Fast Second-order Method for Neural Networks under Small Treewidth Setting

Xiaoyu Li
Department of Computer Science
Stevens Institute of Technology
Hoboken, NJ
xli216@stevens.edu

Jiangxuan Long
School of Software Engineering
South China University of Technology
Guangzhou, China
lungchianghsuan@gmail.com

Zhao Song
The Simons Institute for the Theory of Computing
University of California, Berkeley
Berkeley, CA
magic.linuxkde@gmail.com

Tianyi Zhou
Department of Computer Science
University of Southern California
Los Angeles, CA
tzhou029@usc.edu

***Abstract*—Training neural networks is a fundamental problem in theoretical machine learning. Second-order methods are rarely used in practice due to their high computational cost, even they converge much faster than first-order methods. The state-of-the-art result for the second-order method to train an over-parameterized neural network can run in $O(\log(1/\epsilon))$ iterations and each iteration has $O(mnd + n^3)$ running time, where n represent the number of data points, d is the dimension of the feature space of each data and m is the width of the neural network [Brand, Peng, Song and Weinstein ITCS 2021]. In this work, we further improve the convergence rate and the running time for each iteration to $O(\log\log(1/\epsilon))$ and $O(md\tau^2)$ respectively, where $m = \Omega(n^4)$ and τ is the treewidth of the data matrix and usually very small. Our algorithm has a quadratic convergence rate, so it can be regarded as a truly second-order algorithm. To the best of our knowledge, our algorithm is the first to achieve a quadratic convergence rate for neural network training, with a per-iteration running time of $O(md\tau^2)$.**

***Index Terms*—Neural Network Training, Newton's Method, Cholesky Decomposition**

I. Introduction

Despite the immense success of deep neural networks (DNN) across a multitude of applications, their non-convex nature presents significant challenges in theoretically analyzing their convergence behavior. First-order methods, notably Stochastic Gradient Descent (SGD), have emerged as the predominant choice for training these networks. These methods are particularly favored due to their computational efficiency, stemming from their reliance solely on gradient calculations. A specific merit of SGD lies in its approach to gradient computation. Instead of processing the entirety of training data at each iteration, SGD efficiently computes the gradient using just a mini-batch, offering a blend of practical importance and computational efficiency. This focus on first-order methods and over-parameterized neural networks, has been the subject of numerous recent works [1], [2], [3], [4], [5], given that stochastic gradient descent remains the most widely used training technique for DNNs.

Ref	Method	#Iters	Cost/iter
[6]	Gradient descent	$n^2\log(1/\epsilon)$	mn
[7]	Gradient descent	$n^2\log(1/\epsilon)$	mn
[8]	Adaptive gradient descent	$n\log(1/\epsilon))$	mn
[9]	Gram-Gaussian-Newton (GGN)	$\log\log(1/\epsilon)$	mn^2
[9]	Batch-GGN	$n^2\log(1/\epsilon)$	m
[10]	Natural gradient descent	$\log(1/\epsilon)$	mn^2
[11]	Gram-Gaussian-Newton	$\log(1/\epsilon)$	mn
Ours	Theorem I.1	$\log\log(1/\epsilon)$	$m\tau^2$

TABLE I: Comparison to prior algorithms for training two-layer neural networks. We denote n as the number of input data points and denote d as the dimension of the feature space. We require that the accuracy of the training loss should be controlled within ϵ. We igonre the big-O notation, assume $d = O(1)$ and omit $\mathrm{poly}(\log n, 1/\lambda)$ terms for clarity. We note that the result from [9] only applies to smooth activation gates and not to ReLU networks. We ignore SGD algorithms because they have a slower convergence rate than gradient descent methods and a stronger assumption on the width of the neural network for convergence [12], [13], [14].

However, most first-order methods suffer from a linear convergence rate, that is, in order to reduce the training error below ϵ, it requires $O(\log(1/\epsilon))$ many iterations. Addressing this need for improvement, there has been a surge of accelerated methods to boost the convergence rate [10], [9], [4]. Of these, second-order methods warrant special attention. Known for their efficacy in the convex setting, these methods boast a quadratic convergence rate, specifically $O(\log\log(1/\epsilon))$ [15].

One critical issue for second-order methods is that the computation requires information on the second-order moment, or the Hessian matrix of the loss function, which could be time-consuming. In a ground-breaking study, Brand, Peng, Song, and Weinstein tackled this challenge head-on [11]. They introduced a second-order algorithm tailored for an over-parameterized ReLU-activated neural network. Their innovative method not

only outpaced the leading first-order methods but achieved it with a runtime of $O(mn)$ per iteration, where m represents the width of the network and n the size of the training data set. Despite this accomplishment, a lingering question remains: can the conventional second-order methods, which typically achieve $O(\log\log(1/\epsilon))$ convergence rates, be further enhanced?

In this work, we make the first step towards answering this question. We show that if the training data admits certain good properties, then the quadratic convergence rate is plausible. Especially, we consider the *treewidth* of the graph associated with the training data. Roughly speaking, the treewidth measures the sparsity of a matrix from the tree structure generated by the matrix. A detailed introduction of treewidth can be found in Section II-B.

In practical settings, treewidth typically has a small value. Let n denote the number of constraints. A study by [16] analyzing the MATPOWER data set for power system analysis identified the largest problem size as $(n = 12659, m = 20467)$, with a maximum treewidth of $\tau = 35$. These findings suggest that treewidth-efficient algorithms are more effective for practical applications than general-purpose matrix algorithms. Small treewidth is also a common assumption for accelerating optimization problems including linear system solvers [17], [18] and semidefinite programs [19].

Following [9], [11], we only consider two-layer neural networks (not necessarily overparameterized) which is also the only interesting scenario for constant training error [11].

We state our main result as follows

Theorem I.1 (Our result, informal version of Theorem IV.5)**.** *Given a two-layer neural network with m neurons and assume data matrix with treewidth τ, then to train the accuracy of neural network to ϵ, there is an algorithm that runs in $O(\log\log(1/\epsilon))$ iterations and each iteration takes $O(md\tau^2)$ time.*

A. Related work

a) Second Order Methods: In recent years, second-order methods have been increasingly applied to neural network optimization. For instance, [20] introduced Hessian-free optimization, which leverages conjugate gradient methods to approximate the Newton update. Similarly, [21] proposed a Krylov subspace descent approach for directly approximating the Newton update. For natural gradient methods, which provide a steepest descent in the space of network outputs rather than parameters, [22] showed a connection between second-order optimization and the Fisher information matrix. Building on this, [23] introduced K-FAC, a method that utilizes a block-diagonal approximation of the Fisher matrix for efficient natural gradient updates, with [24] extending this approach to convolutional neural networks. More recently, [25] combined natural gradient methods with trust region techniques to enhance stability and performance.

Despite these advancements, second-order neural network optimization remains an active research area, with challenges such as scaling Hessian approximations, addressing the complexities of non-convex optimization landscapes, and automating hyperparameter selection still unresolved [26], [27], [28], [29]. Our work builds on these prior methods while introducing novel techniques to address these ongoing challenges.

b) Over-Parametrized Neural Networks and NTK: In the deep learning community, significant attention has been devoted to understanding the geometry and convergence behavior of optimization algorithms in over-parametrized neural networks [12], [6], [13], [14], [11], [30], [31], [32], [33], [34], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44].

The seminal work of [45] initiated the study of the Neural Tangent Kernel (NTK), a powerful analytical tool for proving the convergence of training in over-parametrized networks, which employs the first-order Taylor expansion to examine over-parametrized neural networks from their initial states. By over-parametrizing the network width to a relatively large value, i.e. $m \geq \Omega(n^4)$, the training dynamics of a neural network can be shown to closely resemble those of the NTK [46]. Moreover, sketching is a power tool to speed up the training of neural networks [47], [46], [48], [49], [50].

The NTK technique has found widespread application across various domains, including preprocessing analysis [34], [51], [52], federated learning [53], differential privacy [54], [55], attention models [56], [57], [58], [59], [60], [61], [62], graph neural networks [63], [64], [65], [66], [67], diffusion models [68], [69], [70] and LoRA adaptation for large language models [71], [72].

B. Technique Overview

a) Gram-Gauss-Newton Method: The Gram-Gauss-Newton Method (GGN) is an extension of Newton's method for training deep neural networks for regression problems with square loss, which converges much faster and has better performance than SGD. It update the weight of the neural networks by: $W_{t+1} = W_t - J_t^\top G_t^{-1}(f_t - y)$ where the Gram matrix $G_t = J_t J_t^\top$, and Jacobian matrix $J_t \in \mathbb{R}^{n \times md}$, whose i-th row contains the gradient of network gates.

The most time-consuming computation above is multiplying the inverse of the Gram matrix $G_t = J_t J_t^\top$ with $J_t^\top$ in each iteration, which takes $O(mdn^2)$ time. In our algorithm, we use the Cholesky decomposition to speed up the computation, which only takes $O(md\tau^2)$, where τ is the treewidth of the data matrix.

b) Cholesky Decomposition: Given that the data matrix X has small treewidth τ, by using Cholesky decomposition, we decomposed $XX^\top$ by $XX^\top = L_X L_X^\top$. Its Cholesky factor L_X satisfies the property that each column is τ-sparse by Lemma II.3. From Lemma II.9, we know that the Gram matrix G_t is as sparse as or even sparser than matrix $XX^\top$. Hence, the Cholesky factor L_G for $G_t = L_G L_G^\top$ is as sparse as or sparser than L_X. By leveraging the small treewidth property, we compute $J_t J_t^\top$ is $O(md\tau^2)$ time (see detail in Lemma II.5). As the Choleksy decomposition factor L is τ-sparse in columns, the time to compute $J_t^\top (L_G L_G^\top)^{-1}$ is $O(md\tau)$ (see

We provide an example of tree decomposition as defined in Definition II.1.

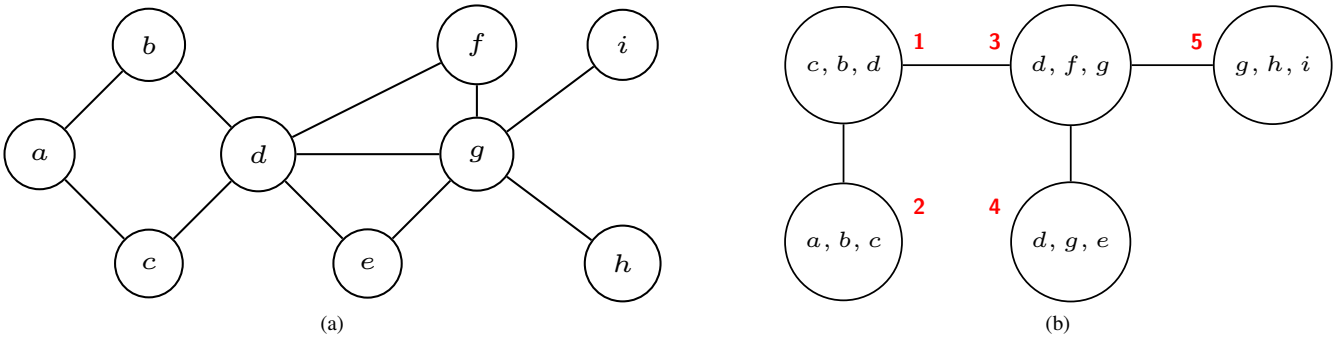


Fig. 1: (a) A graph $G(V, E)$ (b) The corresponding tree decomposition. Here, we illustrate how the three rules in Definition II.2 are satisfied. First, the union of the vertices in all bags are nodes $a, \cdots, i$, which is the same as $V(G)$. Second, for every edge $u, v \in V(G)$, we can find at least one bag containing both u and v. Take edge (b, d) in graph G as an example, bag 1 contains both b and d. For edge (d, g), bag 3 and 4 both contain vertices d and g. For edge (i, g), bag 5 contains vertices i and g. Third, for each node in G, the bags containing it is a connected subgraph of T. Taking node g as an example, it is contained by bag $3, 4, 5$ in tree T, which is a connected subgraph of T. Similarly, for node c, it is contained by bag $1, 2$, which is a also a connected subgraph of T.

detail in Lemma II.6). Hence, the total time for multiplying the inverse of the Gram matrix $G_t = JJ_t^\top$ with $J_t^\top$ is

$$O(md\tau^2) + O(md\tau) = O(md\tau^2),$$

as desired.

c) Roadmap: We first introduce all required preliminary in Section II. Then, we present a toy example to show how small treewidth assumption help speed up convex optimization algorithm in Section III. We provide our fast neural network training algorithm and the corresponding theorem in Section IV. We conclude our contribution in Section V.

II. Preliminary

We first introduce some basic notation in Section II-A, Then, we present background knowledge of treewidth in Section II-B. Finally, we introduce our setup of neural networks in Section II-C.

A. Notation

The normal distribution with mean μ and variance σ^2 is denoted by $\mathcal{N}(\mu, \sigma^2)$. For a random variable X, we use $\mathbb{E}[X]$ to denote its expectation. We use $\Pr[]$ to denote the probability. The inner product between vectors x and y in $\mathbb{R}^d$ is represented as $\langle x, y \rangle$, which equals $\sum_{i=1}^d x_i y_i$. The set of integers from 1 to n is represented as $[n]$. The ℓ_2 norm of a vector x is denoted as $\|x\|_2$, which is defined as $(\sum_{i=1}^n x_i^2)^{1/2}$. A vector with at most τ non-zero entries is said to be τ-sparse. For a matrix $A \in \mathbb{R}^{d \times d}$, we say $A \succeq 0$ (positive semi-definite) if for all $x \in \mathbb{R}^d$ we have $x^\top A x \geq 0$. The transpose of matrix A is represented as $A^\top$. The number of non-zero entries in matrix A is denoted as $\mathrm{nnz}(A)$. The inverse of a square and full rank matrix A is represented as A^{-1}. The notation $\widetilde{O}(f)$ is used to denote a function $f \cdot \mathrm{poly}(\log f)$. For two scalars a, b, We define $\mathbf{1}_{a \geq b} = 1$ if $a \geq b$ and $\mathbf{1}_{a \geq b} = 0$ otherwise. For scalars a, b, c, d, we also define $\mathbf{1}_{a \geq b, c \geq d} = \mathbf{1}_{a \geq b} \cdot \mathbf{1}_{c \geq d}$.

B. Treewidth

We first introduce the definition of treewidth which is a well-studied graph parameter. Small treewidth graphs have small vertex separators and will be easily converted to a tree structure. Treewidth is defined by the tree-decomposition of a graph as below. An example of tree decomposition is given in Figure 1.

Definition II.1. *A tree-decomposition is a mapping from graphs into trees. For a graph G, the tree-decomposition is defined to be a pair (M, T), where T is a tree, and $M : V(T) \to 2^{V(G)}$ is a family of subsets of $V(G)$ called bags labelling the vertices of T, satisfies that*

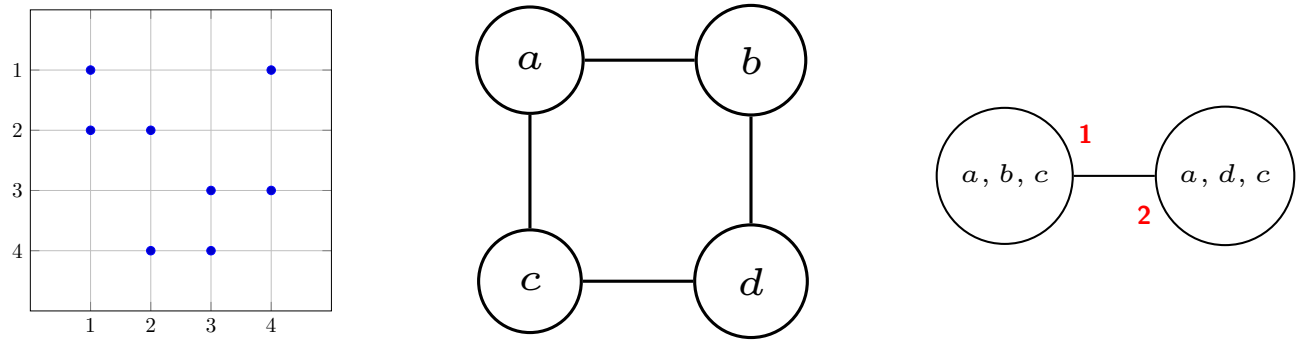
- $\cup_{t \in V(T)} M(t) = V(G)$
- *for each $e = (u, v) \in V(G)$, there is a node $t \in V(T)$ such that $u, v \in M(t)$*
- *for each $v \in V(G)$, the nodes $t \in V(T)$ with $v \in M(t)$ induces a connected subgraph of T*

where $V(\cdot)$ denote the vertex set of a graph. The width of a tree-decomposition (M, T) is $\max\{|M(t)| - 1 : t \in T\}$. The treewidth of G is the minimum width over all tree-decompositions of G.

Next, we show how to define the treewidth of a matrix by converting the matrix into a graph.

Definition II.2 (Treewidth τ)**.** *Let A be a matrix in $\mathbb{R}^{d \times n}$, we associate it with a graph $G = (V, E)$ as follows: The vertex set is $[n]$ indicating each column; An edge $(i, j) \in E$ if and only if there exists $k \in [d]$ such that $A_{k,i} \neq 0, A_{k,j} \neq 0$. Then, the treewidth of the matrix A is defined to be the treewidth of the graph G.*

In Figure 2, we give an example of the a small-treewidth matrix D. Under the small treewidth condition, many classic algorithms can be accelerated due to the sparsity pattern.



(a) (b) (c)

Fig. 2: Example of small treewidth matrix and it's corresponding graph and tree decomposition. (a) A 2-sparse 4 by 4 data matrix D, where the blue dots represent the non-zero elements. (b) The corresponding graph G of data matrix D. By Definition II.2, The corresponding graph of matrix D will be a simple cycle of four nodes a, b, c, d. Here we assume each node in G represents one column in D. (c) The corresponding tree decomposition will only have two bags: $1 : \{a, b, c\}$ and $2 : \{b, c, d\}$. By Definition II.1, we know that the treewidth of matrix D is 2, which satisfies the sparsity of matrix D.

We state the lemma for fast Cholesky factorization in small treewidth case.

Lemma II.3 ([17], [18]). *For any positive diagonal matrix $H \in \mathbb{R}^{n\times n}$, for any matrix $A \in \mathbb{R}^{d\times n}$ with treewidth τ (in particular, every row of A can be regarded as τ-sparse), we can compute the Cholesky factorization $AHA^\top = LL^\top \in \mathbb{R}^{d\times d}$ in $O(d\tau^2)$ time, where $L \in \mathbb{R}^{d\times d}$ is a lower-triangular matrix with real positive entries. Every column of L and L^{-1} is τ-sparse.*

Remark II.4. *The upper-triangular version of Cholesky factorization $AHA^\top = UU^\top$ can also be computed in $O(d\tau^2)$ time, where $U \in \mathbb{R}^{d\times d}$ is a upper-triangular matrix with real positive entries.*

Lemma II.5 ([73]). *For a positive definite matrix $M \in \mathbb{R}^{d\times d}$, we can compute its Cholesky factorization $M = LL^\top$ in time $\Theta(\sum_{j=1}^d |\mathcal{L}_j|^2)$, where $|\mathcal{L}_j|$ denotes the number of nonzero entries in the j-th column of L.*

Lemma II.6 ([74]). *For any positive diagonal matrix $H \in \mathbb{R}^{n\times n}$, given matrix $A \in \mathbb{R}^{d\times n}$ with treewidth τ and the Cholesky factorization $AHA^\top = LL^\top \in \mathbb{R}^{d\times d}$, for any vector $x \in \mathbb{R}^d$, we can compute $(LL^\top)^{-1}x$ in $O(d\tau^2)$ and compute $L^\top x$ in $O(d\tau)$.*

C. Setup of Neural Network

a) Notation: Let n, d represent the number of data points and the dimension of feature space respectively. Let $\{x_1, \cdots, x_n\}$ denote the data sets and let m denote the width of the neural network. In the following sections, we consider two-layers neural networks with ReLU activated function and m neurons and assume that all the data points are normalized (i.e. $\|x_i\|_2 = 1, i = 1, \cdots, n$).

$$f(W, x, a) = \frac{1}{\sqrt{m}} \sum_{r=1}^m a_r \phi(w_r^\top x),$$

where $x \in \mathbb{R}^d$ is the input, $w_1, \cdots, w_m \in \mathbb{R}^d$ are weight vectors in the first layer, $a_1, \cdots, a_m \in \{\pm 1\}$ are weights in the second layer, ϕ is ReLU function $\phi(x) = \max\{x, 0\}$. The above settings are natural in deep learning theory [12], [6], [13], [75], [7].

Let $r \in [m]$, we have

$$\frac{\partial f(w, x, a)}{\partial w_r} = \frac{1}{\sqrt{m}} a_r x \mathbf{1}_{w_r^\top x \geq 0}. \tag{1}$$

Given labels $Y = y_1, \cdots, y_n \subset \mathbb{R}$ and corresponding data sets $X = \{x_1, , \cdots, x_n\} \subset \mathbb{R}^d$, the quadratic loss function $\mathcal{L}$ is defined as $\mathcal{L}(W) = \frac{1}{2}\sum_{i=1}^n (y_i - f(W, x_i, a))^2$.

We obtain the gradient of $\mathcal{L}$ with respect to w_r

$$\frac{\partial \mathcal{L}(W)}{\partial w_r} = \frac{1}{\sqrt{m}} \sum_{i=1}^n (f(W, x_i, a) - y_i) a_r x_i \mathbf{1}_{w_r^\top x_i \geq 0} \tag{2}$$

We define the prediction function $f_t : \mathbb{R}^{d\times n} \to \mathbb{R}^n$ at time t as follow

$$f_t = \begin{bmatrix} \frac{1}{\sqrt{m}} \sum_{r=1}^m a_r \cdot \phi(\langle w_r(t), x_1 \rangle) \\ \frac{1}{\sqrt{m}} \sum_{r=1}^m a_r \cdot \phi(\langle w_r(t), x_2 \rangle) \\ \vdots \\ \frac{1}{\sqrt{m}} \sum_{r=1}^m a_r \cdot \phi(\langle w_r(t), x_n \rangle) \end{bmatrix}$$

where $W_t = [w_1(t)^\top, w_2(t)^\top, \cdots, w_m(t)^\top]^\top \in \mathbb{R}^{md}$ and $X = [x_1, x_2, \cdots, x_n] \in \mathbb{R}^{d\times n}$. For each time t, the Jacobian matrix $J \in \mathbb{R}^{n\times md}$ is defined via the following formulation:

$$\frac{1}{\sqrt{m}} \cdot \begin{bmatrix} a_1 x_1^\top \mathbf{1}_{\langle w_1(t), x_1 \rangle \geq 0} & \cdots & a_m x_1^\top \mathbf{1}_{\langle w_m(t), x_1 \rangle \geq 0} \\ a_1 x_2^\top \mathbf{1}_{\langle w_1(t), x_2 \rangle \geq 0} & \cdots & a_m x_2^\top \mathbf{1}_{\langle w_m(t), x_2 \rangle \geq 0} \\ \vdots & \ddots & \vdots \\ a_1 x_n^\top \mathbf{1}_{\langle w_1(t), x_n \rangle \geq 0} & \cdots & a_m x_n^\top \mathbf{1}_{\langle w_m(t), x_n \rangle \geq 0} \end{bmatrix}$$

We denote the Gram matrix as $G_t = J_t J_t^\top$, whose (i, j)-th entry is $\langle \frac{f\langle W_t, x_i \rangle}{\partial W}, \frac{f\langle W_t, x_j \rangle}{\partial W} \rangle$. From the crucial observation of

[45], [6], we know that the asymptotic of the Gram matrix is a positive semidefinite kernel matrix $K \in R^{n \times n}$, where

$$K(x_i, x_j) = \mathop{\mathbb{E}}_{w \in \mathcal{N}(0,1)}[x_i^\top x_j \mathbf{1}_{\langle w, x_i \rangle \geq 0, \langle w, x_j \rangle \geq 0}]. \quad (3)$$

Next, we present our assumptions on the training setting.

Assumption II.7. *We assume the least eigenvalue λ of the kernel matrix K defined in Eq. (3) satisfies $\lambda > 0$ and denote its least eigenvalue as*

$$\lambda_0 := \lambda_{\min}(K) > 0$$

We note that the positive definiteness assumption on kernel matrix is standard in literature.

Assumption II.8. *We assume that data matrix $X \in \mathbb{R}^{d \times n}$ has small treewidth τ.*

By above assumption, we claim that the Gram matrix is as sparse as or even sparser than matrix $XX^\top$.

Lemma II.9 (Sparsity of Gram matrix)**.** *The support of $J_t J_t^\top \in \mathbb{R}^{n \times n}$ is a subset of the support of $X^\top X \in \mathbb{R}^{n \times n}$.*

Proof. For $G_t = J_t J_t^\top$, we have

$$G_t(i,j) = \frac{1}{\sqrt{m}} \sum_{k=1}^m x_i^\top x_j \mathbf{1}_{\langle w_k(t), x_i \rangle \geq 0, \langle w_k(t), x_j \rangle \geq 0}.$$

When $x_i^\top x_j = 0$, $G_t(i,j)$ must be 0. Thus, the support of G_t is a subset of the support of $XX^\top$. □

D. Tools from previous work

Lemma II.10 (Least Eigenvalue in the Optimization Scope, [9])**.** *For $W \in B(R)$, suppose the events in Lemma IV.2 and IV.3 holds, there is a constant C' so that for $M \geq \frac{C' n^2 R^2}{\lambda_0^2}$, we have $\|G_W - G_{W_0}\|_2 \leq \frac{\lambda_0}{4}$ and thus combined with Lemma IV.2, we know that G_W remains invertible when $W \in B(R)$ and satisfies $\|G_W^{-1}\|_2 \leq \frac{2}{\lambda_0}$.*

III. Toy Example

Before stepping into our results for fast neural networks training, we first show an application of small treewidth assumption for convex optimization problem, which is easier to solve. We show how to exploit the sparse structure of Hessian matrix and apply our technique to speed up convex optimization algorithm. Following [3], we assume f is the γ-strongly convex, β-smooth and its Hessian matrix $\nabla^2 f(x)$ is L Lipschitz continuous and consider the problem $\min_x f(x)$.

Definition III.1 (γ-strongly convex)**.** *The function f is γ-strongly convex if*

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\gamma}{2}\|y - x\|_2^2$$

Definition III.2 (β-smooth)**.** *The function f is β-smooth if*

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\beta}{2}\|y - x\|_2^2$$

Definition III.3 (L Lipschitz continuous Hessian)**.** *The Hessian matrix of function is L Lipschitz continuous is*

$$\|\nabla^2 f(x) - \nabla^2 f(y)\| \leq L\|x - y\|_2.$$

Let $f : \mathbb{R}^d \to \mathbb{R}$. For example, assuming that the objective function $f(x)$ can be written in the form of $f(x) = g(Ax)$ where $A \in \mathbb{R}^{n \times d}$ and the function $g : \mathbb{R}^n \to \mathbb{R}$ has the form of

$$g(Ax) = \sum_{i=1}^n g_i(\langle a_i, x \rangle),$$

then the square root of Hessian is given by

$$B(x) = D_g(x) A \in \mathbb{R}^{n \times d}$$

where $D_g(x) \in \mathbb{R}^{n \times n}$ is a diagonal matrix such that the i-th of $D_g(X)$ is $\sqrt{g_i''(\langle a_i, x \rangle)}$. For more examples, we refer interested reader to Section 3.3 in [3]. Naive implementation of Newton method needs to compute

$$\nabla^2 f(x) = B(x)^\top B(x)$$

and it costs $O(nd^2)$ time. However, under the treewidth assumption on $A^\top$, we can construct an oracle which can take an arbitrary vector h as input, and output $(\nabla^2 f(x))^{-1} \cdot h$ in $O(n\tau^2)$. For this oracle, we don't need to explicitly calculate $\nabla^2 f(x)$, as L^{-1} and $L^{-\top}$ remains τ-sparse in the algorithm.

The detailed implementation is shown in Algorithm 1. Next, we state our result for the toy example.

Algorithm 1 Fast Newton Update

procedure FastNewtonUpdate(f, x_0) ▷ Theorem III.4
  ▷ x_0 is an initial point that is satisfying $\|x_0 - x^*\|_2 \leq O(1/\kappa)$
  for $t = 1 \to T$ **do**
    Compute $B(x_t) \in \mathbb{R}^{n \times d}$ and $\nabla f(x_t) \in \mathbb{R}^d$.
    Compute the Choleksky factorization

$$B(x_t)^\top B(x_t) = LL^\top$$

    Find a solution g_t to the regression problem:

$$\min_{g_t \in \mathbb{R}^n} \|(B(x_t)^\top B(x_t)) \cdot g_t - \nabla f(x_t)\|_2.$$

    $g_t \leftarrow (LL^\top)^{-1} \nabla f(x_t)$
    $x_{t+1} \leftarrow x_t - g_t$
  end for
  return x_T
end procedure

Theorem III.4. *Suppose function f is γ-strongly convex, β-smooth and its Hessian is L Lipschitz continuous. Let $\kappa = L/\gamma$. We assume that $A^\top$ has small treewidth τ. Given an initialization point x_0 satisfying $\|x_0 - x^\star\|_2 \leq 1/\kappa$, there exist an algorithm (procedure* FastNewtonUpdate *in Algorithm 1) which achieves*

$$\|x_{t+1} - x^\star\|_2 \leq \kappa \|x_t - x^\star\|_2^2. \quad (4)$$

Consequently, in order to find an ϵ approximate optimal solution, this algorithm takes $\log\log(1/\epsilon)$ iterations. In addition, each iteration of the algorithm takes $O(d\tau^2)$ time.

Proof. We first show the correctness of our algorithm, and then give an analysis on the running time. Denote

$$\begin{aligned} g_t &= \nabla^2 f(x_t)^{-1} \nabla f(x_t) \\ &= (B(x_t)^\top B(x_t))^{-1} \nabla f(x_t). \end{aligned} \tag{5}$$

We have

$$\begin{aligned} & \|\nabla^2 f(x_t)(x_{t+1} - x^\star)\|_2 \\ = & \|\nabla^2 f(x_t)(x_t - x^\star + x_{t+1} - x_t)\|_2 \\ = & \|\nabla^2 f(x_t)(x_t - x^\star) - \nabla^2 f(x_t) g_t\|_2 \end{aligned} \tag{6}$$

The first step follows from the commutative property and the second step follows from the definition of g_t.

Further, we have

$$\begin{aligned} & \|\nabla^2 f(x_t)(x_t - x^\star) - \nabla^2 f(x_t) g_t\|_2 \\ = & \|\nabla^2 f(x_t)(x_t - x^\star) - \nabla f(x_t)\|_2 \\ = & \|\nabla^2 f(x_t)(x_t - x^\star) \\ & - \int_{s=0}^1 \nabla^2 f(x^\star + s(x_t - x^\star))(x_t - x^\star) \mathrm{d}s\|_2 \\ = & \|\int_{s=0}^1 (\nabla^2 f(x_t) - \nabla^2 f(x^\star + s(x_t - x^\star)))(x_t - x^\star) \mathrm{d}s\|_2 \\ \leq & \int_{s=0}^1 \|\nabla^2 f(x_t) - \nabla^2 f(x^\star + s(x_t - x^\star))\| \mathrm{d}s \cdot \|x_t - x^\star\|_2 \\ \leq & \int_{s=0}^1 L(1-s)\|x_t - x^\star\|_2 \mathrm{d}s \cdot \|x_t - x^\star\|_2 \\ \leq & L\|x_t - x^\star\|_2^2. \end{aligned} \tag{7} \tag{8}$$

The first step follows from the definition of $\widetilde{g_t}$ in Eq. (5), the second step follows from $\nabla f(x^\star) = 0$, the third step follows from the distributive law of multiplication, the fourth step follows from Cauchy-Schwartz inequality, the fifth step follows from the definition of L Lipschitz continuous Hessian and the final step follows from the definition of integral.

$$\begin{aligned} \|x_{t+1} - x^\star\| &\leq \frac{1}{\gamma} \|\nabla^2 f(x_t)(x_{t+1} - x_t)\|_2 \\ &\leq \frac{1}{\gamma}(L\|x_t - x^\star\|_2^2) \\ &= \frac{L}{\gamma}\|x_t - x^\star\|_2^2. \end{aligned} \tag{9}$$

The first step follows from the convexity of f and the second step follows from Eq. (7).

Let $r_t := \|x_t - x^*\|_2$. Then we have

$$\begin{aligned} r_T &\leq \kappa \cdot r_{T-1}^2 \\ &\leq \kappa \cdot (\kappa \cdot r_{T-2}^2)^2 \\ &= \kappa^{1+2^1} \cdot r_{T-2}^{2^2} \\ &\leq \ldots \\ &\leq \kappa^{2^T - 1} \cdot r_0^{2^T} \\ &\leq (\kappa r_0)^{2^T}, \end{aligned}$$

where the first and second steps both follow from Eq. (9), the third step follows from reorganization. We repeat the first three steps recursively and derive the final step.

Suppose $r_0 < R$ and $R < 1/\kappa$, to make sure $r_T < \epsilon$, we need to choose $T = \log\log(1/\epsilon)$.

First computing $B(x_t)$ and $\nabla f(x_t) \in \mathbb{R}^d$ takes $O(d\tau)$ and $O(d)$ time respectively. Then, computing the Choleksky factorization $B(x_t)^\top B(x_t) = LL^\top$ takes $O(d\tau^2)$ by using Lemma II.3. Next, solving regression problem takes $O(d\tau^2)$ by using Lemma II.6. Finally, the update steps takes $O(d)$ time. Hence, the total running time per iteration is $O(d\tau^2)$. □

IV. Neural Network

In this section, we state our result for fast neural network training algorithm in the small treewidth setting. We first present our algorithm in Section IV-A. In Section IV-B, we prove the quadratic convergence of our algorithm and show that $O(\log\log(1/\epsilon))$ iterations suffice for ϵ error. In Section IV-C, we show the running time per iteration of our algorithm.

A. Algorithm

In [11], they first reformulate the Gauss-Newton iteration as an ℓ_2-regression problem and then reduce the dimension of the optimization problem to find an approximate solution of g_t. Our algorithm (Algorithm 1) shares the same update step with them while we solve the regression problem exactly and faster. Under small treewidth, we are able to compute Cholesky factorization for the Gram matrix then solve the regression problem directly. Note that the acceleration has two aspects: one is the efficient solution to regression problem that reduces per iteration cost, another is the exact solution that leads to a quadratic convergence rate.

Algorithm 2 Fast Neural Network Training

procedure FastNN
  $W_0 \leftarrow$ random Gaussian matrix ▷ $W_0 \in \mathbb{R}^{m \times d}$
  for $t = 0 \to T$ **do**
    Compute the Jacobian matrix J_t ▷ $J_t \in \mathbb{R}^{n \times md}$
    Compute the Cholesky factorization $J_t J_t^\top = LL^\top$
    Solve the following regression problem:

$$\min_{g_t} \|J_t J_t^\top g_t - (f_t - y)\|_2$$

    $g_t \leftarrow (LL^\top)^{-1}(f_t - y)$
    Update $W_{t+1} \leftarrow W_t - J_t^\top g_t$
  end for
end procedure

B. Convergence

To begin with, we state the convergence lemma for our algorithm. This lemma provides guarantee for quadratic contraction per iteration. Thus, we can achieve ϵ approximation in $O(\log\log(1/\epsilon))$ iterations.

Lemma IV.1 (Bounds on Norms at Initialization)**.** *If $M = \Omega(d\log(16n/\delta))$, where d is the dimension of feature space of the inputs and n in the number of data points, the three following statements are true with probability at least $1-\delta/2$*

a) $\|W_0\|_2 = O(\sqrt{M})$.
b) $f(W_0, x_i) = O(1)$, *for* $i \in [n]$.
c) $\|J_{W_0,x_i}\|_F = O(1)$, *for* $i \in [n]$.

Proof. a). The lemma is a well-known result concerning of the estimation of singular values of Gaussian random matrices. Given $W_0 \in \mathbb{R}^{M\times d}$ is a Guassian random matrix, we can derive the following with probability $1-2e^{-t^2/2}$

$$\|W_0\|_2 \le \sqrt{M} + \sqrt{d} + t.$$

By choosing $M = \max(d, \sqrt{2\log(8/\delta)})$, we obtain $\|W_0\|_2 \le 3\sqrt{M}$ with probability $1-\delta/4$.

b). First, $a_r, r \in [M]$ are Rademacher variables, thereby 1-sub-Guassian, so we get $\frac{1}{M}\sum_{r=1}^{M} a_r \le t$ with probability $1-2e^{-Mt^2/2}$. This means if we choose $M = \Omega(\log(16/\delta))$, we get

$$\Pr[\frac{1}{\sqrt{M}}\sum_{r=1}^{M} a_r = O(1)] \ge 1 - \frac{\delta}{8}. \tag{10}$$

We know the vector $v_i = \frac{1}{\|x_i\|_2} W_0 x_i \in \mathbb{R}^{M\times 1}$ is a standard Gaussian vector. Assuming that the activation $\sigma(\cdot)$ is l-Lipschitiz and l is $O(1)$, when vector a is fixed, the function

$$\phi : \mathbb{R}^M \to \mathbb{R}, v_i \mapsto \frac{1}{\sqrt{M}} a^\top \sigma(\|x_i\|_2 v_i) = f(W_0, x_i)$$

has a Lipschitz parameter of $l\|x_i\|_2/\sqrt{M} = O(1/\sqrt{M})$. According to the classic result on the concentration of a Lipschitz function over Guassian variables, we have

$$\Pr[|\phi(v_i) - \mathbb{E}_{W_0}(\phi(v_i))| \ge t] \le 2\exp(-\frac{Mt^2}{2l^2\|x_i\|^2}),$$

which means if $M = \Omega(\log(16n/\delta))$,

$$|\frac{1}{\sqrt{M}} a^\top \sigma(W_0 x_i) - \frac{1}{\sqrt{M}}(\sum_{r=1}^{M} a_r)\mathbb{E}_{w\sim\mathcal{N}(0,\mathbb{I}_d)}[\sigma(wx_i)]|$$
$$= O(1). \tag{11}$$

holds jointly for all $i \in [n]$ with probability $1-\delta/8$. Note that

$$|\mathbb{E}_{w\sim\mathcal{N}(0,\mathbb{I}_d)}[\sigma(wx_i)]| \le |\sigma(0)| + l \cdot \mathbb{E}_{\xi\sim\mathcal{N}(0,\|\mathbf{x}_i\|_2^2))}[|\xi|]$$
$$= O(1). \tag{12}$$

Plugging in Eq. (10) and Eq. (12) into Eq. (11), we see that as long as $M = \Omega(\log(16n/\delta))$, then with probability $1-\delta/4$, for all $i \in [n]$,

$$f(W_0, x_i) = \frac{1}{\sqrt{M}} a^\top \sigma(W_0 x_i) = O(1).$$

c). Since σ is $O(1)$-Lipschitz, we have $\|d_{W,x_i}\|_\infty = O(1)$. The formula for J is

$$J_{W,x} = \frac{1}{\sqrt{M}}((d_{W,x} \circ a)x^\top).$$

Then, we can easily know that

$$\|J_{W,x_i}\|_F \le \frac{1}{\sqrt{M}}\|\operatorname{diag}(d)\|_2\|a\|_2\|x\|_2 = O(1).$$

Thus we complete the proof. □

Next, we show the bound on the least eigenvalue of the Gram matrix at initialization.

Lemma IV.2 (Bound on the Least Eigenvalue of the Gram Matrix at Initialization)**.** *If the width $M = \Omega(\frac{n^2\log(2n/\delta)}{\lambda_0^2})$, then the following statement is true with probability at least $1-\delta/2$ over random initialization*

$$\lambda_{\min}(G_{W_0}) \ge \frac{3}{4}\lambda_0.$$

Proof. Given σ is Lipschitz, $\sigma'(wx_i)\sigma'(wx_j)$ is bounded by $O(1)$. For each fixed (ij) pair, at initialization $G_{i,j}$ is an average of independent random variables, and by Hoeffding's inequality, applying union bound for all n^2 of (i,j) pairs, with probability $1-\delta/2$ at initializaion we have

$$|G_{ij} - K_{ij}| \le O(\sqrt{\frac{\log(2n/\delta)}{M}})$$

and then

$$\|G_{W_0} - K\|_2^2 \le \|G_{W_0} - K\|_F^2 = O(\frac{n^2\log(2n/\delta)}{M}).$$

Hence, if $M = \Omega(\frac{n^2\log(2n/\delta)}{\lambda_0^2})$ we can have

$$\|G_{W_0} - K\|_2 \le \frac{1}{4}\lambda_0$$

and thus $\lambda_{\min}(G_{W_0}) \ge \frac{3}{4}\lambda_0$. □

Lemma IV.3 (Bounds on Norms in the Optimization Scope, [9])**.** *Suppose the events in Lemma IV.1 hold. There is a constant $C > 0$ so that if $M \ge CR^2$, we have the following:*

a) For any $W \in B(R)$, we have

$$\|W\|_2 = O(\sqrt{M}). \tag{13}$$

b) For any $W_1, W_2 \in B(R)$, if $\|W_1 - W_2\|_F \le R'$, then we have

$$\|J_{W_1,x_i} - J_{W_2,x_i}\|_F = O(\frac{R'}{\sqrt{M}}), \forall i,$$
$$\text{and } \|J_{W_1} - J_{W_2}\|_2 = O(R'\sqrt{\frac{n}{M}}). \tag{14}$$

Also, for any $W \in B(R)$, we have

$$\|J_W\|_F = O(1) \text{ and } \|J_W\|_2 = O(\sqrt{n}). \tag{15}$$

Lemma IV.4 (Convergence Lemma)**.** *Assume Assumption II.7 holds. Assume the scale of the data is $\|x_i\|_2 = O(1), |y_i| = O(1)$ for $i \in \{1, 2, \cdots, n\}$. If the network width $M = \Omega(\lambda_0^{-4}n^4)$ then with probability $1-\delta$ over the random initialization, Algorithm 2 satisfies the following:*

1) The Gram matrix G_t at each iteration is invertable

2) The loss converges to zero in a way that

$$\|f_{t+1} - y\|_2 \leq \frac{C}{\sqrt{M}} \|f_t - y\|_2^2$$

for some C that is independent of M, which is a second-order convergence.

Proof. We use $W_t, t \in \{0, 1, \cdots\}$ to denote the parameter W after t iterations. We use J_t, G_t, f_t to represent $J_{W_t}, G_{W_t}, f_{W_t}$ respectively for short. We introduce the integral of Jacobian over convex combination as

$$J_{t,t+1} = \int_0^1 J((1-s)W_t + sW_{t+1})\,\mathrm{d}s. \tag{16}$$

For each iteration, if G_t is invertible, we have

$$\begin{aligned} f_{t+1} - y &= f_t - y + J_{t,t+1}\,\mathrm{vec}(W_{t+1} - W_t) \\ &= f_t - y - J_{t,t+1} J_t^\top G_t^{-1}(f_t - y) \\ &= (J_t - J_{t,t+1}) J_t^\top G_t^{-1}(f_t - y) \end{aligned}$$

hence

$$\|f_{t+1} - y\|_2 \leq \|J_t - J_{t,t+1}\|_2 \|J_t^\top\|_2 \|G_t^{-1}\|_2 \|f_t - y\|_2 \tag{17}$$

Then we control the first term of the right hand side based on Lemma IV.3 in the following form

$$\begin{aligned} \|J_t - J_{t,t+1}\|_2 &\leq \frac{\widehat{C}}{\sqrt{M}} \|W_t - W_{t+1}\|_2 \\ &= \frac{\widehat{C}}{\sqrt{M}} \|J_t^\top G_t^{-1}(f_t - y)\|_2 \\ &\leq \frac{\widehat{C}}{\sqrt{M}} \|J_t^\top\|_2 \|G_t^{-1}\|_2 \|f_t - y\|_2, \end{aligned}$$

Let $R_t = \|W_t - W_{t+1}\|_F$ for $t \in \{0, 1, \cdots\}$. We take $R = \Theta(\frac{n}{\lambda_0})$ in Lemma IV.3 and II.10. We prove that $M = \Omega(\max(\frac{n^4}{\lambda_0^4}, \frac{n^2 d \log(16n/\delta)}{\lambda_0^2}))$ (with enough constant) suffices for second-order convergence.

First we can verify that all the requirements for M in Lemma IV.1-II.10 is satisfied. Hence, with probability $1 - \delta$ all the events in Lemma IV.1-II.10 hold. Conditioned on this, we do induction on t to show the following:

- (a). $W_t \in B(R)$
- (b). If $t \geq 0$, then $\|f_{t+1} - y\|_2 \leq \frac{n^{3/2}}{\lambda_0^2 \sqrt{M}} \|f_t - y\|_2^2$

As long as (b) is true for all t, we can choose large enough M to make sure the series $\{\|f_t - y\|_2\}_{t=0}^\infty$ converges to zero. Then, we obtain the second-order convergence property.

For $t = 0$, (a) and (b) hold by definition. Suppose the proposition holds for $t = 0, \cdots, T$. Then for $t = 0, \cdots, T$, G_t is invertible. Recal that the update rule is $\mathrm{vec}(W_{t+1}) = \mathrm{vec}(W_t) - J_t^\top G_t^{-1}(f_t - y)$, we have

$$\begin{aligned} R_t = &\ \|W_t - W_{t+1}\|_F \\ = &\ \|\,\mathrm{vec}(W_t) - \mathrm{vec}(W_{t+1})\|_2 \\ \leq &\ \|J_t^\top\|_2 \|G_t^{-1}\|_2 \|f_t - y\|_2 \\ \leq &\ O(\frac{\sqrt{n}}{\lambda_0} \|f_t - y\|_2). \end{aligned} \tag{18}$$

where the last step follows from Lemma IV.3 and II.10. According to Lemma IV.1(b) and the assumption that the target label $y_i = O(1)$, we have $\|f_0 - y\|_2^2 = O(n)$. When $T > 1$, the decay ratio at the first step is bounded as

$$\frac{\|f_1 - y\|_2}{\|f_0 - y\|_2} \leq O(\frac{n^{\frac{3}{2}}}{\lambda_0^2 \sqrt{M}}) \|f_0 - y\|_2 \leq O(\frac{n^2}{\lambda_0^2 \sqrt{M}}) =: r$$

Taking $M = \Omega(\frac{n^2}{\lambda_0^4})$ with enough constant can ensure that r is a constant less than 1. In this case, the second-order convergence property (b) will ensure a faster ratio of decay at each subsequent step in $\|f_t - y\|_{t=0}^T$.

Combining Eq. (18), we have

$$\begin{aligned} \sum_{t=0}^T R_t &\leq O(\sum_{t=0}^T \frac{\sqrt{n}}{\lambda_0} \|f_t - y\|_2) \\ &\leq O(\frac{n}{\lambda_0} \sum_{t=1}^\infty r^{t-1}) \\ &= O(\frac{n}{\lambda_0}) \\ &\leq R. \end{aligned} \tag{19}$$

Therefore, $W_{T+1} \in B(R)$. Eq. (19) also holds when $T = 0$, so (a) is true.

Since $B(R)$ is convex, we also have $sW_T + (1-s)W_{T+1} \in B(R)$ for $s \in [0, 1]$. Hence we bound the difference of the Jacobian as follows

$$\begin{aligned} &\|J_{t,t+1} - J_T\|_2 \\ \leq &\int_0^1 \|J(sW_t + (1-s)W_{t+1}) - J(W_t)\|_2\,\mathrm{d}s \\ \leq &\ O(\int_0^1 sR_t \sqrt{\frac{n}{m}}\,\mathrm{d}s) \\ = &\ O(\frac{n}{\lambda_0 \sqrt{M}} \|f_t - y\|_2). \end{aligned} \tag{20}$$

where the second step follows from Lemma IV.3 and the last step follows from Eq. (18).

Using the bound of Eq. ((17)), we obtain

$$\begin{aligned} \|f_{t+1} - y\|_2 &\leq \|J_t - J_{t,t+1}\|_2 \|J_t^\top\|_2 \|G_t t^{-1}\|_2 \|f_t - y\| \\ &\leq \frac{n^{3/2}}{\lambda_0^2 \sqrt{M}} \|f_t - y\|_2^2, \end{aligned}$$

where the second inequality follows from Eq. (20) and Lemma IV.3,II.10 and it proves (b). □

C. Running time

In this section, we show the running time for our algorithm. We prove that $O(\log\log(1/\epsilon))$ iterations suffice for ϵ error and each iteration takes $O(md\tau^2)$ time.

Theorem IV.5 (Our result, formal version of Theorem I.1)**.** *Given a two-layer neural network with m neurons and assume data matrix has treewidth τ, then to train the accuracy of neural network to ϵ, there is an algorithm that runs in $O(\log\log(1/\epsilon))$ iterations and each iteration takes $O(md\tau^2)$ time.*

Proof. It directly follows from Lemmas IV.6 and IV.7 . □

We first show the running time for Algorithm 2.

Lemma IV.6. *Let n, d represent the number of data points and the dimension of feature space respectively. Let m denote the width of the neaural network. Given the data matrix has treewidth τ, the cost per iteration of Algorithm 2 is*

$$O(md\tau^2)$$

Proof. Note that there are at most $d\tau$ nonzero entries in X. Hence, it take $O(md\tau)$ time to compute all necessary $\langle w_j(t), x_i \rangle$. Hence it takes $O(md\tau)$ time to construct J_t. It is not hard to see that J_t has (column-wise) treewidth τ. By Lemma II.3, computing the Cholesky decomposition of $J_t J_t^\top$ takes $O(md\tau^2)$ time. Note that our definition of tree-width is row-wise, we need one more step here to get the Cholesky factorization of $J_t J_t^\top$. See Remark II.4. We compute the upper-triangular version of Cholesky factorization $J_t^\top J_t = U U^\top$ for some upper triangular matrix U. Then we get the Choleske factorization $J_t J_t^\top = L L^\top$ where $L = U^\top$ is a lower-triangular matrix. Next, solving regression problem takes $O(d\tau^2)$ time by Lemma II.6. Finally, the update step takes $O(md\tau)$ time due to sparse structure of J_t. Thus, the total running time per iteration is $O(md\tau + md\tau^2 + d\tau^2 + md\tau) = O(md\tau^2)$. □

Next, we show the iterations needed for Algorithm 2.

Lemma IV.7. *To train the accuracy of the network to error ϵ, the number of iterations for Algorithm 2 to converge is $O(\log\log(1/\epsilon))$*

Proof. By the second point in Lemma IV.4, we set $\kappa = \frac{C}{\sqrt{M}}$. Let $r_t := \|f_t - y\|_2$. Then we have

$$\begin{aligned} r_T &\leq \kappa \cdot r_{T-1}^2 \\ &\leq \kappa \cdot (\kappa \cdot r_{T-2}^2)^2 \\ &= \kappa^{1+2^1} \cdot r_{T-2}^{2^2} \\ &\leq \ldots \\ &\leq \kappa^{2^T - 1} \cdot r_0^{2^T} \\ &\leq (\kappa r_0)^{2^T} \end{aligned}$$

By setting $T = O(\log\log(1/\epsilon))$, we obtain $\|f_T - y\|_2 < \epsilon$. □

V. Conclusion

Our paper provides a second-order algorithm for training a two-layer neural networks with $O(\log\log(1/\epsilon))$ convergence rate. Our main technical contribution is speeding up the calculation of multiplying the inverse of the Gram matrix $G_t = J_t J_t^\top$ with $J_t^\top$ in each iteration. We make an assumption that data matrix has small treewidth τ, which is common assumption for accelerating optimization problems. Leveraging the properties of small treewidth and using Cholesky decomposition, the cost per iteration in our algorithm is $O(md\tau^2)$, where $m = \Omega(n^4)$.

References

[1] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning (still) requires rethinking generalization," *Communications of the ACM*, vol. 64, no. 3, pp. 107–115, 2021.

[2] M. Hardt, B. Recht, and Y. Singer, "Train faster, generalize better: Stability of stochastic gradient descent," in *International conference on machine learning*. PMLR, 2016, pp. 1225–1234.

[3] M. Pilanci and M. J. Wainwright, "Newton sketch: A near linear-time optimization algorithm with linear-quadratic convergence," *SIAM Journal on Optimization*, vol. 27, no. 1, pp. 205–245, 2017.

[4] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of machine learning research*, vol. 12, no. 7, 2011.

[5] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[6] S. S. Du, X. Zhai, B. Poczos, and A. Singh, "Gradient descent provably optimizes over-parameterized neural networks," *arXiv preprint arXiv:1810.02054*, 2019.

[7] Z. Song and X. Yang, "Quadratic suffices for over-parametrization via matrix chernoff bound," *arXiv preprint arXiv:1906.03593*, 2019.

[8] X. Wu, S. S. Du, and R. Ward, "Global convergence of adaptive gradient methods for an over-parameterized neural network," *arXiv preprint arXiv:1902.07111*, 2019.

[9] T. Cai, R. Gao, J. Hou, S. Chen, D. Wang, D. He, Z. Zhang, and L. Wang, "Gram-gauss-newton method: Learning overparameterized neural networks for regression problems," *arXiv preprint arXiv:1905.11675*, 2019.

[10] G. Zhang, J. Martens, and R. B. Grosse, "Fast convergence of natural gradient descent for over-parameterized neural networks," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[11] J. v. d. Brand, B. Peng, Z. Song, and O. Weinstein, "Training (overparametrized) neural networks in near-linear time," in *ITCS*, 2021.

[12] Y. Li and Y. Liang, "Learning overparameterized neural networks via stochastic gradient descent on structured data," *Advances in neural information processing systems*, vol. 31, 2018.

[13] Z. Allen-Zhu, Y. Li, and Z. Song, "A convergence theory for deep learning via over-parameterization," in *International Conference on Machine Learning*. PMLR, 2019, pp. 242–252.

[14] ——, "On the convergence rate of training recurrent neural networks," *Advances in neural information processing systems*, vol. 32, 2019.

[15] S. Bubeck, "Convex optimization: Algorithms and complexity," *Foundations and Trends® in Machine Learning*, vol. 8, no. 3-4, pp. 231–357, 2015.

[16] R. Y. Zhang and J. Lavaei, "Sparse semidefinite programs with guaranteed near-linear time complexity via dualized clique tree conversion," *Mathematical programming*, vol. 188, pp. 351–393, 2021.

[17] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks, "Approximating treewidth, pathwidth, frontsize, and shortest elimination tree," *Journal of Algorithms*, vol. 18, no. 2, pp. 238–255, 1995.

[18] T. A. Davis, *Direct methods for sparse linear systems*. SIAM, 2006.

[19] Y. Gu and Z. Song, "A faster small treewidth sdp solver," *arXiv preprint arXiv:2211.06033*, 2022.

[20] J. Martens, "Deep learning via hessian-free optimization," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, 2010, pp. 735–742.

[21] O. Vinyals and D. Povey, "Krylov subspace descent for deep learning," in *Artificial intelligence and statistics*. PMLR, 2012, pp. 1261–1268.

[22] S.-I. Amari, "Natural gradient works efficiently in learning," *Neural computation*, vol. 10, no. 2, pp. 251–276, 1998.

[23] J. Martens and R. Grosse, "Optimizing neural networks with kronecker-factored approximate curvature," in *International conference on machine learning*. PMLR, 2015, pp. 2408–2417.

[24] R. Grosse and J. Martens, "A kronecker-factored approximate fisher matrix for convolution layers," in *International Conference on Machine Learning*. PMLR, 2016, pp. 573–582.

[25] J. Ba, R. Grosse, and J. Martens, "Distributed second-order optimization using kronecker-factored approximations," in *International conference on learning representations*, 2022.

[26] Z. Song, W. Wang, and J. Yin, "A unified scheme of resnet and softmax," *arXiv preprint arXiv:2309.13482*, 2023.

[27] Y. Gao, Z. Song, W. Wang, and J. Yin, "A fast optimization view: Reformulating single layer attention in llm based on tensor and svm

trick, and solving it in matrix multiplication time," *arXiv preprint arXiv:2309.07418*, 2023.
[28] S. Bian, Z. Song, and J. Yin, "Federated empirical risk minimization via second-order method," *arXiv preprint arXiv:2305.17482*, 2023.
[29] Z. Li, Z. Song, Z. Wang, and J. Yin, "Local convergence of approximate newton method for two layer nonlinear regression," *arXiv preprint arXiv:2311.15390*, 2023.
[30] J. Lee, L. Xiao, S. Schoenholz, Y. Bahri, R. Novak, J. Sohl-Dickstein, and J. Pennington, "Wide neural networks of any depth evolve as linear models under gradient descent," *Advances in neural information processing systems*, vol. 32, 2019.
[31] J. D. Lee, R. Shen, Z. Song, M. Wang *et al.*, "Generalized leverage score sampling for neural networks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 10 775–10 787, 2020.
[32] S. Oymak and M. Soltanolkotabi, "Toward moderate overparameterization: Global convergence guarantees for training shallow neural networks," *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 84–105, 2020.
[33] B. Huang, X. Li, Z. Song, and X. Yang, "Fl-ntk: A neural tangent kernel-based framework for federated learning analysis," in *International Conference on Machine Learning*. PMLR, 2021, pp. 4423–4434.
[34] Z. Song, S. Yang, and R. Zhang, "Does preprocessing help training over-parameterized neural networks?" *Advances in Neural Information Processing Systems*, vol. 34, pp. 22 890–22 904, 2021.
[35] Y. Liang, Z. Sha, Z. Shi, Z. Song, and Y. Zhou, "Looped relu mlps may be all you need as practical programmable computers," *arXiv preprint arXiv:2410.09375*, 2024.
[36] Y. Ke, X. Li, Y. Liang, Z. Shi, and Z. Song, "Advancing the understanding of fixed point iterations in deep neural networks: A detailed analytical study," *arXiv preprint arXiv:2410.11279*, 2024.
[37] J. Y.-C. Hu, P.-H. Chang, H. Luo, H.-Y. Chen, W. Li, W.-P. Wang, and H. Liu, "Outlier-efficient hopfield layers for large transformer-based models," in *Forty-first International Conference on Machine Learning (ICML)*, 2024.
[38] J. Y.-C. Hu, B.-Y. Chen, D. Wu, F. Ruan, and H. Liu, "Nonparametric modern hopfield models," *arXiv preprint arXiv:2404.03900*, 2024.
[39] H. Luo, J. Yu, W. Zhang, J. Li, J. Y.-C. Hu, X. Xing, and H. Liu, "Decoupled alignment for robust plug-and-play adaptation," *arXiv preprint arXiv:2406.01514*, 2024.
[40] J. Yu, H. Luo, J. Y.-C. Hu, W. Guo, H. Liu, and X. Xing, "Enhancing jailbreak attack against large language models through silent tokens," *arXiv preprint arXiv:2405.20653*, 2024.
[41] J. Y.-C. Hu, M. Su, E.-J. Kuo, Z. Song, and H. Liu, "Computational limits of low-rank adaptation (lora) for transformer-based models," *arXiv preprint arXiv:2406.03136*, 2024.
[42] E. Liu, J. Y.-C. Hu, A. Reneau, Z. Song, and H. Liu, "Differentially private kernel density estimation," *arXiv preprint arXiv:2409.01688*, 2024.
[43] J. Y.-C. Hu, W. Wu, Z. Li, S. Pi, , Z. Song, and H. Liu, "On statistical rates and provably efficient criteria of latent diffusion transformers (dits)," in *Thirty-eighth Conference on Neural Information Processing Systems (NeurIPS)*, 2024.
[44] J. Y.-C. Hu, D. Wu, and H. Liu, "Provably optimal memory capacity for modern hopfield models: Transformer-compatible dense associative memories as spherical codes," in *Thirty-eighth Conference on Neural Information Processing Systems (NeurIPS)*, 2024.
[45] A. Jacot, F. Gabriel, and C. Hongler, "Neural tangent kernel: Convergence and generalization in neural networks," *Advances in neural information processing systems*, vol. 31, 2018.
[46] Z. Song and Z. Yu, "Oblivious sketching-based central path method for linear programming," in *Proceedings of the 38th International Conference on Machine Learning*, 2021, pp. 9835–9847.
[47] D. P. Woodruff *et al.*, "Sketching as a tool for numerical linear algebra," *Foundations and Trends® in Theoretical Computer Science*, vol. 10, no. 1–2, pp. 1–157, 2014.
[48] X. Li, Z. Song, and J. Yu, "Quantum speedups for approximating the john ellipsoid," *arXiv preprint arXiv:2408.14018*, 2024.
[49] X. Li, Y. Liang, Z. Shi, Z. Song, and J. Yu, "Fast john ellipsoid computation with differential privacy optimization," *arXiv preprint arXiv:2408.06395*, 2024.
[50] Z. Song, X. Yang, Y. Yang, and L. Zhang, "Sketching meets differential privacy: fast algorithm for dynamic kronecker projection maintenance," in *International Conference on Machine Learning*. PMLR, 2023, pp. 32 418–32 462.
[51] Y. Deng, H. Hu, Z. Song, O. Weinstein, and D. Zhuo, "Training overparametrized neural networks in sublinear time," *arXiv preprint arXiv:2208.04508*, 2022.
[52] Y. Gao, L. Qin, Z. Song, and Y. Wang, "A sublinear adversarial training algorithm," in *The Twelfth International Conference on Learning Representations*, 2024.
[53] X. Li, Z. Song, and J. Yang, "Federated adversarial learning: A framework with convergence analysis," in *International Conference on Machine Learning*. PMLR, 2023, pp. 19 932–19 959.
[54] Y. Liang, Z. Sha, Z. Shi, and Z. Song, "Differential privacy mechanisms in neural tangent kernel regression," *arXiv preprint arXiv:2407.13621*, 2024.
[55] Y. Liang, Z. Shi, Z. Song, and Y. Zhou, "Differential privacy of cross-attention with provable guarantee," *arXiv preprint arXiv:2407.14717*, 2024.
[56] B. Chen, Y. Liang, Z. Sha, Z. Shi, and Z. Song, "Hsr-enhanced sparse attention acceleration," *arXiv preprint arXiv:2410.10165*, 2024.
[57] Y. Liang, J. Long, Z. Shi, Z. Song, and Y. Zhou, "Beyond linear approximations: A novel pruning approach for attention matrix," *arXiv preprint arXiv:2410.11261*, 2024.
[58] Y. Gao, Z. Song, X. Yang, and Y. Zhou, "Differentially private attention computation," in *Neurips Safe Generative AI Workshop 2024*, 2024.
[59] Y. Liang, Z. Sha, Z. Shi, Z. Song, and Y. Zhou, "Multi-layer transformers gradient can be approximated in almost linear time," *arXiv preprint arXiv:2408.13233*, 2024.
[60] X. Li, Y. Liang, Z. Shi, Z. Song, and Y. Zhou, "Fine-grained attention i/o complexity: Comprehensive analysis for backward passes," *arXiv preprint arXiv:2410.09397*, 2024.
[61] X. Li, Y. Liang, Z. Shi, and Z. Song, "A tighter complexity analysis of sparsegpt," *arXiv preprint arXiv:2408.12151*, 2024.
[62] B. Chen, X. Li, Y. Liang, Z. Shi, and Z. Song, "Bypassing the exponential dependency: Looped transformers efficiently learn in-context by multi-step gradient descent," *arXiv preprint arXiv:2410.11268*, 2024.
[63] S. S. Du, K. Hou, R. R. Salakhutdinov, B. Poczos, R. Wang, and K. Xu, "Graph neural tangent kernel: Fusing graph neural networks with graph kernels," *Advances in neural information processing systems*, vol. 32, 2019.
[64] L. Qin, Z. Song, and B. Sun, "Is solving graph neural tangent kernel equivalent to training graph neural network?" *arXiv preprint arXiv:2309.07452*, 2023.
[65] Y.-T. Chang, Z. Hu, X. Li, S. Yang, J. Jiang, and N. Sun, "Dihan: A novel dynamic hierarchical graph attention network for fake news detection," in *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, 2024, pp. 197–206.
[66] S. Krishnagopal and L. Ruiz, "Graph neural tangent kernel: Convergence on large graphs," in *International Conference on Machine Learning*. PMLR, 2023, pp. 17 827–17 841.
[67] Y. Liang, Z. Shi, Z. Song, and Y. Zhou, "Tensor attention training: Provably efficient learning of higher-order transformers," *arXiv preprint arXiv:2405.16411*, 2024.
[68] C. Li, Y. Liang, Z. Shi, and Z. Song, "Exploring the frontiers of softmax: Provable optimization, applications in diffusion model, and beyond," *arXiv preprint arXiv:2405.03251*, 2024.
[69] Y. Liang, Z. Shi, Z. Song, and Y. Zhou, "Unraveling the smoothness properties of diffusion models: A gaussian mixture perspective," *arXiv preprint arXiv:2405.16418*, 2024.
[70] ——, "Unraveling the smoothness properties of diffusion models: A gaussian mixture perspective," *arXiv preprint arXiv:2405.16418*, 2024.
[71] Z. Xu, Z. Shi, J. Wei, F. Mu, Y. Li, and Y. Liang, "Towards few-shot adaptation of foundation models via multitask finetuning," in *The Twelfth International Conference on Learning Representations*, 2024.
[72] Z. Shi, Y. Ming, Y. Fan, F. Sala, and Y. Liang, "Domain generalization via nuclear norm regularization," in *Conference on Parsimony and Learning*. PMLR, 2024, pp. 179–201.
[73] A. George, J. Liu, and E. Ng, "Computer solution of sparse linear systems," *Oak Ridge National Laboratory*, 1994.
[74] Y. Gu, Z. Song, and L. Zhang, "A nearly-linear time algorithm for structured support vector machines," *arXiv preprint arXiv:2307.07735*, 2023.
[75] Z. Allen-Zhu, Y. Li, and Y. Liang, "Learning and generalization in overparameterized neural networks, going beyond two layers," *Advances in neural information processing systems*, vol. 32, 2019.