OverSketched Newton: Fast Convex Optimization for Serverless Systems

Vipul Gupta[1], Swanand Kadhe[1], Thomas Courtade[1], Michael Mahoney[2], and Kannan Ramchandran[1]

[1]Department of EECS, University of California, Berkeley
[2]ICSI and Statistics Department, University of California, Berkeley

Abstract

We develop OverSketched Newton, a randomized Hessian-based optimization algorithm to solve large-scale smooth and strongly-convex problems in serverless systems. OverSketched Newton leverages matrix sketching ideas from Randomized Numerical Linear Algebra to compute the Hessian approximately. The use of such sketching schemes also leads to inbuilt resiliency against stragglers that are a characteristic of serverless architectures. We establish that OverSketched Newton has a linear-quadratic convergence rate, and we empirically validate our results by solving large-scale supervised learning problems on real-world datasets. Experiments demonstrate a reduction of ∼50% in total running time on AWS Lambda compared to state-of-the-art distributed optimization schemes.

1 Introduction

Compared to first-order optimization algorithms, second-order methods — which use the gradient as well as Hessian information — enjoy superior convergence rates both in theory and practice. For instance, Newton's method converges quadratically for strongly convex and smooth problems compared to the linear convergence of gradient descent. Moreover, second-order methods do not require step-size tuning and unit step-size provably works for most problems. These methods, however, are not vastly popular among practitioners due to higher computational complexity as they require the calculation of Hessian in each iteration, which can be prohibitive when the training data is large.

Recently, there has been a lot of interest in distributed second-order optimization methods for server-based systems, as they substantially decrease the number of iterations and hence reduce communication at the cost of more computation per iteration [1–6], which is desired in distributed computing. However, these methods are approximate as they do not calculate the exact Hessian due to data being stored distributedly among workers, and each worker calculates an approximation of the Hessian using the data that is stored locally. Such methods, while still better than gradient descent, forego the quadratic convergence property enjoyed by exact Newton's method.

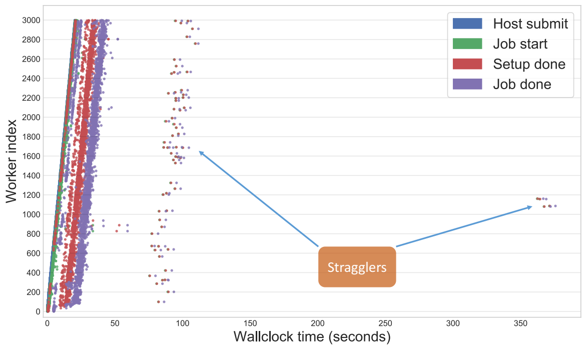


Figure 1: Job times for 3000 AWS Lambda nodes where the median job time is around 40 seconds, and around 5% of the nodes take 100 seconds, and two nodes take as much as 375 seconds to complete the same job [11].

In recent years, there has been tremendous growth in users performing distributed computing operations on the cloud due to extensive and inexpensive commercial offerings like Amazon Web Services (AWS), Google Cloud, Microsoft Azure, etc. Serverless platforms—such as AWS Lambda and Cloud functions—penetrate a large user base by provisioning and managing the servers on which the computation is performed. This abstracts away the need for maintaining servers since this is done by the cloud provider and is hidden from the user– hence the name serverless. Moreover, allocation of these servers is done expeditiously which provides greater elasticity and easy scalability. For example, up to ten thousand machines can be allocated on AWS Lambda in less than ten seconds [7–9]. Using serverless computation for large-scale computation has been the central theme of several recent works [10–13].

Due to several crucial differences between HPC/server-based and serverless architectures, existing distributed algorithms cannot, in general, be extended to serverless computing. Unlike server-based computing, the number of inexpensive workers in serverless platforms is flexible, often scaling into the thousands [9]. This heavy gain in the computation power, however, comes with the following two challenges.

The first challenge is that, unlike cluster nodes, the commodity workers in the serverless architecture are ephemeral, have low memory and do not communicate amongst themselves[1]. The workers read/write data directly from/to a single data storage entity (for example, cloud storage like AWS S3).

Second, unlike HPC/server-based systems, nodes in the serverless systems suffer degradation due to system noise which can be a result of limited availability of shared resources, hardware failure, network latency, etc. [14, 15]. This results in job time variability, and hence a subset of slower nodes often called *stragglers*, which significantly slow the computation, especially in large or iterative jobs. Empirical statistics for worker job times for 3000 workers for a representative example are shown in Fig. 1 for the AWS Lambda system. These experiments consistently demonstrate that ∼5% workers take significantly longer than the median job time, severely degrading the overall efficiency of the system.

Due to the aforementioned issues, first-order methods tend to perform poorly on distributed serverless architectures. In fact, their slower convergence is made worse on serverless platforms due to persistent stragglers. The straggler effect incurs heavy slow down due to the accumulation of tail times as a result of a subset of slow workers occurring in each iteration. Moreover, each call to serverless computing platforms such as AWS Lambda requires invocation time (during which AWS assigns Lambda workers), setup time (where the required python packages are downloaded) and communication time (where the data is downloaded from the cloud). The ephemeral nature of the workers in serverless systems requires that new workers should be invoked every few iterations and data should be communicated to them.

In this paper, we argue that second-order methods are highly compatible with serverless systems that provide extensive computing power by invoking thousands of workers but are limited by the communication costs and hence the number of iterations [9, 10]. To address the challenges of ephemeral workers and stragglers in serverless systems, we propose and analyze a randomized and distributed second-order optimization algorithm, called *OverSketched Newton*. OverSketched Newton uses the technique of matrix sketching from Randomized Numerical Linear Algebra (RandNLA) [16, 17] to obtain a good approximation for the Hessian. In particular, we use the sparse sketching scheme proposed by [11], which is based on approximate randomized matrix multiplication [18], for straggler-resilient Hessian calculation in serverless systems. Such randomization also provides inherent straggler resiliency while calculating the Hessian. For straggler mitigation during gradient calculation, we use the recently proposed technique of employing error-correcting codes to create redundant computation [19, 20]. We prove that OverSketched Newton has a linear-quadratic convergence when the Hessian is calculated using the sparse sketching scheme proposed in [11]. Our experiments on AWS Lambda demonstrate empirically that OverSketched Newton is significantly faster than existing distributed optimization schemes and vanilla Newton's method that calculates exact Hessian.

1.1 Related Work

Existing Straggler Mitigation Schemes: Strategies like speculative execution have been traditionally used to mitigate stragglers in popular distributed computing frameworks like Hadoop MapReduce [21] and Apache Spark [22]. Speculative execution works by detecting workers that are running slower than expected and then allocating their tasks to new workers without shutting down the original straggling task. The worker that finishes first communicates its results. This has several drawbacks. For example, constant monitoring of tasks is required, where the worker pauses its job and provides its running status. Additionally, it is possible that a worker will straggle only at the end of the task, say, while communicating the results. By the time the task is reallocated, the overall efficiency of the system would have suffered already.

Recently, many coding-theoretic ideas have been proposed to introduce redundancy into the distributed computation for straggler mitigation [19, 20, 23–27], many of them catering to distributed matrix-vector multiplication [19, 20, 28]. We use tools from [20] to compute gradients in a distributed straggler-resilient manner, and compare the performance with speculative execution.

Approximate Newton Methods: Several works in the literature prove convergence guarantees for Newton's method when the Hessian is computed approximately using ideas from RandNLA [29–31]. However, these algorithms are designed for a single machine. Our goal in this paper is to use ideas from RandNLA to design a distributed approximate Newton method for a serverless system.

Distributed Second-Order Methods: There has been a growing research interest in designing and analyzing distributed (synchronous) implementations of second-order methods [1–6]. However, these implementations are tailored for server-based distributed systems. Our focus, on the other hand, is on serverless systems. Our motivation behind considering serverless systems stems from their usability benefits, cost efficiency, and extensive and inexpensive commercial offerings [9, 10]. We implement our algorithms using

[1]For example, AWS Lambda nodes have a memory of 3 GB and a maximum runtime of 300 seconds (may change over time).

the recently developed serverless framework called PyWren [9]. While there are works that evaluate existing algorithms on serverless systems [12, 32], this is the first work that proposes a large-scale distributed optimization algorithm for serverless systems. We exploit the advantages offered serverless systems while mitigating the drawbacks such as stragglers and additional overhead per invocation of workers.

2 OverSketched Newton

We are interested in solving a problem of the following form distributedly on serverless systems in a straggler-resilient fashion

$$f(\mathbf{w}^*) = \min_{\mathbf{w}\in\mathcal{C}} f(\mathbf{w}), \tag{1}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is closed and twice-differentiable convex function bounded from below, and $\mathcal{C} \subseteq \mathbb{R}^d$ is a given convex and closed set. We assume that the minimizer $\mathbf{w}^*$ exists and is uniquely defined. Let γ and β denote, respectively, the maximum and minimum eigenvalues of the Hessian evaluated at the minimum, i.e., $\nabla^2 f(\mathbf{w}^*)$. In addition, it is assumed that the Hessian $\nabla^2 f(\mathbf{w})$ is Lipschitz continuous with modulus L, that is

$$||\nabla^2 f(\mathbf{w}+\boldsymbol{\Delta}) - \nabla^2 f(\mathbf{w})||_{op} \le L||\boldsymbol{\Delta}||_2, \tag{2}$$

where $||\cdot||_{op}$ is the operator norm. Newton's update at the $(t+1)$-th iteration is obtained by minimizing the Taylor's expansion of the objective function $f(\cdot)$ at $\mathbf{w}_t$ in the constraint set $\mathcal{C}$, that is

$$\begin{aligned}\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}\in\mathcal{C}} \Big\{ & f(\mathbf{w}_t) + \nabla f(\mathbf{w}_t)^T(\mathbf{w}-\mathbf{w}_t) \\ & + \frac{1}{2}(\mathbf{w}-\mathbf{w}_t)^T\nabla^2 f(\mathbf{w}_t)(\mathbf{w}-\mathbf{w}_t)\Big\}.\end{aligned} \tag{3}$$

For the unconstrained case, that is when $\mathcal{C} = \mathbb{R}^d$, Eq. (3) becomes

$$\mathbf{w}_{t+1} = \mathbf{w}_t - [\nabla^2 f(\mathbf{w}_t)]^{-1}\nabla f(\mathbf{w}_t). \tag{4}$$

Given a good initial point $\mathbf{w}_0$ such that

$$||\mathbf{w}_0 - \mathbf{w}^*||_2 \le \frac{\gamma}{2L}, \tag{5}$$

Newton's method satisfies the following update

$$||\mathbf{w}_{t+1} - \mathbf{w}^*||_2 \le \frac{2L}{\gamma}||\mathbf{w}_t - \mathbf{w}^*||_2^2, \tag{6}$$

implying quadratic convergence [33, 34].

In many applications like machine learning where the training data itself is noisy, exact multiplication is not necessary. Many results in the literature prove convergence guarantees for Newton's method when the Hessian is computed approximately using ideas from RandNLA for a single machine [29–31]. In particular, these methods perform a form of dimensionality reduction for the Hessian using random matrices, called sketching matrices. Many popular sketching schemes have been proposed in the literature, for example, sub-Gaussian, Hadamard, random row sampling, sparse Johnson-Lindenstrauss, etc. [16, 35].

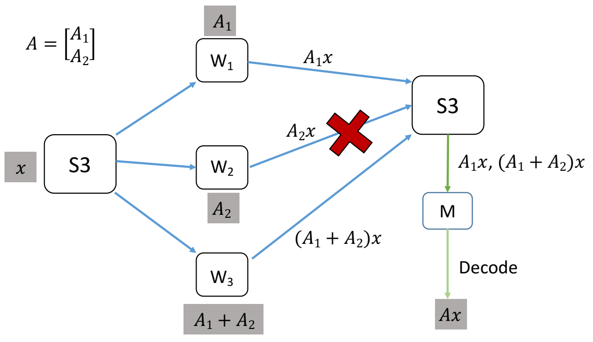


Figure 2: **Coded matrix-vector multiplication**: Matrix $\mathbf{A}$ is divided into 2 row chunks $\mathbf{A}_1$ and $\mathbf{A}_2$. During encoding, redundant chunk $\mathbf{A}_1 + \mathbf{A}_2$ is created. Three workers obtain $\mathbf{A}_1$,$\mathbf{A}_2$ and $\mathbf{A}_1 + \mathbf{A}_2$ from the cloud storage S3, respectively, and multiply by $\mathbf{x}$ and write back the result to the cloud. The master M can decode $\mathbf{Ax}$ from the results of any two workers, thus being resilient to one straggler (W_2 in this case).

Next, we present OverSketched Newton for solving problems of the form (1) distributedly in serverless systems using ideas from RandNLA.

2.1 OverSketched Newton: Description

OverSketched Newton computes the full gradient in each iteration by using tools from error-correcting codes [19, 20, 23]. The exact method of obtaining the full gradient in a distributed straggler-resilient way depends on the optimization problem at hand. Our key observation is that, for several commonly encountered optimization problems, gradient computation relies on matrix-vector multiplication for a large matrix (see Sec. 3 for examples). We then leverage coded matrix multiplication technique from [20] to perform the matrix-vector multiplication in a distributed straggler-resilient manner. The main idea of coded matrix multiplication is explained in Fig. 2; see [20] for details.

For computing the Hessian approximately, we propose to use the sketching matrix called *OverSketch* from [11]. OverSketch is a sparse sketching matrix based on Count-Sketch [35]. It has similar accuracy guarantees as that of the Count-Sketch with two additional properties: it is amenable to distributed implementation and resilient to stragglers. In particular, using OverSketch, a $d \times n$ matrix $\mathbf{A}$ can be sketched into a $d \times m$ matrix, where $d \ll n$, in a distributed way such that each worker takes $O(nnz(A))$ time.[2] Here,

[2]This can be further improved by dividing $\mathbf{A}$ into row-blocks and employing more workers. Note that this is better than

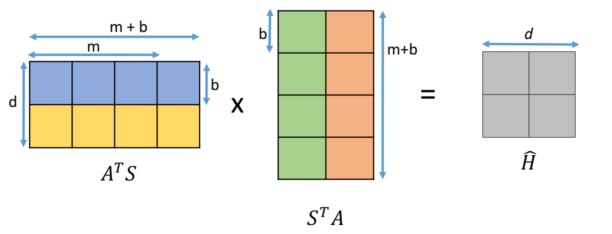


Figure 3: **OverSketch-based approximate Hessian computation:** First, the matrix $\mathbf{A}$—satisfying $\mathbf{A}^T\mathbf{A} = \nabla^2 f(\mathbf{w}_t)$—is sketched in parallel using the sketch in (7). Then, each worker receives a block each of the sketched matrices $\mathbf{A}^T\mathbf{S}$ and $\mathbf{S}^T\mathbf{A}$, multiplies them, and communicates back its results for reduction. During reduction, stragglers can be ignored by the virtue of "over" sketching. For example, here the desired sketch dimension m is increased by b for obtaining straggler resiliency.

$nnz(A)$ is the number of non-zero entries in A. More specifically, the OverSketch matrix is given as [11]:

$$\mathbf{S} = \frac{1}{\sqrt{N}}(\mathbf{S}_1, \mathbf{S}_2, \cdots, \mathbf{S}_{N+e}), \tag{7}$$

where $\mathbf{S}_i \in \mathbb{R}^{n\times b}$, for all $i \in [1, N+e]$, are i.i.d. Count-Sketch matrices with sketch dimension b, and e is the number of stragglers per $N+e$ blocks. Each of the Count-Sketch matrices $\mathbf{S}_i$ is constructed (independently of others) as follows. For every row j, $j \in [n]$, of $\mathbf{S}_i$, first independently choose a column $h(j) \in [b]$. Then, select a uniformly random element from $\{-1, +1\}$, denoted as $\sigma(i)$. Finally, set $\mathbf{S}_i(j, h(j)) = \sigma(i)$ and set $\mathbf{S}_i(j, l) = 0$ for all $l \neq h(j)$ (see [11, 35] for details).

Similar to the gradient computation, the particular method of obtaining the sketched Hessian in a distributed straggler-resilient way depends on the optimization problem at hand. For several commonly encountered optimization problems, Hessian computation involves matrix-matrix multiplication for a pair of large matrices (see Sec. 3 for several example problems). Thus, we can leverage the straggler resiliency of OverSketch to obtain the sketched Hessian in a distributed straggler-resilient manner. An illustration is provided in Fig. 3. For details, see Algorithm 3 in [11].

The model update for OverSketched Newton is given by

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}\in\mathcal{C}} \Big\{ f(\mathbf{w}_t) + \nabla f(\mathbf{w}_t)^T(\mathbf{w} - \mathbf{w}_t) + \frac{1}{2}(\mathbf{w} - \mathbf{w}_t)^T \hat{\mathbf{H}}_t (\mathbf{w} - \mathbf{w}_t) \Big\}, \tag{8}$$

where $\hat{\mathbf{H}}_t = \mathbf{A}_t^T \mathbf{S}_t \mathbf{S}_t^T \mathbf{A}_t$, $\mathbf{A}_t$ is the square root of the Hessian $\nabla^2 f(\mathbf{w}_t)$, and $\mathbf{S}_t$ is an independent realization of (7) at the t-th iteration.

Next, we prove the following convergence guarantee for OverSketched Newton, that uses the sketch matrix in (7) and full gradient for approximate Hessian computation. We refer m to be the resultant sketch dimension of $\mathbf{S}$ obtained after ignoring the stragglers, that is, $m = Nb$.

Theorem 2.1. *Let $\mathbf{w}^*$ be the optimal solution of* (1) *and γ and β be the minimum and maximum eigenvalues of $\nabla^2 f(\mathbf{w}^*)$, respectively. For $\epsilon \in (0, 1/2]$ and a sketch dimension $m = \Omega(\frac{d^{1+\mu}}{\epsilon^2})$ for OverSketch with column-blocks $N = \Theta_\mu(1/\epsilon)$, the updates for OverSketched Newton with initialization satisfying* (5) *follow*

$$||\mathbf{w}_{t+1} - \mathbf{w}^*||_2 \leq \frac{5L}{\gamma}||\mathbf{w}_t - \mathbf{w}^*||_2^2 + \frac{8\epsilon\beta}{\gamma}||\mathbf{w}_t - \mathbf{w}^*||_2$$

with probability at least $1 - O(1/d)$.

Proof. See Appendix A. □

Theorem 2.1 implies that the convergence is linear-quadratic in error $\Delta_t = \mathbf{w}_t - \mathbf{w}^*$. Initially, when $||\Delta_t||_2$ is large, the first term of the RHS will dominate and the convergence will be quadratic, that is, $||\Delta_{t+1}||_2 \lesssim \frac{5L}{\gamma}||\Delta_t||_2^2$. In later stages, when $||\mathbf{w}_t - \mathbf{w}^*||_2$ becomes sufficiently small, the second term of RHS will start to dominate and the convergence will be linear, that is, $||\Delta_{t+1}||_2 \lesssim \frac{8\epsilon\beta}{\gamma}||\Delta_t||_2$. At this stage in practice, the sketch dimension can be increased to reduce ϵ to diminish the effect of the linear term and improve the convergence rate.

Remark 1. *We note that the conventional distributed second-order methods for server-based systems—which distribute training examples evenly across workers (such as [1, 5])—typically find a 'local approximation' of second-order update at each worker and then aggregate it. OverSketched Newton, on the other hand, utilizes the massive storage and compute power in serverless systems to find a 'global approximation'. As we demonstrate in Sec. 4, it performs significantly better than existing server-based methods.*

3 Example Problems

In this section, we describe several commonly encountered optimization problems that can be solved using OverSketched Newton. We give details for logistic regression in Sec. 3.1.

Logistic Regression: The optimization problem for supervised learning using Logistic Regression takes the form

$$\min_{\mathbf{w}\in\mathbb{R}^d} \Big\{ f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \log(1 + e^{-y_i \mathbf{w}^T \mathbf{x}_i}) + \frac{\lambda}{2}\|\mathbf{w}\|^2 \Big\}. \tag{9}$$

Here, $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^{d\times 1}$ and $y_1, \cdots, y_n \in \mathbb{R}$ are training sample vectors and labels, respectively. The goal is to learn the feature vector $\mathbf{w}^* \in \mathbb{R}^{d\times 1}$. Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n] \in \mathbb{R}^{d\times n}$ and $\mathbf{y} = [y_1, \cdots, y_n] \in \mathbb{R}^{n\times 1}$ be the feature and label matrices, respectively. Using

sub-Gaussian and Hadamard schemes that take $O(mnd)$ and $O(md\log n)$ time, respectively.

Newton's method to solve (9) first requires evaluation of the gradient

$$\nabla f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\frac{-y_i\mathbf{x}_i}{1+e^{y_i\mathbf{w}_i^T\mathbf{x}_i}} + \lambda\mathbf{w}.$$

Calculation of $\nabla f(\mathbf{w})$ involves two matrix-vector products, $\boldsymbol{\alpha} = \mathbf{X}^T\mathbf{w}$ and $\nabla f(\mathbf{w}) = \frac{1}{n}\mathbf{X}\boldsymbol{\beta} + \lambda\mathbf{w}$, where $\beta_i = \frac{-y_i}{1+e^{y_i\alpha_i}}\ \forall\ i \in [1,n]$. These matrix-vector products are performed distributedly. Faster convergence can obtained by second-order methods which will additionally compute the Hessian $\mathbf{H} = \frac{1}{n}\mathbf{X}\boldsymbol{\Lambda}\mathbf{X}^T + \lambda\mathbf{I}_d$, where $\boldsymbol{\Lambda}$ is a diagonal matrix with entries given by $\Lambda(i,i) = \frac{e^{y_i\alpha_i}}{(1+e^{y_i\alpha_i})^2}$. The product $\mathbf{X}\boldsymbol{\Lambda}\mathbf{X}^T$ is computed approximately in a distributed straggler-resilient manner using (7). We elaborate on the details in Sec. 3.1.

Interior point methods based linear program: The following linear program can be solved using OverSketched Newton

$$\underset{\mathbf{Ax}\leq\mathbf{b}}{\text{minimize}}\ \mathbf{c}^T\mathbf{x}, \tag{10}$$

where $\mathbf{x} \in \mathbb{R}^{m\times1}, \mathbf{c} \in \mathbb{R}^{m\times1}, \mathbf{b} \in \mathbb{R}^{n\times1}$ and $\mathbf{A} \in \mathbb{R}^{n\times m}$ is the constraint matrix with $n > m$. In interior point methods based algorithm, the following sequence of problems using Newton's method

$$\min_{\mathbf{x}\in\mathbb{R}^m} f(\mathbf{x}) = \min_{\mathbf{x}\in\mathbb{R}^m}\left(\tau\mathbf{c}^T\mathbf{x} - \sum_{i=1}^{n}\log(b_i - \mathbf{a}_i\mathbf{x})\right), \tag{11}$$

where $\mathbf{a}_i$ is the i-th row of $\mathbf{A}$, τ is increased geometrically such that when τ is very large, the logarithmic term does not affect the objective value and serves its purpose of keeping all intermediates solution inside the constraint region. The update in the t-th iteration is given by $\mathbf{x}_{t+1} = \mathbf{x}_t - (\nabla^2 f(\mathbf{x}_t))^{-1}\nabla f(\mathbf{x}_t)$, where $\mathbf{x}_t$ is the estimate of the solution in the t-th iteration. The gradient can be written as $\nabla f(\mathbf{x}) = \tau\mathbf{c} + \mathbf{A}^T\boldsymbol{\beta}$ where $\beta_i = 1/(b_i - \alpha_i)$ and $\boldsymbol{\alpha} = \mathbf{Ax}$.

The Hessian for the objective in (11) is given by

$$\nabla^2 f(\mathbf{x}) = \mathbf{A}^T\text{diag}\frac{1}{(b_i-\alpha_i)^2}\mathbf{A}. \tag{12}$$

This can be computed approximately using the sketch matrix in (7).

Lasso (compressed sensing): The optimization problem takes the following form

$$\min_{\mathbf{w}\in\mathbb{R}^d}\frac{1}{2}||\mathbf{Xw} - \mathbf{y}||_2^2 + \lambda||\mathbf{w}||_1, \tag{13}$$

where $\mathbf{X} \in \mathbb{R}^{n\times d}$ is the measurement matrix, the vector $\mathbf{y} \in \mathbb{R}^n$ contains the measurements, $\lambda \geq 0$ and $d \gg n$. To solve (13), we consider its dual variation

$$\min_{||\mathbf{X}^T\mathbf{z}||_\infty\leq\lambda,\mathbf{z}\in\mathbb{R}^n}\frac{1}{2}||\mathbf{y} - \mathbf{z}||_2^2,$$

which is amenable to interior point methods and can be solved by optimizing the following sequence of problems where τ is increased geometrically

$$\min_{\mathbf{z}} f(\mathbf{z}) = \min_{\mathbf{z}}\left(\frac{\tau}{2}||\mathbf{y} - \mathbf{z}||_2^2 - \sum_{j=1}^{d}\log\frac{\lambda - \mathbf{x}_j^T\mathbf{z}}{\lambda + \mathbf{x}_j^T\mathbf{z}}\right),$$

where $\mathbf{x}_j$ is the j-th column of $\mathbf{X}$. The gradient can be expressed in few matrix-vector multiplications as $\nabla f(\mathbf{z}) = \tau(\mathbf{z} - \mathbf{y}) + \mathbf{X}(\boldsymbol{\beta} - \boldsymbol{\gamma})$, where $\beta_i = 1/(\lambda - \alpha_i)$, $\gamma_i = 1/(\lambda + \alpha_i)$, and $\boldsymbol{\alpha} = \mathbf{X}^T\mathbf{z}$. Similarly, the Hessian can be written as $\nabla^2 f(\mathbf{z}) = \tau\mathbf{I} + \mathbf{X}\boldsymbol{\Lambda}\mathbf{X}^T$, where $\boldsymbol{\Lambda}$ is a diagonal matrix whose entries are given by $\Lambda_{ii} = 1/(\lambda - \alpha_i)^2 + 1/(\lambda + \alpha_i)^2\ \forall\ i \in [1,n]$.

Other common problems where OverSketched Newton is applicable include Linear Regression, Support Vector Machines (SVMs), Semidefinite programs, etc.

3.1 Logistic Regression using OverSketched Newton on Serverless Systems

Algorithm 1 OverSketched Newton: Logistic Regression for Serverless Computing

```
 1: Input Data: Example Matrix $\mathbf{X} \in \mathbb{R}^{d\times n}$ and vector $\mathbf{y} \in \mathbb{R}^{n\times1}$ (stored in cloud storage), regularization parameter $\lambda$, number of iterations $T$, Sketch $\mathbf{S}$ as defined in Eq. (7)
 2: Initialization: Define $\mathbf{w}^1 = \mathbf{0}^{d\times1}, \boldsymbol{\beta} = \mathbf{0}^{n\times1}, \boldsymbol{\gamma} = \mathbf{0}^{n\times1}$, Encode $\mathbf{X}$ and $\mathbf{X}^T$ as described in [20]
 3: for $t = 1$ to $T$ do
 4:     $\boldsymbol{\alpha} = \mathbf{X}\mathbf{w}^t$ {Compute in parallel using codes}
 5:     for $i = 1$ to $n$ do
 6:         $\beta_i = \frac{-y_i}{1+e^{y_i\alpha_i}}$;
 7:     end for
 8:     $\mathbf{g} = \mathbf{X}^T\boldsymbol{\beta}$ { Compute in parallel using codes}
 9:     $\nabla f(\mathbf{w}^t) = \mathbf{g} + \lambda\mathbf{w}^t$;
10:     for $i = 1$ to $n$ do
11:         $\gamma(i) = \frac{e^{y_i\alpha_i}}{(1+e^{y_i\alpha_i})^2}$;
12:     end for
13:     $\mathbf{A} = \sqrt{diag(\boldsymbol{\gamma})}\mathbf{X}$
14:     $\mathbf{H} = \mathbf{A}^T\mathbf{A}$ { Compute in parallel using OverSketch}
15:     $\mathbf{H} = \frac{1}{n}\mathbf{H} + \lambda\mathbf{I}_d$;
16:     $\mathbf{w}^{t+1} = \mathbf{w}^t - \mathbf{H}^{-1}\nabla f(\mathbf{w}^t)$;
17: end for
output $\mathbf{w}^* = \mathbf{w}_{T+1}$
```

We provide a detailed description of OverSketched Newton for large-scale logistic regression for serverless systems in Algorithm 1. Here, steps 4, 8 and 14 are computed in parallel on AWS Lambda. All other steps are simple vector operations that can be performed locally at the master, for instance, the user's laptop. We assume that the number of features are small enough to fit the Hessian matrix $\mathbf{H}$ locally at the master and compute the update $\mathbf{w}^{t+1} = \mathbf{w}^t - \mathbf{H}^{-1}\nabla f(\mathbf{w}^t)$ efficiently. Steps 4 and 8 are executed in a straggler-resilient fashion using the coding scheme in [20], as illustrated in Fig. 1.

We use the coding scheme in [20] since the encoding can be implemented in parallel and requires less communication per worker compared to the scheme in [19] that uses Maximum Distance Separable (MDS) codes. Note that since the example matrix $\mathbf{X}$ is constant, the encoding of $\mathbf{X}$ is done only once before starting the optimization algorithm and, thus, the encoding cost is amortized over iterations. Moreover, for decoding over the resultant product vector requires negligible time and space, even when n is scaling into the millions.

The same is, however, not true for the matrix multiplication for Hessian calculation (step 14 of Algorithm 1), as the matrix $\mathbf{L}$ changes in each iteration, thus encoding costs will be incurred in every iteration if error-correcting codes are used. Moreover, encoding and decoding a huge matrix stored in the cloud incurs heavy communication cost and becomes prohibitive. Motivated by this, we use OverSketch in step 14, proposed in [11], to calculate an approximate matrix multiplication, and hence the Hessian, efficiently in serverless systems with inbuilt straggler resiliency. We also evaluate the exact Hessian-based algorithm with speculative execution and compare it with OverSketched Newton in the next section.

4 Experimental Results

In this section, we evaluate OverSketched Newton on AWS Lambda using real-world and synthetic datasets and compare it with state-of-the-art distributed optimization algorithms. We use the serverless computing framework, Pywren, developed recently in [9]. Our experiments are focused on logistic regression, a popular supervised learning problem, but can be reproduced for other problems described in Section 3.

4.1 Comparison with Existing Second-order Methods

For comparison of OverSketched Newton with existing distributed optimization schemes, we choose recently proposed Globally Improved Approximate Newton Direction (GIANT) [5] as a representative algorithm. This is because GIANT boasts a better convergence rate than many existing distributed second-order methods for linear and logistic regression when $n \gg d$. In GIANT, and other similar distributed second-order algorithms, the training data is evenly divided among workers, and the algorithms proceed in two stages. First, the workers compute partial gradients using local training data, which is then aggregated by the master to compute the exact gradient. Second, the workers receive the full gradient to calculate their local second-order estimate, which is then

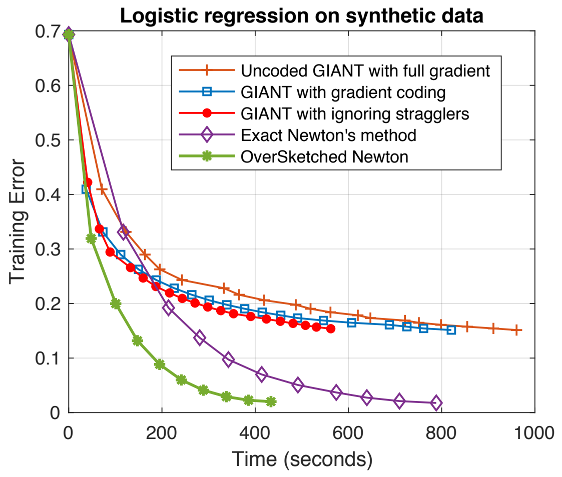


Figure 4: Convergence comparison of GIANT (employed with different straggler mitigation methods), exact Newton's method and OverSketched Newton for Logistic regression on AWS Lambda. The synthetic dataset considered has 300,000 examples and 3000 features.

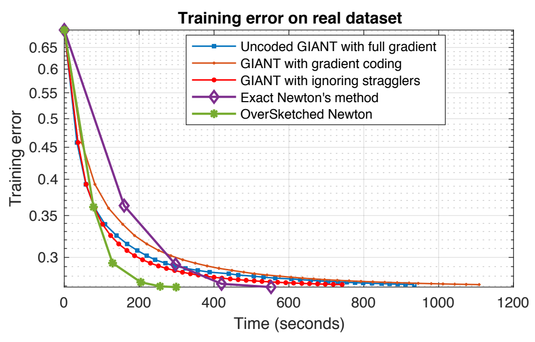
averaged by the master.

For straggler mitigation in such algorithms, [23] proposes a scheme for coding gradient updates called *gradient coding*, where the data at each worker is repeated multiple times to compute redundant copies of the gradient. We use *gradient coding* scheme with GIANT for comparison with OverSketched Newton. We also evaluate exact Newton's method with speculative execution for straggler mitigation, that is, reassigning and recomputing the work for straggling workers, and compare its convergence rate with OverSketched Newton.

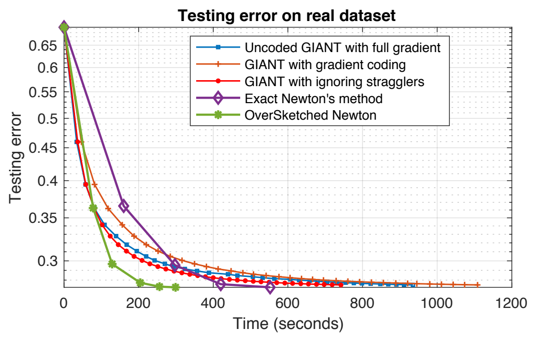
4.2 Experiments on Synthetic Data

In Figure 4, we present our experimental results on randomly generated dataset with $n = 300,000$ and $d = 3000$ for logistic regression on AWS Lambda. The orange, blue and red curves demonstrate the convergence for GIANT with the full gradient (that waits for all the workers), *gradient coding* and mini-batch gradient (that ignores the stragglers while calculating gradient and second-order updates) schemes, respectively. The gradient coding scheme is applied for one straggler, that is the data is repeated twice at each worker. We use 60 Lambda workers for executing GIANT in parallel. Similarly, for Newton's method, we use 60 workers for matrix-vector multiplication in steps 4 and 8 of Algorithm 1, 3600 workers for exact Hessian computation and 600 workers for sketched Hessian computation with a sketch dimension of $10d = 30,000$.

An important point to note from Fig. 4 is that the uncoded scheme (that is, the one that waits for all stragglers) has the worst performance. Moreover, due



(a) Training error for logistic regression on EPSILON dataset



(b) Testing error for logistic regression on EPSILON dataset

Figure 5: Comparison of training and testing errors for several Newton based schemes with straggler mitigation. Testing error closely follows training error.

to faults in certain iterations, the algorithm had to be restarted manually. The implication is that good straggler/fault mitigation algorithms are essential for computing in the serverless setting. Secondly, the mini-batch scheme outperforms the gradient coding scheme by 25%. This is because gradient coding requires additional communication of data to serverless workers (twice when coding for one straggler, see [23] for details) at each invocation to AWS Lambda. However, even though the exact Newton's method describe requires more time per iteration, it converges much faster than GIANT.

The green plot shows the convergence of OverSketched Newton. It can be noted the number of iterations required for convergence for OverSketched Newton and exact Hessian is similar, but the OverSketched Newton converges in almost half the time due to significant time savings during the computation of Hessian, which is the computational bottleneck in each iteration.

4.3 Experiments on Real Data

In Figure 5, we plot a similar convergence of training and testing errors of schemes described above for logistic regression but on the real-world classification dataset EPSILON obtained from [36], with $n = 400,000$ and $d = 2000$. Here, we used 100 workers for GIANT, and 100 workers for matrix-vector multiplications for gradient calculation in OverSketched Newton. Gradient coding for three stragglers was employed in GIANT for straggler mitigation but performs worse than uncoded GIANT that waits for all the stragglers due to the repetition of training data at workers. Hence, the communication costs dominate the straggling costs. Mini-batch gradient that ignores the stragglers still outperforms gradient coding and uncoded schemes for GIANT.

Exact Hessian computation with required 10,000 serverless workers and speculative execution was used to mitigate stragglers (i.e., recomputing the straggling jobs). OverSketch required 1500 workers with a sketch dimension of $15d = 30,000$. OverSketch requires significantly less number of workers as once the square root of Hessian is sketched in a distributed fashion, it can be copied into local memory due to dimension reduction and the Hessian can be calculated locally. There is little difference between the convergence of training and testing error and important conclusions remain the same as in Figure 4. OverSketched Newton significantly outperforms GIANT and exact Newton-based optimization in terms of running time.

4.4 Coded computing versus Speculative Execution

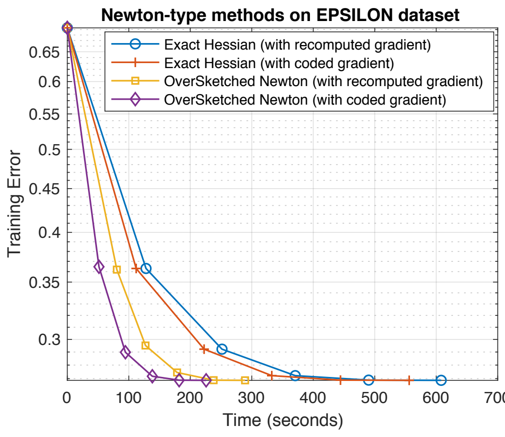


Figure 6: Comparison of speculative execution and coded computing schemes for computing the gradient and Hessian for the EPSILON dataset. OverSketched Newton, that is, coding the gradients and sketching the Hessian, outperforms all other schemes.

In Figure 6, we compare the effect of straggler mitigation schemes on convergence rate, namely speculative execution, that is, restarting the jobs with straggling workers, and coded computing. We regard OverSketch based matrix multiplication as a coding scheme as some redundancy is introduced during "over" sketching for matrix multiplication. There

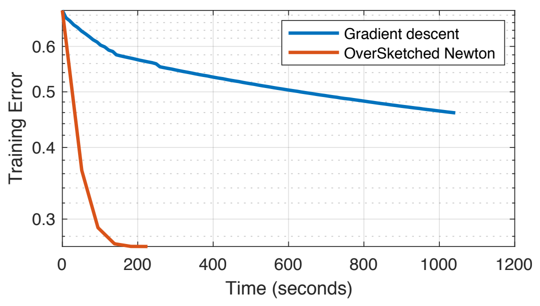


Figure 7: Convergence rate comparison of gradient descent and OverSketched Newton for the EPSILON dataset.

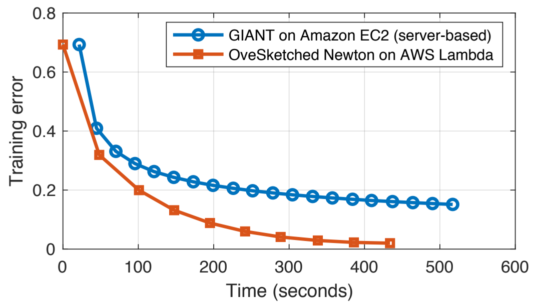


Figure 8: Convergence rate comparison of MPI-based AWS EC2 and AWS Lambda that use GIANT and OverSketched Newton, respectively, to solve a large scale logistic regression problem.

are four different cases, corresponding to gradient and hessian calculation using either speculative execution or coded computing. For speculative execution, we wait for at least 90% of the workers to return (this works well as the number of stragglers is generally less than 5%) and restart the jobs that did not return till this point. For both exact Hessian and OverSketch based optimization, coding the computation of gradient outperforms speculative execution based straggler mitigation. Also, computing the Hessian using OverSketch is considerably better than exact computation in terms of running time as calculating the Hessian is the computational bottleneck in each iteration.

4.5 Comparison with Gradient Descent on AWS Lambda

In Figure 7, we compare gradient descent with OverSketched Newton for logistic regression on EPSILON dataset. The statistics for OverSketched Newton were obtained as described in the previous section. We observed that for first-order methods, there is only a slight difference in convergence for a mini-batch gradient when the batch size is 95%. Hence, for gradient descent, we use 100 workers in each iteration while ignoring the stragglers. The step-size was chosen using the method of backtracking line search, which determines the maximum amount to move in given a descent direction. As can be noted, OverSketched Newton significantly outperforms gradient descent.

4.6 Comparison with Server-based Optimization

In Fig. 8, we compare OverSketched Newton on AWS Lambda with existing distributed optimization algorithm GIANT in server-based systems (AWS EC2 in our case). The results are plotted on synthetically generated data for logistic regression. For server-based programming, we use Message Passing Interface (MPI) with one c3.8xlarge master and 60 t2.medium workers in AWS EC2. In [10], authors observed that many large-scale linear algebra operations on serverless systems take at least 30% more time compared to MPI-based computation on server-based systems. However, as shown in Fig. 8, we observe that OverSketched Newton outperforms MPI-based optimization that uses existing state-of-the-art optimization algorithm. This is because OverSketched Newton exploits the flexibility and massive scale at disposal in serverless, and thus produces a better approximation of the second-order update than GIANT.

5 Conclusions

We proposed OverSketched Newton, a straggler-resilient distributed optimization algorithm for serverless systems. It uses the idea of matrix sketching from RandNLA to find an approximate second-order update in each iteration. We proved that OverSketched Newton has a linear-quadratic convergence rate, where the dependence on the linear term can be made to diminish by increasing the sketch dimension. By exploiting the massive scalability of serverless systems, OverSketched Newton produces a *global approximation* of the second-order update. Empirically, this translates into faster convergence than state-of-the-art distributed optimization algorithms on AWS Lambda.

A Proof of Theorem 2.1

As $\mathbf{w}_{t+1}$ is the optimal solution of RHS in (8), we have, for any $\mathbf{w}$ in $\mathcal{C}$

$$\begin{aligned}
&f(\mathbf{w}_t) + \nabla f(\mathbf{w}_t)^T(\mathbf{w} - \mathbf{w}_t) + \frac{1}{2}(\mathbf{w} - \mathbf{w}_t)^T\hat{\mathbf{H}}_t(\mathbf{w} - \mathbf{w}_t)\\
&\qquad \geq f(\mathbf{w}_t) + \nabla f(\mathbf{w}_t)^T(\mathbf{w}_{t+1} - \mathbf{w}_t) + \frac{1}{2}(\mathbf{w}_{t+1} - \mathbf{w}_t)^T\hat{\mathbf{H}}(\mathbf{w}_{t+1} - \mathbf{w}_t)\\
&\Rightarrow \nabla f(\mathbf{w}_t)^T(\mathbf{w} - \mathbf{w}_{t+1}) + \frac{1}{2}(\mathbf{w} - \mathbf{w}_t)^T\hat{\mathbf{H}}_t(\mathbf{w} - \mathbf{w}_t) - \frac{1}{2}(\mathbf{w}_{t+1} - \mathbf{w}_t)^T\hat{\mathbf{H}}_t(\mathbf{w}_{t+1} - \mathbf{w}_t) \geq 0\\
&\Rightarrow \nabla f(\mathbf{w}_t)^T(\mathbf{w} - \mathbf{w}_{t+1}) + \frac{1}{2}\Big[(\mathbf{w} - \mathbf{w}_t)^T\hat{\mathbf{H}}_t(\mathbf{w} - \mathbf{w}_{t+1}) + (\mathbf{w} - \mathbf{w}_{t+1})^T\hat{\mathbf{H}}_t(\mathbf{w}_{t+1} - \mathbf{w}_t)\Big] \geq 0
\end{aligned}$$

Substituting $\mathbf{w}$ by $\mathbf{w}^*$ in above and calling $\Delta_t = \mathbf{w}^* - \mathbf{w}_{t+1}$, we get

$$\begin{aligned}
&-\nabla f(\mathbf{w}_t)^T\Delta_{t+1} + \frac{1}{2}\Big[\Delta_{t+1}^T\hat{\mathbf{H}}_t(2\Delta_t - \Delta_{t+1})\Big] \geq 0\\
&\Rightarrow \Delta_{t+1}^T\hat{\mathbf{H}}_t\Delta_t - \nabla f(\mathbf{w}_t)^T\Delta_{t+1} \geq \frac{1}{2}\Delta_{t+1}^T\hat{\mathbf{H}}_t\Delta_{t+1}
\end{aligned}$$

Now, due to optimality of $\mathbf{w}^*$, we have $\nabla f(\mathbf{w}^*)^T\Delta_{t+1} \geq 0$. Hence

$$\Delta_{t+1}^T\hat{\mathbf{H}}_t\Delta_t - (\nabla f(\mathbf{w}_t) - \nabla f(\mathbf{w}^*))^T\Delta_{t+1} \geq \frac{1}{2}\Delta_{t+1}^T\hat{\mathbf{H}}_t\Delta_{t+1}$$

Also, using the property $\nabla f(\mathbf{w}_t) - \nabla f(\mathbf{w}^*) = \Big(\int_0^1 \nabla^2 f(\mathbf{w}^* + p(\mathbf{w}_t - \mathbf{w}^*))dp\Big)(\mathbf{w}_t - \mathbf{w}^*)$ in above inequality, we get

$$\Delta_{t+1}^T(\hat{\mathbf{H}}_t - \nabla^2 f(\mathbf{w}_t))\Delta_t + \Delta_{t+1}^T\Big(\nabla^2 f(\mathbf{w}_t) - \int_0^1 \nabla^2 f(\mathbf{w}^* + p(\mathbf{w}_t - \mathbf{w}^*))dp\Big)\Delta_t \geq \frac{1}{2}\Delta_{t+1}^T\hat{\mathbf{H}}_t\Delta_{t+1}$$

Using Cauchy-Schwartz inequality in the RHS above, we get

$$||\Delta_{t+1}||_2||\Delta_t||_2\Big(||\hat{\mathbf{H}}_t - \nabla^2 f(\mathbf{w}_t)||_{op} + \int_0^1 ||\nabla^2 f(\mathbf{w}_t) - \nabla^2 f(\mathbf{w}^* + p(\mathbf{w}_t - \mathbf{w}^*))||_{op}dp\Big) \geq \frac{1}{2}\Delta_{t+1}^T\hat{\mathbf{H}}_t\Delta_{t+1}$$

Now, using the Lipschitz property of $f(\cdot)$ in (2) in the inequality above, we get

$$\begin{aligned}
&\frac{1}{2}\Delta_{t+1}^T\hat{\mathbf{H}}_t\Delta_{t+1} \leq ||\Delta_{t+1}||_2||\Delta_t||_2||\hat{\mathbf{H}}_t - \nabla^2 f(\mathbf{w}_t)||_{op} + \frac{L}{2}||\Delta_{t+1}||_2||\Delta_t||_2^2\int_0^1(1-p)dp\\
&\Rightarrow \frac{1}{2}\Delta_{t+1}^T\hat{\mathbf{H}}_t\Delta_{t+1} \leq ||\Delta_{t+1}||_2\Big(||\Delta_t||_2||\hat{\mathbf{H}}_t - \nabla^2 f(\mathbf{w}_t)||_{op} + \frac{L}{2}||\Delta_t||_2^2\Big) \qquad (14)
\end{aligned}$$

To complete the proof, we will need the following two lemmas. The first lemma would define an upper bound on the first term of the RHS.

Lemma A.1. *Let $\hat{\mathbf{H}}_t = \mathbf{A}_t^T\mathbf{S}_t\mathbf{S}_t^T\mathbf{A}_t$ where $\mathbf{S}_t$ is the sparse sketch matrix in (7) with sketch dimension $m = \Omega(d^{1+\mu}/\epsilon^2)$ and $N = \Theta_\mu(1/\epsilon)$. Then, the following holds*

$$||\hat{\mathbf{H}}_t - \nabla^2 f(\mathbf{w}_t)||_{op} \leq \epsilon||\mathbf{A}_t||_{op}^2 = \epsilon||\nabla^2 f(\mathbf{w}_t)||_{op} \qquad (15)$$

with probability at least $1 - \frac{1}{poly(d)}$.

Proof. Note that for the positive definite matrix $\nabla^2 f(\mathbf{w}_t) = \mathbf{A}_t^T\mathbf{A}_t$, we have $||\mathbf{A}_t||_{op}^2 = ||\nabla^2 f(\mathbf{w}_t)||_{op}$. Moreover,

$$||\hat{\mathbf{H}}_t - \nabla^2 f(\mathbf{w}_t)||_{op} = ||\mathbf{A}_t^T(\mathbf{S}_t\mathbf{S}_t^T - \mathbf{I})\mathbf{A}_t||_{op} \leq ||\mathbf{A}_t||_{op}^2||\mathbf{S}_t\mathbf{S}_t^T - \mathbf{I}||_{op}$$

Next, we note than N is the number of non-zero elements per row in the sketch $\mathbf{S}_t$ in (7) after ignoring stragglers. Moreover, we use Theorem 8 in [37] to bound the singular values for the sparse sketch $\mathbf{S}_t$ in (7) with sketch dimension $m = \Omega(d^{1+\mu}/\epsilon^2)$ and $N = \Theta(1/\epsilon)$. It says that $\mathbb{P}(\forall \mathbf{x} \in \mathbb{R}^n, ||\mathbf{S}_t\mathbf{x}||_2 \in (1 \pm \epsilon/3)||\mathbf{x}||_2) > 1 - 1/d^\alpha$, where the constants in $\Theta(\cdot)$ and $\Omega(\cdot)$ depend on α and μ. Thus, $||\mathbf{S}_t\mathbf{x}||_2 \in (1 \pm \epsilon/3)||\mathbf{x}||_2$, which implies

$$||\mathbf{S}_t\mathbf{x}||_2^2 \in (1 + \epsilon^2/9 \pm 2\epsilon/3)||\mathbf{x}||_2^2,$$

with probability at least $1 - 1/d^\alpha$. For $\epsilon \leq 1/2$, this leads to the following inequality

$$||\mathbf{S}_t\mathbf{x}||_2^2 \in (1 \pm \epsilon)||\mathbf{x}||_2^2 \Rightarrow |\mathbf{x}^T(\mathbf{S}_t\mathbf{S}_t^T - \mathbf{I})\mathbf{x}| \leq \epsilon||\mathbf{x}||_2^2 \; \forall \; x \in \mathbb{R}^n.$$

This implies $||\mathbf{S}_t\mathbf{S}_t^T - \mathbf{I}||_{op} \leq \epsilon$ with probability $1 - 1/d^\alpha$, which proves the desired result. □

Next lemma would provide a lower bound for the LHS in (14).

Lemma A.2. *For the sketch matrix* $\mathbf{S}_t \in \mathbb{R}^{n\times m}$ *in* (7) *with sketch dimension* $m = \Omega(d/\epsilon^2)$*, the following holds*

$$\mathbf{x}^T\mathbf{S}_t\mathbf{S}_t^T\mathbf{x} \geq (1-\epsilon)||\mathbf{x}||_2^2 \ \forall \ \mathbf{x} \in \mathbb{R}^n, \tag{16}$$

with probability at least $1 - 1/d$.

Proof. The result follows directly by substituting $\mathbf{A} = \mathbf{x}^T$, $\mathbf{B} = \mathbf{x}$ and $\theta = 1/d$ in Theorem 4.1 of [11]. □

Using Lemma A.1 to bound the RHS of (14), we have, with probability at least $1 - 1/poly(d)$

$$\frac{1}{2}\Delta_{t+1}^T\hat{\mathbf{H}}_t\Delta_{t+1} \leq ||\Delta_{t+1}||_2\Big(\epsilon||\nabla^2 f(\mathbf{w}_t)||_{op}||\Delta_t||_2 + \frac{L}{2}||\Delta_t||_2^2\Big).$$

Since $\hat{\mathbf{H}}_t = \mathbf{A}_t^T\mathbf{S}_t^T\mathbf{S}_t\mathbf{A}_t$ and sketch dimension $m = \Omega(d^{1+\mu}/\epsilon^2)$ implies $m = \Omega(d/\epsilon^2)$, using Lemma A.2 in above inequality, we get, with probability at least $1 - O(1/d)$,

$$\begin{aligned}\frac{1}{2}(1-\epsilon)||\mathbf{A}\Delta_{t+1}||_2^2 &\leq ||\Delta_{t+1}||_2\Big(\epsilon||\nabla^2 f(\mathbf{w}_t)||_{op}||\Delta_t||_2 + \frac{L}{2}||\Delta_t||_2^2\Big)\\ \Rightarrow \ \frac{1}{2}(1-\epsilon)\Delta_{t+1}^T\nabla^2 f(\mathbf{w}_t)\Delta_{t+1} &\leq ||\Delta_{t+1}||_2\Big(\epsilon||\nabla^2 f(\mathbf{w}_t)||_{op}||\Delta_t||_2 + \frac{L}{2}||\Delta_t||_2^2\Big),\end{aligned}$$

Now, since γ and β are the minimum and maximum eigenvalues of $\nabla^2 f(\mathbf{w}^*)$, we get

$$\frac{1}{2}(1-\epsilon)||\Delta_{t+1}||_2(\gamma - L||\Delta_t||_2) \leq \epsilon(\beta + L||\Delta_t||_2)||\Delta_t||_2 + \frac{L}{2}||\Delta_t||_2^2$$

by the Lipschitzness of $\nabla^2 f(\mathbf{w})$, that is, $|\Delta_{t+1}^T(\nabla^2 f(\mathbf{w}_t) - \nabla^2 f(\mathbf{w}^*))\Delta_{t+1}| \leq L||\Delta_t||_2||\Delta||_{t+1}^2$. Rearranging for $\epsilon \leq 1/2$, we get

$$||\Delta_{t+1}||_2 \leq \frac{4\epsilon\beta}{\gamma - L||\Delta_t||_2}||\Delta_t||_2 + \frac{5L}{2(\gamma - L||\Delta_t||_2)}||\Delta_t||_2^2.$$

Now, using the fact that $||\Delta_t||_2 \leq \gamma/2L$, we get the desired result

$$||\Delta_{t+1}||_2 \leq 8\epsilon\frac{\beta}{\gamma}||\Delta_t||_2 + \frac{5L}{\gamma}||\Delta_t||_2^2.$$

References

[1] O. Shamir, N. Srebro, and T. Zhang, "Communication-efficient distributed optimization using an approximate newton-type method," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pp. II–1000–II–1008, JMLR.org, 2014.

[2] V. Smith, S. Forte, C. Ma, M. Takác, M. I. Jordan, and M. Jaggi, "Cocoa: A general framework for communication-efficient distributed optimization," *arXiv preprint arXiv:1611.02189*, 2016.

[3] Y. Zhang and X. Lin, "Disco: Distributed optimization for self-concordant empirical loss," in *Proceedings of the 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37 of *Proceedings of Machine Learning Research*, (Lille, France), pp. 362–370, PMLR, 07–09 Jul 2015.

[4] S. J. Reddi, A. Hefny, S. Sra, B. Pöczos, and A. Smola, "On variance reduction in stochastic gradient descent and its asynchronous variants," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, (Cambridge, MA, USA), pp. 2647–2655, MIT Press, 2015.

[5] S. Wang, F. Roosta-Khorasani, P. Xu, and M. W. Mahoney, "Giant: Globally improved approximate newton method for distributed optimization," *arXiv preprint arXiv:1709.03528*, 2017.

[6] C. Duenner, A. Lucchi, M. Gargiani, A. Bian, T. Hofmann, and M. Jaggi, "A distributed second-order algorithm you can trust," in *Proceedings of the 35th International Conference on Machine Learning* (J. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, (Stockholmsmssan, Stockholm Sweden), pp. 1358–1366, PMLR, 10–15 Jul 2018.

[7] I. Baldini, P. C. Castro, K. S.-P. Chang, P. Cheng, S. J. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. M. Rabbah, A. Slominski, and P. Suter, "Serverless computing: Current trends and open problems," *CoRR*, vol. abs/1706.03178, 2017.

[8] J. Spillner, C. Mateos, and D. A. Monge, "Faaster, better, cheaper: The prospect of serverless scientific computing and HPC," in *Latin American High Performance Computing Conference*, pp. 154–168, Springer, 2017.

[9] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the cloud: distributed computing for the 99%," in *Proceedings of the 2017 Symposium on Cloud Computing*, pp. 445–451, ACM, 2017.

[10] V. Shankar, K. Krauth, Q. Pu, E. Jonas, S. Venkataraman, I. Stoica, B. Recht, and J. Ragan-Kelley, "numpywren: serverless linear algebra," *ArXiv e-prints*, Oct. 2018.

[11] V. Gupta, S. Wang, T. Courtade, and K. Ramchandran, "Oversketch: Approximate matrix multiplication for the cloud," *IEEE International Conference on Big Data, Seattle, WA, USA*, 2018.

[12] A. Aytekin and M. Johansson, "Harnessing the Power of Serverless Runtimes for Large-Scale Optimization," *arXiv e-prints*, p. arXiv:1901.03161, Jan. 2019.

[13] L. Feng, P. Kudva, D. D. Silva, and J. Hu, "Exploring serverless computing for neural network training," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, vol. 00, pp. 334–341, Jul 2018.

[14] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, pp. 74–80, Feb. 2013.

[15] T. Hoefler, T. Schneider, and A. Lumsdaine, "Characterizing the influence of system noise on large-scale applications by simulation," in *Proc. of the ACM/IEEE Int. Conf. for High Perf. Comp., Networking, Storage and Analysis*, pp. 1–11, 2010.

[16] M. W. Mahoney, *Randomized algorithms for matrices and data.* Foundations and Trends in Machine Learning, Boston: NOW Publishers, 2011.

[17] P. Drineas and M. W. Mahoney, "RandNLA: Randomized numerical linear algebra," *Communications of the ACM*, vol. 59, pp. 80–90, 2016.

[18] P. Drineas, R. Kannan, and M. W. Mahoney, "Fast Monte Carlo algorithms for matrices I: Approximating matrix multiplication," *SIAM Journal on Computing*, vol. 36, pp. 132–157, 2006.

[19] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2018.

[20] T. Baharav, K. Lee, O. Ocal, and K. Ramchandran, "Straggler-proofing massive-scale distributed matrix multiplication with d-dimensional product codes," in *IEEE Int. Sym. on Information Theory (ISIT), 2018*, IEEE, 2018.

[21] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107–113, Jan. 2008.

[22] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, pp. 10–10, 2010.

[23] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, pp. 3368–3376, PMLR, 2017.

[24] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," in *IEEE Int. Sym. on Information Theory (ISIT), 2017*, pp. 2418–2422, IEEE, 2017.

[25] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), pp. 4403–4413, Curran Associates, Inc., 2017.

[26] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," *arXiv preprint arXiv:1801.10292*, 2018.

[27] J. Zhu, Y. Pu, V. Gupta, C. Tomlin, and K. Ramchandran, "A sequential approximation framework for coded distributed optimization," in *Annual Allerton Conf. on Communication, Control, and Computing, 2017*, pp. 1240–1247, IEEE, 2017.

[28] Y. Yang, P. Grover, and S. Kar, "Coded distributed computing for inverse problems," in *Advances in Neural Information Processing Systems 30*, pp. 709–719, Curran Associates, Inc., 2017.

[29] M. Pilanci and M. J. Wainwright, "Newton sketch: A near linear-time optimization algorithm with linear-quadratic convergence," *SIAM Jour. on Opt.*, vol. 27, pp. 205–245, 2017.

[30] F. Roosta-Khorasani and M. W. Mahoney, "Sub-Sampled Newton Methods I: Globally Convergent Algorithms," *arXiv e-prints*, p. arXiv:1601.04737, Jan. 2016.

[31] F. Roosta-Khorasani and M. W. Mahoney, "Sub-Sampled Newton Methods II: Local Convergence Rates," *arXiv e-prints*, p. arXiv:1601.04738, Jan. 2016.

[32] V. Ishakian, V. Muthusamy, and A. Slominski, "Serving deep learning models in a serverless platform," *arXiv e-prints*, p. arXiv:1710.08460, Oct. 2017.

[33] Y. Nesterov, "A method of solving a convex programming problem with convergence rate O(1/sqr(k))," *Soviet Mathematics Doklady*, vol. 27, pp. 372–376, 1983.

[34] S. Boyd and L. Vandenberghe, *Convex Optimization.* New York, NY, USA: Cambridge University Press, 2004.

[35] D. P. Woodruff, "Sketching as a tool for numerical linear algebra," *Found. Trends Theor. Comput. Sci.*, vol. 10, pp. 1–157, 2014.

[36] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM transactions on intelligent systems and technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[37] J. Nelson and H. L. Nguyen, "Osnap: Faster numerical linear algebra algorithms via sparser subspace embeddings," in *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pp. 117–126, Oct 2013.