Distributed Second Order Methods with Fast Rates and Compressed Communication

Rustem Islamov [1 2] Xun Qian [1] Peter Richtárik [1]

Abstract

We develop several new communication-efficient second-order methods for distributed optimization. Our first method, NEWTON-STAR, is a variant of Newton's method from which it inherits its fast local quadratic rate. However, unlike Newton's method, NEWTON-STAR enjoys the same per iteration communication cost as gradient descent. While this method is impractical as it relies on the use of certain unknown parameters characterizing the Hessian of the objective function at the optimum, it serves as the starting point which enables us to design practical variants thereof with strong theoretical guarantees. In particular, we design a stochastic sparsification strategy for learning the unknown parameters in an iterative fashion in a communication efficient manner. Applying this strategy to NEWTON-STAR leads to our next method, NEWTON-LEARN, for which we prove local linear and superlinear rates independent of the condition number. When applicable, this method can have dramatically superior convergence behavior when compared to state-of-the-art methods. Finally, we develop a globalization strategy using cubic regularization which leads to our next method, CUBIC-NEWTON-LEARN, for which we prove global sublinear and linear convergence rates, and a fast superlinear rate. Our results are supported with experimental results on real datasets, and show several orders of magnitude improvement on baseline and state-of-the-art methods in terms of communication complexity.

*Equal contribution [1]King Abdullah University of Science and Technology, Thuwal, Saudi Arabia [2]Moscow Institute of Physics and Technology, Dolgoprudny, Russia. Correspondence to: Rustem Islamov <islamov.ri@phystech.edu>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

1. Introduction

The prevalent paradigm for training modern supervised machine learning models is based on (regularized) empirical risk minimization (ERM) (Shalev-Shwartz and Ben-David, 2014), and the most commonly used optimization methods deployed for solving ERM problems belong to the class of stochastic first order methods (Robbins and Monro, 1951; Nemirovski et al., 2009). Since modern training data sets are very large and are becoming larger every year, it is increasingly harder to get by without relying on modern computing architectures which make efficient use of distributed computing. However, in order to develop efficient distributed methods, one has to keep in mind that communication among the different parallel workers (e.g. processors or compute nodes) is typically very slow, and almost invariably forms the main bottleneck in deployed optimization software and systems (Bekkerman et al., 2011). For this reason, further advances in the area of communication efficient distributed first order optimization methods for solving ERM problems are highly needed, and research in this area constitutes one of the most important fundamental endeavors in modern machine learning. Indeed, this research field is very active, and numerous advances have been made over the past decade (Seide et al., 2014; Wen et al., 2017; Alistarh et al., 2017; Bernstein et al., 2018; Mishchenko et al., 2019; Stich and Karimireddy, 2019; Tang et al., 2019).

1.1. Distributed optimization

We consider L2 regularized empirical risk minimization problems of the form

$$\min_{x\in\mathbb{R}^d} \left[P(x) := f(x) + \tfrac{\lambda}{2}\|x\|^2\right], \qquad (1)$$

where $f:\mathbb{R}^d\to\mathbb{R}$ is a smooth[1] convex function of the "average of averages" structure

$$f(x) := \tfrac{1}{n}\sum_{i=1}^n f_i(x), \quad f_i(x) := \tfrac{1}{m}\sum_{j=1}^m f_{ij}(x), \qquad (2)$$

[1]Function $\phi:\mathbb{R}^d\to\mathbb{R}$ is *smooth* if it is differentiable, and has L_ϕ Lipschitz gradient: $\|\nabla\phi(x)-\nabla\phi(y)\|\le L_\phi\|x-y\|$ for all $x,y\in\mathbb{R}^d$. We say that L_ϕ is the *smoothness constant* of ϕ.

and $\lambda \geq 0$ is a regularization parameter. Here n is the number of parallel workers (nodes), and m is the number of training examples handled by each node[2]. The value $f_{ij}(x)$ denotes the loss of the model parameterized by vector $x \in \mathbb{R}^d$ on the j^{th} example owned by the i^{th} node. This example is denoted as $a_{ij} \in \mathbb{R}^d$, and the corresponding loss function is $\varphi_{ij} : \mathbb{R} \to \mathbb{R}$, and hence we have

$$f_{ij}(x) := \varphi_{ij}(a_{ij}^\top x). \tag{3}$$

Thus, f represents the average loss/risk over all nm training datapoints, and problem (1) seeks to find the model whose (L2 regularized) empirical risk is minimized. We make the following assumption throughout the paper.

Assumption 1.1. *Problem (1) has at least one optimal solution x^*. For all i and j, the loss function $\varphi_{ij} : \mathbb{R} \to \mathbb{R}$ is γ-smooth, twice differentiable, and its second derivative $\varphi_{ij}'' : \mathbb{R} \to \mathbb{R}$ is ν-Lipschitz continuous.*

Note that in view of (3), the Hessian of f_{ij} at point x is

$$\mathbf{H}_{ij}(x) := \nabla^2 f_{ij}(x) = h_{ij}(x) a_{ij} a_{ij}^\top, \tag{4}$$

where

$$h_{ij}(x) := \varphi_{ij}''(a_{ij}^\top x). \tag{5}$$

In view of Assumption 1.1, we have $|\varphi_{ij}''(t)| \leq \gamma$ for all $t \in \mathbb{R}$, and

$$|h_{ij}(x) - h_{ij}(y)| \leq \nu |a_{ij}^\top x - a_{ij}^\top y| \leq \nu \|a_{ij}\| \|x - y\| \tag{6}$$

for all $x, y \in \mathbb{R}^d$. Let $R := \max_{ij} \|a_{ij}\|$. The Hessian of f_i is given by

$$\mathbf{H}_i(x) \overset{(2)}{=} \frac{1}{m} \sum_{j=1}^m \mathbf{H}_{ij}(x) \overset{(4)}{=} \frac{1}{m} \sum_{j=1}^m h_{ij}(x) a_{ij} a_{ij}^\top, \tag{7}$$

and the Hessian of f is given by

$$\mathbf{H}(x) \overset{(2)}{=} \frac{1}{n} \sum_{i=1}^n \mathbf{H}_i(x) \overset{(7)}{=} \frac{1}{nm} \sum_{i=1}^n \sum_{j=1}^m h_{ij}(x) a_{ij} a_{ij}^\top. \tag{8}$$

1.2. The curse of the condition number

All first order methods—distributed or not—suffer from a dependence on an appropriately chosen notion of a *condition number*[3]—a number that describes the difficulty of solving the problem by the method at hand. A condition number is a function of the goal we are trying to achieve (e.g., minimize the number of iterations vs minimize the number of communications), choice of the loss function, structure of the model we are trying to learn, and last but not least, the size and properties of the training data. In fact, most research in this area is motivated by the desire to design methods that would have a *reduced* dependence on the condition number. This is the case for many of the tricks heavily studied in the literature, including minibatching (Takáč et al., 2013), importance sampling (Needell et al., 2015; Zhao and Zhang, 2015), random reshuffling (Mishchenko et al., 2020), variance reduction (Schmidt et al., 2017; Johnson and Zhang, 2013; Xiao and Zhang, 2014; Defazio et al., 2014), momentum (Loizou and Richtárik, 2017a;b), adaptivity (Malitsky and Mishchenko, 2019), communication compression (Alistarh et al., 2017; Bernstein et al., 2018; Mishchenko et al., 2019), and local computation (Ma et al., 2017; Stich, 2020; Khaled et al., 2020). Research in this area is becoming saturated, and new ideas are needed to make further progress.

1.3. Newton's method to the rescue?

One of the ideas that undoubtedly crossed everybody's mind is the trivial observation that there *is* a very old and simple method which does *not* suffer from any conditioning issues: Newton's method. Indeed, when it works, Newton's method has a fast *local quadratic convergence rate* which is entirely independent of the condition number of the problem (Beck, 2014). While this is a very attractive property, developing scalable distributed variants of Newton's method that could also *provably outperform* gradient based methods remains a largely unsolved problem. To highlight the severity of the issues with extending Newton's method to stochastic and distributed settings common in machine learning, we note that until recently, we did not even have any Newton-type analogue of SGD that could provably work with small minibatch sizes, let alone minibatch size one (Kovalev et al., 2019). In contrast, SGD with minibatch size one is one of the simplest and well understood variants thereof (Needell et al., 2015), and much of modern development in the area of SGD methods is much more sophisticated. Most variants of Newton's method proposed for deployment in machine learning are heuristics, which is to say that they are not supported with any convergence guarantees, or have convergence guarantees without explicit rates, or suffer from rates that are worse than the rates of first order methods.

1.4. Contributions summary

We develop *several new fundamental Newton-type methods* which we hope make a marked step towards the ultimate goal of developing practically useful and *communication efficient distributed second order methods*. Our methods are designed with the explicit goal of supporting *efficient communication* in a distributed setting, and in sharp contrast

[2]All our results can be extended in a straightforward way to the more general case when node i contains m_i training examples. We decided to present the results in the special case $m = m_i$ for all i in order to simplify the notation.

[3]Example: if one wishes to minimize an L-smooth μ-strongly convex function and one cares about the number of gradient type iterations, the appropriate notion of a condition number is $\kappa := \frac{L}{\mu}$.

Table 1. Summary of algorithms proposed and convergence results proved in this paper.

Method	**Convergence**			**Rate independent of the condition number?**	**Theorem**
	result †	**type**	**rate**		
NEWTON-STAR (NS) (12)	$r_{k+1} \leq c r_k^2$	local	quadratic	✓	2.1
MAX-NEWTON (MN) Algorithm 4	$r_{k+1} \leq c r_k^2$	local	quadratic	✓	H.1
NEWTON-LEARN (NL1) Algorithm 1		local	linear	✓	3.2
	$r_{k+1} \leq c\theta_1^k r_k$	local	superlinear	✓	3.2
NEWTON-LEARN (NL2) Algorithm 2	$\Phi_2^k \leq \theta_2^k \Phi_2^0$	local	linear	✓	3.5
	$r_{k+1} \leq c\theta_2^k r_k$	local	superlinear	✓	3.5
CUBIC-NEWTON-LEARN (CNL) Algorithm 3	$\Delta_k \leq \frac{c}{k}$	global	sublinear	✗	F.3
	$\Delta_k \leq c\exp(-k/c)$	global	linear	✗	F.4
	$\Phi_3^k \leq \theta_3^k \Phi_3^0$	local	linear	✓	F.5
	$r_{k+1} \leq c\theta_3^k r_k$	local	superlinear	✓	F.5

Quantities for which we prove convergence: (i) distance to solution $r_k := \left\|x^k - x^*\right\|$; (ii) Lyapunov function $\Phi_q^k := \left\|x^k - x^*\right\|^2 + c_q \sum_{i=1}^n \sum_{j=1}^m (h_{ij}^k - h_{ij}(x^*))^2$ for $q = 1, 2, 3$, where $h_{ij}(x^*) = \varphi_{ij}''(a_{ij}^\top x^*)$ (see (5)); (iii) Function value suboptimality $\Delta_k := P(x^k) - P(x^*)$

† constant c is possibly different each time it appears in this table. Refer to the precise statements of the theorems for the exact values.

with most recent work, their design was heavily influenced by our desire to equip them with *strong convergence guarantees* typical for the classical Newton's method (Wallis, 1685; Raphson, 1697) and cubically regularized Newton's method (Griewank, 1981; Nesterov and Polyak, 2006). Our convergence results are summarized in Table 1.

• **First new method and its local quadratic convergence.** We first show that if we know the Hessian of the objective function at the optimal solution, then we can use it instead of the typical Hessian appearing in Newton's method, and the resulting algorithm, which we call NEWTON-STAR (NS), inherits local quadratic convergence behavior of Newton's method (see Theorem 2.1). In a distributed setting with a central orchestrating sever, each compute node only needs to send the local gradient to the server node, and no matrices need to be sent. While this method is not practically useful, it acts as a stepping stone to our next method, in which these deficiencies are removed. This method is described in Section 2. A somewhat different method with similar properties, which we call MAX-NEWTON[4], is described in Section H.

• **Second new method and its local linear and superlinear convergence.** Motivated by the above result, we propose a *learning scheme which enables us to learn the Hessian at the optimum iteratively in a communication efficient manner.* This scheme gives rise to our second new method: NEWTON-LEARN (NL). We analyze this method in two cases: (i) all individual loss functions are convex and $\lambda > 0$ (giving rise to the NL1 method), and (ii) the aggregate loss function P is strongly convex (giving rise to the NL2 method). Besides the local full gradient, each worker node needs to send additional information to the server node in order to learn the Hessian at the optimum. However, our learning scheme supports *compressed communication* with arbitrary compression level. This level can be chosen so that in each iteration, each node sends an equivalent of a few gradients to the server only. That is, we can achieve $O(d)$ communication complexity in each iteration. In both cases, we prove local linear convergence for a carefully designed Lyapunov function, and local superlinear convergence for the squared distance to optimum (see Theorems 3.2 and 3.5). Remarkably, all these rates are *independent of the condition number.*[5] The NL1 and NL2 methods and the associated theory are described in Section 3.

• **Third new method and its global convergence.** Next, we equip our learning scheme with a cubic regularization strategy (Griewank, 1981; Nesterov and Polyak, 2006), which leads to a new *globally* convergent method: CUBIC-NEWTON-LEARN (CNL). We establish global sublinear and linear convergence (for function values) guarantees for convex and strongly convex problems, respectively. The method can also achieve a fast local linear (for a Lyapunov function) and superlinear (for squared distance to solution) convergence in the strongly convex case. We describe this method and the associated theory in Section F.

• **Experiments.** Our theory is corroborated with numerical experiments showing the superiority of our methods to several state-of-the-art benchmarks, including DCGD (Khirirat et al., 2018), DIANA (Mishchenko et al., 2019; Horváth et al., 2019b), ADIANA (Li et al., 2020), BFGS (Broyden, 1967; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970), and DINGO (Crane and Roosta, 2019). Our methods can achieve com-

[4]In fact, this was the first method we developed, in Summer 2020, when we embarked on the research which eventually lead to the results presented in this paper.

[5]However, the size of the *neighborhood* of local convergence is not independent of the condition number.

munication complexity which is *several orders of magnitude better* than competing methods (see Section 4).

2. Three Steps Towards an Efficient Distributed Newton Type Method

In order to better explain the algorithms and results of this paper, we will proceed through several steps in a gradual explanation of the ideas that ultimately lead to our methods. While this is not the process we used to come up with our methods, in retrospect we believe that our methods and results will be understood more easily when seen as having been arrived at in this way. In other words, we have constructed what we believe is a plausible discovery story, one enabling faster and better comprehension. If these ideas seem to follow naturally, it is because we made a conscious effort to make then appear that way. The goal of this paper is to develop communication efficient variants of Newton's method for solving the distributed optimization problem (1).

2.1. Naive distributed implementation of Newton's method

Newton's method applied to problem (1) performs the iteration

$$\begin{aligned} x^{k+1} &= x^k - \left(\nabla^2 P(x^k)\right)^{-1} \nabla P(x^k) \\ &\overset{(1)}{=} x^k - \left(\mathbf{H}(x^k) + \lambda \mathbf{I}\right)^{-1} \nabla P(x^k). \end{aligned} \tag{9}$$

A naive way to implement this method in the *parameter server* framework is for each node i to compute the Hessian $\mathbf{H}_i(x^k)$ and gradient $\nabla f_i(x^k)$ and to communicate these objects to the server. The server then averages the local Hessians $\mathbf{H}_i(x^k)$ to produce $\mathbf{H}(x^k)$ via (8), and averages the local gradients $\nabla f_i(x^k)$ to produce $\nabla f(x^k)$. The server then adds $\lambda \mathbf{I}$ to the Hessian, producing $\mathbf{H}(x^k) + \lambda \mathbf{I} = \nabla^2 P(x^k)$, adds λx^k to the gradient, producing $\nabla P(x^k) = \nabla f(x^k) + \lambda x^k$, and subsequently performs the Newton step (9). The resulting vector x^{k+1} is then broadcasted to the nodes and the process is repeated.

This implementation mirrors the way GD and many other first order methods are implemented in the parameter server framework. However, unlike in the case of GD, where only $O(d)$ floats need to be sent and received by each node in each iteration, the upstream communication in Newton's method requires $O(d^2)$ floats to be communicated by each worker to the server. Since d is typically very large, this is prohibitive in practice. Moreover, computation of the Newton's step by the parameter server is much more expensive than simple averaging of the gradients performed by gradient type methods. However, in this paper we will not be concerned with the cost of the Newton step itself, as we will assume the server is powerful enough and the network connection is slow enough for this step not to be the main bottleneck of the iteration. Instead, we assume that the communication steps in general, and the $O(d^2)$ communication of the Hessian matrices in particular, is what forms the bottleneck. The $O(d)$ per node communication cost of the local gradients is negligible, and so is the $O(d)$ broadcast of the updated model.

2.2. A better implementation taking advantage of the structure of $\mathbf{H}_{ij}(x)$

The above naive implementation can be improved in the setting when $m < d^2$ by taking advantage of the explicit structure (7) of the local Hessians as a conic combination of positive semidefinite rank one matrices:

$$\mathbf{H}_i(x) = \tfrac{1}{m} \sum_{j=1}^{m} h_{ij}(x) a_{ij} a_{ij}^\top. \tag{10}$$

Indeed, assuming that the server has direct access to all the training data vectors $a_{ij} \in \mathbb{R}^d$ (these vectors can be sent to the server at the start of the process), node i can send the m coefficients $h_{i1}(x), \dots, h_{im}(x)$ to the server instead, and the server is then able to reconstruct the Hessian matrix $\mathbf{H}_i(x)$ from this information. This way, each node sends $O(m+d)$ floats to the server, which is a substantial improvement on the naive implementation in the regime when $m \ll d^2$. However, when $m \gg d$, the upstream communication cost is still substantially larger than the $O(d)$ cost of GD. If the server does not have enough memory to store all vectors a_{ij}, this procedure does not work.

2.3. NEWTON-STAR: Newton's method with a single Hessian

We now introduce a simple idea which, surprisingly, enables us to *remove the need to iteratively communicate any coefficients altogether.* Assume, for the sake of argument, that we know the values $h_{ij}(x^*)$ for all i, j. That is, assume the server has access to coefficients $h_{ij}(x^*)$ for all i, j, and that each node i has access to coefficients $h_{ij}(x^*)$ for $j = 1, \dots, m$, i.e., to the vector

$$h_i(x) := (h_{i1}(x), \dots, h_{im}(x)) \in \mathbb{R}^m \tag{11}$$

for $x = x^*$. Next, consider the following new Newton-like method which we call NEWTON-STAR (NS), where the "star" points to the method's reliance on the knowledge of the optimal solution x^*:

$$\begin{aligned} x^{k+1} &= x^k - \left(\nabla^2 P(x^*)\right)^{-1} \nabla P(x^k) \\ &\overset{(1)}{=} x^k - \left(\mathbf{H}(x^*) + \lambda \mathbf{I}\right)^{-1} \nabla P(x^k). \end{aligned} \tag{12}$$

Since the server knows $\mathbf{H}(x^*)$, all that the nodes need to communicate are the local gradients $\nabla f_i(x^k)$, which costs

$O(d)$ per node. The server then computes x^{k+1}, broadcasts it back to the nodes, and the process is repeated. This method has the same per-iteration $O(d)$ communication complexity as GD. However, as we show next, the number of iterations (which is the same as the number of communications) of NEWTON-STAR does not depend on the condition number – a property it borrows from the classical Newton's method. The following theorem says that NEWTON-STAR enjoys *local quadratic convergence*.

Theorem 2.1 (Local quadratic convergence). *Let Assumption 1.1 hold, and assume that $\mathbf{H}(x^*) \succeq \mu^*\mathbf{I}$ for some $\mu^* \geq 0$ (for instance, this holds if f is μ^*-strongly convex) and that $\mu^* + \lambda > 0$. Then for any starting point $x^0 \in \mathbb{R}^d$, the iterates of* NEWTON-STAR *for solving problem (1) satisfy the following inequality*[6]*:*

$$\|x^{k+1}-x^*\| \leq \frac{\nu}{2(\mu^*+\lambda)} \cdot \left(\frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m}\|a_{ij}\|^3\right) \cdot \|x^k-x^*\|^2.$$

Note that we do not need to assume f to be convex or strongly convex. Instead, all we need to assume is positive definiteness of the Hessian at the optimum. This implies local strong convexity, and since our convergence result is local, that is all we need.

Remark. Besides NEWTON-STAR, we have designed another new Newton-type method with a local quadratic rate. This method, which we call MAX-NEWTON, is similar to NEWTON-STAR in that it relies on the knowledge of the coefficients $h_{ij}(x^*)$ for $j = 1, \dots, m$. We describe this method in Appendix H.

3. NEWTON-LEARN: Learning the Hessian and Local Convergence Theory

In Sections 2.1, 2.2 and 2.3 we have gone through three steps in our story, with the first true innovation and contribution of this paper being the NEWTON-STAR method and its rate. We have now sufficiently prepared the ground to motivate our first *key* contribution: the NEWTON-LEARN method. We only outline the basic insights behind this method here; the details are included in Section 3.

3.1. The main iteration

In NEWTON-LEARN we maintain a sequence of vectors

$$h_i^k = (h_{i1}^k, \dots, h_{im}^k) \in \mathbb{R}^m,$$

for all $i = 1, \dots, n$ throughout the iterations $k \geq 0$ with the goal of *learning* the values $h_{ij}(x^*)$ for all i, j. That is, we construct the sequence with the explicit intention to enforce

$$h_{ij}^k \to h_{ij}(x^*) \qquad \text{as} \qquad k \to \infty. \tag{13}$$

Using $h_{ij}^k \approx h_{ij}(x^*)$ we estimate the Hessian $\mathbf{H}(x^*)$ via

$$\mathbf{H}(x^*) \approx \mathbf{H}^k := \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m} h_{ij}^k a_{ij}a_{ij}^\top, \tag{14}$$

and then perform a similar iteration to (12):

$$x^{k+1} = x^k - \left(\mathbf{H}^k + \lambda\mathbf{I}\right)^{-1}\nabla P(x^k). \tag{15}$$

3.2. Learning the coefficients: the idea

To complete the description of the method, we need to explain how the vectors h_i^{k+1} are updated. This is also the place where we can force the method to be communication efficient. Indeed, if we can design a rule that would enforce the update vectors $h_i^{k+1} - h_i^k$ to be *sparse*, say

$$\|h_i^{k+1} - h_i^k\|_0 \leq s \tag{16}$$

for some $1 \leq s \leq m$ and all i and k, then the upstream communication by each node in each iteration would be of the order $O(s+d)$ only (provided the server has access to all vectors a_{ij})! That is, each node i only needs to communicate s entries of the update vector $h_i^{k+1} - h_i^k$ as the rest are equal to zero, and each node also needs to communicate the d dimensional gradient $\nabla f_i(x^k)$. Note that $O(s+d)$ can be interpreted as an *interpolation* between the $O(m+d)$ per-iteration communication complexity of the structure-aware implementation of Newton's method from Section 2.2, and of the $O(d)$ per-iteration communication complexity of NEWTON-STAR described in Section 2.3.

In the more realistic regime when the server does *not* have access to the data $\{a_{ij}\}$, we ask each worker i to additionally send the corresponding s vectors a_{ij}, which costs extra $O(sd)$ in communication per node. However, when $s = O(1)$, this is the same per-iteration communication effort as that of GD.

We develop two different update rules defining the evolution of the vectors $h_1^k, \dots, h_n^k$. This first rule (see (19)) applies to the $\lambda > 0$ case and leads to our first variant of NEWTON-LEARN which we call NL1 (see Algorithm 1). This rule and the method are described in Section 3.5. The second rule applies also to the $\lambda = 0$ cases and leads to our second variant of NEWTON-LEARN which we call NL2 (see Algorithm 2). This rule and the method are described in Section 3.6.

3.3. Outline of fast local convergence theory

We show in Theorem 3.2 (covering NL1) and Theorem 3.5 (covering NL2) that NEWTON-LEARN enjoys a local linear

[6]While this inequality holds for *any* $x^0 \in \mathbb{R}^d$, it is only meaningful if it leads to a contraction in the distance to optimum, and this means that Theorem 2.1 ensures convergence to x^* only if x^0 is sufficiently close to x^*. That is, the theorem implies *local* quadratic convergence only. To see this, let $c > 0$ be the constant on the right hand side of the inequality, so that $\|x^{k+1} - x^*\| \leq c\|x^k - x^*\|^2$ for all k. It is easy to see that requiring $\|x^0 - x^*\| \leq \frac{1}{2c}$ suffices to ensure convergence.

rate wrt a certain Lyapunov function which involves the term $\|x^k - x^*\|^2$ and also all terms of the form $\|h_i^k - h_i(x^*)\|^2$. This means that i) the main iteration (15) works, i.e., x^k converges to x^* at a local linear rate, and that ii) the learning procedure works, and the desired convergence described in (13) occurs at a local linear rate. In addition, we also establish a local superlinear rate of $\|x^k - x^*\|^2$. Remarkably, these rates are *independent of any condition number, which is in sharp contrast with virtually all results on distributed Newton-type methods we are aware of.*

Moreover, we wish to remark that second order methods are not typically analyzed using a Lyapunov style analysis. Indeed, we only know of a couple works that do so. First, Kovalev et al. (2019) develop stochastic Newton and cubic Newton methods of a different structure and scope from ours. They do not consider distributed optimization nor communication compression. Second, Kovalev et al. (2020) develop a stochastic BFGS method. Again, their method and scope is very different from ours. Hence, *our analysis may be of independent interest as it adds to the arsenal of theoretical tools which could be used in a more precise analysis of other second order methods.*

3.4. Compressed learning

Instead of merely relying on sparse updates for the vectors h_i^k (see (16)), we provide a more general communication compression strategy which includes sparsification as a special case (Alistarh et al., 2017). We do so via the use of a *random compression* operator. We say that a randomized map $\mathcal{C} : \mathbb{R}^m \to \mathbb{R}^m$ is a *compression operator (compressor)* if there exists a constant $\omega \geq 0$ such that the following relations hold for all $x \in \mathbb{R}^m$:

$$\mathbb{E}\left[\mathcal{C}(x)\right] = x \tag{17}$$

$$\mathbb{E}\left[\|\mathcal{C}(x)\|^2\right] \leq (\omega + 1)\|x\|^2. \tag{18}$$

The identity compressor $\mathcal{C}(x) \equiv x$ satisfies these relations with $\omega = 0$. The larger the variance parameter ω is allowed to be, the easier it can be to construct a compressor $\mathcal{C}$ for which the value $\mathcal{C}(x)$ can be encoded using a small number of bits only. We refer the reader to (Beznosikov et al., 2020) for a list of several compressors and their properties.

3.5. NL1 (learning in the $\lambda > 0$ case)

We now consider the case where all loss functions φ_{ij} are convex and $\lambda > 0$.

Assumption 3.1. *Each φ_{ij} is convex, $\lambda > 0$.*

When combined with Assumption 1.1, Assumption 3.1 implies that $\varphi_{ij}''(t) \geq 0$ for all t, hence $h_{ij}(x) = \varphi_{ij}''(a_i^\top x) \geq 0$ for all $x \in \mathbb{R}^d$. In particular, $h_{ij}(x^*) \geq 0$ for all i, j. Since we wish to construct a sequence of vectors $h_i^k = (h_{i1}^k, \ldots, h_{im}^k) \in \mathbb{R}^m$ satisfying $h_{ij}^k \to h_{ij}(x^*)$, it makes sense to try to enforce all vectors in this sequence to have *nonnegative entries*: $h_{ij}^k \geq 0$.

Since $\mathbf{H}^k$ arises as a linear combination of the rank-one matrices $a_{ij}a_{ij}^\top$ (see (14)), this makes $\mathbf{H}^k$ positive semidefinite, which in turn means that the matrix $\mathbf{H}^k + \lambda\mathbf{I}$ appearing in the main iteration (15) of NEWTON-LEARN is invertible, and hence the iteration is *well defined.*[7]

The learning iteration and the NL1 algorithm. In particular, in NEWTON-LEARN each node i computes the vector $h_i(x^k) \in \mathbb{R}^m$ of second derivatives defined in (11), and then performs the update

$$\boxed{h_i^{k+1} = \left[h_i^k + \eta\mathcal{C}_i^k(h_i(x^k) - h_i^k)\right]_+,} \tag{19}$$

where $\eta > 0$ is a learning rate, $\mathcal{C}_i^k$ is a freshly sampled compressor by node i at iteration k. By $[\cdot]_+$ we denote the positive part function applied element-wise, defined for scalars as follows: $[t]_+ = t$ if $t \geq 0$ and $[t]_+ = 0$ otherwise.

We remark that it is possible to interpret the learning procedure (19) as one step of projected stochastic gradient descent (SGD) applied to a certain quadratic optimization problem whose unique solution is the vector $h_i(x^k)$.

The NL1 algorithm (Algorithm 1) arises as the combination of the Newton-like update (15) (adjusted to take account of the explicit regularizer) and the learning procedure (19). It is easy to see that the update rule for $\mathbf{H}^k$ in NL1 is designed to ensure that $\mathbf{H}^k$ remains of the form $\mathbf{H}^k = \frac{1}{n}\sum_{i=1}^n \mathbf{H}_i^k$, where $\mathbf{H}_i^k = \frac{1}{m}\sum_{j=1}^m h_{ij}^k a_{ij}a_{ij}^\top$. The update rule for x^k, performed by the server, is identical to (15), with an extra provision for the regularizer. The vector x^{k+1} is broadcasted to all workers. Let us comment on how the key communication step is implemented. If the server does not have direct access to the training data vectors $\{a_{ij}\}$, we choose Option 1, otherwise we choose Option 2. A key property of NL1 is that the server is able to maintain copies of the learning vectors h_i^k *without the need for these vectors to be communicated by the workers to the server.* Indeed, provided the workers and the server agree on the same set of initial vectors $h_1^0, \ldots, h_n^0$, update (19) can be independently computed by the server as well from its memory state h_i^k and the compressed message $\mathcal{C}_i^k(h_i(x^k) - h_i^k)$ received from node i. This strategy is reminiscent of the way the key step in the first-order method DIANA (Mishchenko et al., 2019; Horváth et al., 2019b) is executed. In this sense, NL1 can be seen as arising from a successful marriage of Newton's method and the DIANA trick.

[7]Positive definiteness of Hessian estimates is enforced in several popular quasi-Newton methods as well; for instance, in the BFGS method (Broyden, 1967; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970). However, quasi-Newton methods operate in a markedly different manner, and the way in which positive definiteness is enforced there is also different.

Algorithm 1 NL1: NEWTON-LEARN ($\lambda > 0$ case)

```
**Parameters:** learning rate $\eta > 0$
**Initialization:** $x^0 \in \mathbb{R}^d$; $h_1^0, \ldots, h_n^0 \in \mathbb{R}_+^m$; $\mathbf{H}^0 = \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m} h_{ij}^0 a_{ij}a_{ij}^\top \in \mathbb{R}^{d\times d}$
**for** $k = 0, 1, 2, \ldots$ **do**
    Broadcast $x^k$ to all workers
    **for** each node $i = 1, \ldots, n$ **do**
        Compute local gradient $\nabla f_i(x^k)$
        $h_i^{k+1} = [h_i^k + \eta \mathcal{C}_i^k(h_i(x^k) - h_i^k)]_+$
        Send $\nabla f_i(x^k)$ and $m_i^k = \mathcal{C}_i^k(h_i(x^k) - h_i^k)$ to server
        **Option 1:** Send $\{a_{ij} : h_{ij}^{k+1} - h_{ij}^k \neq 0\}$ to server
        **Option 2:** Do nothing if server knows $\{a_{ij} : \forall j\}$
    **end for**
    $x^{k+1} = x^k - \left(\mathbf{H}^k + \lambda \mathbf{I}\right)^{-1}\left(\frac{1}{n}\sum_{i=1}^{n}\nabla f_i(x^k) + \lambda x^k\right)$
    For each $i$, compute $h_i^{k+1}$ via $h_i^{k+1} = [h_i^k + \eta m_i^k]_+$
    $\mathbf{H}^{k+1} = \mathbf{H}^k + \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m}(h_{ij}^{k+1} - h_{ij}^k)a_{ij}a_{ij}^\top$
**end for**
```

Theory. In our theoretical results we rely on the Lyapunov function

$$\Phi_1^k := \|x^k - x^*\|^2 + \tfrac{1}{3mn\eta\nu^2R^2}\mathcal{H}^k,$$

where $\mathcal{H}^k := \sum_{i=1}^n \|h_i^k - h_i(x^*)\|^2$. Our main theorem follows.

Theorem 3.2 (Convergence of NL1). *Let Assumptions 1.1 and 3.1 hold. Let $\eta \leq \frac{1}{\omega+1}$ and assume that $\|x^k - x^*\|^2 \leq \frac{\lambda^2}{12\nu^2R^6}$ for all $k \geq 0$. Then for Algorithm 1 we have the inequalities*

$$\begin{aligned}\mathbb{E}[\Phi_1^k] &\leq \theta_1^k \Phi_1^0,\\ \mathbb{E}\left[\tfrac{\|x^{k+1}-x^*\|^2}{\|x^k-x^*\|^2}\right] &\leq \theta_1^k\left(6\eta + \tfrac{1}{2}\right)\tfrac{\nu^2R^6}{\lambda^2}\Phi_1^0,\end{aligned}$$

where $\theta_1 := 1 - \min\left\{\frac{\eta}{2}, \frac{5}{8}\right\}$.

Since the stepsize bound $\eta \leq \frac{1}{\omega+1}$ is independent of the condition number, and since from the proof of Lemma 3.3 we have $\frac{\nu^2R^6}{\lambda^2}\Phi_1^0 \leq \frac{1}{12} + \frac{1}{36\eta}$, the linear convergence rates of $\mathbb{E}[\Phi_1^k]$ and $\mathbb{E}\left[\frac{\|x^{k+1}-x^*\|^2}{\|x^k-x^*\|^2}\right]$ are both *independent of the condition number*. Next, we explore under what conditions we can guarantee for all the iterates to stay in a small neighborhood.

Lemma 3.3. *Let Assumptions 1.1 and 3.1 hold. Assume h_{ij}^k is a convex combination of $\{h_{ij}(x^0), h_{ij}(x^1), ..., h_{ij}(x^k)\}$ for all i, j and k. Assume $\|x^0 - x^*\|^2 \leq \frac{\lambda^2}{12\nu^2R^6}$. Then*

$$\|x^k - x^*\|^2 \leq \tfrac{\lambda^2}{12\nu^2R^6} \quad \textit{for all} \quad k \geq 0.$$

It is easy to verify that if we choose $h_{ij}^0 = h_{ij}(x^0)$ and use the random sparsification compressor and $\eta \leq \frac{1}{\omega+1}$, then h_{ij}^k is always a convex combination of $\{h_{ij}(x^0), h_{ij}(x^1), ..., h_{ij}(x^k)\}$ for $k \geq 0$. Thus, from Lemma 3.3 we can guarantee that all the iterates stay in the small neighborhood assumed in Theorem 3.2 as long as the initial point x^0 is in it.

3.6. NL2 (learning in the $\lambda \geq 0$ case)

In this subsection, we consider the case where P is μ-strongly convex. Note that we do not require the components f_{ij} to be convex.

Assumption 3.4. *P is μ-strongly convex, $|h_{ij}^k| \leq \gamma$ for $k \geq 0$.*

The learning iteration and the NL2 algorithm. As in Algorithm 1, we use a sequence of vectors $\{h_i^k\}_{k\geq 0}$ to learn $h_i(x^*)$. However, this time we rely on a different technique for enforcing positive definiteness of the Hessian estimator. Since λ can be zero, our previous technique aimed at forcing the coefficients h_{ij}^k to be nonnegative will not work. So, we give up on this, and instead of (19) we use the simpler update

$$\boxed{h_i^{k+1} = h_i^k + \eta\mathcal{C}_i^k(h_i(x^k) - h_i^k).} \tag{20}$$

In order to guarantee positive definiteness of the Hessian estimator $\mathbf{H}^k + \lambda\mathbf{I}$ we instead rely on the second part of Assumption 3.4. Provided that there exists $\gamma > 0$ such that $|h_{ij}^k| \leq \gamma$ for all i, j, note that $\frac{h_{ij}(x^k)+2\gamma}{h_{ij}^k+2\gamma}$ is always positive. Noticing that each $a_{ij}a_{ij}^\top$ is positive semidefinite and that $\nabla^2 f(x^k)$ can be expressed in the form

$$\tfrac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m}\left(\tfrac{h_{ij}(x^k)+2\gamma}{h_{ij}^k+2\gamma}\cdot(h_{ij}^k + 2\gamma) - 2\gamma\right)a_{ij}a_{ij}^\top,$$

for $\beta^k := \max_{i,j}\frac{h_{ij}(x^k)+2\gamma}{h_{ij}^k+2\gamma}$, we get the inequality $\frac{1}{nm}\sum_{i=1}^n\sum_{j=1}^m\left[\beta^k(h_{ij}^k + 2\gamma) - 2\gamma\right]a_{ij}a_{ij}^\top - \nabla^2 f(x^k) = \frac{1}{nm}\sum_{i=1}^n\sum_{j=1}^m\left[\beta^k - \frac{h_{ij}(x^k)+2\gamma}{h_{ij}^k+2\gamma}\right](h_{ij}^k + 2\gamma)a_{ij}a_{ij}^\top \succeq \mathbf{0}$, where $\mathbf{0}$ is the $d \times d$ zero matrix, and $\mathbf{A} \succeq \mathbf{B}$ means $\mathbf{A} - \mathbf{B}$ is positive semidefinite. Thus, if we can maintain the Hessian estimator in the form

$$\mathbf{H}^k := \tfrac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m}\left[\beta^k(h_{ij}^k + 2\gamma) - 2\gamma\right]a_{ij}a_{ij}^\top,$$

then $\mathbf{H}^k + \lambda\mathbf{I} \succeq \nabla^2 f(x^k) + \lambda\mathbf{I} = \nabla^2 P(x^k) \succeq \mu\mathbf{I}$, where the last inequality follows from Assumption 3.4. To achieve this goal, we use an auxiliary matrix $\mathbf{A}^k$, and maintain $\mathbf{A}^k = \frac{1}{nm}\sum_{i=1}^n\sum_{j=1}^m(h_{ij}^k + 2\gamma)a_{ij}a_{ij}^\top$, and $\mathbf{H}^k = \beta^k\mathbf{A}^k - 2\gamma\cdot\frac{1}{nm}\sum_{i=1}^n\sum_{j=1}^m a_{ij}a_{ij}^\top$. The rest of Algorithm 2 is the same as Algorithm 1.

Theory. Our analysis of NL2 relies on the Lyapunov function

$$\Phi_2^k := \|x^k - x^*\|^2 + \tfrac{1}{3mn\eta\nu^2R^2}\mathcal{H}^k,$$

Algorithm 2 NL2: NEWTON-LEARN (general case)

```
**Parameters:** $\eta > 0$; $\gamma > 0$
**Initialization:** $x^0 \in \mathbb{R}^d$; $h_i^0 \in \mathbb{R}_+^m$; $\mathbf{A}^0 = \frac{1}{nm}\sum_{i=1}^n \sum_{j=1}^m (h_{ij}^0 + 2\gamma) a_{ij} a_{ij}^\top \in \mathbb{R}^{d\times d}$
for $k = 0, 1, 2, \dots$ do
    broadcast $x^k$ to all workers
    for $i = 1, \dots, n$ do
        Compute local gradient $\nabla f_i(x^k)$
        $h_i^{k+1} = h_i^k + \eta \mathcal{C}_i^k(h_i(x^k) - h_i^k)$
        $\beta_i^k = \max_{j\in[m]} \frac{h_{ij}(x^k)+2\gamma}{h_{ij}^k + 2\gamma}$
        Send $\nabla f_i(x^k)$, $\beta_i^k$, and $\eta\mathcal{C}_i^k(h_i(x^k) - h_i^k)$ to server
        **Option 1:** Send $\{a_{ij} : h_{ij}^{k+1} - h_{ij}^k \neq 0\}$ to server
        **Option 2:** Do nothing if server knows $\{a_{ij} : \forall j\}$
    end for
    $\beta^k = \max_i\{\beta_i^k\}$
    $\mathbf{H}^k = \beta^k \mathbf{A}^k - 2\gamma \cdot \frac{1}{nm}\sum_{i=1}^n\sum_{j=1}^m a_{ij}a_{ij}^\top \in \mathbb{R}^{d\times d}$
    $x^{k+1} = x^k - \left(\mathbf{H}^k + \lambda \mathbf{I}\right)^{-1}\left(\frac{1}{n}\sum_{i=1}^n \nabla f_i(x^k) + \lambda x^k\right)$
    $\mathbf{A}^{k+1} = \mathbf{A}^k + \frac{1}{nm}\sum_{i=1}^n\sum_{j=1}^m (\eta\mathcal{C}_i^k(h_i(x^k) - h_i^k))_j a_{ij}a_{ij}^\top$
end for
```

where $\mathcal{H}^k := \sum_{i=1}^n \|h_i^k - h_i(x^*)\|^2$. We now present our main convergence result for NL2.

Theorem 3.5 (Convergence of NL2). *Let Assumptions 1.1 and 3.4 hold. Assume $\eta \le \frac{1}{\omega+1}$ and $\|x^k - x^*\|^2 \le \frac{\mu^2}{432mn\nu^2R^6}$ for all $k \ge 0$. Then for Algorithm 2 we have the inequalities*

$$\begin{aligned} \mathbb{E}[\Phi_2^k] &\le \theta_2^k \Phi_2^0, \\ \mathbb{E}\left[\frac{\|x^{k+1}-x^*\|^2}{\|x^k - x^*\|^2}\right] &\le \theta_2^k (3mn\eta + 1)\frac{72\nu^2R^6}{\mu^2}\Phi_2^0, \end{aligned}$$

where $\theta_2 := 1 - \min\left\{\frac{\eta}{2}, \frac{1}{2}\right\}$.

As before, we give sufficient conditions guaranteeing that the iterates stay in a small neighborhood of the optimum.

Lemma 3.6. *Let Assumptions 1.1 and 3.4 hold. Assume h_{ij}^k is a convex combination of $\{h_{ij}(x^0), h_{ij}(x^1), ..., h_{ij}(x^k)\}$ for all i, j and k. Assume $\|x^0 - x^*\|^2 \le \frac{\mu^2}{432mn\nu^2R^6}$. Then*

$$\|x^k - x^*\|^2 \le \frac{\mu^2}{432mn\nu^2R^6} \quad \textit{for all} \quad k \ge 0.$$

If we choose $h_{ij}^0 = h_{ij}(x^0)$, use a random compressor with variance ω, and choose stepsize $\eta \le \frac{1}{\omega+1}$, then h_{ij}^k is a convex combination of $\{h_{ij}(x^0), h_{ij}(x^1), ..., h_{ij}(x^k)\}$ for all $k \ge 0$. Thus, via Lemma 3.6 we can guarantee all the iterates to be in the small neighborhood required by Theorem 3.5 as long as the initial point x^0 is in it.

4. Experiments

We now study the empirical performance of our second order methods NL1, NL2 and CNL, and compare them with relevant benchmarks and with state-of-the-art methods. We test on the regularized logistic regression problem

$$\min_{x\in\mathbb{R}^d}\left\{\frac{1}{n}\sum_{i=1}^n \frac{1}{m}\sum_{j=1}^m \log\left(1+\exp(-b_{ij}a_{ij}^\top x)\right) + \frac{\lambda}{2}\|x\|^2\right\},$$

where $\{a_{ij}, b_{ij}\}_{j\in[m]}$ are data samples at the i-th node.

Data sets. In our experiments we use four standard datasets from the LIBSVM library: `a2a`, `a7a`, `a9a`, and `w8a`. More experiments are provided in the appendix.

Compression operators. For the first order methods we use three compression operators: random sparsification (Stich et al., 2018), random dithering (Alistarh et al., 2017), and natural compression (Horváth et al., 2019a). For random-r sparsification, the number of communicated bits per iteration is $32r + \log_2\binom{d}{r}$, and we choose $r = d/4$. For random dithering, we choose $s = \sqrt{d}$, which means the number of communicated bits per iteration is $2.8d + 32$. For natural compression, the number of communicated bits per iteration is $9d$ bits. For NL1 and NL2 we use the random-r sparsification operator with different values of r. For CNL we use the random sparsification $\mathcal{C}_p$ (the definition is given in the appendix) with $p = 1/20$ and $r = 1$.

Parameter setting. In our experiments, we use the theoretical parameters (e.g., stepsizes) for all the three algorithms: vanilla Distributed Compressed Gradient Descent (DCGD) (Khirirat et al., 2018), DIANA (Mishchenko et al., 2019), and ADIANA (Li et al., 2020). As the initial approximation of the Hessian in BFGS (Broyden, 1967; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970), we use $\mathbf{H}^0 = \nabla^2 P(x^0)$, and the stepsize is 1. We set the same constants in DINGO (Crane and Roosta, 2019) as they did: $\theta = 10^{-4}, \phi = 10^{-6}, \rho = 10^{-4}$, and use backtracking line search for DINGO to select the largest stepsize in $\{1, 2^{-1}, 2^{-2}, 2^{-4}, \dots, 2^{-10}\}$. We conduct experiments for two values of the regularization parameter λ: $10^{-3}, 10^{-4}$. For the `a2a` dataset, we set number of nodes to $n = 15$ and the size of local dataset to $m = 151$. For the remaining datasets we choose: `a7a` ($n = 100, m = 161$), `a9a` ($n = 80, m = 407$), `w8a` ($n = 142, m = 350$). In the figures we plot the relation of the optimality gap $P(x^k) - P(x^*)$ and the number of accumulated transmitted bits or iterations. The optimal value $P(x^*)$ in each case is the function value at the 20-th iterate of standard Newton's method. In all plots, "communicated bits" refers to the total number of bits that all nodes send to the server. We adopt the realistic setting where the server does not have access to the local data (**Option 1**).

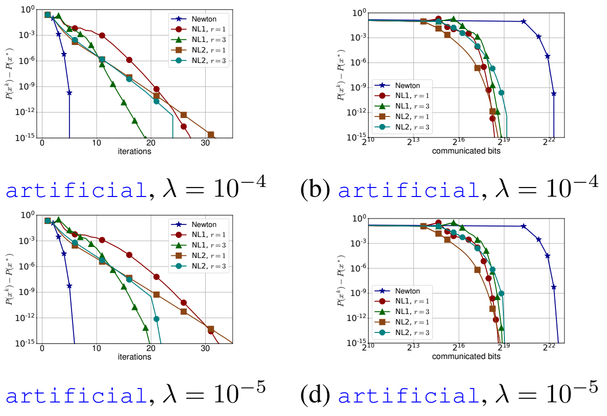

Figure 1. Comparison of NL1 and NL2, with Newton's method in terms of iteration complexity for (a), (c); in terms of communication complexity for (b), (d).

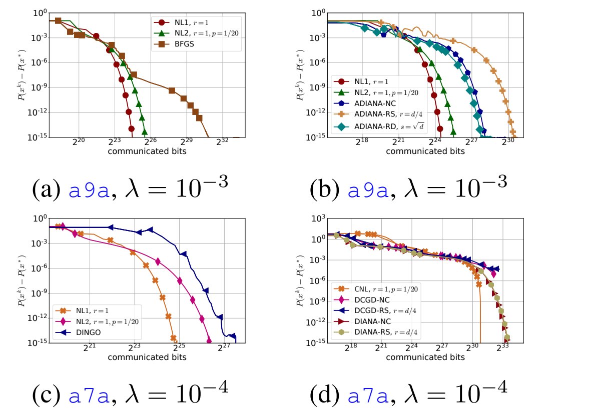

Figure 2. Comparison of NL1 and NL2 (a) with BFGS; (b) with ADIANA; (c) with DINGO; (d) CNL with DIANA and DCGD; in terms of communication complexity.

4.1. Comparison with Newton's method

In our next experiment we compare NL1 and NL2, using different values of r for random-r compression, with Newton's method; see Figure 1. We clearly see that Newton's method performs better than NL1 and NL2 in terms of iteration complexity, as expected. However, our methods have better communication efficiency than Newton's method, by *several orders of magnitude*. Moreover, we see that the smaller r is, the better NL1 and NL2 perform in terms of communication complexity.

4.2. Comparison with BFGS

In our next test, we compare NL1 and NL2 using $r = 1$ with BFGS in Figure 2(a). The experimental results clearly show that our methods have faster convergence rate in terms of communication complexity.

4.3. Comparison with ADIANA

Next, we compare NL1 and NL2 with ADIANA with three different compressors: random sparsification, random dithering, and natural compression; see Figure 2(b). The results indeed show that our methods converge to the optimum using fewer bits than ADIANA for all three compressors. For brevity, we denote ADIANA with natural compression, random sparsification, and random dithering by ADIANA-NC, ADIANA-RS, and ADIANA-RD, respectively.

4.4. Comparison with DINGO

We now compare NL1 and NL2 with DINGO. The results in Figure 2(c) show that our methods are more communication efficient than DINGO, by *many orders of magnitude.*

4.5. Comparison with DCGD and DIANA

Finally, we compare CNL using the random sparsification $\mathcal{C}_p$ with DCGD and DIANA with two compression operators: random sparsification and natural compression in Figure 2(d). According to the numerical experiments, CNL is more communication efficient method than others. We use the same notation as mentioned above.

References

D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic. QSGD: Communication-efficient SGD via gradient quantization and encoding. *Advances in Neural Information Processing Systems*, pages 1709–1720, 2017.

Amir Beck. *Introduction to Nonlinear Optimization: Theory, Algorithms, and Applications with MATLAB*. Society for Industrial and Applied Mathematics, USA, 2014. ISBN 1611973643.

Ron Bekkerman, Mikhail Bilenko, and John Langford. *Scaling up machine learning: Parallel and distributed approaches*. Cambridge University Press, 2011.

J. Bernstein, Y. X. Wang, K. Azizzadenesheli, and A. Anandkumar. SignSGD: Compressed optimisation for non-convex problems. *The 35th International Conference on Machine Learning*, pages 560–569, 2018.

Aleksandr Beznosikov, Samuel Horváth, Peter Richtárik, and Mher Safaryan. On biased compression for distributed learning. *arXiv:2002.12410*, 2020.

Charles G Broyden. Quasi-Newton methods and their application to function minimisation. *Mathematics of Computation*, 21(99):368–381, 1967.

Rixon Crane and Fred Roosta. DINGO: Distributed Newton-type method for gradient-norm optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 9498–9508. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/9718db12cae6be37f7349779007ee589-Paper.pdf.

Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 1646–1654. Curran Associates, Inc., 2014.

Rodger Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–323, 1970.

Avishek Ghosh, Raj Kumar Maity, Arya Mazumdar, and Kannan Ramchandran. Communication efficient distributed approximate Newton method. In *IEEE International Symposium on Information Theory (ISIT)*, 2020. doi: 10.1109/ISIT44484.2020.9174216.

Donald Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24(109):23–26, 1970.

Nicholas IM Gould, Daniel P Robinson, and H Sue Thorne. On solving trust-region and other regularised subproblems in optimization. *Mathematical Programming Computation*, 2(1):21–57, 2010.

Andreas Griewank. The modification of Newton's method for unconstrained optimization by bounding cubic terms. Technical report, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, 1981. Technical Report NA/12.

Filip Hanzely, Nikita Doikov, Yurii Nesterov, and Peter Richtarik. Stochastic subspace cubic Newton method. In *International Conference on Machine Learning*, pages 4027–4038. PMLR, 2020.

Samuel Horváth, Chen-Yu Ho, Ľudovít Horváth, Atal Narayan Sahu, Marco Canini, and Peter Richtárik. Natural compression for distributed deep learning. *arXiv preprint arXiv:1905.10988*, 2019a.

Samuel Horváth, Dmitry Kovalev, Konstantin Mishchenko, Sebastian Stich, and Peter Richtárik. Stochastic distributed learning with gradient quantization and variance reduction. *arXiv preprint arXiv:1904.05115*, 2019b.

Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *NIPS*, pages 315–323, 2013.

Ahmed Khaled, Konstantin Mishchenko, and Peter Richtárik. Tighter theory for local SGD on identical and heterogeneous data. In *The 23rd International Conference on Artificial Intelligence and Statistics (AISTATS 2020)*, 2020.

Sarit Khirirat, Hamid Reza Feyzmahdavian, and Mikael Johansson. Distributed learning with compressed gradients. In *arXiv preprint arXiv:1806.06573*, 2018.

Dmitry Kovalev, Konstanting Mishchenko, and Peter Richtárik. Stochastic Newton and cubic Newton methods with simple local linear-quadratic rates. In *NeurIPS Beyond First Order Methods Workshop*, 2019.

Dmitry Kovalev, Robert M. Gower, Peter Richtárik, and Alexander Rogozin. Fast linear convergence of randomized BFGS. *arXiv:2002.11337*, 2020.

Zhize Li, Dmitry Kovalev, Xun Qian, and Peter Richtárik. Acceleration for compressed gradient descent in distributed and federated optimization. In *International Conference on Machine Learning*, 2020.

Hongzhou Lin, Julien Mairal, and Zaid Harchaoui. A universal catalyst for first-order optimization. *arXiv preprint arXiv:1506.02186*, 2015.

Nicolas Loizou and Peter Richtárik. Linearly convergent stochastic heavy ball method for minimizing generalization error. In *NIPS Workshop on Optimization for Machine Learning*, 2017a.

Nicolas Loizou and Peter Richtárik. Momentum and stochastic momentum for stochastic gradient, Newton, proximal point and subspace descent methods. *arXiv:1712.09677*, 2017b.

Chenxin Ma, Jakub Konečný, Martin Jaggi, Virginia Smith, Michael I. Jordan, Peter Richtárik, and Martin Takáč. Distributed optimization with arbitrary local solvers. *Optimization Methods and Software*, 32(4):813–848, 2017.

Yura Malitsky and Konstantin Mishchenko. Adaptive gradient descent without descent. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 6702–6712. PMLR, 13–18 Jul 2019.

Konstantin Mishchenko, Eduard Gorbunov, Martin Takáč, and Peter Richtárik. Distributed learning with compressed gradient differences. *arXiv preprint arXiv:1901.09269*, 2019.

Konstantin Mishchenko, Ahmed Khaled, and Peter Richtárik. Random reshuffling: Simple analysiswith vast improvements. In *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, 2020.

Deanna Needell, Nathan Srebro, and Rachel Ward. Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm. *Mathematical Programming*, 155(1–2):549–573, 2015.

Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19(4):1574–1609, 2009.

Yurii Nesterov and Boris T. Polyak. Cubic regularization of Newton method and its global performance. *Mathematical Programming*, 108(1):177–205, 2006.

Boris Polyak and Andrey Tremba. New versions of Newton method: step-size choice, convergence domain and under-determined equations. *arXiv preprint arXiv:1703.07810*, 2019.

Boris Polyak and Andrey Tremba. New versions of newton method: step-size choice, convergence domain and under-determined equations. *Optimization Methods and Software*, 35(6):1272–1303, 2020.

Josepho Raphson. Analysis aequationum universalis seu ad aequationes algebraicas resolvendas methodus generalis, & expedita, ex nova infinitarum serierum methodo, deducta ac demonstrata. *Oxford: Richard Davis*, 1697.

Sashank J. Reddi, Jakub Konečný, Peter Richtárik, Barnabás Póczos, and Alex Smola. AIDE: fast and communication efficient distributed optimization. *arXiv:1608.06879*, 2016.

H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

M. Schmidt, N. Le Roux, and F. Bach. Minimizing finite sums with the stochastic average gradient. *Math. Program.*, 162(1-2):83–112, 2017.

F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu. 1-bit stochastic gradient descent and its application to data- parallel distributed training of speech DNNs. *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: from theory to algorithms*. Cambridge University Press, 2014.

Ohad Shamir, Nati Srebro, and Tong Zhang. Communication-efficient distributed optimization using an approximate Newton-type method. In *Proceedings of the 31st International Conference on Machine Learning, PMLR*, volume 32, pages 1000–1008, 2014.

David F Shanno. Conditioning of quasi-Newton methods for function minimization. *Mathematics of computation*, 24(111):647–656, 1970.

S. U. Stich and S. P. Karimireddy. The error-feedback framework: Better rates for SGD with delayed gradients and compressed communication. *arXiv: 1909.05350*, 2019.

Sebastian U. Stich. Local SGD converges fast and communicates little. In *International Conference on Learning Representations*, 2020.

Sebastian U. Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. Sparsified SGD with memory. In *Advances in Neural Information Processing Systems*, pages 4447–4458, 2018.

Martin Takáč, Avleen Bijral, Peter Richtárik, and Nathan Srebro. Mini-batch primal and dual methods for SVMs. In *30th International Conference on Machine Learning*, pages 537–552, 2013.

H. Tang, X. Lian, T. Zhang, and J. Liu. DoubleSqueeze: Parallel stochastic gradient descent with double-pass error-compensated compression. In *Proceedings of the 36th International Conference on Machine Learning*, pages 6155–6165, 2019.

John Wallis. A treatise of algebra, both historical and practical. *Philosophical Transactions of the Royal Society of London*, 15(173):1095–1106, 1685. doi: 10.1098/rstl.1685.0053. URL `https://royalsocietypublishing.org/doi/abs/10.1098/rstl.1685.0053`.

Shusen Wang, Fred Roosta, Peng Xu, and Michael W Mahoney. Giant: Globally improved approximate newton method for distributed optimization. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, pages 2332–2342. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper/2018/file/dabd8d2ce74e782c65a973ef76fd540b-Paper.pdf`.

W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, and H. Li. Terngrad: Ternary gradients to reduce communication in distributed deep learning. *Advances in Neural Information Processing Systems*, pages 1509–1519, 2017.

Lin Xiao and Tong Zhang. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4):2057–2075, 2014.

Jiaqi Zhang, Keyou You, and Tamer Başar. Distributed adaptive Newton methods with globally superlinear convergence. *arXiv preprint arXiv:2002.07378*, 2020.

Yuchen Zhang and Lin Xiao. DiSCO: Distributed optimization for self-concordant empirical loss. In *Proceedings of the 32nd International Conference on Machine Learning, PMLR*, volume 37, pages 362–370, 2015.

Peilin Zhao and Tong Zhang. Stochastic optimization with importance sampling. *The 32nd International Conference on Machine Learning*, 37:1–9, 2015.