OverSketched Newton: Fast Convex Optimization for Serverless Systems

Vipul Gupta[1], Swanand Kadhe[1], Thomas Courtade[1], Michael W. Mahoney[2] and Kannan Ramchandran[1]
[1]Department of EECS, UC Berkeley
[2] ICSI and Statistics Department, UC Berkeley
Email: {vipul_gupta, swanand.kadhe, courtade, kannanr}@eecs.berkeley.edu, mmahoney@stat.berkeley.edu

2020 IEEE International Conference on Big Data (Big Data) | 978-1-7281-6251-5/20/$31.00 ©2020 IEEE | DOI: 10.1109/BigData50022.2020.9378289

***Abstract*—Motivated by recent developments in serverless systems for large-scale computation as well as improvements in scalable randomized matrix algorithms, we develop OverSketched Newton, a randomized Hessian-based optimization algorithm to solve large-scale convex optimization problems in serverless systems. OverSketched Newton leverages matrix sketching ideas from Randomized Numerical Linear Algebra to compute the Hessian approximately. These sketching methods lead to inbuilt resiliency against stragglers that are a characteristic of serverless architectures. Depending on whether or not the problem is strongly convex, we propose different iteration updates using the approximate Hessian. For both cases, we establish convergence guarantees for OverSketched Newton, and we empirically validate our results by solving large-scale supervised learning problems on real-world datasets. Experiments demonstrate a reduction of $\sim$50% in total running time on AWS Lambda, compared to state-of-the-art distributed optimization schemes.**

***Index Terms*—serverless computing, second-order optimization, matrix sketching, coded computing**

I. Introduction

In recent years, there has been tremendous growth in users performing distributed computing operations on the cloud, largely due to extensive and inexpensive commercial offerings like Amazon Web Services (AWS), Google Cloud, Microsoft Azure, etc. Serverless platforms—such as AWS Lambda, Cloud functions and Azure Functions—penetrate a large user base by provisioning and managing the servers on which the computation is performed. These platforms abstract 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 [1]–[4].

The use of serverless systems is gaining significant research traction, primarily due to its massive scalability and convenience in operation. It is forecasted that the market share of serverless will grow by USD 9.16 billion during 2019-2023 (at a CAGR of 11%) [5]. Indeed, according to the *Berkeley view on Serverless Computing* [6], serverless systems are expected to dominate the cloud scenario and become the default computing paradigm in the coming years while client-server based computing will witness a considerable decline. For these reasons, using serverless systems for large-scale computation has garnered significant attention from the systems community [3], [4], [7]–[12].

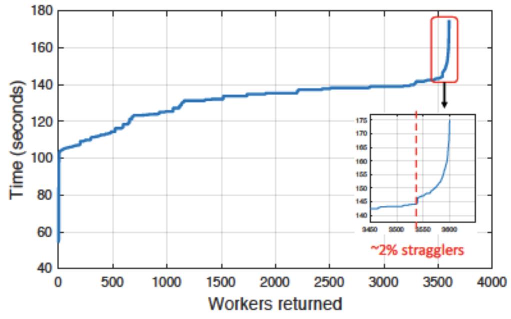


Fig. 1: Average job times for 3600 AWS Lambda nodes over 10 trials for distributed matrix multiplication. The median job time is around 135 seconds, and around 2% of the nodes take up to 180 seconds on average.

Due to several crucial differences between the traditional High Performance Computing (HPC) / serverful and serverless architectures, existing distributed algorithms cannot, in general, be extended to serverless computing. First, unlike *serverful* computing, the number of inexpensive workers in serverless platforms is flexible, often scaling into the thousands [3], [4]. This heavy gain in the computation power, however, comes with the disadvantage that the commodity workers in serverless architecture are ephemeral and have low memory.[1] 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. Moreover, the workers do not communicate amongst themselves, and instead they read/write data directly from/to a single high-latency data storage entity (e.g., cloud storage like AWS S3 [3]).

Second, unlike HPC/serverful systems, nodes in the serverless systems suffer degradation due to what is known as *system noise*. This can be a result of limited availability of shared resources, hardware failure, network latency, etc. [13], [14]. This results in job time variability, and hence a subset of much slower nodes, often called *stragglers*. These stragglers significantly slow the overall computation time, especially in

[1]For example, serverless nodes in AWS Lambda, Google Cloud Functions and Microsoft Azure Functions have a maximum memory of 3 GB, 2 GB and 1.5 GB, respectively, and a maximum runtime of 900 seconds, 540 seconds and 300 seconds, respectively (these numbers may change over time).

large or iterative jobs. In Fig. 1, we plot the running times for a distributed matrix multiplication job with 3600 workers on AWS Lambda and demonstrate the effect of stragglers on the total job time. In fact, our experiments consistently demonstrate that at least 2% workers take significantly longer than the median job time, severely degrading the overall efficiency of the system.

Due to these issues, first-order methods, e.g., gradient descent and Nesterov Accelerated Gradient (NAG) methods, tend to perform poorly on distributed serverless architectures [15]. Their slower convergence is made worse on serverless platforms due to persistent stragglers. The straggler effect incurs heavy slowdown due to the accumulation of tail times as a result of a subset of slow workers occurring in each iteration.

Compared to first-order optimization algorithms, second-order methods—which use the gradient as well as Hessian information—enjoy superior convergence rates. For instance, Newton's method enjoys quadratic convergence for strongly convex and smooth problems, compared to the linear convergence of gradient descent [16]. Moreover, second-order methods do not require step-size tuning and unit step-size provably works for most problems. These methods have a long history in optimization and scientific computing (see, e.g., [16]), but they are less common in machine learning and data science. This is partly since stochastic first order methods suffice for downstream problems [17] and partly since naive implementations of second order methods can perform poorly [18]. However, recent theoretical work has addressed many of these issues [19]–[23], and recent implementations have shown that high-quality implementations of second order stochastic optimization algorithms can beat state-of-the-art in machine learning applications [24]–[28] in traditional systems.

A. Main Contributions

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; and, 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 Sub-Sampled Newton (SSN) methods [19]–[22], which are based on sketching methods from Randomized Numerical Linear Algebra (RandNLA) [29]–[31], to obtain a good approximation for the Hessian, instead of calculating the full Hessian.

OverSketched Newton has two key components. For straggler-resilient Hessian calculation in serverless systems, we use the sparse sketching based randomized matrix multiplication method from [32]. For straggler mitigation during gradient calculation, we use the recently proposed technique based on error-correcting codes to create redundant computation [33]–[35]. We prove that, for strongly convex functions, the local convergence rate of OverSketched Newton is linear-quadratic, while its global convergence rate is linear. Then, going beyond the usual strong convexity assumption for second-order methods, we adapt OverSketched Newton using ideas from [22]. For such functions, we prove that a linear convergence rate can be guaranteed with OverSketched Newton.

We extensively evaluate OverSketched Newton on AWS Lambda using several real-world datasets obtained from the LIBSVM repository [36], and we compare OverSketched Newton with several first-order (gradient descent, Nesterov's method, etc.) and second-order (exact Newton's method [16], GIANT [24], etc.) baselines for distributed optimization. We further evaluate and compare different techniques for straggler mitigation, such as speculative execution, coded computing [33], [34], randomization-based sketching [32] and gradient coding [37]. We demonstrate that OverSketched Newton is at least 9x and 2x faster than state-of-the-art first-order and second-order schemes, respectively, in terms of end-to-end training time on AWS Lambda. Moreover, we show that OverSketched Newton on serverless systems outperforms existing distributed optimization algorithms in serverful systems by at least 30%.[2]

B. Related Work

Our results tie together three quite different lines of work, each of which we review here briefly.

Existing Straggler Mitigation Schemes: Strategies like speculative execution have been traditionally used to mitigate stragglers in popular distributed computing frameworks like Hadoop MapReduce [39] and Apache Spark [40]. 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, e.g., constant monitoring of tasks is required and late stragglers can still hurt the efficiency.

Recently, many coding-theoretic ideas have been proposed to introduce redundancy into the distributed computation for straggler mitigation (e.g., see [33]–[35], [37], [41], [42]). The idea of coded computation is to generate redundant copies of the result of distributed computation by encoding the input data using error-correcting-codes. These redundant copies are then used to decode the output of the missing stragglers. Our algorithm to compute gradients in a distributed straggler-resilient manner uses codes to mitigate stragglers, and we compare our performance with speculative execution.

Approximate Second-order Methods: In many machine learning applications, where the data itself is noisy, using the exact Hessian is not necessary. Indeed, using ideas from RandNLA, one can prove convergence guarantees for SSN methods on a single machine, when the Hessian is computed approximately [19]–[21], [23]. To accomplish this, many sketching schemes can be used (sub-Gaussian, Hadamard, random row sampling, sparse Johnson-Lindenstrauss, etc. [29], [30]), but these methods cannot tolerate stragglers, and thus they do not perform well in serverless environments.

[2]A longer technical report version of this paper is available at [38].

This motivates the use of the *OverSketch* sketch from our recent work in [32]. OverSketch has many nice properties, like subspace embedding, sparsity, input obliviousness, and amenability to distributed implementation. To the best of our knowledge, this is the first work to prove and evaluate convergence guarantees for algorithms based on OverSketch. Our guarantees take into account the amount of communication at each worker and the number of stragglers, both of which are a property of distributed systems.

There has also been a growing research interest in designing and analyzing distributed implementations of stochastic second-order methods [24], [43]–[46]. However, these implementations are tailored for serverful distributed systems. Our focus, on the other hand, is on serverless systems.

Distributed Optimization on Serverless Systems: Optimization over the serverless framework has garnered significant interest from the research community. However, these works either evaluate and benchmark existing algorithms (e.g., see [9]–[11]) or focus on designing new systems frameworks for faster optimization (e.g., see [12]) on serverless. To the best of our knowledge, this is the first work that proposes a large-scale distributed optimization algorithm that specifically caters to *serverless architectures* with *provable convergence guarantees*. We exploit the advantages offered by serverless systems while mitigating the drawbacks such as stragglers and additional overhead per invocation of workers.

II. Newton's Method: An Overview

We are interested in solving on serverless systems in a distributed and straggler-resilient manner problems of the form:

$$f(\mathbf{w}^*) = \min_{\mathbf{w}\in\mathbb{R}^d} f(\mathbf{w}), \tag{1}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is a closed and convex function bounded from below. In the Newton's method, the update at the $(t+1)$-th iteration is obtained by minimizing the Taylor's expansion of the objective function $f(\cdot)$ at $\mathbf{w}_t$, that is

$$\mathbf{w}_{t+1} = \arg\min_{\mathbf{w}\in\mathbb{R}^d} \Big\{ f(\mathbf{w}_t) + \nabla f(\mathbf{w}_t)^T(\mathbf{w}-\mathbf{w}_t) + \frac{1}{2}(\mathbf{w}-\mathbf{w}_t)^T\nabla^2 f(\mathbf{w}_t)(\mathbf{w}-\mathbf{w}_t)\Big\}. \tag{2}$$

For strongly convex $f(\cdot)$, that is, when $\nabla^2 f(\cdot)$ is invertible, Eq. (2) becomes $\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{H}_t^{-1}\nabla f(\mathbf{w}_t)$, where $\mathbf{H}_t = \nabla^2 f(\mathbf{w}_t)$ is the Hessian matrix at the t-th iteration. Given a good initialization and assuming that the Hessian is Lipschitz, the Newton's method satisfies the update $||\mathbf{w}_{t+1} - \mathbf{w}^*||_2 \leq c||\mathbf{w}_t - \mathbf{w}^*||_2^2$, for some constant $c > 0$, implying quadratic convergence [16].

One shortcoming for the classical Newton's method is that it works only for strongly convex objective functions. In particular, if f is weakly-convex[3], that is, if the Hessian matrix is not positive definite, then the objective function in (2) may be unbounded from below. To address this shortcoming, authors in [22] recently proposed a variant of Newton's method, called Newton-Minimum-Residual (Newton-MR). Instead of (1), Newton-MR considers the following auxiliary optimization problem:

$$\min_{\mathbf{w}\in\mathbb{R}^d} ||\nabla f(\mathbf{w})||^2.$$

Note that the minimizers of this auxiliary problem and (1) are the same when $f(\cdot)$ is convex. Then, the update direction in the $(t+1)$-th iteration is obtained by minimizing the Taylor's expansion of $||\nabla f(\mathbf{w}_t + \mathbf{p})||^2$, that is,

$$\mathbf{p}_t = \arg\min_{\mathbf{w}\in\mathbb{R}^d} ||\nabla f(\mathbf{w}_t) + \mathbf{H}_t\mathbf{p}||^2.$$

The general solution of the above problem is given by $\mathbf{p} = -[\mathbf{H}_t]^\dagger\nabla f(\mathbf{w}_t) + (\mathbf{I} - \mathbf{H}_t[\mathbf{H}_t]^\dagger)\mathbf{q},\ \forall\ \mathbf{q} \in \mathbb{R}^d$, where $[\cdot]^\dagger$ is the Moore-Penrose inverse. Among these, the minimum norm solution is chosen, which gives the update direction in the t-th iteration as $\mathbf{p}_t = -\mathbf{H}_t^\dagger\nabla f(\mathbf{w}_t)$. Thus, the model update is

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \mathbf{p}_t = \mathbf{w}_t - [\nabla^2 f(\mathbf{w}_t)]^\dagger\nabla f(\mathbf{w}_t). \tag{3}$$

OverSketched Newton considers both of these variants.

III. OverSketched Newton

We present OverSketched Newton, a stochastic second order algorithm for solving—*on serverless systems, in a distributed, straggler-resilient manner*—problems of the form (1).

Distributed straggler-resilient gradient computation: OverSketched Newton computes the full gradient in each iteration using tools from error-correcting codes [33], [34]. Our key observation is that, for several commonly encountered optimization problems, gradient computation relies on matrix-vector multiplications (see Sec. IV for examples). We leverage coded matrix multiplication technique from [34] to perform the large-scale matrix-vector multiplication in a distributed straggler-resilient manner. The idea of coded matrix multiplication is explained in Fig. 2; detailed algorithm is provided in the technical report version of this paper [38].

Distributed straggler-resilient approximate Hessian computation: For several commonly encountered optimization problems, Hessian computation involves matrix-matrix multiplication for a pair of large matrices (see Sec. IV for several examples). For computing the large-scale matrix-matrix multiplication in parallel in serverless systems, we use a straggler-resilient scheme from [32] called *OverSketch*. It uses a sparse sketching matrix based on Count-Sketch [29]. It has similar computational efficiency and accuracy guarantees as that of the Count-Sketch, with two additional properties: it is amenable to distributed implementation; and it is resilient to stragglers. More specifically, the OverSketch matrix is constructed as follows.

Recall that the Hessian $\nabla^2 f(\cdot) \in \mathbb{R}^{d\times d}$. First choose the desired sketch dimension m (which depends on d), block-size b (which depends on the memory of the workers), and straggler tolerance $\zeta > 0$ (which depends on the distributed system). Then, define $N = m/b$ and $e = \zeta N$, for some constant $\zeta > 0$. Here ζ is the fraction of stragglers that we want our algorithm

[3]For the sake of clarity, we call a convex function weakly-convex if it is not strongly convex.

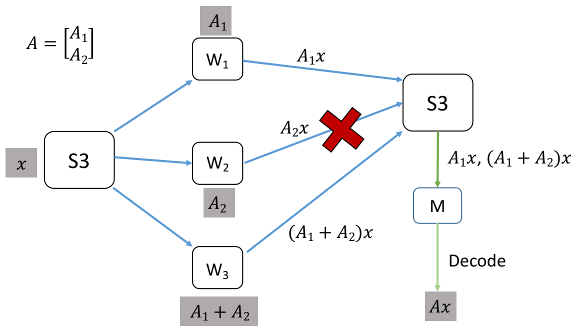


Fig. 2: **Coded matrix-vector multiplication**: Matrix $\mathbf{A}$ is divided into 2 row chunks $\mathbf{A}_1$ and $\mathbf{A}_2$. During encoding, redundant chunk $\mathbf{A}_1+\mathbf{A}_2$ is created. Three workers obtain $\mathbf{A}_1$,$\mathbf{A}_2$ and $\mathbf{A}_1+\mathbf{A}_2$ from

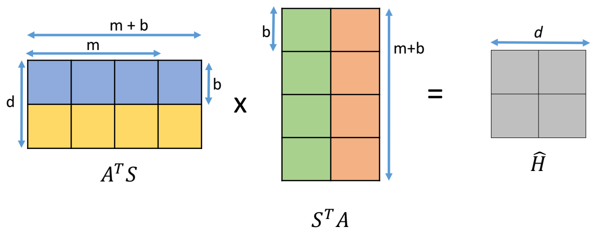


Fig. 3: **OverSketch-based approximate Hessian computation:** First, the matrix $\mathbf{A}$—satisfying $\mathbf{A}^T\mathbf{A} = \nabla^2 f(\mathbf{w}_t)$—is sketched in parallel using the sketch in (4). Then, each worker receives a block of each of the sketched matrices $\mathbf{A}^T\mathbf{S}$ and $\mathbf{S}^T\mathbf{A}$, 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 block-size b for obtaining resiliency against one straggler for each block of $\hat{\mathbf{H}}$.

to tolerate. Thus, e is the maximum number of stragglers per $N+e$ workers that can be tolerated. The sketch $\mathbf{S}$ is given by

$$\mathbf{S} = \frac{1}{\sqrt{N}}(\mathbf{S}_1, \mathbf{S}_2, \cdots, \mathbf{S}_{N+e}), \tag{4}$$

where $\mathbf{S}_i \in \mathbb{R}^{n\times b}$, for all $i \in [1, N+e]$, are i.i.d. Count-Sketch matrices[4] with sketch dimension b. Note that $\mathbf{S} \in \mathbb{R}^{n\times(m+eb)}$, where $m = Nb$ is the required sketch dimension and e is the over-provisioning parameter to provide resiliency against e stragglers per $N+e$ workers. We leverage the straggler resiliency of OverSketch to obtain the sketched Hessian in a distributed straggler-resilient manner. An illustration of OverSketch is provided in Fig. 3; see the technical report version of this paper [38] for a detailed algorithm.

Model update: Let $\hat{\mathbf{H}}_t = \mathbf{A}_t^T\mathbf{S}_t\mathbf{S}_t^T\mathbf{A}_t$, where $\mathbf{A}_t$ is the square root of the Hessian $\nabla^2 f(\mathbf{w}_t)$, and $\mathbf{S}_t$ is an independent realization of (4) at the t-th iteration. For strongly-convex functions, the update direction is $\mathbf{p}_t = -\hat{\mathbf{H}}_t^{-1}\nabla f(\mathbf{w}_t)$. We use line-search to choose the step-size, that is, find

$$\alpha_t = \max_{\alpha\leq 1} \ \alpha \quad \text{such that}$$
$$f(\mathbf{w}_t + \alpha\mathbf{p}_t) \leq f(\mathbf{w}_t) + \alpha\beta\mathbf{p}_t^T\nabla f(\mathbf{w}_t), \tag{5}$$

for some constant $\beta \in (0, 1/2]$. For weakly-convex functions, the update direction (inspired by Newton-MR [22]) is $\mathbf{p}_t = -\hat{\mathbf{H}}_t^{\dagger}\nabla f(\mathbf{w}_t)$, where $\hat{\mathbf{H}}_t^{\dagger}$ is the Moore-Penrose inverse of $\hat{\mathbf{H}}_t$. To find the update $\mathbf{w}_{t+1}$, we find the right step-size α_t using line-search in (5), but with $f(\cdot)$ replaced by $||\nabla f(\cdot)||^2$ and $\nabla f(\mathbf{w}_t)$ replaced by $2\hat{\mathbf{H}}_t\nabla f(\mathbf{w}_t)$, according to the objective in $||\nabla f(\cdot)||^2$. More specifically, for some constant $\beta \in (0, 1/2]$, find

$$\alpha_t = \max_{\alpha\leq 1} \ \alpha \qquad \text{such that}$$
$$||\nabla f(\mathbf{w}_t + \alpha\mathbf{p}_t)||^2 \leq ||\nabla f(\mathbf{w}_t)||^2 + 2\alpha\beta\mathbf{p}_t^T\hat{\mathbf{H}}_t\nabla f(\mathbf{w}_t). \tag{6}$$

Note that for OverSketched Newton, we use $\hat{\mathbf{H}}_t$ in the line-search since the exact Hessian is not available. The update in the t-th iteration in both cases is given by

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t\mathbf{p}_t.$$

Note that line-search in Eq. (5) can be solved approximately using Armijo backtracking line search (see [16] for a general algorithm and [24] for a distributed implementation). OverSketched Newton is concisely described in Algorithm 1. Next, we prove convergence guarantees for OverSketched Newton.

Algorithm 1: OverSketched Newton: An Outline

```
Input: Convex function $f$; Initial iterate $\mathbf{w}_0 \in \mathbb{R}^d$;
       Line search parameter $0 < \beta \leq 1/2$; Number
       of iterations $T$
1  for $t = 1$ to $T$ do
2      Compute full gradient $\mathbf{g}_t$ in a distributed
         straggler-resilient manner
3      Compute sketched Hessian matrix $\hat{\mathbf{H}}_t$ in a
         distributed fashion using OverSketch
4      if $f$ is strongly-convex then
5          Compute the update direction at the master as:
             $\mathbf{p}_t = -[\hat{\mathbf{H}}_t]^{-1}\nabla f(\mathbf{w}_t)$
6          Compute step-size $\alpha_t$ satisfying the line-search
             condition (5) in a distributed fashion
7      else
8          Compute the update direction at the master as:
             $\mathbf{p}_t = -[\hat{\mathbf{H}}_t]^{\dagger}\nabla f(\mathbf{w}_t)$
9          Find step-size $\alpha_t$ satisfying the line-search
             condition (6) in a distributed fashion
10     end
11     Compute the model update $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_t\mathbf{p}_t$ at
         the master
12 end
```

A. Convergence Guarantees

First, we focus our attention to strongly convex functions. We consider the following assumptions. We note that these

[4]Each of the Count-Sketch matrices $\mathbf{S}_i$ is constructed (independently of others) as follows. First, for every row j, $j \in [n]$, of $\mathbf{S}_i$, 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 [29], [32] for details.)

assumptions are standard for analyzing approximate Newton methods, (e.g., see [19], [20], [23]).

Assumptions:

1. f is twice-differentiable;
2. f is k-strongly convex ($k > 0$), that is, $\nabla^2 f(\mathbf{w}) \succeq k\mathbf{I}$;
3. f is M-smooth ($k \leq M < \infty$), that is, $\nabla^2 f(\mathbf{w}) \preceq M\mathbf{I}$;
4. The Hessian is L-Lipschitz continuous, that is, for any $\mathbf{\Delta} \in \mathbb{R}^d$, we have $||\nabla^2 f(\mathbf{w}+\mathbf{\Delta}) - \nabla^2 f(\mathbf{w})||_2 \leq L||\mathbf{\Delta}||_2$, where $||\cdot||_2$ is the spectral norm for matrices.

We can prove the following "global" convergence guarantee which shows that OverSketched Newton would converge from any random initialization of $\mathbf{w}_0 \in \mathbb{R}^d$ with high probability.[5]

Theorem III.1 (**Global convergence for strongly-convex** f). *Consider Assumptions 1, 2, and 3 and step-size α_t given by Eq.* (5). *Let $\mathbf{w}^*$ be the optimal solution of* (1). *Let ϵ and μ be positive constants. Then, using the sketch in* (4) *with a sketch dimension $Nb + eb = \Omega(\frac{d^{1+\mu}}{\epsilon^2})$ and the number of column-blocks $N + e = \Theta_\mu(1/\epsilon)$, the updates for OverSketched Newton, for any $\mathbf{w}_t \in \mathbb{R}^d$, satisfy*

$$f(\mathbf{w}_{t+1}) - f(\mathbf{w}^*) \leq (1-\rho)(f(\mathbf{w}_t) - f(\mathbf{w}^*)),$$

with probability at least $1 - 1/d^\tau$, where $\rho = \frac{2\alpha_t\beta k}{M(1+\epsilon)}$ and $\tau > 0$ is a constant depending on μ and constants in $\Omega(\cdot)$ and $\Theta(\cdot)$. Moreover, α_t satisfies $\alpha_t \geq \frac{2(1-\beta)(1-\epsilon)k}{M}$.

Theorem III.1 guarantees the global convergence of OverSketched Newton starting with any initial estimate $\mathbf{w}_0 \in \mathbb{R}^d$ to the optimal solution $\mathbf{w}^*$ with at least a linear rate.

Next, we can also prove an additional "local" convergence guarantee for OverSketched Newton, under the assumption that $\mathbf{w}_0$ is sufficiently close to $\mathbf{w}^*$.

Theorem III.2 (**Local convergence for strongly-convex** f). *Consider Assumptions 1, 2, and 4 and step-size $\alpha_t = 1$. 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. Let $\epsilon \in (0, \gamma/(8\beta)]$ and $\mu > 0$. Then, using the sketch in* (4) *with a sketch dimension $Nb + eb = \Omega(\frac{d^{1+\mu}}{\epsilon^2})$ and the number of column-blocks $N + e = \Theta_\mu(1/\epsilon)$, the updates for OverSketched Newton, with initialization $\mathbf{w}_0$ such that $||\mathbf{w}_0 - \mathbf{w}^*||_2 \leq \frac{\gamma}{8L}$, follow*

$$||\mathbf{w}_{t+1} - \mathbf{w}^*||_2 \leq \frac{25L}{8\gamma}||\mathbf{w}_t - \mathbf{w}^*||_2^2 + \frac{5\epsilon\beta}{\gamma}||\mathbf{w}_t - \mathbf{w}^*||_2,$$

for $t = 1, 2, \cdots, T$, with probability $\geq 1 - T/d^\tau$, where $\tau > 0$ is a constant depending on μ and constants in $\Omega(\cdot)$ and $\Theta(\cdot)$.

Theorem III.2 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, $||\Delta_{t+1}||_2 \lesssim \frac{25L}{8\gamma}||\Delta_t||_2^2$. 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, $||\Delta_{t+1}||_2 \lesssim \frac{5\epsilon\beta}{\gamma}||\Delta_t||_2$. At this stage, the sketch dimension can be increased to reduce ϵ to diminish the effect of the linear term and improve the convergence rate in practice. Note that, for second order methods, the number of iterations T is in the order of tens in general, while the number of features d is typically in thousands. Hence, the probability of failure is generally small (and can be made negligible by choosing τ appropriately).

[5]Due to space, proofs are deferred to the technical report version [38].

Finally, we consider the case of weakly-convex functions. For this case, we consider two more assumptions on the Hessian matrix, similar to [22]. These assumptions are a relaxation of the strongly-convex case.

Assumptions:

5. There exists some $\eta > 0$ such that, $\forall\ \mathbf{w} \in \mathbb{R}^d$, $||(\nabla^2 f(\mathbf{w}))^\dagger||_2 \leq 1/\eta$. This assumption establishes regularity on the pseudo-inverse of $\nabla^2 f(\mathbf{x})$.
6. Let $\mathbf{U} \in \mathbb{R}^{d\times d}$ be any arbitrary orthogonal basis for Range($\nabla^2 f(\mathbf{w})$), there exists $0 < \nu \leq 1$, such that,

$$||\mathbf{U^T}\nabla \mathbf{f}(\mathbf{w})||^\mathbf{2} \geq \nu||\nabla \mathbf{f}(\mathbf{w})||^\mathbf{2} \quad \forall \quad \mathbf{w} \in \mathbb{R}^\mathbf{d}.$$

This assumption ensures that there is always a non-zero component of the gradient in the subspace spanned by the Hessian, and, thus, ensures that the model update $-\hat{\mathbf{H}}_t^\dagger \nabla f(\mathbf{w}_t)$ will not be zero. Note that these assumptions are always satisfied by strongly-convex functions. Under these assumptions, we prove global convergence of OverSketched Newton when the objective is weakly-convex.

Theorem III.3 (**Global convergence for weakly-convex** f). *Consider Assumptions 1,3,4,5 and 6 and step-size α_t given by Eq.* (6). *Let $\epsilon \in \left(0, \frac{(1-\beta)\nu\eta}{2M}\right]$ and $\mu > 0$. Then, using an OverSketch matrix with a sketch dimension $Nb + eb = \Omega(\frac{d^{1+\mu}}{\epsilon^2})$ and the number of column-blocks $N + e = \Theta_\mu(1/\epsilon)$, the updates for OverSketched Newton, for any $\mathbf{w}_t \in \mathbb{R}^d$, satisfy*

$$||\nabla f(\mathbf{w}_{t+1})||^2 \leq \left(1 - 2\beta\alpha\nu\frac{(1-\epsilon)\eta}{M(1+\epsilon)}\right)||\nabla f(\mathbf{w}_t)||^2,$$

with probability at least $1 - 1/d^\tau$, where $\alpha = \frac{\eta}{2Q}\left[(1-\beta)\nu\eta - 2\epsilon M\right], Q = (L||\nabla f(\mathbf{w}_0)|| + M^2)$, $\mathbf{w}_0$ is the initial iterate of the algorithm and $\tau > 0$ is a constant depending on μ and constants in $\Omega(\cdot)$ and $\Theta(\cdot)$.

IV. OverSketched Newton: Examples

Here, we describe several examples where our general approach can be applied.

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

$$\min_{\mathbf{w}\in\mathbb{R}^d} \left\{ f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \log(1 + e^{-y_i\mathbf{w}^T\mathbf{x}_i}) + \frac{\lambda}{2}||\mathbf{w}||_2^2 \right\}. \quad (7)$$

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 example and label matrices, respectively.

It is straightforward to see that the calculation of $\nabla f(\mathbf{w})$ involves two matrix-vector products, $\boldsymbol{\alpha} = \mathbf{X}^T\mathbf{w}$ and $\nabla f(\mathbf{w}) =$

Algorithm 2: OverSketched Newton: Logistic Regression on Serverless Systems

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

$\frac{1}{n}\mathbf{X}\boldsymbol{\beta} + \lambda\mathbf{w}$, where $\beta_i = \frac{-y_i}{1+e^{y_i\alpha_i}}\ \forall\ i \in [1,\cdots,n]$. When the example matrix is large, these matrix-vector products are performed distributedly using codes. Faster convergence is 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 the sketch matrix in (4). Using the result of distributed multiplication, the Hessian matrix $\mathbf{H}$ is calculated at the master and the model is updated as $\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{H}^{-1}\nabla f(\mathbf{w}_t)$. In practice, an efficient algorithm like conjugate gradient, that provides a good estimate in a small number of iterations, can be used locally at the master to solve for $\mathbf{w}_{t+1}$ [47].[6]

We provide a detailed description of OverSketched Newton for large-scale logistic regression for serverless systems in Algorithm 2. Steps 4, 8, and 14 of the algorithm are computed in parallel on AWS Lambda. All other steps are simple vector operations that can be performed locally at the master. Steps 4 and 8 are executed in a straggler-resilient fashion using the coding scheme in [34], as illustrated in Fig. 1. Since the example matrix $\mathbf{X}$ is constant in this example, the encoding of $\mathbf{X}$ is done only once before starting the optimization algorithm. Thus, the encoding cost can be amortized over iterations. 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 2), as the matrix $\mathbf{A}$ 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, as illustrated in Fig. 3, to calculate an approximate matrix multiplication, and hence the Hessian, efficiently in serverless systems with inbuilt straggler resiliency.[7]

Softmax Regression: We take unregularized softmax regression as an illustrative example for the weakly convex case. The goal is to find the weight matrix $\mathbf{W} = [\mathbf{w}_1, \cdots, \mathbf{w}_K]$ that fits the training data $\mathbf{X} \in \mathbb{R}^{d\times N}$ and $\mathbf{y} \in \mathbb{R}^{K\times N}$. Here $\mathbf{w}_i \in \mathbb{R}^d$ represents the weight vector for the k-th class for all $i \in [1, K]$ and K is the total number of classes. Hence, the resultant feature dimension for softmax regression is dK. The optimization problem is

$$f(\mathbf{W}) = \sum_{n=1}^{N}\left[\sum_{k=1}^{K} y_{kn}\mathbf{w}_k^T\mathbf{x}_n - \log\sum_{l=1}^{K}\exp\left(\mathbf{w}_l^T\mathbf{x}_n\right)\right]. \quad (8)$$

The gradient vector for the i-th class is

$$\nabla f_i(\mathbf{W}) = \sum_{n=1}^{N}\left[\frac{\exp\left(\mathbf{w}_i^T\mathbf{x}_n\right)}{\sum_{l=1}^{K}\exp\left(\mathbf{w}_l^T\mathbf{x}_n\right)} - y_{in}\right]\mathbf{x}_n, \quad (9)$$

which can be written as matrix products $\boldsymbol{\alpha_i} = \mathbf{X}^T\mathbf{w}_i$ and $\nabla f_i(\mathbf{W}) = \mathbf{X}\boldsymbol{\beta}_i$, where the entries of $\boldsymbol{\beta}_i \in \mathbb{R}^N$ are given by $\beta_{in} = \left(\frac{\exp(\alpha_{in})}{\sum_{l=1}^{K}\exp(\alpha_{ln})} - y_{in}\right)$. Thus, the full gradient matrix is given by $\nabla f(\mathbf{W}) = \mathbf{X}\boldsymbol{\beta}$ where the entries of $\boldsymbol{\beta} \in \mathbb{R}^{N\times K}$ are dependent on $\boldsymbol{\alpha} \in \mathbb{R}^{N\times K}$ as above and the matrix $\boldsymbol{\alpha}$ is given by $\boldsymbol{\alpha} = \mathbf{X}^T\mathbf{W}$. We assume that the number of classes K is small enough such that tall matrices $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are small enough for the master to do local calculations on them.

Since the effective number of features is $d\times K$, the Hessian matrix is of dimension $dK \times dK$. The (i,j)-th component of the Hessian, say $\mathbf{H}_{ij}$, is

$$\mathbf{H}_{ij}(\mathbf{W}) = \frac{d}{d\mathbf{w}_j}\nabla f_i(\mathbf{W}) = \frac{d}{d\mathbf{w}_j}\mathbf{X}\boldsymbol{\beta_i} = \mathbf{X}\frac{d}{d\mathbf{w}_j}\boldsymbol{\beta}_i = \mathbf{X}\mathbf{Z_{ij}}\mathbf{X}^T \quad (10)$$

where $\mathbf{Z}_{ij} \in \mathbb{R}^{N\times N}$ is a diagonal matrix whose n-th diagonal entry is

$$Z_{ij}(n) = \frac{\exp(\alpha_{in})}{\sum_{l=1}^{K}\exp(\alpha_{ln})}\left(\mathbb{I}(i=j) - \frac{\exp(\alpha_{jn})}{\sum_{l=1}^{K}\exp(\alpha_{ln})}\right), \quad (11)$$

for all $n \in [1, N]$, where $\mathbb{I}(\cdot)$ is the indicator function and $\boldsymbol{\alpha} = \mathbf{X}\mathbf{W}$ was defined above. The full Hessian matrix is obtained

[6]For simplicity, we assume here that the number of features is small enough to perform the model update locally at the master. This is not necessary, and straggler resilient schemes, such as in [35], can be used to perform distributed conjugate gradient in serverless systems when the assumption does not hold.

[7]We also evaluate the exact Hessian-based algorithm with speculative execution, i.e., recomputing the straggling jobs, and compare it with OverSketched Newton in Sec. V.

by putting together all such $\mathbf{H}_{ij}$'s in a $dK \times dK$ matrix. It can be expressed in a matrix-matrix multiplication form as

$$\nabla^2 f(\mathbf{W}) = \begin{bmatrix} \mathbf{H}_{11} & \cdots & \mathbf{H}_{1K} \\ \vdots & \ddots & \vdots \\ \mathbf{H}_{K1} & \cdots & \mathbf{H}_{KK} \end{bmatrix} = \begin{bmatrix} \mathbf{X}\mathbf{Z}_{11}\mathbf{X}^T & \cdots & \mathbf{X}\mathbf{Z}_{1K}\mathbf{X}^T \\ \vdots & \ddots & \vdots \\ \mathbf{X}\mathbf{Z}_{K1}\mathbf{X}^T & \cdots & \mathbf{X}\mathbf{Z}_{KK}\mathbf{X}^T \end{bmatrix} = \bar{\mathbf{X}}\bar{\mathbf{Z}}\bar{\mathbf{X}}^T, \quad (12)$$

where $\bar{\mathbf{X}} \in \mathbb{R}^{dK \times NK}$ is a block diagonal matrix that contains $\mathbf{X}$ in the diagonal blocks and $\bar{\mathbf{Z}} \in \mathbb{R}^{NK \times NK}$ is formed by stacking all the $\mathbf{Z}_{ij}$'s for $i, j \in [1, K]$. In OverSketched Newton, we compute this multiplication using sketching in serverless systems for efficiency and resiliency to stragglers. Assuming $d \times K$ is small enough, the master can then calculate the update $\mathbf{p}_t$ using efficient algorithms such the minimum-residual method [22], [48].

Other common problems where OverSketched Newton is applicable include linear Regression, lasso, linear programming via interior point methods, support vector machines, semidefinite programs, etc. (see the technical report version of this paper [38] for more details).

V. Experimental Results

In this section, we evaluate OverSketched Newton on AWS Lambda using real-world and synthetic datasets, and we compare it with state-of-the-art distributed optimization algorithms. We use the serverless computing framework Pywren [3]. Our experiments are focused on logistic and softmax regression, which are popular supervised learning problems, but they can be reproduced for other problems such as linear program, lasso, linear regression, etc. We present experiments on the following datasets:

Dataset	Training Samples	Features	Testing samples
Synthetic	$300,000$	3000	$100,000$
EPSILON	$400,000$	2000	$100,000$
WEBPAGE	$48,000$	300	$15,000$
a9a	$32,000$	123	$16,000$
EMNIST	$240,000$	7840	$40,000$

For comparison of OverSketched Newton with existing distributed optimization schemes, we choose recently-proposed Globally Improved Approximate Newton Direction (GIANT) [24]. The reason is that 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 averaged by the master.

For straggler mitigation in such server-based algorithms, [37] 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 implement GIANT with three different ways of dealing with stragglers: (1) we use gradient coding to mitigate stragglers; (2) we wait for all the workers to return; and (3) we simply ignore the stragglers. We compare the convergence of OverSketched Newton and GIANT with these straggler mitigation schemes. We further evaluate and compare the convergence exact Newton's method (employed with speculative execution, that is, reassigning and recomputing the work for straggling workers).

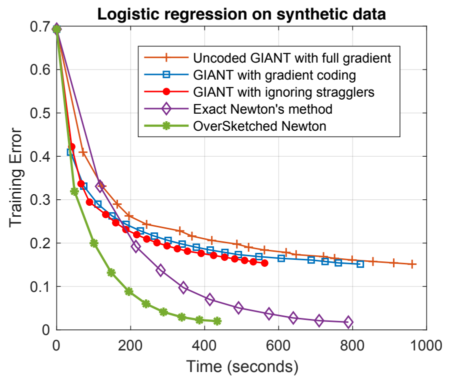


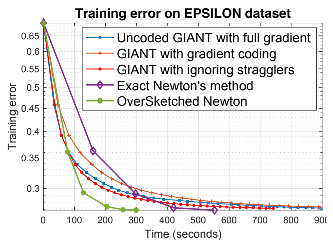
Fig. 4: Convergence comparison of GIANT (employed with different straggler mitigation methods), exact Newton's method and OverSketched Newton for Logistic regression on AWS Lambda. The synthetic dataset considered has 300,000 examples and 3000 features.

A. Comparisons with Second-Order Methods on AWS Lambda

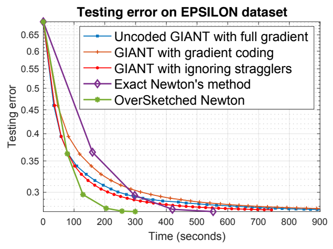
In Figure 4, we present our results on a synthetic dataset with $n = 300,000$ and $d = 3000$ for logistic regression on AWS Lambda. Each column $\mathbf{x}_i \in \mathbb{R}^d$, for all $i \in [1, n]$, is sampled uniformly randomly from the cube $[-1, 1]^d$. The labels y_i are sampled from the logistic model, that is, $\mathbb{P}[y_i = 1] = 1/(1 + exp(\mathbf{x}_i\mathbf{w} + b))$, where the weight vector $\mathbf{w}$ and bias b are generated randomly from the normal distribution.

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 purple and green curves depict the convergence for the exact Newton's method and OverSketched Newton, 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 2, 3600 workers for exact Hessian computation and 600 workers for sketched Hessian computation with a sketch dimension of $10d = 30,000$ in step 14 of Algorithm 2.

Remark 1. *In our experiments, we choose the number of workers in such a way that each worker receives approximately the same amount of data to work with, regardless of the underlying scheme. This is motivated by the fact that the limited memory at each worker is the bottleneck in serverless*

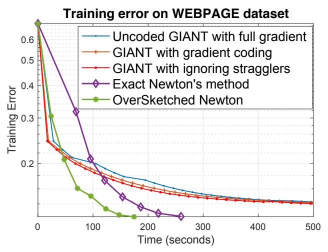


(a) Training error for logistic regression on EPSILON dataset

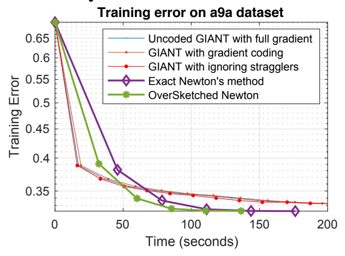


(b) Testing error for logistic regression on EPSILON dataset

Fig. 5: Training and testing errors for logistic regression on EPSILON dataset with several Newton based schemes on AWS Lambda. OverSketched Newton outperforms others by at least 46%.



(a) Logistic regression on WEBPAGE dataset



(b) Logistic regression on a9a dataset

Fig. 6: Logistic regression on WEBPAGE and a9a datasets with several Newton based schemes on AWS Lambda. OverSketched Newton outperforms others by at least 25%.

systems. Note that this is unlike serverful/HPC systems, where the number of workers is the bottleneck.

In all cases, unit step-size was used to update the model.[8]

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. 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 [37] for details) at each invocation to AWS Lambda. On the other hand, the exact Newton's method converges much faster than GIANT, even though it requires more time per iteration.

The number of iterations needed for convergence for OverSketched Newton and exact Newton (that exactly computes the Hessian) is similar, but OverSketched Newton converges in almost half the time due to an efficient computation of (approximate) Hessian (which is the computational bottleneck and thus reduces time per iteration).

1) Logistic Regression on EPSILON, WEBPAGE and a9a Datasets: In Figure 5, we repeat the above experiment with EPSILON classification dataset obtained from [36], with $n = 400,000$ and $d = 2000$. We plot training and testing errors for logistic regression for the schemes described in the previous section. We use 100 workers for GIANT, and 100 workers for matrix-vector multiplications for gradient calculation in OverSketched Newton. We use gradient coding designed for three stragglers in GIANT. This scheme performs worse than uncoded GIANT that waits for all the stragglers due to the repetition of training data at workers. Hence, one can conclude that the communication costs dominate the straggling costs. In fact, it can be observed that the mini-batch gradient scheme that ignores the stragglers outperforms the gradient coding and uncoded schemes for GIANT.

[8]Line-search in Section III was mainly introduced to prove theoretical guarantees. In our experiments, we observe that constant step-size works well for OverSketched Newton.

During exact Hessian computation, we use $10,000$ serverless workers with speculative execution to mitigate stragglers (i.e., recomputing the straggling jobs) compared to OverSketched Newton that uses 1500 workers with a sketch dimension of $15d = 30,000$. OverSketched Newton requires a significantly smaller number of workers, as once the square root of Hessian is sketched in a distributed fashion, it can be copied into local memory of the master due to dimension reduction, and the Hessian can be calculated locally. Testing error follows training error closely, and important conclusions remain the same as in Figure 4. OverSketched Newton outperforms GIANT and exact Newton-based optimization by at least 46% in terms of running time.

We repeated the above experiments for classification on the WEBPAGE ($n = 49,749$ and $d = 300$) and a9a ($n = 32,561$ and $d = 123$) datasets [36]. For both datasets, we used 30 workers for each iteration in GIANT and any matrix-vector multiplications. Exact hessian calculation invokes 900 workers as opposed to 300 workers for OverSketched Newton, where the sketch dimension was $10d = 3000$. The results for training loss on logistic regression are shown in Figure 6. Testing error closely follows the training error in both cases. OverSketched Newton outperforms exact Newton and GIANT by at least $\sim 25\%$ and $\sim 75\%$, respectively, which is similar to the trends witnessed heretofore.

Remark 2. *Note that conventional distributed second-order methods for serverful systems—which distribute training examples evenly across workers (such as [24], [43]–[46])—typically find a "localized approximation" (localized to each machine) of second-order update at each worker and then aggregate it. OverSketched Newton, on the other hand, uses the massive storage and compute power in serverless systems to find a more "globalized approximation" (globalized in the sense of across machine). Thus, it performs better in practice.*

B. Softmax Regression on EMIST

In Fig. 7, we solve unregularized softmax regression, which is weakly convex. We use the Extended MNIST (EMNIST) dataset [49] with $N = 240,000$ training examples, $d = 784$ features for each of the $K = 10$ classes. Note that GIANT cannot be applied here as the objective function is not strongly convex. We compare the convergence rate of OverSketched Newton, exact Hessian and gradient descent based schemes.

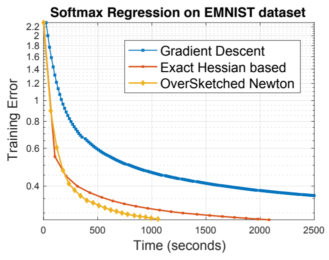


Fig. 7: Convergence comparison of different schemes for Softmax regression.

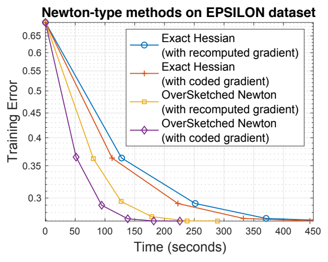


Fig. 8: Speculative execution versus coded computing for distributed computation.

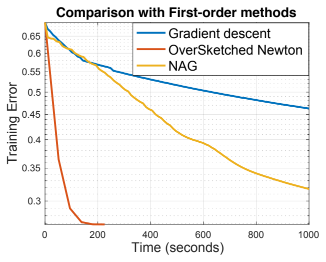


Fig. 9: Convergence of distributed gradient descent, NAG and OverSketched Newton.

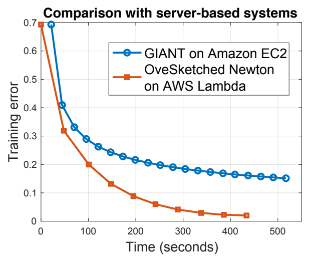


Fig. 10: Comparison of GIANT on AWS EC2 and OverSketched Newton on AWS Lambda.

For gradient computation in all three schemes, we use 60 workers. However, exact Newton scheme requires 3600 workers to calculate the $dK \times dK$ Hessian and recomputes the straggling jobs, while OverSketched Newton requires only 360 workers to calculate the sketch in parallel with sketch dimension $6dK = 47,040$. The approximate Hessian is then computed locally at the master using its sketched square root, where the sketch dimension is $6dK = 47,040$. The step-size is fixed and is determined by hyperparamter tuning before the start of the algorithm. Even for the weakly-convex case, second-order methods tend to perform better. Moreover, the runtime of OverSketched Newton outperforms both gradient descent and Exact Newton based methods by $\sim 75\%$ and $\sim 50\%$, respectively.

C. Coded computing versus Speculative Execution

In Figure 8, we compare straggler mitigation schemes, namely speculative execution and coded computing, on the convergence rate of logistic regression on the EPSILON dataset. We regard OverSketch based matrix multiplication as a *coding scheme* in which some redundancy is introduced during "over" sketching for matrix multiplication. There 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 10%) and restart the jobs that did not return till this point.

For both exact Hessian and OverSketched Newton, using codes for distributed gradient computation outperforms speculative execution based straggler mitigation. Moreover, computing the Hessian using OverSketch is significantly better than exact computation in terms of running time as calculating the Hessian is the computational bottleneck.

D. Comparison with First-Order Methods on AWS Lambda

In Figure 9, we compare gradient descent and Nesterov Accelerated Gradient (NAG) (while ignoring the stragglers) with OverSketched Newton for logistic regression on EPSILON dataset. 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 and NAG, we use 100 workers in each iteration while ignoring the stragglers.[9] These first-order methods were given the additional advantage of backtracking line-search, which determined the optimal amount to move in given a descent direction.[10] Overall, OverSketched Newton with unit step-size significantly outperforms gradient descent and NAG with backtracking line-search.

E. Comparison with Serverful Optimization

In Fig. 10, we compare OverSketched Newton on AWS Lambda with existing distributed optimization algorithm GIANT in serverful systems (AWS EC2). The results are plotted on synthetically generated data for logistic regression. For serverful programming, we use Message Passing Interface (MPI) with one `c3.8xlarge` master and 60 `t2.medium` workers in AWS EC2. In [4], the authors observed that many large-scale linear algebra operations on serverless systems take at least 30% more time compared to MPI-based computation on serverful systems. However, as shown in Fig. 10, we observe a slightly surprising trend 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.[11]

References

[1] 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.

[2] 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]We note that stochastic methods such as SGD perform worse that gradient descent since their update quality is poor, requiring more iterations (hence, more communication) to converge while not using the massive compute power of serverless. For example, 20% minibatch SGD in the setup of Fig. 9 requires $1.9\times$ more time than gradient descent with same number of workers.

[10]We remark that backtracking line-search required $\sim 13\%$ of the total time for NAG. Hence, as can be seen from Fig. 9, any well-tuned step-size method would still be significantly slower than OverSketched Newton.

[11]We do not compare with exact Newton in serverful sytems since the data is large and stored in the cloud. Computing the exact Hessian would require a large number of workers (e.g., we use 10,000 workers for exact Newton in EPSILON dataset) which is infeasible in existing serverful systems.

[3] 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.

[4] 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.

[5] Technavio, "Serverless architecture market by end-users and geography - global forecast 2019-2023." https://www.technavio.com/report/serverless-architecture-market-industry-analysis.

[6] E. Jonas *et al.*, "Cloud programming simplified: A berkeley view on serverless computing," *arXiv preprint arXiv:1902.03383*, 2019.

[7] 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.

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

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

[10] H. Wang, D. Niu, and B. Li, "Distributed machine learning with a serverless architecture," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 1288–1296, IEEE, 2019.

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

[12] J. Carreira, P. Fonseca, A. Tumanov, A. Zhang, and R. Katz, "Cirrus: a serverless framework for end-to-end ml workflows," in *Proceedings of the ACM Symposium on Cloud Computing*, pp. 13–24, 2019.

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

[14] 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.

[15] J. M. Hellerstein, J. Faleiro, J. E. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, and C. Wu, "Serverless computing: One step forward, two steps back," *arXiv preprint arXiv:1812.03651*, 2018.

[16] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[17] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *Siam Review*, vol. 60, no. 2, pp. 223–311, 2018.

[18] N. S. Wadia, D. Duckworth, S. S. Schoenholz, E. Dyer, and J. Sohl-Dickstein, "Whitening and second order optimization both destroy information about the dataset, and can make generalization impossible," *arXiv e-prints*, p. arXiv:2008.07545, Aug. 2020.

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

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

[21] P. Xu, F. Roosta, and M. W. Mahoney, "Newton-type methods for non-convex optimization under inexact hessian information," 2017.

[22] F. Roosta, Y. Liu, P. Xu, and M. W. Mahoney, "Newton-MR: Newton's method without smoothness or convexity," *arXiv preprint arXiv:1810.00303*, 2018.

[23] 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.

[24] S. Wang, F. Roosta-Khorasani, P. Xu, and M. W. Mahoney, "GIANT: Globally improved approximate Newton method for distributed optimization," in *Advances in Neural Information Processing Systems*, pp. 2332–2342, 2018.

[25] C.-H. Fang, S. B. Kylasa, F. Roosta-Khorasani, M. W. Mahoney, and A. Grama, "Distributed Second-order Convex Optimization," *ArXiv e-prints*, July 2018.

[26] Z. Yao, A. Gholami, K. Keutzer, and M. Mahoney, "Pyhessian: Neural networks through the lens of the hessian," *arXiv preprint arXiv:1912.07145*, 2019.

[27] Z. Yao, A. Gholami, S. Shen, K. Keutzer, and M. W. Mahoney, "Adahessian: An adaptive second order optimizer for machine learning," *arXiv preprint arXiv:2006.00719*, 2020.

[28] R. Anil, V. Gupta, T. Koren, K. Regan, and Y. Singer, "Second order optimization made practical," *arXiv preprint arXiv:2002.09018*, 2020.

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

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

[31] A. Gittens, A. Devarakonda, E. Racah, M. Ringenburg, L. Gerhardt, J. Kottalam, J. Liu, K. Maschhoff, S. Canon, J. Chhugani, *et al.*, "Matrix factorizations at scale: A comparison of scientific data analytics in spark and c+ mpi using three case studies," in *2016 IEEE International Conference on Big Data (Big Data)*, pp. 204–213, IEEE, 2016.

[32] 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.

[33] 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.

[34] 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)*, IEEE, 2018.

[35] V. Gupta, D. Carrano, Y. Yang, V. Shankar, T. Courtade, and K. Ramchandran, "Serverless straggler mitigation using local error-correcting codes," *IEEE International Conference on Distributed Computing and Systems (ICDCS), Singapore*, 2020.

[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] 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.

[38] V. Gupta, S. Kadhe, T. Courtade, M. W. Mahoney, and K. Ramchandran, "Oversketched Newton: Fast convex optimization for serverless systems," *arXiv preprint arXiv:1903.08857*, 2019.

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

[40] 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.

[41] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *Advances in Neural Inf. Processing Systems 30*, pp. 4403–4413, 2017.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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*, vol. 80, pp. 1358–1366, PMLR, 10–15 Jul 2018.

[47] J. R. Shewchuk *et al.*, "An introduction to the conjugate gradient method without the agonizing pain," 1994.

[48] J. Levin, "Note on convergence of minres," *Multivariate behavioral research*, vol. 23, no. 3, pp. 413–417, 1988.

[49] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "Emnist: an extension of mnist to handwritten letters," *arXiv preprint arXiv:1702.05373*, 2017.