Training Multi-Layer Over-Parametrized Neural Network in Subquadratic Time

Zhao Song
Adobe Research, San Jose, CA, USA

Lichen Zhang
Massachusetts Institute of Technology, Cambridge, MA, USA

Ruizhe Zhang
Simons Institute for the Theory of Computing, Berkeley, CA, USA

Abstract

We consider the problem of training a multi-layer over-parametrized neural network to minimize the empirical risk induced by a loss function. In the typical setting of over-parametrization, the network width m is much larger than the data dimension d and the number of training samples n ($m = \mathrm{poly}(n, d)$), which induces a prohibitive large weight matrix $W \in \mathbb{R}^{m \times m}$ per layer. Naively, one has to pay $O(m^2)$ time to read the weight matrix and evaluate the neural network function in both forward and backward computation. In this work, we show how to reduce the training cost per iteration. Specifically, we propose a framework that uses m^2 cost only in the initialization phase and achieves *a truly subquadratic cost per iteration* in terms of m, i.e., $m^{2-\Omega(1)}$ per iteration. Our result has implications beyond standard over-parametrization theory, as it can be viewed as designing an efficient data structure on top of a pre-trained large model to further speed up the fine-tuning process, a core procedure to deploy large language models (LLM).

2012 ACM Subject Classification Theory of computation → Streaming, sublinear and near linear time algorithms; Theory of computation → Machine learning theory; Theory of computation → Nonconvex optimization

Keywords and phrases Deep learning theory, Nonconvex optimization

Digital Object Identifier 10.4230/LIPIcs.ITCS.2024.93

Related Version *Full Version*: `https://arxiv.org/abs/2112.07628`

Funding *Lichen Zhang*: Supported by NSF CCF-1955217 and NSF DMS-2022448.

Acknowledgements We would like to thank Yin Tat Lee for discussing the motivation of this problem. We would also like to thank Jan van den Brand, Binghui Peng, Omri Weinstein, and David P. Woodruff for helpful discussions in the early stage of this project. We would like to thank Pravesh Kothari and Gary Miller for helpful discussions on the data structures.

1 Introduction

Over-parameterized neural networks represent one of the most extensively researched and widely utilized models in contemporary machine learning. These networks are among the first to offer convergence guarantees for prevalent deep network training methodologies, as indicated by numerous studies [47, 34, 26, 3, 4, 25]. Beyond the sound convergence theory, the recent surge of large language models (LLMs) also raises the stakes of training such networks efficiently. In particular, the over-parametrization setting aligns well with the *fine-tuning* process of LLMs. To make it concrete, recall fine-tuning is the procedure of adapting an LLM for a particular set of focused data so that it can be more specialized in certain domains. Since the size of fine-tuning data is much smaller compared to the size of actual training data of an LLM, fine-tuning can typically be implemented in an efficient manner. Moreover, the over-parametrization theory requires the number of parameters of a network (expressed as the *width* of the network) to be orders of magnitude larger than that of the input dataset, which is exactly the setup for fine-tuning as the size of an LLM is much larger than the fine-tuning dataset. Speeding up the fine-tuning process is one of the core algorithmic tasks for large language model applications, and many works attempt to exploit the inherent over-parametrization structure of the process. The work [30] explicitly utilizes the *low-rank* property of the gradient and derives a heuristic fine-tuning procedure that greatly reduces the parameters required.

Meanwhile, developments in the department of convex optimization have led to breakthroughs in tasks such as linear programming [21, 38, 13, 12, 29], empirical risk minimization [46, 54], the cutting plane method [36], semidefinite programming [35, 32, 29], and sum-of-squares optimization [37]. However, the techniques developed in these works are tailored for optimizing convex objectives, which are not applicable to the training of over-parameterized networks. Furthermore, existing research on efficient training for over-parameterized networks primarily focuses on two-layer networks [14, 62, 31, 5], which differs substantially from the networks utilized in real-world applications.

In this work, we take the first step to develop a fast algorithm for training deep over-parameterized networks. Specifically, given n data points of dimension d, we consider training a fully connected neural network with width $m = \mathrm{poly}(n, d)$ and depth $L \geq 2$, using a shifted ReLU activation. To ensure the convergence of the training process, the network width m is typically chosen to be a large polynomial in n and d. For deep neural networks, the current best over-parameterization width is on the order of n^4 [19]. The input layer is represented as a matrix of size $m \times d$, and all intermediate layers are of size $m \times m$. The training typically consists of a forward pass and a backward pass. The forward pass involves multiplying weight matrices $W \in \mathbb{R}^{m \times m}$ with vectors $h \in \mathbb{R}^m$. Without any structure on W and h, this step would take $\Theta(m^2)$ time, which is prohibitively large. We hence ask the following question:

Is it possible to reduce the cost per iteration during the training to truly subquadratic in m?

For training two-layer over-parameterized neural networks, two interesting results have been obtained by van den Brand, Peng, Song, and Weinstein [14] and Song, Yang, and Zhang [62]. However, we note that the settings both of these papers studied differ from ours, specifically:

- In the two-layer case, they only need to train a weight matrix of size $m \times d$. Since evaluating one data point in d-dimensional space for neural network functions takes $O(md)$ time, the cost per iteration bound they aim to match [14] or beat [62] is $O(md)$.
- In the multi-layer case, instead of having only a weight matrix of size $m \times d$, we will have at least one weight matrix of size $m \times m$. Since evaluating one data point in m-dimensional space for neural network functions takes $O(m^2)$ time, the cost per iteration bound we are trying to beat in this work is $O(m^2)$.

In [14], they provide an algorithm that can adaptively choose the step size for different gradient directions associated with different data points, which is one of the goals we want to achieve. However, their method involves forming a Jacobian matrix of the data, which is of size $n \times md$ in the two-layer case, but of size $n \times m^2$ in our case. Hence, their algorithm can only imply an $O(nm^2)$ cost per iteration, which cannot match our subquadratic goal.

In [62], they surpass the nmd barrier by leveraging two key ideas: using a shifted ReLU activation and, through a sparsity analysis, showing that only $o(m)$ number of neurons are fired up[1]. They further use a data structure to preprocess both data points and training

[1] Such a phenomenon has been observed in practice as in [18, 17, 50].

weights. However, their algorithm only works for small d due to the exponential dependence on d in the preprocessing phase. In a multiple-layer neural network, instead of having only a weight matrix of size $m \times d$, we will have at least one weight matrix of size $m \times m$; this directly translates to a high dependence on m in the preprocessing phase. As our hope is for an algorithm with ideally $O(nm^2)$ time to preprocess, their data structure is infeasible for any practical application.

We also note two more recent works [31, 5] that particularly focus on developing efficient algorithms for training two-layer over-parameterized networks. Both of them make use of tree-based data structures to find activated neurons in sublinear time. These approaches, unfortunately, do not scale well to multi-layer settings, as they rely on the input dimension being much smaller than the network width, while in multi-layer settings, the weight matrix is square.

Our method overcomes the shortcomings of these prior approaches, as it can not only choose the step size adaptively but also achieve a per iteration cost of $o(m^2)$. It can be interpreted as a design guideline for efficient fine-tuning of LLMs. As we will see later, our algorithm also leverages the *low-rank* property of the gradient but in a *provable* fashion. Moreover, our approach goes beyond the standard first-order methods, such as gradient descent or stochastic gradient descent, as we provide a highly efficient implementation of the Gauss-Newton method. One can treat our result as a data structure with a manual for LLM fine-tuning. Although our convergence argument works for over-parameterized, shifted ReLU-based networks, we believe it will serve as a foundation for designing even faster *practical* algorithms for fine-tuning.

1.1 Our Results

Our main results can be summarized in the following theorems: the first analyzes the convergence behavior of a general Gram-based optimization framework and the second designs an efficient algorithm to achieve subquadratic cost per iteration.

Throughout this paper, we use n to denote the number of training data points, d to represent the dimension of input data points, m for the width of the network, and L for the number of layers in the network. We denote the smallest eigenvalue of the neural tangent kernel induced by our neural network as λ_L and the prediction of the neural network at time t as $f_t \in \mathbb{R}^n$.

Our first theorem demonstrates the fast convergence rate of our algorithm.

▶ **Theorem 1** (Convergence). *Suppose the width of the neural network satisfies $m = \mathrm{poly}(n, \lambda_L^{-1}, L)$. Then, there exists an algorithm (Algorithm 1) such that, over the randomness of initialization of the network and the algorithm, with probability at least $1 - 1/\mathrm{poly}(n)$, we have*

$$\|f_{t+1} - y\|_2 \leq \frac{1}{3}\|f_t - y\|_2.$$

The above theorem establishes the linear convergence rate of our method. However, compared to the one-hidden layer case, our analysis is more sophisticated as we must control the probability so it does not exponentially blow up with respect to the number of layers.

Our convergence argument follows a novel and systematic approach to analyze the dynamics of multi-layer neural tangent kernels. Previous works either depend exponentially on the number of layers L in network width [25] or directly analyze training dynamics, requiring a data-separable assumption [3]. Our analysis builds upon these works, providing a comprehensive theory of multi-layer NTKs under first- or second-order perturbations, thus greatly extending the landscape of NTK-based convergence theory. The framework we developed is also compatible with sparser, shifted ReLU activations.

We note that since we are essentially learning an NTK, the generalization guarantee of our approach is at least as good as that of an NTK [6]. The generalization bound of the shifted ReLU has also been carefully examined in [69].

The next theorem concerns the *cost per iteration* of our algorithm.

▶ **Theorem 2** (Runtime). *There exists a randomized algorithm (Algorithm 1) that trains a multi-layer neural network of width m with the cost per training iteration being $\widetilde{O}(nm^{2-\Omega(1)})$. Furthermore, if $m \geq n^4$, then the cost per iteration is*

$$\widetilde{O}(m^{2.25-\alpha}),$$

where $\alpha \geq 0.32$ is the dual matrix multiplication exponent [66, 41].

We improve the overall training time of multi-layer over-parametrized networks from $\mathcal{T}_{\text{init}} + T \cdot \widetilde{O}(nm^2)$ to $\mathcal{T}_{\text{init}} + T \cdot o(nm^2)$, where $\mathcal{T}_{\text{init}}$ is the initialization time of training, typically taking $O(nm^2)$. As previously argued, multi-layer over-parametrized networks require m to be on the order of n^4, hence improving the cost per iteration from quadratic to subquadratic is a significant gain in speeding up training. Our algorithmic result can be interpreted as a data structure for fine-tuning LLMs: given a large, pre-trained language model whose weights are "frozen" during fine-tuning, we only need to initialize the data structure once. The data structure can then be queried in a highly-efficient manner to implement fine-tuning steps.

To achieve subquadratic time, we cannot afford to perform matrix-vector products between the weight matrices and any dense vectors. However, matrix-vector product seems unavoidable, as any algorithm requiring first- or second-order information needs to evaluate the network prediction. When implementing the Gauss-Newton method, we also have a Jacobian matrix of size $n \times m^2$, so forming it exactly would already take $\Theta(nm^2)$ time. Moreover, note that the update is also an $m \times m$ matrix, meaning that explicitly applying the update would also cost $\Theta(m^2)$ time. To circumvent these problems, we exploit the fact that the gradient is low-rank (rank n). Thus, one can compute a rank-n factorization and use it to support fast matrix-vector products. We also observe that each row of the Jacobian matrix can be formulated as a tensor product of two vectors. Therefore, we can use fast randomized linear algebra to approximate the tensor product efficiently.

1.2 Related Work

Convex and Non-Convex Optimization

In convex optimization problems, such as linear programming [65, 22, 44, 21, 71, 64, 38], empirical risk minimization [46, 54], the cutting plane method [36], maximum bipartite matching and maxflow [48, 8], and semidefinite programming [45, 35, 32, 29], one typically uses an algorithm that can dynamically adjust the search direction and step size to reduce the iteration count. Due to the prohibitively high cost of implementing one step of these methods, most of these works focus on improving the *cost per iteration.*

In the non-convex setting, there is a vast body of ongoing works [49, 10, 53, 1, 9, 15, 72, 14, 70] that try to improve the iteration count and cost per iteration, especially in the setting of training deep neural networks. As shown in [15], it is possible to exploit the equivalence between over-parametrized networks and neural tangent kernels to optimize an $n \times n$ matrix instead of an $m^2 \times m^2$ matrix, which is an important breakthrough in gaining speedup for such methods. Sketching and sampling-based methods can also be used to accelerate the computation of inverses of the Hessian matrix [53]. In spirit, our work most resembles [15] and [14], in the sense that our optimization also works on an $n \times n$ Gram matrix. Our algorithm also makes use of sketching and sampling, as in [53].

Over-Parametrized Neural Networks

In the deep learning community, understanding the geometry and convergence behavior of various optimization algorithms on over-parametrized neural networks has received a lot of attention [47, 34, 26, 3, 4, 25, 63, 14, 73, 74, 16, 42, 43, 52, 19, 33, 62, 5, 31, 28]. The seminal work of [34] initiates the study of the *neural tangent kernel* (NTK), which is a very useful analytical model to prove the convergence of training on over-parametrized networks. By over-parametrizing the neural network so that the network width is relatively large ($m \geq \Omega(n^4)$), one can show that the training dynamics on a neural network are almost the same as those on an NTK.

Sketching

Using randomized linear algebra to reduce the dimension of problems and speed up algorithms for various problems has been a growing trend in the machine learning community [56, 20, 67] due to its wide range of applications, especially the efficient approximation of kernel matrices [7, 2, 68, 59]. The standard "Sketch-and-Solve" paradigm [20] involves reducing the dimension of the problem via sketching and then using a black box for the original problem to gain computational efficiency. Another line of work uses sketching as a preconditioner [67, 14] to obtain high-precision solutions. Sketching has also been successful in solving symmetric norm regression [58], clustering and coverage [27], low-rank approximation [60, 61], and in distributed settings [11].

Roadmap

In Section 2, we give a preliminary overview of the training setup considered in this paper. In Section 2.1, we introduce the notations that will be used throughout the paper. In Section 2.2, we consider the training setting more specifically. In Section 3, we provide an overview of the techniques employed in this paper. In Section 3.1, we examine the algorithmic tools utilized in this work to achieve subquadratic cost per iteration. In Section 3.2, we demonstrate various techniques to prove the convergence of our second-order method. Finally, in Section 4, we summarize the results of this paper and point out some potential future directions.

2 Preliminaries

2.1 Notations

For any integer $n > 0$, let $[n]$ denote the set $\{1, 2, \cdots, n\}$. Let $\Pr[\cdot]$ denote probability and $\mathbb{E}[\cdot]$ denote expectation. We use $\|x\|_2$ to denote the ℓ_2 norm of a vector x. We use $\|A\|$ and $\|A\|_F$ to denote the spectral norm and the Frobenius norm of matrix A, respectively. We use $A^\top$ to denote the transpose of matrix A. We use I_m to denote the identity matrix of size $m \times m$. For α being a vector or matrix, we use $\|\alpha\|_0$ to denote the number of nonzero entries of α. Given a real square matrix A, we use $\lambda_{\max}(A)$ and $\lambda_{\min}(A)$ to denote its largest and smallest eigenvalues respectively. Given a real matrix A, we use $\sigma_{\max}(A)$ and $\sigma_{\min}(A)$ to denote its largest and smallest singular values respectively. We use $\mathcal{N}(\mu, \sigma^2)$ to denote the Gaussian distribution with mean μ and variance σ^2. We use $\widetilde{O}(f(n))$ to denote $O(f(n) \cdot \mathrm{poly}\log(f(n)))$. We use $\langle \cdot, \cdot \rangle$ to denote the inner product, when applying to two vectors, this denotes the standard dot product between two vectors, and when applying to two matrices, this means $\langle A, B \rangle = \mathrm{tr}[A^\top B]$, i.e., the trace of $A^\top B$. For a vector $x \in \mathbb{R}^n$, we use $\mathrm{diag}(x)$ to denote an $n \times n$ diagonal matrix where diagonal entries are from x. Given two vectors x, y, we use $x \otimes y$ to denote their tensor product and given two matrices A, B, we use $A \otimes B$ to denote their Kronecker product.

2.2 Problem Setup

Architecture

We first describe our network architecture. The network consists of L hidden layers, each represented by a weight matrix $W_\ell \in \mathbb{R}^{m \times m}$ for any $\ell \in [L]$. The output layer consists of a vector $a \in \mathbb{R}^m$. We define the evaluation of the neural network as a prediction function $f : \mathbb{R}^d \to \mathbb{R}$

$$f(W,x) = a^\top \phi(W_L(\phi(\cdots \phi(W_1 x)))),$$

where $\phi : \mathbb{R} \to \mathbb{R}$ is the shifted ReLU activation function ($\sigma_b(x) = \max\{x - b, 0\}$) applied coordinate-wise to a vector.

We measure the loss via the squared-loss function:

$$\mathcal{L}(W) = \frac{1}{2}\sum_{i=1}^{n}(y_i - f(W,x_i))^2.$$

This is also the objective function for our training.

The prediction function $f_t : \mathbb{R}^{d \times n} \to \mathbb{R}^n$ is defined as

$$f_t(X) = \begin{bmatrix} f(W(t),x_1) & f(W(t),x_2) & \cdots & f(W(t),x_n) \end{bmatrix}^\top .$$

Initialization

Our neural networks are initialized as follows:

- For each $\ell \in [L]$, the layer-ℓ's weight parameter $W_\ell(0) \in \mathbb{R}^{m \times m}$ is initialized such that each entry is sampled from $\mathcal{N}(0, \frac{2}{m})$.
- Each entry of a is an i.i.d. sample from $\{-1, +1\}$ uniformly at random.

Gradient

In order to write the gradient in an elegant way, we define some artificial variables: for all $i \in [n]$

$$\begin{aligned} g_{i,1} &= W_1 x_i, \quad h_{i,1} = \phi(W_1 x_i), \\ g_{i,\ell} &= W_\ell h_{i,\ell-1}, \quad h_{i,\ell} = \phi(W_\ell h_{i,\ell-1}), \quad \forall \ell \in [L]\backslash\{1\} \end{aligned}$$

and

$$\begin{aligned} D_{i,1} &= \operatorname{diag}\big(\phi'(W_1 x_i)\big), \quad \forall i \in [n] \\ D_{i,\ell} &= \operatorname{diag}\big(\phi'(W_\ell h_{i,\ell-1})\big), \quad \forall i \in [n], \forall \ell \in [L]\backslash\{1\} \end{aligned}$$

Using the definitions of f and h, we have

$$f(W,x_i) = a^\top h_{i,L}, \quad \in \mathbb{R}, \quad \forall i \in [n]$$

We can compute the gradient of $\mathcal{L}$ in terms of $W_\ell \in \mathbb{R}^{m \times m}$, for all $\ell \geq 2$

$$\frac{\partial \mathcal{L}(W)}{\partial W_\ell} = \sum_{i=1}^{n}(f(W,x_i) - y_i) D_{i,\ell} \cdot \left(\prod_{k=\ell+1}^{L} W_k^\top D_{i,k}\right) a h_{i,\ell-1}^\top \tag{1}$$

Note that the gradient for $W_1 \in \mathbb{R}^{m \times d}$ is slightly different and cannot be written in general form. By the chain rule, the gradient of the variables in W_1 can be expressed as:

$$\frac{\partial \mathcal{L}(W)}{\partial W_1} = \sum_{i=1}^{n} (f(W, x_i) - y_i) D_{i,1} \left(\prod_{k=2}^{L} W_k^\top D_{i,k} \right) a x_i^\top$$

It is worth noting that the gradient matrix is rank n, since it's a sum of n rank-1 matrices.

Jacobian

For each layer $\ell \in [L]$ and time $t \in [T]$, we define the Jacobian matrix $J_{\ell,t} \in \mathbb{R}^{n \times m^2}$ via the following formulation:

$$J_{\ell,t} = \begin{bmatrix} \mathrm{vec}(\frac{\partial f(W(t), x_1)}{\partial W_\ell(t)})^\top \\ \mathrm{vec}(\frac{\partial f(W(t), x_2)}{\partial W_\ell(t)})^\top \\ \vdots \\ \mathrm{vec}(\frac{\partial f(W(t), x_n)}{\partial W_\ell(t)})^\top \end{bmatrix}.$$

The Gram matrix at layer ℓ and time t is then defined as $G_{\ell,t} = J_{\ell,t} J_{\ell,t}^\top \in \mathbb{R}^{n \times n}$ whose (i, j)-th entry is

$$\left\langle \frac{\partial f(W(t), x_i)}{\partial W_\ell}, \frac{\partial f(W(t), x_j)}{\partial W_\ell} \right\rangle.$$

3 Technique Overview

In this section, we give an overview of the techniques employed in this paper. In Section 3.1, we showcase our algorithm and explain various techniques being used to obtain a subquadratic cost per iteration. In Section 3.2, we give an overview of the proof to show the convergence of our algorithm.

3.1 Subquadratic Time

In this section, we study the different techniques being used to achieve the subquadratic cost per iteration.

Our algorithm can be summarized as follows: we use the shifted ReLU activation to ensure that with high probability, only a sublinear number of neurons in m are activated. We then maintain and update the gradient through a lazy, low-rank data structure. When computing the Gauss-Newton direction, we form the Jacobian matrix implicitly and approximately, then use the inexact Jacobian to approximately form Gram and invert it.

Sparsify via a Sparser Activation

We first consider the forward computation phase, in which we need to compute the matrix-vector product

$$(W_L(0) + \Delta W_L) h_{i,L-1}.$$

As we will show later, via a carefully-designed low-rank maintenance data structure, the product $\Delta W_L h_{i,L-1}$ can be performed in $o(m^2)$ time. However, this does not hold for the product $W_L(0) h_{i,L-1}$; the main reason being $W_L(0)$ is initialized as a dense matrix with 0

Algorithm 1 Informal version of our algorithm.

```
1: procedure OurAlgorithm($f, \{x_i, y_i\}_{i\in[n]}$)    ▷ Theorem 1,2
2:     /*Initialization*/
3:     Initialize $W_\ell(0), \forall \ell \in [L]$
4:     Store $h_{i,L-1}$ in memory, $\forall i \in [n]$    ▷ Takes $O(nm^2)$ time
5:     for $t = 0 \to T$ do
6:         /*Forward computation*/
7:         $v_{i,L} \leftarrow h_{i,L-1}, \forall i \in [n]$
8:         $h_{i,L} \leftarrow \phi((W_L(0) + \Delta W_L) h_{i,L-1}), \forall i \in [n]$    ▷ Takes $o(nm^2)$ time
9:         $D_{i,L} \leftarrow \mathrm{diag}(h_{i,L}), \forall i \in [n]$
10:        $f_t \leftarrow [a^\top h_{1,L}, \ldots, a^\top h_{n,L}]^\top$    ▷ Takes $O(nm)$ time
11:        /*Backward computation*/
12:        $u_{i,L} \leftarrow a^\top D_{i,L}$
13:        Form $\widetilde{J}_{L,t}$ that approximates $J_{L,t}$ using $\{u_{i,L}\}_{i=1}^n, \{v_{i,L}\}_{i=1}^n$
14:            ▷ Takes $\widetilde{O}(mn)$ time, $\widetilde{J}_{L,t} \in \mathbb{R}^{n\times s}$ where $s = \widetilde{O}(n)$
15:        Compute $g_L$ that approximates $(\widetilde{J}_{L,t}\widetilde{J}_{L,t}^\top)^{-1}(f_t - y)$
16:        Form $J_{L,t}^\top g_\ell$ via low rank factorization $\sum_{i=1}^n g_{L,i} u_{i,L} v_{i,L}^\top$
17:        Implicitly update
               $\Delta W_L \leftarrow \Delta W_L + \sum_{i=1}^n g_{L,i} u_{i,L} v_{i,L}^\top$
18:    end for
19: end procedure
```

mean Gaussian entries and does not exhibit any particular low-rank structures as ΔW_L. To address this issue, we use a shifted ReLU activation to make sure that the vector $h_{i,L-1}$ is sparse. Note that a coordinate of $h_{i,L-1}$ is nonzero if and only if the corresponding coordinate in $W_\ell h_{i,\ell-1}$ is at least b. We show that, by choosing b as $\sqrt{2\alpha \log m}$ for a parameter $\alpha \in [0, 1)$, we can ensure that the sparsity of $h_{i,\ell}$ is at most $m^{1-\alpha}$ with high probability for all $\ell \in [L-1]$. We also show that using our shifted ReLU activation, the induced distribution on the initial weight matrix $W(0)$ is a truncated Gaussian distribution, and it does not affect convergence behavior except for a slightly worse success probability.

Low-Rank Structure of the Gradient

Consider $\frac{\partial f(W,x_i)}{\partial W_L} \in \mathbb{R}^{m\times m}$, it can be written as (for simplicity, we use $h_{i,0}$ to denote x_i):

$$\frac{\partial f(W, x_i)}{\partial W_L} = h_{i,L-1} \cdot (a^\top D_{i,L})^\top.$$

This means the gradient is essentially an outer product of two vectors, and hence has rank one. This has several interesting consequences: for over-parametrized networks, the gradient is merely of rank n instead of m. The low-rank structure of the gradient can be further utilized, by representing and maintaining the *change of the weight matrix*, ΔW in an implicit fashion so that the matrix-vector product against ΔW can be quickly computed. More specifically, suppose we are given the vectors $\{h_{i,L-1}\}_{i=1}^n$ and $\{D_{i,L-1}a\}_{i=1}^n$, the gradient can be compactly expressed as $\frac{\partial f}{\partial W_L} = UV^\top$.

Gauss-Newton Method, Gram Regression and How to Sketch the Jacobian

We start by recalling the update rule of the Gauss-Newton method for our algorithm:

$$W_L(t+1) \leftarrow W_L(t) - J_{L,t}^\top (J_{L,t} J_{L,t}^\top)^{-1} (f_t - y),$$

where $J_{L,t} \in \mathbb{R}^{n \times m^2}$ is the Jacobian matrix. Note that we cannot even afford to form the Jacobian – as it's an $n \times m^2$ matrix, writing it down would already take $O(nm^2)$ time. On the other hand, $J_{L,t}$ has a structure that can be exploited, as each row of the matrix is in the form of $u_i \otimes v_i$ where $\otimes$ is the tensor product. While exactly forming these tensor products would take $O(nm^2)$ time, one can approximately compute them using tensor-based sketching techniques in time nearly linear in m. Given this much smaller, approximate Jacobian, we can perform subsequent operations much faster.

The next obstacle is to form the Gram matrix and compute the inversion. While the inversion can be computed relatively fast as the Gram matrix is $n \times n$, computing the Gram itself comes at a prohibitively high cost. We utilize our sketched Jacobian and instead perform the multiplication in time roughly $O(n^{3.3})$, which is sublinear in m. Inverting the Gram matrix then only takes $O(n^\omega)$ time, which is efficient. One could interpret our method as an inexact solver for Gram regression where the Jacobian matrices can only be approximated. Nevertheless, we show that tensor-based sketching enables us to solve this problem quickly.

3.2 Convergence Analysis

To prove that our algorithm indeed converges, we need to develop a variety of new machinery beyond the standard NTK [25] and over-parametrization [3] analysis. Specifically, we have to handle the shifted ReLU activation, which induces a different distribution than the more standard Gaussian distribution induced by the ReLU activation. Our algorithm is also based on the Gauss-Newton method, in contrast to the first-order gradient descent or stochastic gradient descent. Prior treatments of this method [15, 14] focus only on analyzing two-layer over-parametrized networks, whose analysis is much simpler and standard. Below, we provide an overview of our convergence analysis in phases.

Initialization

Let $W(0)$ be the random initialization. We first show that for any data point x_i, the initial neural network output $f(W(0), x_i) = \widetilde{O}(1)$. The analysis draws inspiration from [3]. The general idea is that given a fixed unit length vector x, multiplying it with a random Gaussian matrix W will ensure $\|Wx\|_2^2 \approx 2$ with high probability. Since W is a random Gaussian matrix, applying shifted ReLU activation gives a random vector with a truncated Gaussian distribution conditioned on a binomial random variable indicating which neurons are activated. We will end up with $\|\phi(Wx)\|_2 \approx 1$ as well as $\phi(Wx)$ being sparse. Inductively applying this idea to each layer and carefully controlling the error occurring at each layer, we can show that with good probability, $\|h_{i,L}\|_2$ is a constant. We conclude the argument by exploiting the fact that a is a Rademacher random vector so the inner product $\langle a, h_{i,L} \rangle$ concentrates around $\|h_{i,L}\|_2$, and hence with good probability, we have $f(W(0), x_i) = \widetilde{O}(1)$.

Furthermore, we show that the Gram matrix for the multi-layer over-parametrized neural network, defined as $J_{\ell,0} J_{\ell,0}^\top$, has a nontrivial minimum eigenvalue after initialization. In particular, we adapt the neural tangent kernel (NTK) for multi-layer neural networks defined by [25] into our setting by analyzing the corresponding Gaussian process with shifted ReLU activation function. Then, we can prove that with high probability, the smallest eigenvalue of the initial Gram matrix is lower bounded by the smallest eigenvalue of the neural tangent kernel matrix.

Small Perturbation

The next step is to show that if all weight matrices undergo a small perturbation from initialization (in terms of the spectral norm), then the corresponding Jacobian matrix has not changed too much. As long as the perturbation is small enough, it is possible to show that the change of the h vector (in terms of ℓ_2 norm) and the consecutive product (in terms of the spectral norm) is also small. Finally, we use the concentration property of Rademacher random variables and truncated Gaussian random variables to conclude that the change of the Jacobian has a relatively small spectral norm and Frobenius norm.

For learning the NTK of two-layer over-parametrized networks [26], the above argument is rather straightforward. However, this is no longer the case for the NTK of multi-layer networks. Our strategy is to focus on the training of the last layer, a common approach for transfer learning and spurious correlations [39], and our analysis can be viewed as a theoretical explanation of last-layer training.

Connect Everything via a Double Induction

We use a double induction argument, where we assume the perturbation of the weight matrix is small and the gap between f_t and y is at most $1/3$ of the gap between f_{t-1} and y. By carefully bounding various terms and exploiting the fact that the Jacobian matrix *always* has a relatively small spectral norm of $\widetilde{O}(\sqrt{n})$, we first show that the weights are not moving too far from the initialization, then use this fact to derive a final convergence bound for $\|f_t - y\|_2$.

4 Discussion and Future Directions

In this work, we propose and analyze a variant of the Gauss-Newton method to train multi-layer over-parametrized neural networks. Our algorithm achieves a linear convergence rate in terms of training loss and a subquadratic ($o(m^2)$) cost per training iteration. From an analytical perspective, we greatly extend the analysis of [3] to our method, coupled with the use of the equivalence between multi-layer over-parametrized networks and neural tangent kernels [25]. From an algorithmic perspective, we achieve a subquadratic cost per iteration, a significant improvement from $O(m^2)$ time per iteration due to the prohibitively large network width m. Our algorithm combines various techniques, such as training with the Gram matrix, solving the Gram regression via sketching-based preconditioning, fast tensor computation and dimensionality reduction, and low-rank decomposition of weight updates. These techniques are general and can be easily modified into a gradient descent algorithm that runs in time $o(nm^2)$ by using a pre-set scalar value as step size instead of solving a Gram regression at each iteration. Our algorithm is particularly valuable when adapted for a prolonged chain of fine-tuning, and hence, when the number of iterations is large. It can also be viewed as using data structures to speed up the iterative process, a popular trend in recent years [21, 46, 38, 32, 37, 57, 23]. Though our work is mainly theoretical, our techniques draw inspiration from practice, such as sparse activations [15, 17] and last layer training [39]. The algorithm we develop is essentially a data structure for deep, over-parametrized neural networks that fits well for large pre-trained language models as it provides significant speedups for fine-tuning the model. Below, we pose several open problems related to our result.

Nearly-Linear Time Algorithm for Multi-Layer Over-parametrized Network

The first question one might wonder is whether achieving a per iteration cost of $\widetilde{O}(m)$ is possible. In particular, can this runtime be achieved under the current best width of multi-layer over-parametrized networks ($m \geq n^4$)? We note that the major limitation in our method is the *sparsity* of the change of the diagonal matrices (ΔD) is directly related to the magnitude of the change of weights ($\|\Delta W\|$). In our analysis of convergence, we go through a careful double induction argument, which, in fact, imposes a lower bound on $\|\Delta W\|$. It seems that, in order to achieve a nearly linear runtime, one has to adapt a different analytical framework or approach the problem from a different perspective.

Maintain Change of Weights Beyond Low-Rank

In this work, we achieve speedup in the neural network training process by observing that the changes of the weights are small in each iteration. A similar phenomenon also appears in some classical optimization problems (e.g., solving linear program [21, 38] and solving semidefinite program [35, 32]), and they achieve further speedup by using lazy update and amortization techniques to compute the weight changes or using a more complicated data structure to maintain the *changes* of the weight changes. Can we adapt their techniques to neural network training? However, these dynamic maintenance methods can only *approximately* compute the weight changes. Therefore, a deeper understanding of the robustness of the training algorithms is required in order to apply these techniques to neural network training. An orthogonal direction to maintain the change is to design an initialization setup such that while we still have enough randomness to obtain provable guarantees, the matrix-vector product with the initial weight matrix can be performed faster than $O(m^2)$ by sparsifying the Gaussian matrix as in [24] or imposing extra structural assumption such as using circulant Gaussian [55, 51, 40].

Extension to Other Activations

In this paper, we consider the shifted ReLU activation and design our algorithm and analysis around its properties. Is it possible to generalize our algorithm and analysis to various other activation functions, such as sigmoid, tanh, leaky ReLU, or GeLU? If one chooses a smooth activation, can we get a better result in terms of convergence rate? Can we leverage this structure to design faster algorithms?

Network Architecture Beyond Feedforward

Finally, the network architecture considered in this paper is the standard feedforward network. Is it possible to extend our analysis and algorithm to other architectures, such as recurrent neural networks (RNN) or transformers? For RNN, the weight matrices for each layer are the same. Hence, it is trickier to analyze the training dynamics on such networks. Though the convergence of the first-order method on over-parametrized multi-layer RNN has been established, it is unclear whether such analysis can be extended to our method. More generally, can we improve the efficiency of the training process of neural networks even beyond the NTK regime, e.g., in the feature learning regime? We believe new techniques are needed to achieve theoretical time complexity improvements as well as the convergence guarantee.

References

1 Naman Agarwal, Brian Bullins, and Elad Hazan. Second-order stochastic optimization for machine learning in linear time. *The Journal of Machine Learning Research*, 18(1):4148–4187, 2017.

2 Thomas D. Ahle, Michael Kapralov, Jakob Bæk Tejs Knudsen, Rasmus Pagh, Ameya Velingker, David P. Woodruff, and Amir Zandieh. Oblivious sketching of high-degree polynomial kernels. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 141–160, 2020.

3 Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *ICML*, 2019.

4 Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. On the convergence rate of training recurrent neural networks. In *NeurIPS*, 2019.

5 Josh Alman, Jiehao Liang, Zhao Song, Ruizhe Zhang, and Danyang Zhuo. Bypass exponential time preprocessing: Fast neural network training via weight-data correlation preprocessing. In *Advances in Neural Information Processing Systems*, NeurIPS'23, 2023.

6 Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *ICML*. arXiv preprint, 2019. `arXiv:901.08584`.

7 Haim Avron, Huy L. Nguyen, and David P. Woodruff. Subspace embeddings for the polynomial kernel. In *NeurIPS*, 2014.

8 Kyriakos Axiotis, Aleksander Madry, and Adrian Vladu. Faster sparse minimum cost flow by electrical flow localization. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, 2021.

9 Alberto Bernacchia, Mate Lengyel, and Guillaume Hennequin. Exact natural gradient in deep linear networks and its application to the nonlinear case. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2018.

10 Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical Gauss-Newton optimisation for deep learning. In *Proceedings of the 34th International Conference on Machine Learning*, pages 557–565, 2017.

11 Christos Boutsidis, David P. Woodruff, and Peilin Zhong. Optimal principal component analysis in distributed and streaming models. In *STOC'16—Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, 2016.

12 Jan van den Brand. A deterministic linear program solver in current matrix multiplication time. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 259–278. SIAM, 2020.

13 Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. Solving tall dense linear programs in nearly linear time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, pages 775–788, 2020.

14 Jan van den Brand, Binghui Peng, Zhao Song, and Omri Weinstein. Training (overparametrized) neural networks in near-linear time. In *ITCS*, 2021. `arXiv:2006.11648`.

15 Tianle Cai, Ruiqi Gao, Jikai Hou, Siyu Chen, Dong Wang, Di He, Zhihua Zhang, and Liwei Wang. Gram-gauss-newton method: Learning overparameterized neural networks for regression problems. *arXiv preprint*, 2019. `arXiv:1905.11675`.

16 Yuan Cao and Quanquan Gu. Generalization bounds of stochastic gradient descent for wide and deep neural networks. In *NeurIPS*, pages 10835–10845, 2019.

17 Beidi Chen, Zichang Liu, Binghui Peng, Zhaozhuo Xu, Jonathan Lingjie Li, Tri Dao, Zhao Song, Anshumali Shrivastava, and Christopher Re. {MONGOOSE}: A learnable {lsh} framework for efficient neural network training. In *Proceedings of the Nineth International Conference on Learning Representations (ICLR'2021)*, 2021.

18 Beidi Chen, Tharun Medini, Sameh Gobriel James Farwell, Charlie Tai, and Anshumali Shrivastava. SLIDE : In defense of smart algorithms over hardware acceleration for large-scale deep learning systems. In *MLSys'2020*, 2020.

19 Zixiang Chen, Yuan Cao, Difan Zou, and Quanquan Gu. How much over-parameterization is sufficient to learn deep ReLU networks? In *International Conference on Learning Representations (ICLR)*, 2021.

20 Kenneth L. Clarkson and David P. Woodruff. Low rank approximation and regression in input sparsity time. In *Symposium on Theory of Computing Conference (STOC)*, pages 81–90, 2013.

21 Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. In *STOC*, 2019.

22 Samuel I Daitch and Daniel A Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the fortieth annual ACM symposium on Theory of computing (STOC)*, pages 451–460, 2008.

23 Yichuan Deng, Zhao Song, Omri Weinstein, and Ruizhe Zhang. Fast distance oracles for any symmetric norm. In *NeurIPS*, 2022.

24 Michał Dereziński, Jonathan Lacotte, Mert Pilanci, and Michael W. Mahoney. Newton-less: Sparsification without trade-offs for the sketched newton update, 2021. `arXiv:2107.07480`.

25 Simon S Du, Jason D Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. In *International Conference on Machine Learning (ICML)*, 2019.

26 Simon S Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. In *ICLR*, 2019.

27 Alessandro Epasto, Mohammad Mahdian, Vahab Mirrokni, and Peilin Zhong. Improved sliding window algorithms for clustering and coverage via bucketing-based sketches. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2022.

28 Yeqi Gao, Lianke Qin, Zhao Song, and Yitan Wang. A sublinear adversarial training algorithm. *arXiv preprint*, 2022. `arXiv:2208.05395`.

29 Yuzhou Gu and Zhao Song. A faster small treewidth sdp solver. *arXiv preprint*, 2022. `arXiv:2211.06033`.

30 Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL: `https://openreview.net/forum?id=nZeVKeeFYf9`.

31 Hang Hu, Zhao Song, Omri Weinstein, and Danyang Zhuo. Training overparametrized neural networks in sublinear time. *arXiv preprint*, 2022. `arXiv:2208.04508`.

32 Baihe Huang, Shunhua Jiang, Zhao Song, Runzhou Tao, and Ruizhe Zhang. Solving sdp faster: A robust ipm framework and efficient implementation. In *FOCS*, 2022.

33 Baihe Huang, Xiaoxiao Li, Zhao Song, and Xin Yang. Fl-ntk: A neural tangent kernel-based framework for federated learning convergence analysis. In *ICML*, 2021.

34 Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: convergence and generalization in neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems (NeurIPS)*, pages 8580–8589, 2018.

35 Haotian Jiang, Tarun Kathuria, Yin Tat Lee, Swati Padmanabhan, and Zhao Song. A faster interior point method for semidefinite programming. In *FOCS*, 2020.

36 Haotian Jiang, Yin Tat Lee, Zhao Song, and Sam Chiu-wai Wong. An improved cutting plane method for convex optimization, convex-concave games and its applications. In *STOC*, 2020.

37 Shunhua Jiang, Bento Natura, and Omri Weinstein. A faster interior-point method for sum-of-squares optimization. In *ICALP*, 2022.

38 Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. Faster dynamic matrix inverse for faster lps. In *STOC*, 2021.

39 Polina Kirichenko, Pavel Izmailov, and Andrew Gordon Wilson. Last layer re-training is sufficient for robustness to spurious correlations, 2022.

40 Felix Krahmer, Shahar Mendelson, and Holger Rauhut. Suprema of chaos processes and the restricted isometry property. *Communications on Pure and Applied Mathematics*, 67(11):1877–1904, 2014.

41 Francois Le Gall. Faster rectangular matrix multiplication by combination loss analysis. In *SODA*, 2024.

42 Jaehoon Lee, Lechao Xiao, Samuel S. Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. 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 8570–8581, 2019.

43 Jason D Lee, Ruoqi Shen, Zhao Song, Mengdi Wang, and Zheng Yu. Generalized leverage score sampling for neural networks. In *NeurIPS*, 2020.

44 Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in õ(sqrt(rank)) iterations and faster algorithms for maximum flow. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 424–433, 2014.

45 Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 1049–1065. IEEE, 2015.

46 Yin Tat Lee, Zhao Song, and Qiuyi Zhang. Solving empirical risk minimization in the current matrix multiplication time. In *Conference on Learning Theory (COLT)*, pages 2140–2157. PMLR, 2019.

47 Yuanzhi Li and Yingyu Liang. Learning overparameterized neural networks via stochastic gradient descent on structured data. In *NeurIPS*, 2018.

48 S. Cliff Liu, Zhao Song, Hengjie Zhang, Lichen Zhang, and Tianyi Zhou. Space-efficient interior point method, with applications to linear programming and maximum weight bipartite matching. In *ICALP*, 2023.

49 James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 2408–2417. JMLR.org, 2015.

50 Iman Mirzadeh, Keivan Alizadeh, Sachin Mehta, Carlo C Del Mundo, Oncel Tuzel, Golnoosh Samei, Mohammad Rastegari, and Mehrdad Farajtabar. Relu strikes back: Exploiting activation sparsity in large language models, 2023. `arXiv:2310.04564`.

51 Jelani Nelson and Huy L Nguyen. Sparsity lower bounds for dimensionality reducing maps. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 101–110, 2013.

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

53 Mert Pilanci and Martin J. Wainwright. Newton sketch: A near linear-time optimization algorithm with linear-quadratic convergence. *SIAM J. Optim.*, 27:205–245, 2017.

54 Lianke Qin, Zhao Song, Lichen Zhang, and Danyang Zhuo. An online and unified algorithm for projection matrix vector multiplication with application to empirical risk minimization. In *AISTATS*, 2023.

55 Holger Rauhut, Justin Romberg, and Joel A Tropp. Restricted isometries for partial random circulant matrices. *Applied and Computational Harmonic Analysis*, 32(2):242–254, 2012.

56 Tamas Sarlos. Improved approximation algorithms for large matrices via random projections. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 143–152. IEEE, 2006.

57 Zhao Song, Baocheng Sun, Omri Weinstein, and Ruizhe Zhang. Sparse fourier transform over lattices: A unified approach to signal reconstruction. *arXiv preprint*, 2022. `arXiv:2205.00658`.

58 Zhao Song, Ruosong Wang, Lin Yang, Hongyang Zhang, and Peilin Zhong. Efficient symmetric norm regression via linear sketching. *Advances in Neural Information Processing Systems*, 32, 2019.

59 Zhao Song, David P. Woodruff, Zheng Yu, and Lichen Zhang. Fast sketching of polynomial kernels of polynomial degree. In *ICML*, 2021.

60 Zhao Song, David P Woodruff, and Peilin Zhong. Low rank approximation with entrywise l1-norm error. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 688–701, 2017.

61 Zhao Song, David P Woodruff, and Peilin Zhong. Relative error tensor low rank approximation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2772–2789. SIAM, 2019.

62 Zhao Song, Shuo Yang, and Ruizhe Zhang. Does preprocessing help training over-parameterized neural networks? In *Thirty-Fifth Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

63 Zhao Song and Xin Yang. Quadratic suffices for over-parametrization via matrix chernoff bound. *arXiv preprint*, 2019. `arXiv:1906.03593`.

64 Zhao Song and Zheng Yu. Oblivious sketching-based central path method for linear programming. In *International Conference on Machine Learning (ICML)*, pages 9835–9847. PMLR, 2021.

65 Pravin M Vaidya. Speeding-up linear programming using fast matrix multiplication. In *30th Annual Symposium on Foundations of Computer Science*, pages 332–337. IEEE, 1989.

66 Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In *SODA*, 2024.

67 David P. Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends in Theoretical Computer Science*, 10(1–2):1–157, 2014.

68 David P Woodruff and Amir Zandieh. Near input sparsity time kernel embeddings via adaptive sampling. In *ICML*, 2020.

69 Hongru Yang, Ziyu Jiang, Ruizhe Zhang, Zhangyang Wang, and Yingbin Liang. Convergence and generalization of wide neural networks with large bias, 2023. `arXiv:2301.00327`.

70 Zhewei Yao, Amir Gholami, Sheng Shen, Mustafa Mustafa, Kurt Keutzer, and Michael Mahoney. Adahessian: An adaptive second order optimizer for machine learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(12):10665–10673, May 2021.

71 Guanghao Ye. Fast algorithm for solving structured convex programs. *University of Washington Undergraduate Thesis*, 2020.

72 Guodong Zhang, James Martens, and Roger B Grosse. Fast convergence of natural gradient descent for over-parameterized neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

73 Difan Zou, Yuan Cao, Dongruo Zhou, and Quanquan Gu. Gradient descent optimizes over-parameterized deep relu networks. In *Machine Learning*, 2020.

74 Difan Zou and Quanquan Gu. An improved analysis of training over-parameterized deep neural networks. In *NeurIPS*, pages 2053–2062, 2019.