mathbf{x} + b_i) + c_0,$$

where $\theta = [(c_i, \boldsymbol{w}_i, b_i)_{i=1}^N, c_0]$, and the second term on the right is a constant function on $B_r(0)$. Let $Z(x) = \sum_{i=n+1}^N c_i \phi(\boldsymbol{w}_i^T \mathbf{x} + b_i)$. Given a training data set $\mathcal{T}_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ where $\{\mathbf{x}_i\}_{i=1}^m \subset B_r(0)$, and a loss metric $\ell : \mathbb{R}^{d_{\text{out}}} \times \mathbb{R}^{d_{\text{out}}} \mapsto \mathbb{R}$, the loss function is $\mathcal{L}(\theta; \mathcal{T}_m) = \sum_{i=1}^m \ell[\mathcal{N}(\mathbf{x}_i;\theta), y_i]$. The gradients of the loss function $\mathcal{L}$ with respect to parameters are

$$\frac{\partial}{\partial \theta}\mathcal{L}(\theta;\mathcal{T}_m) = \sum_{(x,y)\in\mathcal{T}_m} \ell'[\mathcal{N}(x;\theta), y] \frac{\partial}{\partial \theta}\mathcal{N}(x;\theta). \tag{A.1}$$

Then for $i = 1, \cdots, N$, we have $[\partial/(\partial \boldsymbol{w}_i)]\mathcal{N}(x;\theta) = \phi'(\boldsymbol{w}_i^T \mathbf{x} + b_i)\mathbf{x}$ and $[\partial/(\partial b_i)] \times \mathcal{N}(x;\theta) = \phi'(\boldsymbol{w}_i^T \mathbf{x} + b_i)$. Since $Z(\mathbf{x})$ is a constant function on Ω, for $i = n+1, \cdots, N$, we have $\phi'(\boldsymbol{w}_i^T \mathbf{x} + b_i) = 0$ for all $\mathbf{x} \in B_r(0)$. Therefore any gradient-based optimization method does not update $(c_i, \boldsymbol{w}_i, b_i)_{i=n+1}^N$, which makes $Z(\mathbf{x})$ remain a constant function in $B_r(0)$.

It follows from Lemma 10 of Lu et al. (2019) that with probability 1, a network is initialized to be a constant function if and only if there exists a hidden layer such that all neurons are dead. Thus all dead neurons cannot be revived through gradient-based training. □

APPENDIX B. PROOF OF LEMMA 2

Proof. Given a set of nondegenerate m data, $\{\mathbf{x}_i, y_i\}_{i=1}^m$, for $d_{\text{in}} = 1$, suppose $x_1 < x_2 < \cdots < x_m$ and for $d_{\text{in}} > 1$, we choose a vector $\boldsymbol{w}$ such that $\boldsymbol{w}^T \mathbf{x}_1 < \cdots < \boldsymbol{w}^T \mathbf{x}_m$. We note that one can always find such $\boldsymbol{w}$. Let

$$S_{ij} = \{\boldsymbol{w} \in \mathbb{S}^{d_{\text{in}}-1} | \boldsymbol{w}^T(\mathbf{x}_i - \mathbf{x}_j) = 0\}, \quad i \neq j.$$

Since $\mathbf{x}_i$'s are distinct, S_{ij} is a Lebesgue measure zero set. Thus $\cup_{1\le i<j\le m} S_{ij}$ is also a measure zero set. Therefore the Lebesgue measure of $\cap_{1\le i<j\le m} S_{ij}^c$ is positive and thus it is nonempty. Then, any vector $\boldsymbol{w} \in \cap_{1\le i<j\le m} S_{ij}^c$ satisfies the condition.

We recursively define shallow ReLU networks; for $j = 0, \cdots, m$,

$$\mathcal{N}(\mathbf{x}; j) = y_1 + \sum_{i=1}^{j} c_i \phi[\boldsymbol{w}^T(\mathbf{x} - \mathbf{x}_i)], \quad c_i = \frac{y_{i+1} - \mathcal{N}(\mathbf{x}_{i+1}; i-1)}{\boldsymbol{w}^T(\mathbf{x}_{i+1} - \mathbf{x}_i)}.$$

Then it can be checked that $\mathcal{N}(\mathbf{x}_j; m-1) = y_j$ for all j. Since $\boldsymbol{w}^T(x_j - x_k) \le 0$ for all $k \ge j$, $\mathcal{N}(\mathbf{x}_j; m-1) = \mathcal{N}(\mathbf{x}_j; j-1)$. Also, since $\mathcal{N}(\mathbf{x}_j; j-1) = \mathcal{N}(\mathbf{x}_j; j-2) + c_{j-1}\boldsymbol{w}^T(\mathbf{x}_j - \mathbf{x}_{j-1})$ and $c_{j-1} = [y_j - \mathcal{N}(\mathbf{x}_j; j-2)]/[\boldsymbol{w}^T(\mathbf{x}_j - \mathbf{x}_{j-1})]$, we have $\mathcal{N}(\mathbf{x}_j; m-1) = y_j$.

Let $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$ be the set of $(m+1)$ data such that $\mathbf{x}_i = \alpha_i \mathbf{x}_1$ and α_i's are distinct. Also let $\alpha_1 < \cdots < \alpha_m$ (after the reordering if necessary) and

$$\frac{y_{i+2} - y_{i+1}}{\alpha_{i+1} - \alpha_i} \neq \frac{y_{i+1} - y_i}{\alpha_{i+1} - \alpha_i} \neq \frac{y_i - y_{i-1}}{\alpha_i - \alpha_{i-1}}, \quad \forall i = 2, \cdots, m-2. \tag{B.1}$$

Suppose there exists a network $\mathcal{N}(\mathbf{x}; m-2)$ of width $(m-2)$ which interpolates all m data. We note that a shallow ReLU network is a piecewise linear function. That is, whenever a slope in a direction needs to be changed, a new neuron has to be added. Since the number of neurons is $(m-2)$, the number of slope changes is at most $(m-2)$. However, in order to interpolate the data set satisfying (B.1), the minimum number of slope changes is $m-1$. To be more precise, the network in the direction of $\mathbf{x}_1$ can be viewed as a one-dimensional network satisfying

$$\mathcal{N}(\mathbf{x}_i; m-2) = \mathcal{N}(\alpha_i \mathbf{x}_1; m-2) = y_i, \forall i.$$

Since $\mathcal{N}(s\mathbf{x}_1; m-2)$ is a network of width $(m-2)$ in one-dimensional input space (i.e., as a function of s) and it interpolates m data satisfying (B.1), there must be at least $m-1$ slope changes in the interval $[\alpha_1, \alpha_m]$. However, since $\mathcal{N}$ has only $(m-2)$ width, this is impossible. Therefore any shallow ReLU network of width less than $(m-2)$ cannot interpolate m data points which satisfy (B.1). □

APPENDIX C. PROOF OF THEOREM 2

Proof. It had been shown in several existing works (Du et al., 2018b; Li and Liang, 2018; Oymak and Soltanolkotabi, 2019) that with probability at least $1-\delta$ over the initialization, an overparameterized shallow ReLU network can interpolate all training data by the (stochastic) gradient-descent method. In other words, overparameterization is a sufficient condition for interpolating all training data with probability at least $1-\delta$.

By Lemma 2, in order to interpolate $(m+1)$ data points, a shallow ReLU network having at least width m is required. However, the probability that an initialized ReLU network of width m has m active neurons is

$$\Pr(\mathfrak{m}_1 = m) = (1 - \hat{p}_{d_{\text{in}}}(r))^m,$$

which decays exponentially in m. It follows from Lemma 3 that $\hat{p}_{d_{\text{in}}}(r) > (\sin \alpha_r)^{d_{\text{in}}}/(\pi d_{\text{in}})$ where $\alpha_r = \tan^{-1}(1/r)$. From the assumption of (4) we have

$$\Pr(\mathfrak{m}_1 = m) = (1 - \hat{p}_{d_{\text{in}}}(r))^m < \left[1 - \frac{(\sin \alpha_r)^{d_{\text{in}}}}{\pi d_{\text{in}}}\right]^m < 1 - \delta.$$

That is, the trainability is less than $1-\delta$. Therefore, overparameterization is required to guarantee, with probability at least $1-\delta$, that at least m neurons are active at the initialization. Therefore, overparameterization is a necessary condition for interpolating all training data. □

APPENDIX D. PROOF OF THEOREM 1

Proof. It follows from Lemma 4, since $\pi_0 = [0, \cdots, 0, 1]$, it suffices to compute the last row of the stochastic matrix P_1. For completeness, we set $(\mathrm{P}_1)_{i,:} = [1, 0, \cdots, 0]$ for $i = 1, \cdots, n_0$.

Suppose the He initialization without bias is used. Since $\mathbf{x} \in B_r(0)$, for any $\boldsymbol{w}$ there exists some $\mathbf{x} \in B_r(0)$ such that $\boldsymbol{w}^T\mathbf{x} > 0$. Therefore no ReLU neuron will be born dead; hence $(\mathrm{P}_1)_{n_0+1,:} = [0, \cdots, 1]$.

Suppose the He initialization with bias is used. Since each hidden neuron is independent, m_1 follows a binomial distribution $B(n_1, p)$. Here p represents the born dead probability of a single ReLU neuron in the first hidden layer. By Lemma 3, $p = \hat{p}_{n_0}(r)$, and this completes the proof. □

Lemma 3. *Suppose all training data inputs are from $B_r(0) = \{x \in \mathbb{R}^d | \|x\| \leq r\}$ and the weights and the biases are independently initialized from a zero mean normal distribution $N(0, \upsigma^2)$. Then the probability that a single ReLU neuron dies at the initialization is*

$$\hat{p}_d = \frac{1}{\sqrt{\pi}} \frac{\Gamma((d+1)/2)}{\Gamma(d/2)} \int_0^{\alpha_r} (\sin u)^{d-1} du < \frac{1}{2}, \tag{D.1}$$

where $\upalpha_r = \tan^{-1}(r^{-1})$. Furthermore,

$$\frac{1}{\pi d}(\sin \upalpha)^d \leq \hat{p}_d \leq \sqrt{\frac{d}{2\pi}} \upalpha (\sin \upalpha)^{d-1}. \tag{D.2}$$

Proof of Lemma 3. Let $\upphi(\boldsymbol{w}\mathbf{x} + b)$ be a single ReLU neuron where $\boldsymbol{w}_i, b \sim N(0, \upsigma^2)$. Note that in order for a single ReLU neuron to die in $B_r(0)$, for all $\mathbf{x} \in B_r(0)$, $\boldsymbol{w}\mathbf{x} + b < 0$. Therefore it suffices to calculate

$$\hat{p}_{d_{\text{in}}} = \Pr[\boldsymbol{w}\mathbf{x} + b < 0, \forall \mathbf{x} \in B_r(0)].$$

Let $\boldsymbol{v} = [\boldsymbol{w}, b] \in \mathbb{R}^{d_{\text{in}}+1}$. Since $\boldsymbol{w}_i$'s and b are iid normal, $\boldsymbol{s} := \boldsymbol{v}/\|\boldsymbol{v}\|$ follows the uniform distribution on the unit hypersphere $\mathbb{S}^{d_{\text{in}}}$, i.e., $\boldsymbol{s} \sim \mathcal{U}(\mathbb{S}^{d_{\text{in}}})$. Also, since $\boldsymbol{s} \overset{d}{=} -\boldsymbol{s}$, we have

$$\hat{p}_{d_{\text{in}}} = \Pr[\langle \boldsymbol{s}, [\mathbf{x}, 1] \rangle < 0, \forall \mathbf{x} \in B_r(0)] = \Pr[\langle \boldsymbol{s}, [\mathbf{x}, 1] \rangle > 0, \forall \mathbf{x} \in B_r(0)].$$

Let

$$\mathcal{A} = \left\{ \boldsymbol{s} \in \mathbb{S}^{d_{\text{in}}} \middle| \langle \boldsymbol{s}, \boldsymbol{v} \rangle > 0, \forall \boldsymbol{v} \in B_r(0) \times \{1\} \right\}. \tag{D.3}$$

Then $\hat{p}_{d_{\text{in}}} = \Pr(\mathcal{A})$. Let $\boldsymbol{s} \in \mathbb{S}^{d_{\text{in}}}$. If $\boldsymbol{s}_{d_{\text{in}}+1} \leq 0$, then $\boldsymbol{s} \notin \mathcal{A}$. This is because there exists $\boldsymbol{v} = (0, \cdots, 0, 1) \in B_r(0) \times \{1\}$ such that $\langle \boldsymbol{s}, \boldsymbol{v} \rangle \leq 0$. Suppose $\boldsymbol{s}_{d_{\text{in}}+1} > 0$ and let $r_{\boldsymbol{s}} = 1/\boldsymbol{s}_{d_{\text{in}}+1}$. Then $\tilde{\boldsymbol{s}} := (1/\boldsymbol{s}_{d_{\text{in}}+1})\boldsymbol{s} \in B_{r_{\boldsymbol{s}}}(0) \times \{1\}$.

We can express any $\mathbf{x} \in B_r(0)$ in the spherical coordinate system, i.e.,

$$\begin{aligned}
\mathbf{x}_1 &= t\cos(\uptheta_1), \\
\mathbf{x}_2 &= t\sin(\uptheta_1)\cos(\uptheta_2), \\
&\vdots \\
\mathbf{x}_{d_{\text{in}}-1} &= t\sin(\uptheta_1)\cdots\sin(\uptheta_{d_{\text{in}}-2})\cos(\uptheta_{d_{\text{in}}-1}), \\
\mathbf{x}_{d_{\text{in}}} &= t\sin(\uptheta_1)\cdots\sin(\uptheta_{d_{\text{in}}-2})\sin(\uptheta_{d_{\text{in}}-1}).
\end{aligned} \tag{D.4}$$

Since $\boldsymbol{s}$ is a uniform random variable from $\mathbb{S}^{d_{\text{in}}}$, it is coordinate-free. Thus let $\tilde{\boldsymbol{s}} = (r_{\boldsymbol{s}}, 0, \cdots, 0, 1)$ for some $0 \leq r_{\boldsymbol{s}}$ and $\boldsymbol{v} = [\mathbf{x}, 1] \in B_r(0) \times \{1\}$. Then

$$\langle \tilde{\boldsymbol{s}}, \boldsymbol{v} \rangle = 1 + r_{\boldsymbol{s}} t\cos(\uptheta_1), \quad 0 \leq t \leq r, \quad 0 \leq \uptheta_1 \leq 2\pi.$$

In order for $\boldsymbol{s} \in \mathcal{A}$, $r_{\boldsymbol{s}} < 1/r$ has to be satisfied; therefore,

$$\mathcal{A} = \left\{ \frac{\tilde{\boldsymbol{s}}}{\|\tilde{\boldsymbol{s}}\|} \in \mathbb{S}^{d_{\text{in}}} \middle| \tilde{\boldsymbol{s}} \in B_{1/r}(0) \times \{1\} \right\}.$$

Let $\text{Surf}(\mathbb{S}^d)$ be the surface area of $\mathbb{S}^d$. It is known that $\text{Surf}(\mathbb{S}^d) = 2\pi^{(d+1)/2}/\{\Gamma[(d+1)/2]\}$, where Γ is the gamma function. Then

$$\begin{aligned}\hat{p}_{d_{\text{in}}} = \Pr(\mathcal{A}) &= \frac{1}{\text{Surf}(\mathbb{S}^{d_{\text{in}}})} \int_{\mathcal{A}} d^{d_{\text{in}}+1}\mathbf{S} = \frac{1}{\text{Surf}(\mathbb{S}^{d_{\text{in}}})} \int_{\theta_{d_{\text{in}}}=0}^{2\pi} \int_{\theta_{d_{\text{in}}-1}=0}^{\pi} \cdots \int_{\theta_2=0}^{\pi} \int_{\theta_1=0}^{\alpha} d^{d_{\text{in}}+1}\mathbf{S} \\ &= \frac{\text{Surf}(\mathbb{S}^{d_{\text{in}}-1})}{\text{Surf}(\mathbb{S}^{d_{\text{in}}})} \int_0^{\alpha} \sin^{d-1}\theta d\theta,\end{aligned}$$

where $\alpha = \tan^{-1}(1/r)$ and $d^{d_{\text{in}}+1}\mathbf{S} = \sin^{d_{\text{in}}-1}\theta_1 \sin^{d_{\text{in}}-2}\theta_2 \cdots \sin\theta_{d_{\text{in}}-1} d\theta_1 d\theta_2 \cdots d\theta_{d_{\text{in}}}$. Note that $g(d) = [\text{Surf}(\mathbb{S}^{d-1})]/[\text{Surf}(\mathbb{S}^d)]$ is bounded above by $\sqrt{d/(2\pi)}$ for all $d \geq 1$ (Leopardi, 2007) and $g(d)$ is monotonically increasing. Thus we have an upper bound of $\hat{p}_d$ as $\hat{p}_d \leq \sqrt{d/(2\pi)}\alpha(\sin\alpha)^{d-1}$. For a lower bound, it can be shown that for any $\theta \in [0, \pi]$, $\int_0^{\theta} (\sin x)^{d-1} dx \geq (1/d)(\sin\theta)^d$. Thus we have

$$\hat{p}_d \geq g(d)\frac{(\sin\alpha)^d}{d} \geq g(1)\frac{(\sin\alpha)^d}{d} = \frac{(\sin\alpha)^d}{\pi d},$$

which completes the proof. □

APPENDIX E. PROBABILITY DISTRIBUTION OF THE NUMBER OF ACTIVE ReLU NEURONS

In order to calculate the trainability, we first present the results for the distribution of the number of active neurons. Understanding how many neurons will be active at the initialization is not only directly related to the trainability of a ReLU network but also suggests how much overspecification or overparameterization shall be needed for training. Given a L-layer ReLU network with $\boldsymbol{n} = (n_0, n_1, \cdots, n_L)$ architecture, let $\mathfrak{m}_t$ be the number of active neurons at the tth hidden layer and π_t be its probability distribution. Then the distribution of $\mathfrak{m}_t$ can be identified as follows.

Lemma 4. *Let $\boldsymbol{V}^t = [\boldsymbol{W}^t, \boldsymbol{b}^t]$ be the parameter (weight and bias) matrix in the tth layer. Suppose $\{\boldsymbol{V}^t\}_{t=1}^L$ is randomly independently initialized and each row of $\boldsymbol{V}^t$ is independent of any other row and follows an identical distribution. Then the probability distribution of the number of active neurons $\mathfrak{m}_j$ at the jth hidden layer can be expressed as*

$$\pi_j = \pi_0 P_1 P_2 \cdots P_j, \quad (\pi_j)_i = Pr(\mathfrak{m}_j = i), \tag{E.1}$$

where $\pi_0 = [0, \cdots, 0, 1]$, and P_t is the stochastic matrix of size $(n_{t-1}+1) \times (n_t+1)$ whose $(i+1, j+1)$ entry is $Pr(\mathfrak{m}_t = j | \mathfrak{m}_{t-1} = i)$. Furthermore, the stochastic matrix P_t is expressed as

$$(P_t)_{(i+1,j+1)} = \binom{n_t}{j} \mathbb{E}_{t-1}\left[(1 - \mathfrak{p}_t(A_{t-1}^i))^j \mathfrak{p}_t(A_{t-1}^i)^{n_t - j}\right], \tag{E.2}$$

where $\mathbb{E}_{t-1}$ is the expectation with respect to $\{\boldsymbol{W}^i, \boldsymbol{b}^i\}_{i=1}^{t-1}$, and $\mathfrak{p}_t(A_{t-1}^i)$ is the conditional born dead probability (BDP) of a neuron in the tth layer given the event where exactly i neurons are active in the $(t-1)$th layer.

Proof of Lemma 4. By the law of total probability, it readily follows that for $j = 0, \cdots, n_t$,

$$\Pr(\mathfrak{m}_t = j) = \sum_{k=0}^{n_{t-1}} \Pr(\mathfrak{m}_t = j | \mathfrak{m}_{t-1} = k) \Pr(\mathfrak{m}_{t-1} = k),$$

which gives $\pi_t = \pi_{t-1} \mathrm{P}_t$. By recursively applying it, we obtain $\pi_t = \pi_0 \mathrm{P}_1 \cdots \mathrm{P}_t$.

For each t and i, let A_{t-1}^i be the event where exactly i neurons are active in the $(t-1)$th layer and D_t^k be the event where the kth neuron in the tth layer is dead. Since each row of $\boldsymbol{V}^t$ is iid and $\{\boldsymbol{V}^t\}_{t=1}^L$ is independent, $\Pr(D_t^k | A_{t-1}^i) = \Pr(D_t^j | A_{t-1}^i)$ for any k, j. We denote the conditional BDP of a neuron in the tth layer given A_{t-1}^i as $\mathfrak{p}_t(A_{t-1}^i)$. From the independent row assumption, the stochastic matrix P_t can then be expressed as

$$\Pr(\mathfrak{m}_t = j | \mathfrak{m}_{t-1} = i) = (\mathrm{P}_t)_{(i+1,j+1)} = \binom{n_t}{j} \mathbb{E}_{t-1} \left\{ [1 - \mathfrak{p}_t(A_{t-1}^i)]^j \mathfrak{p}_t(A_{t-1}^i)^{n_t - j} \right\},$$

where $\mathbb{E}_{t-1}$ is the expectation with respect to $\{\boldsymbol{W}^i, \boldsymbol{b}^i\}_{i=1}^{t-1}$. □

Lemma E.2 indicates that $\mathfrak{p}_t(A_{t-1}^i)$ is a fundamental quantity for the complete understanding of π_j. As a first step toward understanding π_j, we calculate the exact probability distribution π_1 of the number of active neurons in the first hidden layer.

Lemma 5. *Given a ReLU network having $\boldsymbol{n} = (n_0, n_1, \cdots, n_L)$ architecture, suppose the training input domain is $B_r(0)$. If either the normal (2) or the unit hypersphere (3) initialization without bias is used in the first hidden layer, we have*

$$(\pi_1)_j = \Pr(\mathfrak{m}_1 = j) = \delta_{j,n_1}.$$

If either the normal (2) or the unit hypersphere (3) with bias is used in the first hidden layer, $\mathfrak{m}_1$ follows a binomial distribution with parameters n_1 and $1 - \hat{p}_{n_0}(r)$, where

$$\hat{p}_d(r) = \frac{1}{\sqrt{\pi}} \frac{\Gamma((d+1)/2)}{\Gamma(d/2)} \int_0^{\alpha_r} (\sin\theta)^{d-1} d\theta, \quad \alpha_r = \tan^{-1}(r^{-1}), \tag{E.3}$$

and $\Gamma(x)$ is the Gamma function.

Proof. The proof readily follows from Lemma 3. □

We now calculate π_2 for a ReLU network at $d_{\text{in}} = 1$. Since the bias in each layer can be initialized in different ways, we consider some combinations of them.

Lemma 6. *Given a ReLU network having $\boldsymbol{n} = (1, n_1, n_2, \cdots, n_L)$ architecture, suppose the training input domain is $B_r(0)$.*

- *Suppose the unit hypersphere (3) initialization without bias is used in the first hidden layer.*
  1. *If the normal (2) initialization without bias is used in the second hidden layer, the stochastic matrix P_2 is $(P_2)_{i,:} = [1, 0, \cdots, 0]$ for $1 \le i \le n_1$ and*

$$(P_2)_{n_1+1,j+1} = \binom{n_2}{j} \left[\left(1 - \frac{1}{2^{n_1-1}}\right) \frac{3^j}{4^{n_2}} + \frac{1}{2^{n_1+n_2-1}} \right], \quad 0 \le j \le n_2.$$

2. *If the normal (2) initialization with bias is used in the second hidden layers, the stochastic matrix P_2 is $(P_2)_{i,:} = [1, 0, \cdots, 0]$ for $1 \le i \le n_1$ and*

$$(P_2)_{n_1+1,j+1} = \binom{n_2}{j} \mathbb{E}_s \left[(1 - \mathfrak{p}_2(s))^j \mathfrak{p}_2(s)^{n_2-j} \right], \quad 0 \le j \le n_2,$$

where $s \sim B(n_1, 1/2)$, $\alpha_s = \tan^{-1}(\frac{s}{n_1 - s})$, $g(x) = \sin(\tan^{-1}(x))$, and

$$\mathfrak{p}_2(s) = \frac{1}{2} + \left[\int_{\pi/2}^{\pi+\alpha_s} \frac{g(r\sqrt{s}\cos(\theta))}{4\pi} d\theta + \int_{\pi+\alpha_s}^{2\pi} \frac{g(r\sqrt{n_1 - s}\sin(\theta))}{4\pi} d\theta \right].$$

- *Suppose the unit hypersphere (3) initialization with bias is used in the first hidden layer.*
  1. *If the normal (2) initialization without bias is used in the second hidden layer and $n_1 = 1$, the stochastic matrix P_2 is $(P_2)_{1,:} = [1, 0, \cdots, 0]$ and*

$$(P_2)_{2,:} = \textit{Binomial}(n_2, 1/2).$$

  2. *If the normal (2) initialization with bias is used in the second hidden layers and $n_1 = 1$, the stochastic matrix P_2 is $(P_2)_{1,:} = [1, 0, \cdots, 0]$ and*

$$(P_2)_{2,j+1} = \binom{n_2}{j} \mathbb{E}_\omega \left[(1 - \mathfrak{p}_2(\omega))^j \mathfrak{p}_2(\omega)^{n_2-j} \right], \quad 0 \le j \le n_2,$$

where $\alpha_r = \tan^{-1}(r)$, $\omega \sim \mathrm{Unif}\,(0, \pi/2 + \alpha_r)$, $g(x) = \tan^{-1}(1/\sqrt{r^2+1}\cos(x))$, and

$$\mathfrak{p}_2(\omega) = \begin{cases} \dfrac{1}{4} + \dfrac{g(\omega - \alpha_r)}{2\pi}, \\ \quad \textit{if} \quad \omega \in \left[\dfrac{\pi}{2} - \alpha_r, \dfrac{\pi}{2} + \alpha_r \right), \\ \dfrac{1}{4} + \dfrac{g(\omega - \alpha_r) + \tan^{-1}(\sqrt{r^2+1}\cos(\omega + \alpha_r))}{2\pi}, \\ \quad \textit{if} \quad \omega \in \left[0, \dfrac{\pi}{2} - \alpha_r \right). \end{cases}$$

Then $\pi_2 = \pi_1 P_2$, where π_1 is defined in Lemma 5.

Proof of Lemma 6. Since π_1 is completely characterized in Lemma 5, it suffices to calculate the stochastic matrix P_2, as $\pi_2 = \pi_1 \mathrm{P}_2$. From Eq. (E.2), it suffices to calculate the BDP $\mathfrak{p}_2(A_1^i)$ of a ReLU neuron at the second layer given A_1^i.

We note that if $\boldsymbol{z} = [x, 1]$ where $x \in B_r(0) = [-r, r]$ and $\boldsymbol{v} = [w, b] \sim \mathcal{U}(\mathbb{S}^1)$, then

$$\begin{aligned} P(\phi(\boldsymbol{v}^T \boldsymbol{z}) = \boldsymbol{v}^T \boldsymbol{z}, \forall \boldsymbol{z} \in B_r(0) \times \{1\}) &= \hat{p}_1(r) = \frac{\tan^{-1}(1/r)}{\pi}, \\ P(\phi(\boldsymbol{v}^T \boldsymbol{z}) = \boldsymbol{v}^T \boldsymbol{z} \mathbb{I}_{x \in [a,b]}, \forall \boldsymbol{z} \in B_r(0) \times \{1\}) &= \frac{1}{4} + \frac{\tan^{-1}(1/b) + \tan^{-1}(a)}{2\pi}. \end{aligned} \tag{E.4}$$

First, let us consider the case where the unit hypersphere initialization without bias is used for the first hidden layer. Note that since $x \in [-r, r]$, i.e., $d_{\text{in}} = 1$, we have $A_1^j = \emptyset$ for $0 \le j < n_1$

and $A_1^{n_1} = \{1, -1\}^{n_1}$. Also, note that if w_j's are iid normal, $\sum_{j=1}^{s} w_j \overset{d}{=} \sqrt{s}w$, where $w \overset{d}{=} w_1$. For fixed $\omega \in A_1^{n_1}$, a single neuron in the second layer is

$$\phi\left[\sum_{j=1}^{s} \boldsymbol{w}_{1,j}^2 \phi(x) + \sum_{j=s+1}^{n_1} \boldsymbol{w}_{1,j}^2 \phi(-x) + b\right] \overset{d}{=} \phi\Big[\sqrt{s}w_1\phi(x) + \sqrt{n_1 - s}w_2\phi(-x) + b\Big], \tag{E.5}$$

where s is the number of 1's in ω. If the normal initialization without bias is used for the second hidden layer, we have

$$\mathfrak{p}_2(\omega) = \begin{cases} \dfrac{1}{2}, & \text{if} \quad \omega = \pm[1, \cdots, 1]^T, \\ \dfrac{1}{4}, & \text{otherwise.} \end{cases}$$

Also, $s \sim \text{Binomial}(n_1, 1/2)$. Thus, for $j = 0, \cdots, n_2$,

$$\begin{aligned} \Pr(\mathrm{m}_2 = j | \mathrm{m}_1 = n_1) &= \binom{n_2}{j} \mathbb{E}_1\left[(1 - \mathfrak{p}_2(A_1^{n_1}))^j (\mathfrak{p}_2(A_1^{n_1}))^{n_2 - j}\right] \\ &= \binom{n_2}{j} \mathbb{E}_s\left[(1 - \mathfrak{p}_2(A_1^{n_1}))^j (\mathfrak{p}_2(A_1^{n_1}))^{n_2 - j}\right] \\ &= \binom{n_2}{j}\left[\frac{1}{2^{n_1-1}}\frac{1}{2^{n_2}} + \left(1 - \frac{1}{2^{n_1-1}}\right)\frac{3^j}{4^{n_2}}\right]. \end{aligned}$$

Suppose the normal initialization with bias is used for the second hidden layer. It follows from (E.5) that

$$\mathfrak{p}_2(\omega) = \Pr(w_1\sqrt{s}\phi(x) + w_2\sqrt{n_1 - s}\phi(-x) + b < 0, \forall x \in [-r, r] | \boldsymbol{W}^1 \text{ has } s \text{ 1's}).$$

Let $\boldsymbol{z} = [\sqrt{s}\phi(x), \sqrt{n_1 - s}\phi(-x), 1]$ and $\boldsymbol{v} = (w_1, w_2, b)$. Without loss of generality, we normalize $\boldsymbol{v}$. Then $\boldsymbol{v} \sim \mathbb{S}^2$, and we write it as

$$\boldsymbol{v} = (\cos\theta \sin\alpha, \sin\theta \sin\alpha, \cos\alpha),$$

where $\theta \in [0, 2\pi]$ and $\alpha \in [0, \pi]$. Since $\boldsymbol{v} \overset{d}{=} -\boldsymbol{v}$, it suffices to compute

$$\Pr(\boldsymbol{v}^T\boldsymbol{z} > 0, \forall \boldsymbol{z} | \boldsymbol{W}^1 \text{ has } s \text{ 1's}).$$

Also, note that

$$\begin{aligned} \boldsymbol{v}^T\boldsymbol{z} &= \begin{cases} \sqrt{s}\phi(x)\cos\theta\sin\alpha + \cos\alpha, & \text{if} \quad x > 0, \\ \sqrt{n_1 - s}\phi(-x)\sin\theta\sin\alpha + \cos\alpha, & \text{if} \quad x < 0, \end{cases} \\ &= \begin{cases} \sqrt{1 + sx^2\cos^2\theta}\cos(\alpha - \beta), & \text{if} \quad x > 0, \\ \sqrt{1 + (n_1 - s)x^2\sin^2\theta}\cos(\alpha - \beta), & \text{if} \quad x < 0, \end{cases} \end{aligned}$$

where $\tan\beta = \sqrt{s}x\cos\theta$ if $x > 0$ and $\tan\beta = \sqrt{n_1 - s}\sin\theta$ if $x < 0$. Given $\boldsymbol{W}^1$ which has s 1's, the regime in $\mathbb{S}^2$, where $\boldsymbol{v}^T\boldsymbol{z} > 0$ for all $\boldsymbol{z}$, is

$$\begin{aligned}
&\text{for } \omega \in \left[0, \frac{\pi}{2}\right], \alpha \in \left[0, \frac{\pi}{2}\right], \\
&\text{for } \omega \in \left[\frac{\pi}{2}, \pi + \omega^*\right], \alpha \in \left[0, \tan^{-1}(\sqrt{s}r\cos\theta) + \frac{\pi}{2}\right], \\
&\text{for } \omega \in [\pi + \omega^*, 2\pi], \alpha \in \left[0, \tan^{-1}(\sqrt{n_1 - s}r\sin\theta) + \frac{\pi}{2}\right],
\end{aligned} \tag{E.6}$$

where $\tan\omega^* = s/(n_1 - s)$. By uniformly integrating the above domain in $\mathbb{S}^2$, we have

$$\mathfrak{p}_2(s) = \frac{1}{2} + \left[\int_{\pi/2}^{\pi+\alpha_s} \frac{g(r\sqrt{s}\cos(\theta))}{4\pi}d\theta + \int_{\pi+\alpha_s}^{2\pi} \frac{g(r\sqrt{n_1 - s}\sin(\theta))}{4\pi}d\theta\right],$$

where $g(x) = \sin[\tan^{-1}(x)]$. Thus we obtain

$$(\mathrm{P}_2)_{n_1+1,j+1} = \binom{n_2}{j}\mathbb{E}_s\left[(1 - \mathfrak{p}_2(s))^j\mathfrak{p}_2(s)^{n_2-j}\right], \quad 0 \le j \le n_2.$$

Secondly, let us consider the case where the unit hypersphere initialization with bias is used for the first hidden layer and $n_1 = 1$. Since $[w_1, b_1] \sim \mathbb{S}^1$, we write it as $(\sin\omega, \cos\omega)$ for $\omega \in [-\pi, \pi]$. Since $x \in [-r, r]$, we have

$$\begin{aligned}
&A_1^0 = \{\omega \in [-\pi, \pi] | \phi(\sin\omega x + \cos\omega) = 0, \forall x \in [-r, r]\} = [-\pi + \alpha_r, \pi - \alpha_r], \\
&A_1^1 = (A_1^0)^c = (-\pi + \alpha_r, \pi - \alpha_r),
\end{aligned} \tag{E.7}$$

where $\alpha_r = \tan^{-1}(r)$. If the normal initialization without bias is used for the second hidden layer, since a single neuron in the second layer is $\phi[w^2\phi(w^1x + b^1)]$, for given A_1^1, we have $\mathfrak{p}_2(A_1^1) = 1/2$. Thus

$$\Pr(\mathrm{m}_2 = j|\mathrm{m}_1 = 1) = \binom{n_2}{j}(1/2)^j(1/2)^{n_2-j}, \quad j = 0, \cdots, n_2.$$

If the normal initialization with bias is used for the second hidden layer, it follows from Lemma 7 that for $\omega \in A_1^1$,

$$\mathfrak{p}_2(\omega) = \begin{cases} \dfrac{1}{4} + \dfrac{g(|\omega| - \alpha_r)}{2\pi}, \\ \quad \text{if} \quad |\omega| \in \left[\dfrac{\pi}{2} - \alpha_r, \dfrac{\pi}{2} + \alpha_r\right), \\ \dfrac{1}{4} + \dfrac{g(|\omega| - \alpha_r) + \tan^{-1}\left(\sqrt{r^2+1}\cos(|\omega| + \alpha_r)\right)}{2\pi}, \\ \quad \text{if} \quad |\omega| \in \left[0, \dfrac{\pi}{2} - \alpha_r\right), \end{cases}$$

where $g(x) = \tan^{-1}[1/(\sqrt{r^2+1}\cos(x))]$.

Thus we have

$$\Pr(\mathrm{m}_2 = j|\mathrm{m}_1 = 1) = \binom{n_2}{j}\mathbb{E}_\omega\left[(1 - \mathfrak{p}_2(\omega))^j(\mathfrak{p}_2(\omega))^{n_2-j}\right], \quad j = 0, \cdots, n_2,$$

where $\omega \sim \mathrm{Unif}(A_1^1)$. Then the proof is completed once we have the following lemma.

Lemma 7. *Given a ReLU network having $\boldsymbol{n} = (1, 1, n_2, \cdots, n_L)$, suppose $(w^1, b^1), (w^2, b^2) \sim \mathbb{S}^1$. Given $\{w^1, b^1\}$, let ω be the angle of (w^1, b^1) in $\mathbb{R}^2$. Then the BDP for a ReLU neuron at the second hidden layer is*

$$\mathfrak{p}_2(\omega) = \begin{cases} 1, \quad \textit{if} \quad |\omega| \in [\pi/2 + \alpha_r, \pi], \\ \dfrac{1}{4} + \dfrac{g(|\omega| - \alpha_r)}{2\pi}, \quad \textit{if} \quad |\omega| \in [\pi/2 - \alpha_r, \pi/2 + \alpha_r), \\ \dfrac{1}{4} + \dfrac{g(|\omega| - \alpha_r) + \tan^{-1}(\sqrt{r^2+1}\cos(|\omega| + \alpha_r))}{2\pi}, \\ \quad \textit{if} \quad |\omega| \in [0, \pi/2 - \alpha_r)\,, \end{cases}$$

where $g(x) = \tan^{-1}\{1/[\sqrt{r^2+1}\cos(x)]\}$ and $\alpha_r = \tan^{-1}(r)$.

Proof of Lemma 7. For a fixed $\boldsymbol{v} = [w, b]$ and $\boldsymbol{z} = [x, 1]$, we can write

$$\phi(\boldsymbol{v}^T\boldsymbol{z}) = \|\boldsymbol{z}\|\phi(\boldsymbol{v}^T\boldsymbol{z}/\|\boldsymbol{z}\|) = \|z\|\phi(\cos(\omega - \theta(x))), \quad \theta(x) = \tan^{-1}(x).$$

Since $\boldsymbol{v}$ is uniformly drawn from $\mathbb{S}^1$, it is equivalent to draw $\omega \sim \mathcal{U}(-\pi, \pi)$. Let $0 < \theta_{\max} = \tan^{-1}(r) < \pi/2$. Then

$$\phi(\boldsymbol{v}^T\boldsymbol{z}) = \begin{cases} \boldsymbol{v}^T\boldsymbol{z}, & \forall\theta(x), \text{ if } \omega \in \left(-\dfrac{\pi}{2} + \theta_{\max}, \dfrac{\pi}{2} - \theta_{\max}\right), \\ 0, & \forall\theta(x), \text{ if } \omega \in \left[-\pi, \dfrac{\pi}{2} - \theta_{\max}\right] \cup \left[\dfrac{\pi}{2} + \theta_{\max}, \pi\right], \end{cases}$$

and if

$$\omega \in \left(-\frac{\pi}{2} - \theta_{\max}, -\frac{\pi}{2} + \theta_{\max}\right] \cup \left[\frac{\pi}{2} - \theta_{\max}, \frac{\pi}{2} + \theta_{\max}\right),$$

we have $\phi(\boldsymbol{v}^T\boldsymbol{z}) = \boldsymbol{v}^T\boldsymbol{z}\mathbb{I}_{A(\omega)}(\theta(x))$, where $A(\omega) = \{\theta \in [-\theta_{\max}, \theta_{\max}] || \theta(x) - \omega| \leq \pi/2\}$. Due to symmetry, let us assume that $\omega \sim \mathcal{U}(0, \pi)$. Then it can be checked that $A_1^0 = [\pi/2 + \theta_{\max}, \pi]$ and $A_1^1 = [0, \pi/2 + \theta_{\max})$. Furthermore,

$$\max_{\boldsymbol{z}} \phi(\boldsymbol{v}^T\boldsymbol{z}) = \begin{cases} \sqrt{r^2+1}\cos(\omega - \theta_{\max}), & \text{if} \quad \omega \in \left(0, \dfrac{\pi}{2} + \theta_{\max}\right), \\ 0, & \text{if} \quad \omega \in \left[\dfrac{\pi}{2} + \theta_{\max}, \pi\right], \end{cases}$$

and

$$\min_{\boldsymbol{z}} \phi(\boldsymbol{v}^T\boldsymbol{z}) = \begin{cases} \sqrt{r^2+1}\cos(\omega + \theta_{\max}), & \text{if} \quad \omega \in \left(0, \dfrac{\pi}{2} - \theta_{\max}\right), \\ 0, & \text{if} \quad \omega \in \left[\dfrac{\pi}{2} - \theta_{\max}, \pi\right]. \end{cases}$$

For a fixed ω, let $\mathfrak{p}_2(\omega)$ be the probability that a single neuron at the second layer is born dead, i.e.,

$$\mathfrak{p}_2(\omega) = \Pr[w^2\phi(w^1x + b^1) + b^2 < 0, \quad \forall x \in B_r(0)|w^1, b^1].$$

Also, since $(w^2, b^2) \overset{d}{=} (-w^2, -b^2)$, we have

$$\mathfrak{p}_2(\omega) = \Pr[w^2\phi(w^1x + b^1) + b^2 > 0, \quad \forall x \in B_r(0)|w^1, b^1].$$

It follows from (E.4) that

$$\mathfrak{p}_2(\omega) = \frac{1}{4} + \frac{\tan^{-1}[1/\max_{\boldsymbol{z}} \phi(\boldsymbol{v}^T\boldsymbol{z})] + \tan^{-1}[\min_{\boldsymbol{z}} \phi(\boldsymbol{v}^T\boldsymbol{z})]}{2\pi}.$$

Thus we obtain

$$\mathfrak{p}_2(\omega) = \begin{cases} 1, \quad \text{if} \quad \omega \in \left[\frac{\pi}{2} + \theta_{\max}, \pi\right], \\ \frac{1}{4} + \frac{g(\omega - \theta_{\max})}{2\pi}, \quad \text{if} \quad \omega \in \left[\frac{\pi}{2} - \theta_{\max}, \frac{\pi}{2} + \theta_{\max}\right), \\ \frac{1}{4} + \frac{g(\omega - \theta_{\max}) + \tan^{-1}\left(\sqrt{r^2+1}\cos(\omega + \theta_{\max})\right)}{2\pi}, \\ \quad \text{if} \quad \omega \in \left[0, \frac{\pi}{2} - \theta_{\max}\right), \end{cases}$$

where $g(x) = \tan^{-1}[1/(\sqrt{r^2+1}\cos(x))]$, and this completes the proof. □

□

Lemmas 5 and 6 indicate that the bias initialization could drastically change the active neuron distributions π_j. Since $\pi_j = \pi_1 \mathrm{P}_2 \cdots \mathrm{P}_j = \pi_2 \mathrm{P}_3 \cdots \mathrm{P}_j$, the behaviors of π_1 and π_2 affect the higher layer's distributions π_j. In Fig. E1, we consider a ReLU network with $\boldsymbol{n} = (1, 6, 4, 2, n_4, \cdots, n_L)$ architecture and plot the empirical distributions π_j, $j = 1, 2, 3$, from 10^6 independent simulations at $r = 1$. On the left and the middle, the unit hypersphere (3) initializations without and with bias are employed, respectively, in all layers.

On the right, the unit hypersphere initialization without bias is employed in the first hidden layer, and the normal (2) initialization with bias is employed in all other layers. The theoretically derived distributions, π_1, π_2, are also plotted as references. We see that all empirical results are well matched with our theoretical derivations. When the first hidden layer is initialized with bias, with probability 0.8, at least one neuron in the first hidden layer will be dead. On the other hand, if the first hidden layer is initialized without bias, with probability 1, no neuron will be dead. It is clear that the distributions obtained by three initialization schemes show different behavior.

APPENDIX F. A GENERAL FORMULATION FOR COMPUTING TRAINABILITY

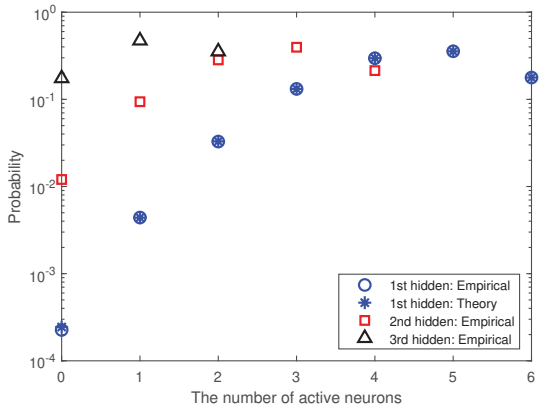
We present a general formulation for computing trainability. Our formulation requires a complete understanding of two types of inhomogeneous stochastic matrices.

Let $\mathfrak{d}_t^b$ be the number of permanently dead neurons at the tth hidden layer. Given $\{n_t, m_t\}_{t=1}^{L-1}$, let $s_t = (n_t - m_t + 1)(m_t - 1)$ for $t > 1$ and $s_1 = n_1 - m_1 + 1$. For convenience, let $\mathcal{T}_{t-1} := [\hat{n}_1] \times [\hat{n}_2] \times [\hat{m}_2] \cdots \times [\hat{n}_{t-1}] \times [\hat{m}_{t-1}]$, where $[\hat{n}_t] = \{0, \cdots, n_t - m_t\}$ and $[\hat{m}_t] = \{1, \cdots, m_t - 1\}$. Let $\hat{T}_t := [\hat{n}_t] \times [\hat{m}_t]$. Let

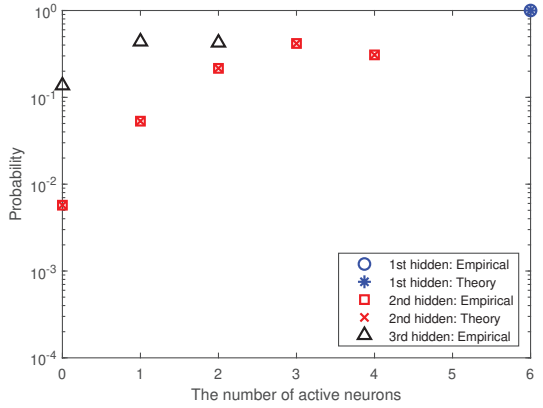
$$\mathrm{k}_{t-1}^l = (k_1^l, k_2^l, k_{2,b}^l, \cdots, k_{t-1}^l, k_{t-1,b}^l)$$

be the lth multi-index of $\mathcal{T}_{t-1}$ (assuming a certain ordering). For $1 < t < L-1$, let $\hat{P}_t$ be a matrix of size $\prod_{j=1}^{t-1} s_j \times \prod_{j=1}^{t} s_j$ defined as follows. For $l = 1, \ldots, \prod_{j=1}^{t-1} s_j$ and $r = 1, \ldots, \prod_{j=1}^{t} s_j$,

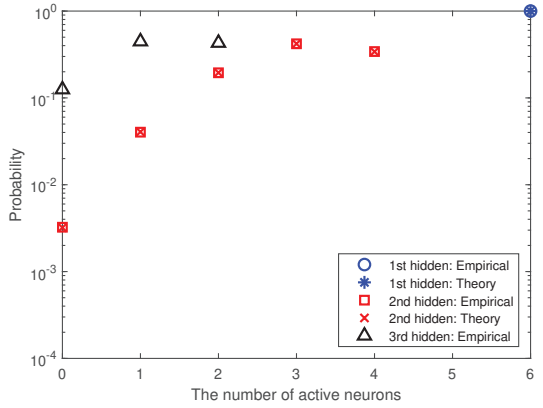
$$[\hat{P}_t]_{l,r} = \Pr(\mathfrak{m}_t = k_t^r, \mathfrak{d}_t^b = k_{t,b}^r | \mathfrak{m}_s = k_s^l, \mathfrak{d}_s^b = k_{s,b}^l, \forall 1 \le s < t) \prod_{j=1}^{t-1} \delta_{k_j^l = k_j^r} \delta_{k_{j,b}^l = k_{j,b}^r}. \quad \text{(F.1)}$$



(a)



(b)



(c)

FIG. E1: The probability distributions of the number of active neurons at different layers are shown for a ReLU network having $\boldsymbol{n} = (1, 6, 4, 2, n_4, \cdots, n_L)$ architecture. (a) All layers are initialized by the unit hypersphere with bias. (b) All layers are initialized by the unit hypersphere without bias. (c) The first hidden layer is initialized by the unit hypersphere without bias. All other layers are initialized by the normal with bias.

For $t = L-1$, let $\hat{P}_{L-1}$ be a matrix of size $\prod_{j=1}^{L-2} s_j \times s_{L-1}$ such that for $l = 1, \ldots, \prod_{j=1}^{L-2} s_j$ and $r = 1, \ldots, s_{L-1}$,

$$[\hat{P}_{L-1}]_{l,r} = \Pr(\mathfrak{m}_{L-1} = \bar{k}^r_{L-1}, \mathfrak{d}^b_{L-1} = \bar{k}^r_{L-1,b} | \mathfrak{m}_s = k^l_s, \mathfrak{d}^b_s = k^l_{s,b}, \forall 1 \le s < L-2), \quad \text{(F.2)}$$

where $\bar{\mathrm{k}}^r_{L-1} = (\bar{k}^r_{L-1}, \bar{k}^r_{L-1,b})$ is the rth multi-index of the lexicographic ordering of $[\hat{n}_{L-1}] \times [\hat{m}_{L-1}]$.

Once the above stochastic matrices are all identified, its corresponding trainability readily follows based on the formulation given below.

Lemma 8. *For a learning task that requires a L-layer ReLU network having at least m_t active neurons in the tth layer, the trainability for a L-layer ReLU network with $\boldsymbol{n} = (n_0, n_1, \cdots, n_L)$ architecture is given as follow. Let $\tilde{n}_t = n_t - m_t + 1$. Then the trainability is given by*

$$\textit{Trainability } = \pi'_1 P'_2 \cdots P'_{L-1} 1_{\tilde{n}_{L-1}} + \pi'_1 \hat{P}_2 \cdots \hat{P}_{L-1} 1_{s_{L-1}},$$

where π'_1 is a $1 \times \tilde{n}_1$ submatrix of π_1 whose first component is $[\pi_1]_{m_t}$, P'_t is a $\tilde{n}_{t-1} \times \tilde{n}_t$ submatrix of P_t whose $(1,1)$ component is $[P_t]_{m_{t-1}, m_t}$, and 1_p is a $p \times 1$ vector whose entries are all 1's. Here π_1 and P_t are defined in Lemma 4, and $\{\hat{P}_t\}$ is defined in (F.1) and (F.2).

Proof of Lemma 8. We observe that

$$\Pr(\mathfrak{m}_t \ge 1, \mathfrak{d}^b_t \le n_t - m_t, \forall 1 \le t < L) = \Pr(\mathfrak{m}_t \ge m_t, \forall 1 \le t < L)$$
$$+ \Pr(\mathfrak{m}_1 \ge m_1, 1 \le \mathfrak{m}_t < m_t, \mathfrak{d}^b_t \le n_t - m_t, \forall 1 \le t < L).$$

From Lemma 4, it can be checked that

$$\Pr(\mathfrak{m}_t \ge m_t, \forall 1 \le t < L) = \pi'_1 \mathrm{P}'_2 \cdots \mathrm{P}'_{L-1} 1_{\tilde{n}_{L-1}}.$$

For convenience, let $\hat{\mathfrak{m}}_t = (\mathfrak{m}_t, \mathfrak{d}^b_t)$ for $t > 1$ and $\hat{\mathfrak{m}}_1 = \mathfrak{m}_t$. Let $\vec{\mathfrak{m}}_t = (\hat{\mathfrak{m}}_1, \hat{\mathfrak{m}}_2, \cdots, \hat{\mathfrak{m}}_t)$. Also, recall that $\mathcal{T}_{t-1} := [\hat{n}_1] \times [\hat{n}_2] \times [\hat{m}_2] \cdots \times [\hat{n}_{t-1}] \times [\hat{m}_{t-1}]$, where $[\hat{n}_t] = \{0, \cdots, n_t - m_t\}$ and $[\hat{m}_t] = \{1, \cdots, m_t - 1\}$. Let $\hat{T}_t := [\hat{n}_t] \times [\hat{m}_t]$. Also let $\vec{\pi}_t = [\Pr(\vec{\mathfrak{m}}_t = \mathrm{k})]_{\mathrm{k} \in \mathcal{T}_t}$ be the distribution of $\vec{\mathfrak{m}}_t$ restricted to $\mathcal{T}_t$. Then,

$$\begin{aligned}
&\Pr(\mathfrak{m}_1 \ge m_1, 1 \le \mathfrak{m}_t < m_t, \mathfrak{d}^b_t \le n_t - m_t, \forall 1 \le t < L) \\
&= \Pr(\vec{\mathfrak{m}}_{L-1} \in \mathcal{T}_{L-1}) = \sum_{\mathrm{k}_{L-1} \in \mathcal{T}_{L-1}} \Pr(\vec{\mathfrak{m}}_{L-1} = \mathrm{k}_{L-1}) \\
&= \sum_{\hat{\mathrm{k}}_{L-1} \in \hat{T}_{L-1}} \sum_{\mathrm{k}_{L-2} \in \mathcal{T}_{L-2}} \Pr(\hat{\mathfrak{m}}_{L-1} = \hat{\mathrm{k}}_{L-1} | \vec{\mathfrak{m}}_{L-2} = \mathrm{k}_{L-2}) \Pr(\vec{\mathfrak{m}}_{L-2} = \mathrm{k}_{L-2}) \\
&= \vec{\pi}_{L-2} \hat{P}_{L-1} 1_{s_{L-1}}.
\end{aligned}$$

It then suffices to identify $\vec{\pi}_t$ for $1 \le t < L-1$. Then note that for each $\mathrm{k}_t = (\mathrm{k}_{t-1}, \hat{\mathrm{k}}_t) \in \mathcal{T}_t$,

$$\Pr(\vec{\mathfrak{m}}_t = \mathrm{k}_t) = \Pr(\hat{\mathfrak{m}}_t = \hat{\mathrm{k}}_t | \vec{\mathfrak{m}}_{t-1} = \mathrm{k}_{t-1}) \Pr(\vec{\mathfrak{m}}_{t-1} = \mathrm{k}_{t-1}).$$

Thus we have $\vec{\pi}_t = \vec{\pi}_{t-1} \hat{P}_t$. Since $\vec{\pi}_1 = \pi'_1$, by recursively applying it, the proof is completed. □

We are now in a position to present our proof of Theorem 3.

Proof of Theorem 3. Given the event A_{t-1}^i that exactly i neurons are active in the $(t-1)$th hidden layer, let $\mathfrak{p}_{t,b}(A_{t-1}^i)$ and $\mathfrak{p}_{t,g}(A_{t-1}^i)$ be the conditional probabilities that a neuron in the tth hidden layer is born dead permanently and born dead tentatively, respectively. Then

$$\mathfrak{p}_t(A_{t-1}^i) = \mathfrak{p}_{t,b}(A_{t-1}^i) + \mathfrak{p}_{t,g}(A_{t-1}^i).$$

Note that since the weights and the biases are initialized from a symmetric probability distribution around 0, we have $\mathfrak{p}_{t,b}(A_{t-1}^i) \geq 2^{-i-1}$. This happens when all the weights and bias are initialized to be nonpositive. Let $\mathfrak{d}_t^g$ and $\mathfrak{d}_t^b$ be the number of tentatively dead and permanently dead neurons at the tth hidden layer. It then can be checked that

$$\begin{aligned}
&\Pr(\mathfrak{d}_t^g = j_1, \mathfrak{d}_t^b = j_2 | \mathfrak{m}_1 = i) \\
&\quad = \binom{n_t}{j_1, j_2, j_3} \mathbb{E}_{t-1} \left\{ [1 - \mathfrak{p}_t(A_{t-1}^i)]^{n_t - j_1 - j_2} [\mathfrak{p}_{t,g}(A_{t-1}^i)]^{j_1} [\mathfrak{p}_{t,b}(A_{t-1}^i)]^{j_2} \right\},
\end{aligned}$$

where $j_3 = n_t - j_1 - j_2$, $\mathbb{E}_{t-1}$ is the expectation with respect to $\mathcal{F}_{t-1}$, and

$$\binom{n}{k_1, k_2, k_3}$$

is a multinomial coefficient. Also note that $\mathfrak{m}_t + \mathfrak{d}_t^g + \mathfrak{d}_t^b = n_t$. It then follows from Lemma 8 that

$$\begin{aligned}
&\Pr(\mathfrak{m}_t \geq 1, \mathfrak{d}_t^b \leq n_t - m_t, \forall 1 \leq t < 3) \\
&\quad = \Pr(\mathfrak{m}_1 \geq m_1, \mathfrak{m}_2 \geq m_2) + \sum_{j=1}^{m_2-1} \sum_{l=0}^{n_2-m_2} \sum_{k=m_1}^{n_1} \Pr(\mathfrak{m}_2 = j, \mathfrak{d}_2^b = l | \mathfrak{m}_1 = k) \\
&\quad \times \Pr(\mathfrak{m}_1 = k) = \Pr(\mathfrak{m}_1 \geq m_1, \mathfrak{m}_2 \geq m_2) + \sum_{j=1}^{m_2-1} \sum_{l=0}^{n_2-m_2} \sum_{k=m_1}^{n_1} \Pr \\
&\quad \times (\mathfrak{d}_2^g = n_2 - j - l, \mathfrak{d}_2^b = l | \mathfrak{m}_1 = k) \Pr(\mathfrak{m}_1 = k) = \Pr(\mathfrak{m}_1 \geq m_1, \mathfrak{m}_2 \geq m_2) \\
&\quad + \sum_{j=1}^{m_2-1} \sum_{l=0}^{n_2-m_2} \sum_{k=m_1}^{n_1} \binom{n_2}{n_2 - j - l, j, l} \mathbb{E}_1 \left[(1 - \mathfrak{p}_2(A_1^k))^j (\mathfrak{p}_{2,g}(A_1^k))^{n_2 - j - l} (\mathfrak{p}_{2,b}(A_1^k))^l \right] \\
&\quad \times \Pr(\mathfrak{m}_1 = k) \geq \Pr(\mathfrak{m}_1 \geq m_1, \mathfrak{m}_2 \geq m_2) + \sum_{j=1}^{m_2-1} \sum_{l=0}^{n_2-m_2} \sum_{k=m_1}^{n_1} \binom{n_2}{n_2 - j - l, j, l} \\
&\quad \times \mathbb{E}_1 \left[(1 - \mathfrak{p}_2(A_1^k))^j (\mathfrak{p}_2(A_1^k) - 2^{-k-1})^{n_2 - j - l} (2^{-k-1})^l \right] \Pr(\mathfrak{m}_1 = k).
\end{aligned}$$

Since $\mathfrak{p}_2(A_1^k)$ is identified by Lemma 6 and $\Pr(\mathfrak{m}_1 = k)$ is identified by Lemma 5, by plugging it into the above, the proof is completed. □

APPENDIX G. PROOF OF COROLLARY 1

Proof. Note that

$$\Pr(\mathfrak{m}_t \geq 1, \mathfrak{d}_t^b \leq n_t - m_t, \forall t = 1, \cdots, L) \leq \Pr(\mathfrak{m}_t \geq 1, \forall t = 1, \cdots, L),$$

and

$$1 - \Pr(\mathfrak{m}_t \geq 1, \forall t = 1, \cdots, L) = \Pr(\exists t, \text{ such that } \mathfrak{m}_t = 0) = \Pr(\mathcal{N}^{L+1}(\mathbf{x}) \text{ is born dead}).$$

It was shown in Theorem 3 of Lu et al. (2019) that

$$\Pr(\mathcal{N}^L(\mathbf{x}) \text{ is born dead}) \geq 1 - \mathfrak{a}_1^{L-2} + \frac{(1-2^{-n+1})(1-2^{-n})}{1+(n-1)2^{-n}}(-\mathfrak{a}_1^{L-2} + \mathfrak{a}_2^{L-2}),$$

where $\mathfrak{a}_1 = 1 - 2^{-n}$ and $\mathfrak{a}_2 = 1 - 2^{-n+1} - (n-1)2^{-2n}$. Thus the proof is completed. □

APPENDIX H. PROOF OF THEOREM 4

Proof. Since $q(\mathbf{x}) = E[\|\mathcal{N}(\mathbf{x})\|^2]/d_{\text{out}}$ and the rows of $\boldsymbol{W}^2$ are independent, without loss of generality let us assume $d_{\text{out}} = 1$. The direct calculation shows that

$$E[\|\mathcal{N}(\mathbf{x}_k)\|^2] = \sum_{i=1}^{N} \sigma_{\text{out}}^2 E\left[\phi(\boldsymbol{w}_i^T \mathbf{x}_k + \boldsymbol{b}_i)^2\right] = \frac{N\sigma_{\text{out}}^2}{N_{\text{train}}} \left\{ \sum_{i=1}^{N_{\text{train}}} E\left[\phi(\boldsymbol{w}_i^T(\mathbf{x}_k - \mathbf{x}_i) + |\epsilon_i|)^2\right] \right\}.$$

Let $\sigma_{k,i}^2 = \sigma_{\text{in}}^2 \|\mathbf{x}_k - \mathbf{x}_i\|^2$ and $\epsilon_{k,i} = |\epsilon_i|/\sigma_{k,i}$. Note that $\boldsymbol{w}_i^T(\mathbf{x}_k - x_i) \sim N(0, \sigma_{k,i}^2)$. Then

$$E\left[\phi(\boldsymbol{w}_i^T(\mathbf{x}_k - \mathbf{x}_i) + |\epsilon_i|)^2 | \epsilon_i\right] = I_1(\epsilon_i) + I_2(\epsilon_i),$$

where

$$I_1(\epsilon) = \int_0^\infty (z+\epsilon)^2 \frac{e^{-z^2/(2\sigma_{k,i}^2)}}{\sqrt{2\pi\sigma_{k,i}^2}} dz, \quad I_2(\epsilon) = \int_{-\epsilon}^0 (z+\epsilon)^2 \frac{e^{-z^2/(2\sigma_{k,i}^2)}}{\sqrt{2\pi\sigma_{k,i}^2}} dz.$$

Then if $\epsilon_i = |e_i|$ where $e_i \sim \text{N}(0, \sigma_{e,i}^2)$, we have

$$I_1(\epsilon_i) = \frac{1}{2}\sigma_{k,i}^2 + \sqrt{\frac{2}{\pi}}\sigma_{k,i}\epsilon_i + \frac{1}{2}\epsilon_i^2 \implies E[I_1(\epsilon_i)] = \frac{1}{2}\sigma_{k,i}^2 + \frac{2}{\pi}\sigma_{k,i}\sigma_{e,i} + \frac{1}{2}\sigma_{e,i}^2.$$

Also, we have

$$I_2(\epsilon) = \int_{-\epsilon}^0 (z+\epsilon)^2 \frac{e^{-z^2/(2\sigma_{k,i}^2)}}{\sqrt{2\pi\sigma_{k,i}^2}} dz = \sigma_{k,i}^2 \int_{-\epsilon_{k,i}}^0 (z+\epsilon_{k,i})^2 \frac{e^{-z^2/2}}{\sqrt{2\pi}} dz$$

$$= \sigma_{k,i}^2 \left[\frac{1}{2}(\epsilon_{k,i}^2 + 1)\text{erf}\left(\frac{\epsilon_{k,i}}{\sqrt{2}}\right) + \frac{\epsilon_{k,i}(e^{-\epsilon_{k,i}^2/2} - 2)}{\sqrt{2\pi}} \right],$$

where $\epsilon_{k,i} = |e_{k,i}|$ and $e_{k,i} \sim \mathrm{N}(0, \sigma_{e,i}^2/\sigma_{k,i}^2)$. Note that if $z = |z'|$ where $z' \sim \mathrm{N}(0, \sigma^2)$,

$$E[z^2 \mathrm{erf}(z/\sqrt{2})] = \frac{2\sigma^2 \tan^{-1}(\sigma)}{\pi} + \frac{2\sigma^3}{\pi(\sigma^2+1)},$$

$$E[\mathrm{erf}(z/\sqrt{2})] = \frac{2\tan^{-1}(\sigma)}{\pi}, \quad E[ze^{-z^2/2}] = \frac{2\sigma}{\sqrt{2\pi}(\sigma^2+1)}, \quad E[z] = \frac{2\sigma}{\sqrt{2\pi}}.$$

Therefore

$$E\left[\frac{1}{2}(z^2+1)\mathrm{erf}(z/\sqrt{2}) + \frac{ze^{-z^2/2} - 2z}{\sqrt{2\pi}}\right] = \frac{(\sigma^2+1)\tan^{-1}(\sigma)}{\pi} - \frac{\sigma}{\pi}.$$

By setting $\sigma = \sigma_{e,i}/\sigma_{k,i}$, we have

$$E[I_2(\epsilon_i)] = \sigma_{k,i}^2 \mathbb{E}_{\epsilon_{k,i}}\left[\frac{1}{2}(\epsilon_{k,i}^2+1)\mathrm{erf}\left(\frac{\epsilon_{k,i}}{\sqrt{2}}\right) + \frac{\epsilon_{k,i}(e^{-\epsilon_{k,i}^2/2} - 2)}{\sqrt{2\pi}}\right],$$

$$= \frac{(\sigma_{e,i}^2 + \sigma_{k,i}^2)\tan^{-1}(\sigma_{e,i}/\sigma_{k,i})}{\pi} - \frac{\sigma_{e,i}\sigma_{k,i}}{\pi} := \gamma_i.$$

Thus we have

$$E\left[\phi(\boldsymbol{w}_i^T(\mathbf{x}_k - \mathbf{x}_i) + |\epsilon_i|)^2\right] = E[I_1(\epsilon_i)] + E[I_2(\epsilon_i)] = \frac{1}{2}\sigma_{k,i}^2 + \frac{2}{\pi}\sigma_{k,i}\sigma_{e,i} + \frac{1}{2}\sigma_{e,i}^2 + \gamma_i,$$

and thus

$$E[\|\mathcal{N}(\mathbf{x}_k)\|^2] = \frac{N\sigma_{\text{out}}^2}{N_{\text{train}}}\sum_{i=1}^{N_{\text{train}}}\left[\frac{1}{2}\sigma_{k,i}^2 + \frac{1}{2}\sigma_{e,i}^2 + \frac{2}{\pi}\sigma_{k,i}\sigma_{e,i} + \gamma_i\right].$$

Let $\sigma_{e,i}^2 = \sigma_e^2 = \sigma_{\text{in}}^2 s^2$ for all i. Then we have

$$E[q(\mathbf{x}_k)] = E[\|\mathcal{N}(\mathbf{x}_k)\|^2] = \frac{N\sigma_{\text{out}}^2\sigma_{\text{in}}^2}{N_{\text{train}}\pi}\sum_{i=1}^{N_{\text{train}}}\left[(s^2 + \Delta_{k,i}^2)\left(\tan^{-1}(s/\Delta_{k,i}) + \pi/2\right) + s\Delta_{k,i}\right],$$

where $\Delta_{k,i} = \|\mathbf{x}_k - \mathbf{x}_i\|_2$. Thus, we obtain

$$\mathbb{E}_{\mathcal{X}_m}[q(\mathbf{x})] = \frac{1}{N_{\text{Ntrain}}}\sum_{k=1}^{N_{\text{train}}} E[q(\mathbf{x}_k)] = \frac{N\sigma_{\text{out}}^2\sigma_{\text{in}}^2}{N_{\text{train}}^2\pi}\sum_{k,i=1}^{N_{\text{train}}}$$
$$\times\left[(s^2 + \Delta_{k,i}^2)\left(\tan^{-1}(s/\Delta_{k,i}) + \pi/2\right) + s\Delta_{k,i}\right],$$

which completes the proof. □