Training (Overparametrized) Neural Networks in Near-Linear Time

Jan van den Brand
KTH Royal Institute of Technology, Stockholm, Sweden
janvdb@kth.se

Binghui Peng
Columbia University, New York, NY, USA
bp2601@columbia.edu

Zhao Song
Princeton University and Institute for Advanced Study, NJ, USA
zhaos@ias.edu

Omri Weinstein
Columbia University, New York, NY, USA
omri@cs.columbia.edu

Abstract

The slow convergence rate and pathological curvature issues of first-order gradient methods for training deep neural networks, initiated an ongoing effort for developing faster *second-order* optimization algorithms beyond SGD, without compromising the generalization error. Despite their remarkable convergence rate (*independent* of the training batch size n), second-order algorithms incur a daunting slowdown in the *cost per iteration* (inverting the Hessian matrix of the loss function), which renders them impractical. Very recently, this computational overhead was mitigated by the works of [79, 23], yielding an $O(mn^2)$-time second-order algorithm for training two-layer overparametrized neural networks of polynomial width m.

We show how to speed up the algorithm of [23], achieving an $\widetilde{O}(mn)$-time backpropagation algorithm for training (mildly overparametrized) ReLU networks, which is near-linear in the dimension (mn) of the full gradient (Jacobian) matrix. The centerpiece of our algorithm is to reformulate the Gauss-Newton iteration as an ℓ_2-regression problem, and then use a Fast-JL type dimension reduction to *precondition* the underlying Gram matrix in time independent of M, allowing to find a sufficiently good approximate solution via *first-order* conjugate gradient. Our result provides a proof-of-concept that advanced machinery from randomized linear algebra – which led to recent breakthroughs in *convex optimization* (ERM, LPs, Regression) – can be carried over to the realm of deep learning as well.

2012 ACM Subject Classification Theory of computation → Nonconvex optimization

Keywords and phrases Deep learning theory, Nonconvex optimization

Digital Object Identifier 10.4230/LIPIcs.ITCS.2021.63

Related Version A full version of the paper is available at `https://arxiv.org/abs/2006.11648`.

Funding *Jan van den Brand*: This project has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation program under grant agreement No 715672. Partially supported by the Google PhD Fellowship Program.
Binghui Peng: Research supported by NSF IIS-1838154, NSF CCF-1703925 and NSF CCF-1763970.
Zhao Song: Research supported by Special Year on Optimization, Statistics, and Theoretical Machine Learning (being led by Sanjeev Arora) at Institute for Advanced Study.
Omri Weinstein: Research supported by NSF CAREER award CCF-1844887.

Acknowledgements The author would like to thank David Woodruff for telling us the tensor trick for computing kernel matrices and helping us improve the presentation of the paper.

1 Introduction

Understanding the dynamics of gradient-based optimization of deep neural networks has been a central focal point of theoretical machine learning in recent years [49, 81, 80, 48, 31, 4, 5, 3, 13, 58, 9, 67, 27, 39, 22]. This line of work led to a remarkable rigorous understanding of the generalization, robustness and convergence rate of *first-order* (SGD-based) algorithms, which are the standard choice for training DNNs. By contrast, the *computational complexity* of implementing gradient-based training algorithms (e.g., backpropagation) in such non-convex landscape is less understood, and gained traction only recently due to the overwhelming size of training data and complexity of network design [55, 32, 51, 23, 79].

The widespread use first-order methods such as (stochastic) gradient descent in training DNNs is explained, to a large extent, by its computational efficiency – recalculating the gradient of the loss function at each iteration is simple and cheap (linear in the dimension of the full gradient), let alone with the advent of minibatch random sampling [37, 23]. Nevertheless, first-order methods have a slow rate of convergence in non-convex settings (typically $\Omega(\text{poly}(n)\log(1/\epsilon))$ for overparametrized networks, see e.g., [79]) for reducing the training error below ϵ, and it is increasingly clear that SGD-based algorithms are becoming a real bottleneck for many practical purposes. This drawback initiated a substantial effort for developing fast training methods beyond SGD, aiming to improve its convergence rate without compromising the generalization error [16, 53, 55, 32, 44, 59, 23, 79].

Second-order gradient algorithms (which employ information about the Hessian of the loss function), pose an intriguing computational tradeoff in this context: On one hand, they are known to converge extremely fast, at a rate *independent* of the input size (i.e., only $O(\log 1/\epsilon)$ iterations [79]), and offer a qualitative advantage in overcoming pathological curvature issues that arise in first-order methods, by exploiting the local geometry of the loss function. This feature implies another practical advantage of second order methods, namely, that they do not require tuning the learning rate [23, 79]. On the other hand, second-order methods have a prohibitive *cost per iteration*, as they involve *inverting* a dynamically-changing dense Hessian matrix. This drawback explains the scarcity of second order methods in *large scale non-convex* optimization, in contrast to its popularity in the convex setting.

The recent works of [23, 79] addressed the computational bottleneck of second-order algorithms in optimizing deep neural nets, and presented a training algorithm for overparametrized neural networks with smooth (resp. ReLU) activations, whose running time is $O(mn^2)$, where m is the width of the neural network, and n is the size of the training data in $\mathbb{R}^d$. The two algorithms, which achieve essentially the same running time, are based on the classic Gauss-Newton algorithm (resp. "Natural gradient" algorithm) combined with the recent introduction of *Neural Tangent Kernels* (NTK) [38]. The NTK formulation utilizes a local-linearization of the loss function for overparametrized neural networks, which reduces the optimization problem of DNNs to that of a *kernel regression* problem: The main insight is that when the network is *overparametrized*, i.e., sufficiently wide $m \gtrsim n^4$ ([67]), the neural network becomes locally convex and smooth, hence the problem is equivalent to a kernel regression problem with respect to the NTK function [38], and therefore solving the latter via (S)GD is guaranteed to converge to a global minimum. The training algorithm of [23] draws upon this equivalence, by designing a second-order variation of the Gauss-Newton algorithm (termed "Gram-Gauss-Newton"), yielding the aforementioned runtime for *smooth activation functions.*

Single vs. Multilayer Network Training. Following [23, 79], we focus on two-layer (i.e., single hidden-layer) neural networks. While our algorithm extends to the multilayer case (with a slight comprise on the width dependence), we argue that, as far as training time, the two-layer case is not only the common case, but in fact the *only* interesting case for constant training error: Indeed, in the multilayer case ($L \geq 2$), we claim that the mere cost of *feed-forward* computation of the network's output is already $\Omega_\epsilon(m^2nL)$. Indeed, the total number of parameters of L-layer networks is $M = (L-1)m^2 + md$, and as such, feed-forward computation requires, at the very least, computing a single product of $m \times m$ (dense) matrices W with a $m \times 1$ vector for each training data, which already costs m^2n time:

$$\widehat{y}_i = a^\top \sigma_L \left(\underbrace{W_L}_{m\times m} \sigma_{L-1} \left(\underbrace{W_{L-1}}_{m\times m} \dots \sigma_1 (\underbrace{W_1}_{m\times d} x_i) \right) \right) .$$

Therefore, sublinear-time techniques (as we present) appear futile in the case of multilayer overparametrized networks, where it is possible to achieve linear time (in M) using essentially direct (lossless) computation (see next subsection). It may still be possible to use sublinear algorithms to improve the running time to $O(m^2nL + \mathrm{poly}(n))$, though in for overparametrized DNNs this seems a minor saving.

1.1 Our Result

Our main result is a quadratic speedup to the algorithm of [23], yielding an *essentially optimal* training algorithm for overparametrized two-layer neural networks. Moreover, in contrast to [23], our algorithm applies to the more complex and realistic case of *ReLU* activation functions. Our main result is shown below (For a more comprehensive comparison, see Table 1 below and references therein).

▶ **Theorem 1.** *Suppose the width of a two layer ReLU neural network satisfies*

$$m = \Omega(\max\{\lambda^{-4}n^4, \lambda^{-2}n^2 d \log(n/\delta)\}),$$

where $\lambda > 0$ denotes the minimum eigenvalue of the Gram matrix (see Eq. (5) below), n is the number of training data, d is the input dimension. Then with probability $1-\delta$ over the random initialization of neural network and the randomness of the training algorithm, our algorithm achieves

$$\|f_{t+1} - y\|_2 \leq \frac{1}{2}\|f_t - y\|_2.$$

The computational cost of each iteration is $\widetilde{O}(mnd + n^3)$, and the running time for reducing the training loss to ϵ is $\widetilde{O}((mnd + n^3)\log(1/\epsilon))$. Using fast matrix-multiplication, the total running time can be further reduced to $\widetilde{O}((mnd + n^\omega)\log(1/\epsilon))$.[1]

▶ Remark 2. We stress that that our algorithm runs in (near) linear time even for networks with width $m \gtrsim n^2$ and in fact, under the common belief that $\omega = 2$, this is true so long as $m \gtrsim n$ (!). This means that the bottleneck for linear-time training of *small-width* DNNs is *not computational*, but rather *analytic*: The overparametrization requirements ($m \gtrsim n^4$) in Theorem 1 stems from current-best analysis of the convergence guarantees of (S)GD-based training of ReLU networks, and any improvement on these bounds would directly yield linear-time training for thinner networks using our algorithm.

[1] Here, $\omega < 2.373$ denotes the fast matrix-multiplication (FMM) constant for multiplying two $n \times n$ matrices [73, 46].

Table 1 Summary of state-of-art algorithms for training two-layer neural networks. n denotes the training batch size (number of input data points in $\mathbb{R}^d$) and ϵ denote the desired accuracy of the training loss. For simplicity, here we assume $d = O(1)$ and omit $\mathrm{poly}(\log n, 1/\lambda)$ terms. The result of [23] applies only to smooth activation gates and not to ReLU networks. Comparison to SGD algorithms is omitted from this table since they require a must stronger assumption on the width m for convergence, and have slower convergence rate than GD [48, 4, 5].

Ref.	**Method**	**#Iters**	**Cost/iter**	**Width**	**ReLU?**
[31]	Gradient descent	$O(n^2 \log(1/\epsilon))$	$O(mn)$	$\Omega(n^6)$	Yes
[67]	Gradient descent	$O(n^2 \log(1/\epsilon))$	$O(mn)$	$\Omega(n^4)$	Yes
[77]	Adaptive gradient descent	$O(n \log(1/\epsilon))$	$O(mn)$	$\Omega(n^6)$	Yes
[23]	Gram-Gaussian-Newton (GGN)	$O(\log\log(1/\epsilon))$	$O(mn^2)$	$\Omega(n^4)$	No
[23]	Batch-GGN	$O(n^2 \log(1/\epsilon))$	$O(m)$	$\Omega(n^{18})$	No
[79]	Natural gradient descent	$O(\log(1/\epsilon))$	$O(mn^2)$	$\Omega(n^4)$	Yes
Ours		$O(\log(1/\epsilon))$	$O(mn)$	$\Omega(n^4)$	Yes

Techniques. The majority of ML optimization literature on overparametrized network training is dedicated to understanding and minimizing the *number of iterations* of the training process [79, 23] as opposed to the *cost per iteration*, which is the focus of our paper. Our work shows that it is possible to harness the toolbox of *randomized linear algebra* – which was heavily used in the past decade to reduce the cost of *convex optimization* tasks – in the nonconvex setting of deep learning as well. A key ingredient in our algorithm is *linear sketching*, where the main idea is to carefully *compress* a linear system underlying an optimization problem, in a way that preserves a good enough solution to the problem yet can be solved much faster in lower dimension. This is the essence of the celebrated *Sketch-and-Solve* (S&S) paradigm [24]. As we explain below, our main *departure* from the classic S&S framework (e.g., [59]) is that we cannot afford to directly solve the underlying compressed regression problem (as this approach turns out to be prohibitively slow for our application). Instead, we use sketching (or sampling) to facilitate *fast preconditioning* of linear systems (in the spirit of [68, 43, 62, 74]), which in turn enables to solve the compressed regression problem to very high accuracy via first-order *conjugate* gradient descent. This approach essentially *decouples* the sketching error from the final precision error of the Gauss-Newton step, enabling a much smaller sketch size. We believe this (somewhat unconventional) approach to non-convex optimization is the most enduring message of our work.

1.2 Related Work

Second-order methods in non-convex optimization. Despite the prevalence of first order methods in deep learning applications, there is a vast body of ongoing work [18, 17, 55, 35, 36, 23, 79] aiming to design more scalable second-order algorithms that overcome the limitations of (S)GD for optimizing deep models. Grosse and Martens [55, 35] designed the K-FAC method, where the idea is to use Kronecker-factors to approximate the Fisher information matrix, combined with natural gradient descent. This approach has been further explored and extended by [78, 34, 54]. Gupta et al. [36] designed the "Shampoo method", based on the idea of *structure-aware preconditioning*. Anil et al. [7] further validate the practical perfromance of Shampoo and incorporated it into hardware. However, despite sporadic empirical evidence of such second-order methods (e.g., K-FAC and Shampoo), these methods generally lack a provable theoretical guarantee on the performance when applied to deep neural networks. Furthermore, in the overparametrized setting, their cost per-iteration in general is at least $\Omega(mn^2)$.

We remark that in the *convex* setting, theoretical guarantees for large-scale second-order algorithms have been established (e.g.,[1, 59, 56, 21]), but such rigorous analysis in non-convex setting was only recently proposed ([23, 79]). Our algorithm bears some similarities to the *NewtonSketch* algorithm of [59], which also incorporates sketching into second order Newton methods. A key difference, however, is that the algorithm of [59] works only for convex problems, and requires access to $(\nabla^2 f(x))^{1/2}$ (i.e., the square-root of the Hessian). Most importantly, though, [59] use the standard (black-box) Sketch-and-Solve paradigm to reduce the computational cost, while this approach incurs large computation overhead in our non-convex setting. By contrast, we use sketching as a subroutine for fast preconditioning. As a by-product, in the full version of this paper we show how to apply our techniques to give a substantial improvement over [59] in the *convex* setting.

The aforementioned works of [79] and [23] are most similar in spirit to ours. Zhang et al. [79] analyzed the convergence rate of Natural gradient descent algorithms for two-layer (overparametrized) neural networks, and showed that the number of iterations is *independent* of the training data size n (essentially $\log(1/\epsilon)$). They also demonstrate similar results for the convergence rate of K-FAC in the overparametrized regime, albeit with larger requirement on the width m. Another downside of K-FAC is the high cost per iteration ($\sim mn^2$). Cai et al. [23] analyzed the convergence rate of the so-called Gram-Gauss-Newton algorithm for training two-layer (overparametrized) neural network with *smooth* activation gates. They proved a quardratic (i.e., doubly-logarithnmic) convergence rate in this setting ($\log(\log(1/\epsilon))$) albeit with $O(mn^2)$ cost per iteration. It is noteworthy that this quadratic convergence rate analysis does not readily extend to the more complex and realistic setting of ReLU activation gates, which is the focus of our work. [23] also prove bounds on the convergence of "batch GGN", showing that it is possible to reduce the cost-per-iteration to m, at the price of $O(n^2 \log(1/\epsilon))$ iterations, for very heavily overparametrized DNNs (currently $m = \Omega(n^{18})$).

Sketching. The celebrated "Sketch and Solve" (S&S) paradigm [24] was originally developed to speed up the cost of solving linear regression and low-rank approximation problems. This dimensionality-reduction technique has since then been widely developed and applied to both convex and non-convex numerical linear algebra problems [20, 61, 76, 6, 14, 12, 72, 28, 65, 64, 15], as well as machine-learning applications [10, 11, 50, 75]. The most direct application of the sketch-and-solve technique is overconstrained regression problems, where the input is a linear system $[A, b] \in \mathbb{R}^{n\times(d+1)}$ with $n \gg d$, and we aim to find an (approximate) solution $\widehat{x} \in \mathbb{R}^d$ so as to minimize the residual error $\|A\widehat{x} - b\|_2$.

In the classic S&S paradigm, the underlying regression solver is treated as a *black box*, and the computational savings comes from applying it on a smaller *compressed* matrix. Since then, sketching (or sampling) has also been used in a non-black-box fashion for speeding-up optimization tasks, e.g., as a subroutine for preconditioning [74, 62, 68, 43] or fast inverse-maintenance in Linear Programming solvers, semi-definite programming, cutting plane methods, and empirical-risk minimization [25, 42, 40, 41, 47].

Overparametrization in neural networks. A long and active line of work in recent deep learning literature has focused on obtaining rigorous bounds on the convergence rate of various local-search algorithms for optimizing DNNs [48, 31, 4, 5, 8, 9, 67, 39]. The breakthrough work of Jacob et al. [38] and subsequent developments[2] introduced the notion of *neural tangent kernels* (NTK), implying that for wide enough networks ($m \gtrsim n^4$), (stochastic) gradient descent provably converges to an optimal solution, with generalization error independent of the number of network parameters.

[2] For a complete list of references, we refer the readers to [8, 9].

2 Technical Overview

We now provide a streamlined overview of our main result, Theorem 1. As discussed in the introduction, our algorithm extends to multi-layer ReLU networks , though we focus on the two-layer case (one-hidden layer), which is the most interesting case where one can indeed hope for linear training time.

The main, and most expensive step, of the GGN (or natural gradient descent) algorithms [23, 79] is multiplying, in each iteration t, the *inverse* of the Gram matrix $G_t := J_t J_t^\top$ with the Jacobian matrix $J_t \in \mathbb{R}^{n \times m}$, whose ith row contains the gradient of the $m = md$ network gates w.r.t the ith datapoint x_i (in our case, under ReLU activation).

Naiively computing G_t would already take mdn^2 time, however, the *tensor product* structure of the Jacobian J in fact allows to compute G_t in $n \cdot \mathcal{T}_{mat}(m, d, n) \ll mn^2$ time, where $\mathcal{T}_{mat}(m, d, n)$ is the cost of fast rectangular matrix multiplication[73, 46, 33].[3] Since the Gram-Gauss-Newton (GGN) algorithm requires $O(\log\log 1/\epsilon)$ iterations to converge to an ϵ-global minimum of the ℓ_2 loss [23], this observation yields an $O(n \cdot \mathcal{T}_{mat}(m, d, n) \log\log 1/\epsilon)$ total time algorithm for reducing the training loss below ϵ. While already nontrivial, this is still far from linear running time ($\gg mdn$).

We show how to carry out each Gauss-Newton iteration in time $\widetilde{O}(mnd + n^3)$, at the price of slightly compromising the number of iterations to $O(\log 1/\epsilon)$, which is inconsequential for the natural regime of constant dimension d and constant ϵ[4]. Our first key step is to reformulate the Gauss-Newton iteration (multiplying G_t^{-1} by the error vector) as an *ℓ_2-regression problem*:

$$\min_{g_t} \|J_t J_t^\top g_t - (f_t - y)\|_2 \tag{1}$$

where $(f_t - y)$ is the training error with respect to the network's output and the training labels y. Since the Gauss-Newton method is robust to small perturbation errors (essentially [71, 70]), our analysis shows that it is sufficient to find an approximate solution g_t' such that $J_t^\top g_t'$ satisfies

$$\|J_t J_t^\top g_t' - y\|_2 \le \gamma \|y\|_2, \quad \text{for} \quad \gamma \approx 1/n. \tag{2}$$

The benefit of this reformulation is that it allows to use *linear sketching* to first compress the linear system, significantly reducing the dimension of the optimization problem and thereby the cost of finding a solution, at the price of a small error in the found solution (this is the essence of the *sketch-and-solve* paradigm [24]). Indeed, a (variation of) the *Fast-JL* sketch [2, 52] guarantees that we can multiply the matrix $J_t^\top \in \mathbb{R}^{m \times n}$ by a much smaller $\widetilde{O}(n/\delta^2) \times m$ matrix S, such that (i) the multiplication takes near-linear time $\widetilde{O}(mn)$ time (using the FFT algorithm), and (ii) $SJ_t^\top$ is a δ-spectral approximation of $J_t^\top$ (i.e., $\|J_t S^\top S J_t^\top x\|_2 = (1 \pm \delta)\|G_t x\|_2$ for every x). Since both computing and inverting the matrix $\widetilde{G}_t := J_t S^\top S J_t^\top$ takes $\widetilde{O}(n^3/\delta^2)$ time, the overall cost of finding a δ-approximate solution to

[3] To see this, observe that the kronecker-product structure of J (here $J \in \mathbb{R}^{n \times md}$ can be constructed from an $n \times m$ matrix and an $n \times d$ matrix) allows computing Jh for any $h \in \mathbb{R}^{md}$ using fast rectangular matrix multiplication in time $\mathcal{T}_{mat}(m, d, n)$ which is near linear time in the dimension of J and h (that is, $n \times m + n \times d$ for J and md for h) so long as $d \le n^\alpha = n^{0.31}$ [33], hence computing $G = JJ^\top$ can be done using n independent invocations of the aforementioned subroutine, yielding $n \cdot \mathcal{T}_{mat}(m, d, n)$ as claimed.

[4] We also remark that this slowdown in the convergence rate is also a consequence of a direct extension of the analysis in [23] to ReLU activation functions.

the regression problem becomes at most $\widetilde{O}(mn + n^3/\delta^2)$. Alas, as noted in Equation (2), the approximation error of the found solution must be *polynomially small* $\gamma \sim 1/n$ in order to guarantee the desired convergence rate (i.e., constant decrease in training error per iteration). This means that we must set $\delta \sim \gamma \sim 1/n$, hence the cost of the naiive "sketch-and-solve" algorithm would be at least $\widetilde{O}(n^3/\delta^2) = \widetilde{O}(n^5)$, which is a prohibitively large overhead in both theory and practice (and in particular, no longer yields linear runtime whenever $m \ll n^4$ which is the current best overparametrization guarantee [67]). Since the $O(1/\delta^2)$ dependence of the JL embedding is known to be tight in general [45], this means we need to take a more clever approach to solve the regression (1). This is where our algorithm departs from the naiive sketch-and-solve method, and is the heart of our work.

Our key idea is to use dimension reduction – not to directly invert the compressed matrix – but rather to *precondition* it quickly. More precisely, our approach is to use a (conjugate) gradient-descent solver for the regression problem itself, with a fast preconditioning step, ensuring exponentially faster convergence to very high (polynomially small) accuracy. Indeed, conjugate gradient descent is guaranteed to find a γ-approximate solution to a regression problem $\min_x \|Ax - b\|_2$ in $O(\sqrt{\kappa} \log(1/\gamma))$ iterations, where $\kappa(A)$ is the *condition number* of A (i.e., the ratio of maximum to minimum eigenvalue). Therefore, if we can ensure that $\kappa(G_t)$ is small, then we can γ-solve the regression problem in $\sim mn \log(1/\gamma) = \widetilde{O}(mn)$ time, since the per-iteration cost of first-order SGD is linear ($\sim mn$).

The crucial advantage of our approach is that it *decouples* the sketching error from the final precision of the regression problem: Unlike the usual "sketch-and-solve" method, where the sketching error δ directly affects the overall precision of the solution to (2), here δ only affects the *quality of the preconditioner* (i.e., the ratio of max/min singular values of the sketch $\widetilde{G}_t$), hence it suffices to take a *constant* sketching error $\delta = 0.1$ (say), while letting the SGD deal with the final precision (at it has logarithmic dependence on γ).

Indeed, by setting the sketching error to $\delta = 0.1$ (say), the resulting matrix $\widetilde{G}_t = J_t S^\top S J_t^\top$ is small enough ($n \times \widetilde{O}(n)$) that we can afford running a standard (QR) algorithm to precondition it, at another $\widetilde{O}(n^3)$ cost per iteration. The output of this step is a matrix $\widetilde{G}'_t := \mathsf{Prec}(\widetilde{G}_t)$ with a *constant* condition number $\kappa(\widetilde{G}'_t)$ which preserves $\widetilde{G}'_t x \approx_{\ell_2} \widetilde{G}_t$ up to $(1 \pm \delta)^2$ relative error. At this point, we can run a (conjugate) gradient descent algorithm, which is guaranteed to find a $\gamma \approx 1/n$ approximate solution to (1) in time $\widetilde{O}((mn \log((1+\delta)/\gamma) + n^3)$, as desired.

We remark that, by definition, the preconditioning step (on the JL sketch) does *not* preserve the eigen-spectrum of G_t, which is in fact necessary to guarantee the fast convergence of the Gauss-Newton iteration . The point is that this preconditioning step is only preformed as a *local subroutine* so as to solve the regression problem, and does *not* affect the convergence rate of the outer loop.

3 Preliminaries

3.1 Model and Problem Setup

We denote by n the number of data points in the training batch, and by d the data dimension/feature-space (i.e., $x_i \in \mathbb{R}^d$). We denote by m the *width* of neural network, and by L the number of layers and by M the number of parameters. We assume the data has been normalized, i.e., $\|x\|_2 = 1$. We begin with the two-layer neural network in the following section, and then extend to multilayer networks. Consider a two-layer ReLU activated neural network with m neurons in the (single) hidden layer:

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

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, $a_1, \cdots, a_m \in \mathbb{R}$ are weights in the second layer. For simplicity, we consider $a \in \{-1,+1\}^m$ is fixed over all the iterations, this is natural in deep learning theory [48, 31, 4, 3, 67]. Recall the ReLU function $\phi(x) = \max\{x, 0\}$. Therefore for $r \in [m]$, we have

$$\frac{\partial f(W,x,a)}{\partial w_r} = \frac{1}{\sqrt{m}} a_r x \mathbf{1}_{w_r^\top x \geq 0}. \tag{3}$$

Given n input data points $(x_1,y_1),(x_2,y_2),\cdots(x_n,y_n) \in \mathbb{R}^d \times \mathbb{R}$. We define the objective function $\mathcal{L}$ as follows

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

We can compute the gradient of $\mathcal{L}$ in terms of w_r

$$\frac{\partial \mathcal{L}(W)}{\partial w_r} = \frac{1}{\sqrt{m}} \sum_{i=1}^n (f(W,x_i,a) - y_i) a_r x_i \mathbf{1}_{w_r^\top x_i \geq 0}. \tag{4}$$

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:

$$J_t = \frac{1}{\sqrt{m}} \begin{bmatrix} a_1 x_1^\top \mathbf{1}_{\langle w_1(t), x_1\rangle \geq 0} & a_2 x_1^\top \mathbf{1}_{\langle w_2(t), x_1\rangle \geq 0} & \cdots & a_m x_1^\top \mathbf{1}_{\langle w_m(t), x_1\rangle \geq 0} \\ a_1 x_2^\top \mathbf{1}_{\langle w_1(t), x_2\rangle \geq 0} & a_2 x_2^\top \mathbf{1}_{\langle w_2(t), x_2\rangle \geq 0} & \cdots & a_m x_2^\top \mathbf{1}_{\langle w_m(t), x_2\rangle \geq 0} \\ \vdots & \vdots & \ddots & \vdots \\ a_1 x_n^\top \mathbf{1}_{\langle w_1(t), x_n\rangle \geq 0} & a_2 x_n^\top \mathbf{1}_{\langle w_2(t), x_n\rangle \geq 0} & \cdots & a_m x_n^\top \mathbf{1}_{\langle w_m(t), x_n\rangle \geq 0} \end{bmatrix}.$$

The Gram matrix G_t is defined as $G_t = J_t J_t^\top$, whose (i,j)-th entry is $\left\langle \frac{f(W_t,x_i)}{\partial W}, \frac{f(W_t,x_j)}{\partial W} \right\rangle$. The crucial observation of [38, 31] is that the asymptotic of the Gram matrix equals a positive semidefinite kernel matrix $K \in \mathbb{R}^{n\times n}$, where

$$K(x_i,x_j) = \mathop{\mathbb{E}}_{w \in \mathcal{N}(0,1)} \left[x_i^\top x_j \mathbf{1}_{\langle w,x_i\rangle \geq 0, \langle w,x_j\rangle \geq 0} \right]. \tag{5}$$

▶ **Assumption 3.** *We assume the least eigenvalue λ of the kernel matrix K defined in Eq.* (5) *satisfies $\lambda > 0$.*

3.2 Subspace embedding

Subspace embedding was first introduced by Sarlós [63], it has been extensively used in numerical linear algebra field over the last decade [24, 57, 19, 66]. For a more detailed survey, we refer the readers to [74]. The formal definition is:

▶ **Definition 4** (Approximate subspace embedding, ASE [63]). *A $(1\pm\epsilon)$ ℓ_2-subspace embedding for the column space of an $N \times k$ matrix A is a matrix S for which for all $x \in \mathbb{R}^k$, $\|SAx\|_2^2 = (1\pm\epsilon)\|Ax\|_2^2$. Equivalently, $\|I - U^\top S^\top S U\|_2 \leq \epsilon$, where U is an orthonormal basis for the column space of A.*

Combining Fast-JL sketching matrix [2, 30, 69, 29, 52, 60] with a classical ϵ-net argument [74] gives subspace embedding,

▶ **Lemma 5** (Fast subspace embedding [52, 74]). *Given a matrix $A \in \mathbb{R}^{N \times k}$ with $N = \mathrm{poly}(k)$, then we can compute a $S \in \mathbb{R}^{k\mathrm{poly}(\log(k/\delta))/\epsilon^2 \times k}$ that gives a subspace embedding of A with probability $1-\delta$, i.e., with probability $1-\delta$, we have :*

$$\|SAx\|_2 = (1 \pm \epsilon)\|Ax\|_2$$

holds for any $x \in \mathbb{R}^n$, $\|x\|_2 = 1$. Moreover, SA can be computed in $O(Nk \cdot \mathrm{poly}\log k)$ time.

4 Our Algorithm

Our main algorithm is shown in Algorithm 1. We have the following convergence result of our algorithm.

▶ **Theorem 6.** *Suppose the width of a ReLU neural network satisfies*

$$m = \Omega(\max\{\lambda^{-4}n^4, \lambda^{-2}n^2 d \log(16n/\delta)\}),$$

then with probability $1-\delta$ over the random initialization of neural network and the randomness of the training algorithm, our algorithm (procedure FasterTwoLayer *in Algorithm 1) achieves*

$$\|f_{t+1} - y\|_2 \leq \frac{1}{2}\|f_t - y\|_2.$$

The computation cost in each iteration is $\widetilde{O}(mnd + n^3)$, and the running time for reducing the training loss to ϵ is $\widetilde{O}((mnd + n^3)\log(1/\epsilon))$. Using fast matrix-multiplication, the total running time can be further reduced to $\widetilde{O}((mnd + n^\omega)\log(1/\epsilon))$.

Algorithm 1 Faster algorithm for two-layer neural network.

```
1: procedure FasterTwoLayer()                                          ▷ Theorem 6
2:     $W_0$ is a random Gaussian matrix                                ▷ $W_0 \in \mathbb{R}^{md}$
3:     while $t < T$ do
4:         Compute the Jacobian matrix $J_t$                            ▷ $J_t \in \mathbb{R}^{n \times md}$
5:         Find an $\epsilon_0$ approximate solution using Algorithm 2  ▷ $\epsilon_0 \in (0, \frac{1}{6}\sqrt{\lambda/n}]$

               $\min_{g_t} \|J_t J_t^\top g_t - (f_t - y)\|_2$          (6)

6:         Update $W_{t+1} \leftarrow W_t - J_t^\top g_t$
7:         $t \leftarrow t + 1$
8:     end while
9: end procedure
```

The main difference between [23, 79] and our algorithm is that we perform an *approximate Newton update* (see line 6). The crucial observation here is that the Newton method is robust to small loss, thus it suffices to present a fine approximation. This observation is well-known in the convex optimization but unclear to the non-convex (but overparameterized) neural network setting. Another crucial observation is that instead of directly approximating the Gram matrix, it is suffices to approximate $(J_t J_t^\top)^{-1} g_t = G_t^{-1} g_t$. Intuitively, this follows from

$$J_t^\top g_t \approx J_t (J_t J_t^\top)^{-1}(f_t - y) = (J_t^\top J_t)^\dagger J_t (f_t - y),$$

where $(J_t^\top J_t)^\dagger$ denotes the pseudo-inverse of $J_t^\top J_t$ and the last term is exactly the Newton update. This observation allows us to formulate the problem a regression problem (see Eq. (6)), on which we can introduce techniques from *randomize linear algebra* and develop fast algorithm that solves it in near linear time.

4.1 Fast regression solver

Algorithm 2 Fast regression.

```
1:  procedure FastRegression($A, \epsilon$)                                    ▷ Lemma 7
2:                  ▷ $A \in \mathbb{R}^{N \times k}$ is a full rank matrix, $\epsilon \in (0, 1/2)$ is the desired precision
3:      Compute a subspace embedding $SA$                                  ▷ $S \in \mathbb{R}^{k\mathrm{poly}(\log k) \times k}$
4:      Compute $R$ such that $SAR$ orthonormal columns via QR decomposition ▷ $R \in \mathbb{R}^{k \times k}$
5:      $z_0 \leftarrow \vec{0} \in \mathbb{R}^k$
6:      while $\|A^\top A R z_t - y\|_2 \geq \epsilon$ do
7:          $z_{t+1} \leftarrow z_t - (R^\top A^\top A R)^\top (R^\top A^\top A R z_t - R^\top y)$
8:      end while
9:  return $Rz_t$
10: end procedure
```

The core component of our algorithm is a fast regression solver (shown in Algorithm 2). The regression solver provides an approximate solution to $\min_x \|A^\top A x - y\|$ where $A \in \mathbb{R}^{N \times k}$ $(N \gg k)$. We perform preconditioning on the matrix of $A^\top A$ (line 3 – 4) and use gradient descent to derive an approximation solution (line 6 – 8).

▶ **Lemma 7.** *Let $N = \Omega(k\mathrm{poly}(\log k))$. Given a matrix $A \in \mathbb{R}^{N \times k}$, let κ denote the condition number of A* [5]*, consider the following regression problem*

$$\min_{x \in \mathbb{R}^k} \|A^\top A x - y\|_2. \tag{7}$$

Using procedure FastRegression *(in Algorithm 2), with probability $1 - \delta$, we can compute an ϵ-approximate solution x' satisfying*

$$\|A^\top A x' - y\|_2 \leq \epsilon \|y\|_2$$

in $\widetilde{O}\left(Nk \log(\kappa/\epsilon) + k^3\right)$ time.

Speedup in Convex Optimization. It should come as no surprise that our techniques can help accelerating a broad class of solvers in *convex optimization* problems as well. In the full version of this paper, we elaborate on this application, and in particular show how our technique improves the runtime of the "Newton-Sketch" algorithm of [59].

5 Conclusion and Open Problems

Our work provides a computationally-efficient (near-linear time) second-order algorithm for training sufficiently overparametrized two-layer neural network, overcoming the drawbacks of traditional first-order gradient algorithms. Our main technical contribution is developing a *faster regression solver* which uses linear sketching for fast preconditioning (in time

[5] $\kappa = \sigma_{\max}(A)/\sigma_{\min}(A)$

independent of the network width). As such, our work demonstrates that the toolbox of randomized linear algebra can substantially reduce the computational cost of second-order methods in *non-convex optimization*, and not just in the convex setting for which it was originally developed (e.g., [59, 74, 25, 42, 40, 41, 47]).

Finally, we remark that, while the running time of our algorithm is $\widetilde{O}(Mn + n^3)$ (or $O(Mn + n^\omega)$ using FMM), it is no longer (near) linear for networks with parameters $M \leq n^2$ (resp. $M \lesssim n^{\omega-1}$). While it is widely believed that $\omega = 2$ [26], FMM algorithms are impractical at present, and it would therefore be very interesting to improve the extra additive term from n^3 to $n^{2+o(1)}$ (which seems best possible for dense $n \times n$ matrices), or even to $n^{3-\epsilon}$ using a practically viable algorithm. Faster preconditioners seem key to this avenue.

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 Nir Ailon and Bernard Chazelle. Approximate nearest neighbors and the fast johnson-lindenstrauss transform. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing (STOC)*, pages 557–563, 2006.

3 Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. In *Advances in neural information processing systems (NeurIPS)*, pages 6155–6166, 2019.

4 Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *ICML*, volume 97, pages 242–252. PMLR, 2019. *arXiv version:* `https://arxiv.org/pdf/1811.03962.pdf`.

5 Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. On the convergence rate of training recurrent neural networks. In *NeurIPS*, pages 6673–6685, 2019. *arXiv version:* `https://arxiv.org/pdf/1810.12065.pdf`.

6 Alexandr Andoni, Chengyu Lin, Ying Sheng, Peilin Zhong, and Ruiqi Zhong. Subspace embedding and linear regression with orlicz norm. In *ICML*, pages 224–233. PMLR, 2018. *arXiv versuion:* `https://arXiv.org/pdf/1806.06430.pdf`.

7 Rohan Anil, Vineet Gupta, Tomer Koren, Kevin Regan, and Yoram Singer. Second order optimization made practical. *arXiv preprint*, 2020. `arXiv:arXiv:2002.09018`.

8 Sanjeev Arora, Simon Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pages 322–332. PMLR, 2019. *arXiv version:* `https://arxiv.org/pdf/1901.08584.pdf`.

9 Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 8139–8148, 2019. *arXiv version:* `https://arxiv.org/pdf/1904.11955.pdf`.

10 Haim Avron, Michael Kapralov, Cameron Musco, Christopher Musco, Ameya Velingker, and Amir Zandieh. Random fourier features for kernel ridge regression: Approximation bounds and statistical guarantees. In *ICML*, volume 70, pages 253–262. PMLR, 2017. *arXiv version:* `https://arxiv.org/pdf/1804.09893.pdf`.

11 Haim Avron, Michael Kapralov, Cameron Musco, Christopher Musco, Ameya Velingker, and Amir Zandieh. A universal sampling method for reconstructing signals with simple fourier transforms. In *STOC*. ACM, 2019. arXiv version: `https://arxiv.org/pdf/1812.08723.pdf`. `doi:10.1145/3313276.3316363`.

12 Ainesh Bakshi, Nadiia Chepurko, and David P Woodruff. Robust and sample optimal algorithms for psd low-rank approximation. *arXiv preprint*, 2019. `arXiv:1912.04177`.

13 Ainesh Bakshi, Rajesh Jayaram, and David P Woodruff. Learning two layer rectified neural networks in polynomial time. In *COLT*, volume 99, pages 195–268. PMLR, 2019. *arXiv version:* `https://arxiv.org/pdf/1811.01885.pdf`.

14 Ainesh Bakshi and David Woodruff. Sublinear time low-rank approximation of distance matrices. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3782–3792, 2018. *arXiv version:* `https://arxiv.org/pdf/1809.06986.pdf`.

15 Frank Ban, David P. Woodruff, and Richard Zhang. Regularized weighted low rank approximation. In *NeurIPS*, pages 4061–4071, 2020. *arXiv version:* `https://arxiv.org/pdf/1911.06958.pdf`.

16 Sue Becker and Yann Le Cun. Improving the convergence of back-propagation learning with second order methods. In *Proceedings of the 1988 connectionist models summer school*, pages 29–37, 1988.

17 Alberto Bernacchia, Máté Lengyel, and Guillaume Hennequin. Exact natural gradient in deep linear networks and its application to the nonlinear case. In *Advances in Neural Information Processing Systems (NIPS)*, pages 5941–5950, 2018.

18 Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical gauss-newton optimisation for deep learning. In *International Conference on Machine Learning (ICML)*, pages 557–565, 2017.

19 Christos Boutsidis and David P Woodruff. Optimal cur matrix decompositions. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 353–362. ACM, 2014. *arXiv version:* `https://arxiv.org/pdf/1405.7910.pdf`.

20 Christos Boutsidis, David P Woodruff, and Peilin Zhong. Optimal principal component analysis in distributed and streaming models. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing (STOC)*, pages 236–249, 2016.

21 Sébastien Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.

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

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

24 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. ACM, 2013. *arXiv version:* `https://arxiv.org/pdf/1207.6365.pdf`.

25 Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. In *STOC*, pages 938–942. ACM, 2019. *arXiv version:* `https://arxiv.org/pdf/1810.07896.pdf`.

26 Henry Cohn, Robert Kleinberg, Balazs Szegedy, and Christopher Umans. Group-theoretic algorithms for matrix multiplication. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 379–388. IEEE, 2005.

27 Amit Daniely. Memorizing gaussians with no over-parameterizaion via gradient decent on neural networks. *arXiv preprint*, 2020. `arXiv:2003.12895`.

28 Huaian Diao, Rajesh Jayaram, Zhao Song, Wen Sun, and David Woodruff. Optimal sketching for kronecker product regression and low rank approximation. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 4739–4750, 2019. *arXiv version:* `https://arxiv.org/pdf/1909.13384.pdf`.

29 Petros Drineas, Malik Magdon-Ismail, Michael W Mahoney, and David P Woodruff. Fast approximation of matrix coherence and statistical leverage. *Journal of Machine Learning Research*, 13(Dec):3475–3506, 2012.

30 Petros Drineas, Michael W Mahoney, and Shan Muthukrishnan. Sampling algorithms for l2 regression and applications. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 1127–1136. Society for Industrial and Applied Mathematics, 2006.

31 Simon S Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. In *ICLR*. OpenReview.net, 2019. *arXiv version:* `https://arxiv.org/pdf/1810.02054.pdf`.

32 John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research (JMLR)*, 12(Jul):2121–2159, 2011.

33 François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1029–1046. SIAM, 2018. *arXiv version:* `https://arxiv.org/pdf/1708.05622.pdf`.

34 Thomas George, César Laurent, Xavier Bouthillier, Nicolas Ballas, and Pascal Vincent. Fast approximate natural gradient descent in a kronecker factored eigenbasis. In *Advances in Neural Information Processing Systems (NIPS)*, pages 9550–9560, 2018.

35 Roger Grosse and James Martens. A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning (ICML)*, pages 573–582, 2016.

36 Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning (ICML)*, pages 1842–1850, 2018.

37 Moritz Hardt, Benjamin Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. In *ICML*, volume 48, pages 1225–1234. JMLR.org, 2016. *arXiv version:* `https://arxiv.org/pdf/1509.01240.pdf`.

38 Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems (NIPS)*, pages 8571–8580, 2018.

39 Ziwei Ji and Matus Telgarsky. Polylogarithmic width suffices for gradient descent to achieve arbitrarily small test error with shallow relu networks. In *ICLR*. OpenReview.net, 2020. *arXiv version:* `https://arxiv.org/pdf/1909.12292.pdf`.

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

41 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*, pages 944–953. ACM, 2020. *arXiv version:* `https://arxiv.org/pdf/2004.04250.pdf`.

42 Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. Faster dynamic matrix inverse for faster lps. *arXiv preprint*, 2020. `arXiv:2004.07470`.

43 Jonathan A Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving sdd systems in nearly-linear time. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing (STOC)*, pages 911–920. ACM, 2013. *arXiv version:* `https://arxiv.org/pdf/1301.6628.pdf`.

44 Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. *arXiv versuion:* `https://arxiv.org/pdf/1412.6980.pdf`.

45 Kasper Green Larsen and Jelani Nelson. Optimality of the johnson-lindenstrauss lemma. In *IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 633–638. IEEE, 2017. *arXiv version:* `https://arxiv.org/pdf/1609.02094.pdf`.

46 François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th international symposium on symbolic and algebraic computation (ISSAC)*, pages 296–303. ACM, 2014.

47 Yin Tat Lee, Zhao Song, and Qiuyi Zhang. Solving empirical risk minimization in the current matrix multiplication time. In *COLT*. `https://arxiv.org/pdf/1905.04447`, 2019.

48 Yuanzhi Li and Yingyu Liang. Learning overparameterized neural networks via stochastic gradient descent on structured data. In *NeurIPS*, pages 8168–8177, 2018. *arXiv version:* `https://arxiv.org/pdf/1808.01204.pdf`.

49 Yuanzhi Li and Yang Yuan. Convergence analysis of two-layer neural networks with ReLU activation. In *Advances in neural information processing systems (NIPS)*, pages 597–607, 2017. *arXiv version:* `https://arxiv.org/pdf/1705.09886.pdf`.

50 Hang Liao, Barak A. Pearlmutter, Vamsi K. Potluru, and David P. Woodruff. Automatic differentiation of sketched regression. In *AISTATS*, 2020.

51 Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint*, 2019. `arXiv:1908.03265`.

52 Yichao Lu, Paramveer Dhillon, Dean P Foster, and Lyle Ungar. Faster ridge regression via the subsampled randomized hadamard transform. In *Advances in neural information processing systems*, pages 369–377, 2013.

53 James Martens. Deep learning via hessian-free optimization. In *ICML*, volume 27, pages 735–742, 2010.

54 James Martens, Jimmy Ba, and Matthew Johnson. Kronecker-factored curvature approximations for recurrent neural networks. In *ICLR*, 2018.

55 James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning (ICML)*, pages 2408–2417, 2015.

56 Philipp Moritz, Robert Nishihara, and Michael Jordan. A linearly-convergent stochastic l-bfgs algorithm. In *Artificial Intelligence and Statistics*, pages 249–258, 2016.

57 Jelani Nelson and Huy L Nguyên. Osnap: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 117–126. IEEE, 2013. *arXiv version:* `https://arxiv.org/pdf/1211.1002.pdf`.

58 Samet Oymak and Mahdi Soltanolkotabi. Towards moderate overparameterization: global convergence guarantees for training shallow neural networks. *IEEE Journal on Selected Areas in Information Theory*, 2019.

59 Mert Pilanci and Martin J Wainwright. Newton sketch: A near linear-time optimization algorithm with linear-quadratic convergence. *SIAM Journal on Optimization*, 27(1):205–245, 2017.

60 Eric Price, Zhao Song, and David P. Woodruff. Fast regression with an ℓ_∞ guarantee. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 80 of *LIPIcs*, pages 59:1–59:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. *arXiv version:* `https://arxiv.org/pdf/1705.10723.pdf`. `doi:10.4230/LIPIcs.ICALP.2017.59`.

61 Ilya Razenshteyn, Zhao Song, and David P Woodruff. Weighted low rank approximations with provable guarantees. In *Proceedings of the 48th Annual Symposium on the Theory of Computing (STOC)*, 2016.

62 Vladimir Rokhlin and Mark Tygert. A fast randomized algorithm for overdetermined linear least-squares regression. *Proceedings of the National Academy of Sciences*, 105(36):13212–13217, 2008.

63 Tamás Sarlós. Improved approximation algorithms for large matrices via random projections. In *Proceedings of 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2006.

64 Zhao Song. *Matrix Theory : Optimization, Concentration and Algorithms.* PhD thesis, The University of Texas at Austin, 2019.

65 Zhao Song, Ruosong Wang, Lin Yang, Hongyang Zhang, and Peilin Zhong. Efficient symmetric norm regression via linear sketching. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 828–838, 2019. *arXiv version:* `https://arxiv.org/pdf/1910.01788.pdf`.

66 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. *arXiv version:* `https://arxiv.org/pdf/1704.08246.pdf`.

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

68 Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing (STOC)*, pages 81–90. ACM, 2004.

69 Joel A Tropp. Improved analysis of the subsampled randomized hadamard transform. *Advances in Adaptive Data Analysis*, 3(01n02):115–126, 2011.

70 Pravin M Vaidya. A new algorithm for minimizing convex functions over convex sets. In *30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 338–343. IEEE, 1989.

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

72 Ruosong Wang and David P Woodruff. Tight bounds for ℓ_p oblivious subspace embeddings. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1825–1843. SIAM, 2019. *arXiv version:* `https://arxiv.org/pdf/1801.04414.pdf`.

73 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing (STOC)*, pages 887–898. ACM, 2012.

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

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

76 David P Woodruff and Peilin Zhong. Distributed low rank approximation of implicit functions of a matrix. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 847–858. IEEE, 2016.

77 Xiaoxia Wu, Simon S Du, and Rachel Ward. Global convergence of adaptive gradient methods for an over-parameterized neural network. *arXiv preprint*, 2019. `arXiv:1902.07111`.

78 Yuhuai Wu, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems (NIPS)*, pages 5279–5288, 2017.

79 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)*, pages 8080–8091, 2019.

80 Kai Zhong, Zhao Song, and Inderjit S Dhillon. Learning non-overlapping convolutional neural networks with multiple kernels. *arXiv preprint*, 2017. `arXiv:1711.03440`.

81 Kai Zhong, Zhao Song, Prateek Jain, Peter L. Bartlett, and Inderjit S. Dhillon. Recovery guarantees for one-hidden-layer neural networks. In *ICML*, volume 70, pages 4140–4149. PMLR, 2017. *arXiv version:* `https://arxiv.org/pdf/1706.03175.pdf`.