599 DL Theory Final Project

Guanghao Ye
University of Washington
ghye@cs.uw.edu

Andrew Wei
University of Washington
nowei@cs.uw.edu

1 Introduction

In the past decade, neural networks have achieved great success on various real-world problems, such as speech recognition, image classification, and medical imaging. Despite its great success in those empirical tasks, the theory of deep learning remains largely under-explored. Hence, understanding the ability of neural networks has been a focus within the theoretical machine learning community.

Empirically, it has been shown that neural networks trained by first-order methods can fit training data fairly well [ZBH+17]. This is not surprising from an expressivity perspective, since those neural networks have much more parameters than the number of samples we are training with.

There is also an implicit assumption that as the number of parameters and layers increases, the modeling capabilities of neural networks increases. As such, modern deep learning practitioners have chased this ideology and neural networks have become deeper and wider. These large models come at a heavy cost, as the time it takes to train and test neural networks has increased dramatically over the last decade. Specialized hardware like GPUs have been employed to speed up the time for training these models. Even so, the increasing size of these models may result in the model not fitting within GPU memories, increasing the cost of training to require multiple GPUs. This direction is unsustainable, as we have limited memory and computational resources. This leads us to the following questions:

- Is the best way to learn really to make networks as deep and as wide as possible?
- Can we learn networks that can achieve similar results to larger neural networks with smaller ones?
- How large should the network be in order to memorize a given dataset?
- Can we train such networks efficiently?

Theoretically, many models are known to be learnable in polynomial time with a polynomial number of parameters unless there are more restrictions on the model [DZPS19, DLL+19, ZG19]. We will begin our investigation by reading [Dan20b], [Dan20a], and [GWZ19] to learn about the memorization abilities of neural networks with (almost) no over-parameterization. We will then look into [OS19a] to learn about the convergence guarantees of shallow neural networks.

In section 2, we will give a review of papers in the space of memorization of data. In section 3, we will cover some elementary results in the space of exact memorization. Lastly, we will go over the main theorems, techniques, and results within [Dan20b].

2 Related Papers

In this section, we will go over some papers that focus on the memorization of training data. We first talk about some work on memorization capacity and take a deeper dive on work surrounding

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

memorization using gradient-based methods. In the following, we use m for the number of training samples, d for the dimensionality of the feature space, and q for the number of neurons in the hidden layer.

2.1 Memorization Capacity

It's known that $O(m + d)$ many neurons are sufficient for memorization with arbitrary non-linearity [ZBH$^+$17]. Baum [Bau88] proved that for threshold activation functions and binary labels, a more compact representation can be achieved — $O(m/d)$ many neurons being sufficient for memorization — by taking advantage of the high-dimensionality of the data. Here, we note that number of neurons $q = m/d$ is optimal by a simple counting argument. Later, this was generalized to ReLU and deeper neural networks by [YSJ19] and [Ver20]. More recently, [BELM20] generalizes this to arbitrary real labels.

2.2 Memorization via Gradient-Based Method

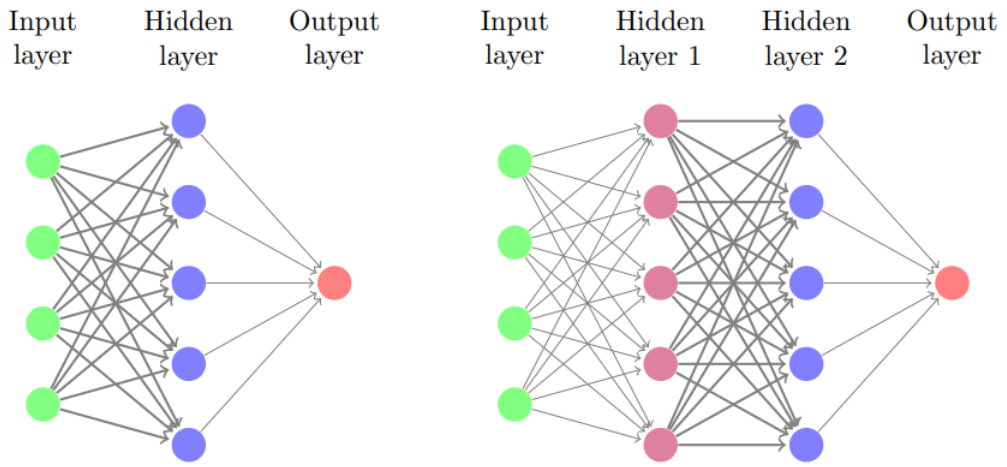
Naturally, another question people are interested in is how to find these networks. Specifically, can this kind of memorization be achieved with via gradient-based optimization on those neural networks? The literature on this line of work is very large. Some early work with minimal assumption include [LL18, DLL$^+$19]. The number of neurons in these works are usually a large polynomial in the number of data points. More recently, the amount of over-paremtrization needed was improved to a small polynomial in m and d [OS19b, KH19].

In later parts of the paper, we will focus on [Dan20b], but we will give a brief summary here. They give an analysis on the convergence of Neural Tangent Kernels to their expectations. This lets them show that SGD on depth-two neural networks initialized with a variant of the Xavier initialization can memorize training data, bounded-weight polynomials, and certain kernel spaces. Their results are mainly constrained on the d-ball. Their results are close to optimal, showing that they use m times some poly-logarithmic factor number of parameters.

In [Dan20a], they show that a single gradient descent step in a depth-two neural network can memorize $m = \Omega\left(dq/\log^4(d)\right)$ independent d-dimensional Gaussian points. They based their analysis on an initially orthonormal weights matrix and activation function with the following constraints: $O(1)$ Lipschitz, piecewise twice differentiable with finitely many pieces and bounded second derivative, and satisfies $\mathbb{E}_{X\sim\mathcal{N}(0,1)}\sigma'(X) = 0$. The main theorem they prove is that for some constrained m and q, there is a lower bound for $y_i h_{W^+}(\mathbf{x}_i) = \Omega(\ln(d))$ with high probability, where W^+ is defined as the weights after the gradient step. This implies that they are able to memorize m points drawn from a Gaussian with high probability.

In [GWZ19], they go on to show that training data can be memorized by mildly overparameterized models, with a constant factor more parameters than training samples and fewer neurons. To offset the growth rate of the number of parameters in a neural network, it is required that the number of neurons per layer is kept small. They achieve this result with polynomial activation functions and perform the analysis using perturbed inputs and Perturbed Gradient Descent. They begin with an inquiry into two-layer neural networks with quadratic loss functions and characterize the optimization landscape. They show that for small enough ϵ, every ϵ-second-order stationary point achieves almost zero training error. They then move on to a three-layer architecture, where the first layer randomly maps the input into k dimensions, where $k = \Theta(\sqrt{m})$, then apply a similar argument to the analysis on the two-layer networks.

In [OS19a], they show that stochastic gradient descent with random initializations converges at a geometric rate to a global optima when the root of the number of network parameters is larger than the size of the training data. The main motivation for this work is the observation that in the overparameterized setting, there are many global optima even though the training landscape is highly non-convex. This is one of the first papers that show a scaling with the number of parameters and the size of the training data instead of the number of hidden units and the size of the training data.



(a) 2-layer neural net structure (b) 3-layer neural net structure

Figure 1: 2-layer and 3-layer neural networks, from [GWZ19].

3 Some elementary results

We starts with prove some elementary results of exactly memorization.

Theorem 1 (Folklore). *Let σ be an non-polynomial activation function. Then, there exists a two-layer neural network with number of neurons $q = m$ such that*

$$h_W(\mathbf{x}_i) = y_i \qquad \forall i \in [m].$$

Proof. The set of function in the form $\mathbf{x} \mapsto u^\top \sigma(W\mathbf{x}+\mathbf{b})$ corresponds to a vector space V spanned by the functions $\mathbf{x} \mapsto \sigma(W\mathbf{x}+\mathbf{b})$. Let Π be a linear operator $\Pi : V \to \mathbb{R}^n$ that corresponds to the evaluation on data points $\mathbf{x}_i$, i.e. $\Pi(h_{\mathbf{W}})_i = h_{\mathbf{W}}(\mathbf{x}_i)$. Since Π is not a polynomial, the image of Π is $\mathbb{R}^n$. Moreover, $\mathrm{Im}(\Pi)$ is spanned by the set of vector $\Pi(h_{\mathbf{W}})$. Since $\dim(\mathrm{Im}(\Pi)) = n$, there is a subset of n such vectors with the same span, which completes the proof.

□

Theorem 2 ([Bau88]). *Let $\sigma(t) = \mathbb{1}\{t \geq 0\}$. Let $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\} \subseteq \mathbb{R}^d$ be in the general position. Let $\{y_1, \ldots, y_m\} \subseteq \{0,1\}^m$. Then, there is a two-layer neural network with number of neurons $q = O(m/d)$ such that*

$$h_W(\mathbf{x}_i) = y_i \qquad \forall i \in [m].$$

Proof. Baum builds a network iteratively: Pick d points with label 1. WLOG, we denote them as $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d$. Let $H = x : u^\top x = b$ be the hyperplane exactly containing those points. Using two neurons, one can build the indicator of a small neighborhood of H, by letting

$$f(\mathbf{x}) = \sigma(u^\top x - (b - \tau)) - \sigma(u^\top x(-b + \tau)$$

for sufficiently small τ. Hence, we have $f(\mathbf{x}_i) = 1$ for $i \leq d$, and $f(\mathbf{x}_i) = 0$ for $i > d$. Hence, we at most need $2 \cdot \frac{m}{d}$ many neurons to perfectly memorize the data.

□

The proof technique above can be used to generalize the result to ReLU with arbitrary real labeling.

Theorem 3 ([BELM20]). *Let σ be the ReLU function. Let $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\} \subseteq \mathbb{R}^d$ be in the general position. Let $\{y_1, \ldots, y_m\} \subseteq \mathbb{R}^m$. Then, there is a two-layer neural network with number of neurons $q = O(m/d)$ such that*

$$h_W(\mathbf{x}_i) = y_i \qquad \forall i \in [m].$$

Proof. Let $f_{\delta,u,v,b}$ defined by:

$$f_{\delta,u,v,b} : x \to \frac{\sigma((u+\delta v)\cdot x - b) - \sigma(u \cdot x - b)}{\delta}.$$

As $\delta \to 0$, the function is equal to

$$f_{u,v,b} = \sigma'(u \cdot x - b)v \cdot x.$$

For ReLU, by taking δ small enough, we have $f_{\delta,u,v,b} = f_{u,v,b}$. Note that it suffices to take

$$\delta = \frac{1}{2} \min_i \frac{|u \cdot \mathbf{x}_i - b|}{|v \cdot \mathbf{x}_i|}.$$

Thus, we can use 2 neurons to implement $f_{u,v,b}$.

Again, pick an arbitrary set of d points. WLOG, we denote them as $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d$. Let $H = x : u^\top x = b$ be the hyperplane exactly containing those points. Using 4 neurons, we can build the function $f = f_{u,v,b-\tau} - f_{u,v,b+\tau}$ for sufficiently small τ, so that $f(\mathbf{x}_i) = \mathbf{x}_i \cdot v$ for $i \leq d$ and $f(\mathbf{x}_i) = 0$ for $i > d$.

Now, it suffices to pick v such that $v \cdot x_i = y_i$ for any $i \leq d$. We complete the proof by noting that the matrix given by $\mathbf{x}_1, \ldots, \mathbf{x}_m$ is full-rank.

□

4 Memorization on random data

We concern ourselves with the task of memorizing points in the training data with slight over-parameterization. In this section, we will go more in-depth with the content in [Dan20b] and give some proof sketches on the theorems involved in the process of deriving their result on the memorization of training data.

4.1 Problem Setting

In this course project, we are interested in the problem of memorization. More specifically, we focus on two-layer (i.e., single hidden-layer) neural networks, where such a network can be written in the form of

$$h_{W,u}(\boldsymbol{x}) = u^\top \sigma(W\boldsymbol{x}).$$

Definition 4 (informal)**.** *Given a dataset $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\} \subset \mathbb{R}^d \times \{\pm 1\}$ where y_i's are randomly drawn from $\{\pm 1\}$ and $\delta > 0$. The function h* memorizes *the dataset S, if $h(x_i) = y_i$ with probability at least $1 - \delta$.*

Algorithm 1 Neural Network Training

Input: Network parameters σ and d, q, loss ℓ, initialization parameter $B > 0$, learning rate $\eta > 0$, batch size b, number of steps $T > 0$, access to samples from a distribution $\mathcal{D}$
Sample $\mathbf{W}^1 \sim \mathcal{I}(d, q, B)$
for $t = 1, \ldots, T$ **do**
  Obtain a mini-batch $S_t = \{(\mathbf{x}_i^t, y_i^t)\}_{i=1}^b \sim \mathcal{D}^b$
  With back-propagation, calculate a stochastic gradient $\nabla \mathcal{L}_{S_t}(\mathbf{W}^t)$ and update

$$\mathbf{W}^{t+1} \leftarrow \mathbf{W}^t - \eta \nabla \mathcal{L}_{S_t}(\mathbf{W}^t)$$

end for
Choose $t \in [T]$ uniformly at random and return $\mathbf{W}_t$

They utilize a depth-two neural network with a non-standard variant of Xavier initializations and show that such a network can memorize samples, learn polynomials with bounded weights, and learn certain kernel spaces. For the purpose of memorization, they show that they can memorize m randomly labeled samples on the d-ball using depth-two neural networks with m and some poly logarithmic factors of parameters. Here, we give some sketches of their proofs.

4.2 Prelims

Definition 5 (Xavier initialization with zero output [GB10])**.** *Let $W' \in \mathbb{R}^{q\times d}$ be a matrix drawn from standard gaussian. Let $u' \in \mathbb{R}^q$ to be the all-B vector. Then, the hidden weight W defined to be*

$$W = \begin{bmatrix} W' \\ W' \end{bmatrix},$$

and the output weight is defined to be

$$u = \begin{bmatrix} u' \\ -u' \end{bmatrix}.$$

We denote this distribution of initialization by $\mathcal{I}(d, q, B)$.

We further remark that for any $\mathbf{W} \sim \mathcal{I}(d, q, B)$, we have $h_{\mathbf{W}}(\mathbf{x}) = 0$ for any $\mathbf{x} \in \mathbb{R}^d$.

Algorithm 2 Neural Tangent Kernel Training

Input: Network parameters σ and d, q, loss ℓ, learning rate $\eta > 0$, batch size b, number of steps $T > 0$, access to samples from a distribution $\mathcal{D}$
Sample $\mathbf{W} \sim \mathcal{I}(d, q, 1)$
Let $\Psi_{\mathbf{W}}(\mathbf{x})$ be the gradient of function $h_{\mathbf{W}}(\mathbf{x})$ w.r.t. the hidden weights.
Define $f_{\Psi_{\mathbf{W}},\mathbf{V}} = \langle \mathbf{V}, \Psi_{\mathbf{W}}(\mathbf{x}) \rangle$
Initialize $\mathbf{V}^1 = 0 \in \mathbb{R}^{2q\times d}$
for $t = 1, \ldots, T$ **do**
  Obtain a mini-batch $S_t = \{(\mathbf{x}_i^t, y_i^t)\}_{i=1}^b \sim \mathcal{D}^b$
  Using back-propagation, calculate the gradient ∇ of $\mathcal{L}_{S_t}(\mathbf{V}) = \mathcal{L}_{S_t}\left(f_{\Psi_{\mathbf{W}},\mathbf{V}}\right)$ at $\mathbf{V}^t$ Update

$$\mathbf{V}^{t+1} \leftarrow \mathbf{V}^t - \eta\nabla$$

end for
Choose $t \in [T]$ uniformly at random and return $f_{\Psi_W,\mathbf{V}_t}$

Let $f_{\Psi,\boldsymbol{v}} = \langle \boldsymbol{v}, \Psi(x) \rangle_{\mathcal{H}}$ and $\|f\|_k = \min\{\|\boldsymbol{v}\|_{\mathcal{H}} : f_{\Psi,\boldsymbol{v}}\}$. For a kernel k and $M > 0$, we denote $\mathcal{H}_k^M = h \in \mathcal{H}_k : \|h\|_k \le M$.

Definition 6. *Fix network parameters σ, d, q and B. The neural tangent kernel corresponding to weights $\mathbf{W}$ is defined to be*

$$\mathrm{tk}_{\mathbf{W}}(\mathbf{x}, \mathbf{y}) = \frac{\langle \nabla_{\mathbf{W}} h_{\mathbf{W}}(\mathbf{x}), \nabla_{\mathbf{W}} h_{\mathbf{W}}(\mathbf{y}) \rangle}{2qB^2}.$$

4.3 Proof sketches

Now, we can state the main theorem.

Theorem 7. *Given $d, M, R, \varepsilon > 0$, there is a choice of $q = \widetilde{O}(\frac{M^2R^2}{d\varepsilon^2})$,$T = O(M^2/\varepsilon^2)$, $B > 0$, and $\eta > 0$, such that for any R-bounded distribution $\mathcal{D}$ and batch size b. The function h returned by Algorithm 1 satisfied*

$$\mathbb{E}\mathcal{L}_{\mathcal{D}}(h) \le \mathcal{L}_{\mathcal{D}}(\mathcal{H}^M_{\mathrm{tk}_\sigma^h}) + \varepsilon.$$

Instead of directly working with this neural networking training algorithm, we'll show that it is actually equivalent to Algorithm 2.

We say an activation function is decent if it is continuous, twice differentiable in almost all points, and there is $M > 0$ such that $|\sigma'(x)| \le M$ and $|\sigma''(x)| \le M$ for every twice differentiable point. We define a loss function as L-decent when if for every $y \in \mathcal{Y}$, the loss function is convex, L-Lipschitz, and twice differentiable in almost all points.

Theorem 8. *Fix a decent activation σ and decent loss ℓ. There is a choice of B, such that for every input distribution we have*

$$|\mathbb{E}\mathcal{L}_{\mathcal{D}}(h_1) - \mathbb{E}\mathcal{L}_{\mathcal{D}}(h_2)| \le \varepsilon.$$

where h_1 is the function returned by Algorithm 1 with parameters $d, q, \eta/B^2, b, B, T$ and h_2 is the function returned by Algorithm 2 with parameters d, q, η, b, T.

Proof. We consider another run of Algorithm 1, where $\mathbf{W} \sim \mathcal{I}(d, q, 1)$. We denote the corresponding function as $\tilde{h}_W(\mathbf{x})$ and denote the function with $\mathbf{W} \sim \mathcal{I}(d, q, B)$ as $h_W(\mathbf{x})$. Then, we have

$$\begin{aligned}\tilde{h}_{W+V}(\mathbf{x}) &= Bh_{W+V}(\mathbf{x}) \\ &= Bh_W(\mathbf{x}) + B\langle \nabla_W h_W(\mathbf{x}), V\rangle + \frac{HB}{2}\|V\|_2^2.\end{aligned}$$

where the second step follows by the first order approximation and H is a uniform bound on the Hessian.

We note that $Bh_W(\mathbf{x}) = 0$ by the initialization scheme. Now, it suffices to show that

$$\frac{HB}{2}\|V\|_2^2 \leq \varepsilon$$

for sufficiently large B. Note that for the run with initialization $\mathbf{W} \sim \mathcal{I}(d, q, 1)$, the movement is in the ball with radius R where R doesn't depends on B. While for the run with initialization $\mathbf{W} \sim \mathcal{I}(d, q, B)$, the gradient is scaled up by B and the step size is $1/B^2$. Hence, we have $\|V\|_2^2 \leq R^2/B^2$. Thus, by choosing $B = \Omega(HR^2)$, we have

$$\tilde{h}_{W+V}(\mathbf{x}) = B\langle \nabla_W h_W(\mathbf{x}), V\rangle + \epsilon.$$

Hence, Algorithm 1 is equivalent to optimization over the linear function $B\langle \nabla_W h_W(\mathbf{x}), V\rangle$ with step size η/B^2 up to a error ϵ.

□

Now, it suffices to prove following theorem.

Theorem 9. *Given $d, M, R, \varepsilon > 0$, there is a choice of $q = \widetilde{O}(\frac{M^2R^2}{d\varepsilon^2})$,$T = O(M^2/\varepsilon^2)$, $B > 0$, and $\eta > 0$, such that for any R-bounded distribution $\mathcal{D}$ and batch size b. The function h returned by Algorithm 2 satisfied $\mathbb{E}\mathcal{L}_{\mathcal{D}}(h) \leq \mathcal{L}_{\mathcal{D}}(\mathcal{H}^M_{\mathrm{tk}^h_\sigma}) + \varepsilon$.*

Algorithm 3 SGD on RFS

Input: RFS $\psi : \Omega \times \mathcal{X} \to \mathbb{R}^d$, number of random features q, loss ℓ, learning rate $\eta > 0$, batch size b, number of steps $T > 0$, access to samples from a distribution $\mathcal{D}$
Sample $\boldsymbol{\omega} \sim \mu^q$
Initialize $\mathbf{v}^1 = 0 \in \mathbb{R}^{q\times d}$
for $t = 1, \ldots, T$ **do**
  Obtain a mini-batch $S_t = \{(\mathbf{x}_i^t, y_i^t)\}_{i=1}^b \sim \mathcal{D}^b$
  Update $\mathbf{v}_{t+1} = \mathbf{v}_t - \eta\nabla\mathcal{L}_{S_t}(\mathbf{v}_t)$ where $\mathcal{L}_{S_t}(\mathbf{v}) = \mathcal{L}_{S_t}(f_{\Psi_{\boldsymbol{\omega}},\mathbf{v}})$.
end for
Choose $t \in [T]$ uniformly at random and return $f_{\Psi_{\boldsymbol{\omega}},\mathbf{v}_t}$

We note that Algorithm 2 is SGD on the random embedding Ψ_ω, which is composed of q i.i.d. random mappings $\psi_\omega(\mathbf{x}) = (\sigma'(\omega^\top\mathbf{x})\mathbf{x}, -\sigma'(\omega^\top x)x)$ where $\omega \in \mathbb{R}^d$ is drawn from the standard gaussian. We note that if we change the mapping to $\psi_\omega(\mathbf{x}) = \sigma'(\omega^\top\mathbf{x})\mathbf{x}$, the inner product between the different samples only change by a multiplicative factor of $\sqrt{2}$.

Random feature schemes map the input data to a randomized low-dimensional feature space and have been used to speed up the time to train kernel machines [RR07]. The vector random feature scheme used here allows us to approximate functions, giving us the ability to obtain an ϵ-approximation of f. When SGD is applied to these embeddings with a convex and Lipschitz loss, we get an approximation that is not too bad. This can be applied to the Neural Tangent Kernel Random Feature Scheme and to neural network learning. Intuitively, this scheme returning good

approximations makes sense because enough random vectors would give us a wide variety of embeddings for the function being approximated.

To get the tight bound on the memorization, we cannot directly apply theorem above. We note that it suffices to show that for $q = \tilde{O}(m/d) = \tilde{O}(d^{c-1})$ with probability $1 - o(1)$ over the choice of S and $\boldsymbol{\omega}$, there is $\mathbf{v} \in \mathbb{R}^{dq}$ such that

$$\langle \mathbf{v}, \Psi_{\boldsymbol{\omega}}(\mathbf{x}_i) \rangle = y_i + o(1) \text{ for all } i \text{ and } \|v\|_2^2 = \tilde{O}(m). \tag{1}$$

We define

$$f(\mathbf{x}) = \sum_{i=1}^{m} y_i (\langle \mathbf{x}_i, \mathbf{x} \rangle)^{c'}$$

where $c' > 4c + 2$.
Let $\boldsymbol{v}$ be defined to be

$$\boldsymbol{v} = \frac{1}{\sqrt{q}} (\breve{f}(\omega_1), \ldots, \breve{f}(\omega_q)).$$

Note that, we have

$$\mathbb{E}\|\boldsymbol{v}\|_2^2 = \|f\|_{k_\sigma}^2 \qquad \text{and} \qquad \mathbb{E}[\boldsymbol{v}^\top \Psi_\omega(\mathbf{x})] = \mathbb{E}[f_\omega(\mathbf{x})] = f(\mathbf{x}).$$

This implies Eq. (1) holds for large q by Chebyshev's inequality. However, this gives us $q = \widetilde{O}(m^2/d)$ instead of $q = \widetilde{O}(m/d)$. The original author resolves this issue by giving a more fine-grained analysis on the rate of f_ω approximating f in this case. Here, we omit that part of details of proof (see A.4 of the supplement of [Dan20b]).

5 Conclusion and Future Work

The number of parameters in a neural network is loosely tied to how long it takes to train the model, but increasing the number of parameters is thought to increase the expressivity and power of the model. Thus, using a minimal number of parameters while maintaining decent performance and expressivity is important for the purposes of efficient model feedback. Contrary to this goal, modern neural networks have grown wider and deeper. This leads us to focus on learning more about minimizing the number of parameters necessary to learn a model or to memorize the training data.

Intuitively, to memorize m distinct points, we need at least m parameters. Otherwise, we would not be able to distinguish each data point separately. We have seen several schemes to memorize training data points, but many of them fall within well-defined settings, which is not applicable to most real-world settings. The overparameterized setting is also not the most general, as data collection has become cheaper. Even so, their results may still be useful in applications where the training data is limited due to scarcity/privacy of data, like with medical datasets, or give us better intuitions on good models with a minimal number of parameters. Lastly, learning how this applies to the underparameterized setting may help give us some insights on the undiscovered potential of neural networks.

We have only begun our exploration into the memorization capabilities of neural networks, but it seems like a hopeful direction for a future towards efficient and lower-cost training.

References

[Bau88] Eric B. Baum. On the capabilities of multilayer perceptrons. *J. Complex.*, 4(3):193–215, 1988.

[BELM20] Sébastien Bubeck, Ronen Eldan, Yin Tat Lee, and Dan Mikulincer. Network size and size of the weights in memorization with two-layers neural networks. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[Dan20a] Amit Daniely. Memorizing gaussians with no over-parameterizaion via gradient decent on neural networks. *CoRR*, abs/2003.12895, 2020.

[Dan20b] Amit Daniely. Neural networks learning and memorization with (almost) no over-parameterization. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[DLL+19] Simon S. Du, Jason D. Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 1675–1685. PMLR, 2019.

[DZPS19] Simon S. Du, Xiyu Zhai, Barnabás Póczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. JMLR Workshop and Conference Proceedings.

[GWZ19] Rong Ge, Runzhe Wang, and Haoyu Zhao. Mildly overparametrized neural nets can memorize training data efficiently, 2019.

[KH19] Kenji Kawaguchi and Jiaoyang Huang. Gradient descent finds global minima for generalizable deep neural networks of practical sizes. In *57th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2019, Monticello, IL, USA, September 24-27, 2019*, pages 92–99. IEEE, 2019.

[LL18] Yuanzhi Li and Yingyu Liang. Learning overparameterized neural networks via stochastic gradient descent on structured data. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 8168–8177, 2018.

[OS19a] Samet Oymak and Mahdi Soltanolkotabi. Towards moderate overparameterization: global convergence guarantees for training shallow neural networks, 2019.

[OS19b] Samet Oymak and Mahdi Soltanolkotabi. Towards moderate overparameterization: global convergence guarantees for training shallow neural networks. *CoRR*, abs/1902.04674, 2019.

[RR07] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*, NIPS'07, page 1177–1184, Red Hook, NY, USA, 2007. Curran Associates Inc.

[Ver20] Roman Vershynin. Memory capacity of neural networks with threshold and relu activations. *CoRR*, abs/2001.06938, 2020.

[YSJ19] Chulhee Yun, Suvrit Sra, and Ali Jadbabaie. Small relu networks are powerful memorizers: a tight analysis of memorization capacity. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors,

Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, pages 15532–15543, 2019.

[ZBH+17] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[ZG19] Difan Zou and Quanquan Gu. An improved analysis of training over-parameterized deep neural networks. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 2053–2062, 2019.