99% of Worker-Master Communication in Distributed Optimization Is Not Needed

Konstantin Mishchenko
KAUST
Thuwal, Saudi Arabia

Filip Hanzely
KAUST
Thuwal, Saudi Arabia

Peter Richtárik
KAUST
Thuwal, Saudi Arabia

Abstract

In this paper we discuss sparsification of worker-to-server communication in large distributed systems. We improve upon algorithms that fit the following template: a local gradient estimate is computed independently by each worker, then communicated to a master, which subsequently performs averaging. The average is broadcast back to the workers, which use it to perform a gradient-type step to update the local version of the model. We observe that the above template is fundamentally inefficient in that too much data is unnecessarily communicated from the workers to the server, which slows down the overall system. We propose a fix based on a new update-sparsification method we develop in this work, which we suggest being used on top of existing methods. Namely, we develop a new variant of parallel block coordinate descent based on independent sparsification of the local gradient estimates before communication. We demonstrate that with only m/n blocks sent by each of n workers, where m is the total number of parameter blocks, the theoretical iteration complexity of the underlying distributed methods is essentially unaffected. As an illustration, this means that when $n = 100$ parallel workers are used, the communication of 99% blocks is redundant, and hence a waste of time. Our theoretical claims are supported through extensive numerical experiments which demonstrate an almost perfect match with our theory on a number of synthetic and real datasets.

1 Introduction

In this work we are concerned with parallel/distributed algorithms for solving finite sum minimization problems

$$\min_{x\in\mathbb{R}^d} f(x) \triangleq \frac{1}{n}\sum_{i=1}^{n} f_i(x), \tag{1}$$

where each f_i is convex and smooth. In particular, we are interested in methods which employ n parallel units/workers/nodes/processors, each of which has access to a single function f_i and its gradients (or unbiased estimators thereof). Let x^* be an optimal solution of (1). In many practical scenarios, f_i is often of the form

$$f_i(x) = \mathbb{E}_\xi \phi_i(x;\xi), \tag{2}$$

where the expectation is with respect to a distribution of training examples stored locally at machine i. More typically, however, each machine contains a very large but finite number of examples (for simplicity, say there are l examples on each machine), and f_i is of the form

$$f_i(x) = \frac{1}{l}\sum_{j=1}^{l} f_{ij}(x). \tag{3}$$

In the rest of this section, we provide some basic motivation and intuitions in support of our approach. To this purpose, assume, for simplicity of exposition, that f_i is of the finite-sum form (3). In typical modern machine learning workloads, the number of machines n is much smaller than the number of data points on each machine l. In a large scale regime (i.e., when the model size d, the number of data points nl, or both are large), problem (1) needs to be solved by a combination of efficient methods and modern hardware. In recent years there has been a lot of progress in designing new algorithms for solving this problem using techniques such as stochastic approximation [35], variance reduction [36, 18, 10], coordinate descent [28, 33, 43] and acceleration [26], resulting in excellent theoretical and practical performance.

Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI), PMLR volume 124, 2020.

The computational power of the hardware is increasing as well. In recent years, a very significant amount of such increase is due to parallelism. Since many methods, such as minibatch Stochastic Gradient Descent (SGD), are embarrassingly parallel, it is very simple to use them in big data applications. However, it has been observed in practice that adding more resources beyond a certain limit does not improve iteration complexity significantly. Moreover, having more parallel units makes their synchronization harder due to the so-called communication bottleneck. Minibatch versions of most variance reduced methods[1] such as SAGA [10] or SVRG [18] scale even worse in parallel setting – they do not guarantee, in the worst case, any speedup from using more than one function at a time. Unfortunately, numerical experiments show that this is not a proof flaw, but rather a real property of these methods [13]. A similar observation was made for SVRG by [44], where it was shown that only a small number of partial derivatives are needed at each iteration.

Since there are too many possible situations, we choose to focus on black-box optimization, although we admit that much can be achieved by assuming the sparsity structure. In fact, for any method there exists a toy situation where the method would scale perfectly – one simply needs to assume that each function f_i depends on its own subset of coordinates and minimize each f_i independently. This can be generalized assuming sparsity patterns [19, 20] to get almost linear scaling if any coordinate appears in a small number of functions. Our interest, however, is in explaining situations as in [13] where the models almost do not scale.

In this paper, we demonstrate that a simple trick of *independent* block sampling can remedy the problem of scaling, to a substantial but limited extent. To illustrate one of the key insights of our paper on a simple example, in what follows consider a thought experiment in which GD is a baseline method we would want to improve on.

From gradient descent to block coordinate descent and back

A simple benchmark in the distributed setting is a parallel implementation of gradient descent (GD). GD arises as a special case of the more general class of block coordinate descent methods (BCD) [34]. The conventional way to run BCD for problem (1) is to update a single or several blocks[2] of x, chosen at random, on all n machines [34, 11], followed by an update aggregation step. Such updates on each worker typically involve a gradient step on a subspace corresponding to the selected blocks. Importantly, and this is a key structural property of BCD methods, *the same set of blocks is updated on each machine*. If communication is expensive, it often makes sense to do more work on each machine, which in the context of BCD means updating more blocks. A particular special case is to update *all* blocks, which leads to parallel implementation of GD for problem (1), as mentioned above. Moreover, it is known that the theoretical iteration complexity of BCD improves as the number of blocks updated increases [34, 29, 30]. For these and similar reasons, GD (or one of its variants, such as GD with momentum), is often the preferable method to BCD (in terms of iteration complexity). Having said that, we did not choose to describe BCD only to discard it at this point; we shall soon return to it, albeit with a twist.

From gradient descent to independent block coordinate descent

Because of what we just said, iteration complexity of GD will not improve by any variant running BCD; it can only get worse. Despite this, *we propose to run a variant of BCD, modified to allow each worker to sample an independent subset of blocks* instead. This variant of BCD for (1) was not considered before. As we shall show, our *independent sampling* approach leads to a better-behaved aggregated gradient estimator when compared to that of BCD, which in turn leads to better overall iteration complexity. We call our method *independent block coordinate descent (IBCD)*. We provide a unified analysis of our method, allowing for a random subset of τm out of a total of m blocks to be sampled on each machine, independently from other machines. GD arises as a special case of this method by setting $\tau = 1$. However, as we show (see Corollary 1), *the same iteration complexity guarantee can be obtained by choosing τ as low as $\tau = 1/n$.* The immediate consequence of this result is that *it is suboptimal to run GD in terms of communication complexity.* Indeed, GD needs to communicate all m blocks per machine, while IBCD achieves the same rate with m/n blocks per machine only. Coming back to the abstract, consider an example with $n = 100$ machines. In this case, when compared to GD, IBCD only communicates 1% of the data. Because the iteration complexities of the two methods are the same, and if communication cost is dominant, this means that the problem can be solved in just 1% of the time. In contrast, and when compared to the potential of IBCD, parallel implementation of GD inevitably wastes 99% of the time.

The intuition behind our approach lies in the law of

[1] We shall mention that there are already a few variance reduced methods that scale, up to some level, linearly in a parallel setup: Quartz for sparse data [31], Katyusha [3], or SAGA/SVRG/SARAH with importance sampling for non-convex problems [17].

[2] Assume the entries of x are partitioned into several non-overlapping blocks.

large numbers. By averaging independent noise, we reduce the total variance of the resulting estimator by the factor of n. If, however, the noise is already tiny, as, in non-accelerated variance reduced methods, there is no improvement. On the other hand, (uniform) block coordinate descent (CD) has variance proportional to $1/\tau$ [41], where $\tau < 1$ is the ratio of used blocks. Therefore, after the averaging step the variance is $1/\tau n$, which illustrates why setting any $\tau > 1/n$ should not yield a significant speedup when compared to the choice $\tau = 1/n$. It also indicates that it should be possible to throw away a $(1 - 1/n)$ fraction of blocks while keeping the same convergence rate.

Beyond gradient descent and further contributions

The goal of the above discussion was to introduce one of the ideas of this paper in a gentle way. However, our independent sampling idea has immense consequences beyond the realm of GD, as we show in the rest of the paper. Let us summarize the contributions here:

• We show that the independent sampling idea can be coupled with *variance reduction*/SAGA (see Sec 4), SGD for problem (1)+(2) (see Sec D), *acceleration* (under mild assumption on stochastic gradients; see Sec E) and *regularization*/SEGA (see Sec 5). We call the new methods ISAGA, ISGD, IASGD and ISEGA, respectively. We also develop ISGD variant for *asynchronous* distributed optimization – IASGD (Sec F).

• We present two versions of SAGA coupled with IBCD. The first one is for a distributed setting, where each machine owns a subset of data and runs a SAGA iteration with block sampling locally, followed by aggregation. The second version is in a shared data setting, where each machine has access to all functions. This allows for *linear convergence even if* $\nabla f_i(x^*) \neq 0$.

• We show that when combined with IBCD, the SEGA trick [15] leads to a method that enjoys a *linear rate for problems where* $\nabla f_i(x^*) \neq 0$ and allows for more general objectives which may include a non-separable non-smooth regularizer.

A comprehensive summary of all proposed algorithms is given in Table 1.

Identical sparsification and memory If the sampling of coordinate blocks was identical (instead of independent) across the workers, the aggregated update would be sparse and consequently, server-to-worker communication would be reduced. However, such a naive strategy suffers worse convergence guarantees due to the blow-up of the gradient estimator variance. Our method can be thus seen as a strategy to reduce the variance for the price of full worker-to-server communication. An alternative approach to decrease the variance of identical sparsification is to memorize the sparsified information and reuse it later on [2, 38]. We do not explore this direction in our paper.

2 Practical Implications and Limitations

In this section, we outline some further limitations and practical implications of our framework. **Main limitation**

The main limitation of this work is that independent sampling does not generally result in a sparse aggregated update. Indeed, since each machine might sample a different subset of blocks, all these updates add up to a dense one, and this problem gets worse as n increases, other things equal. For instance, if every parallel unit updates a single unique block[3], the total number of updated blocks is equal n. In contrast, standard BCD, one that samples the *same* block on each worker, would update a single block only. For simple linear problems, such as logistic regression, sparse updates allow for a fast implementation of BCD via memorization of the residuals. However, this limitation is not crucial in common settings where broadcast is much faster than reduce.

Practical implications

The main body of this work focuses on theoretical analysis and on verifying our claims via experiments. However, there are several straightforward and important applications of our technique.

Distributed synchronous learning. A common way to run a distributed optimization method is to perform a local update, communicate the result to a parameter server using a 'reduce' operation, and inform all workers using 'broadcast'. Typically, if the number of workers is significantly large, the bottleneck of such a system is communication. In particular, the 'reduce' operation takes much more time than 'broadcast' as it requires to add up different vectors computed locally, while 'broadcast' informs the workers about *the same* data (see [23] for a numerical validation that 'broadcast' is 10-20 times faster across a wide range of dimensions). Nevertheless, if every worker can instead send to the parameter server only $\tau = 1/n$ fraction of the d-dimensional update, essentially the server node will receive just one full d-dimensional vector, and thus our approach can compete against methods like QSGD [1], signSGD [4], TernGrad [42], DGC [21] or ATOMO [40]. In fact, our approach may completely remove the communication bottleneck.

Distributed asynchronous learning. The main difference with the synchronous case is that only one-to-one communications will be used instead of

[3] Assume x is partitioned into several "blocks" of variables.

#	Name	Origin	$\nabla f_i(x^*) \neq 0$	Lin. rate	Stochastic gradient	Note
1	IBCD	I+ CD [28]	✗	✓	✗	Simplest
3	ISEGA	I + SEGA [15]	✓	✓	✗	Allows prox
4	IBGD	I+ GD	✗	✓	✗	Bernoulli
2	ISAGA	I+ SAGA [10]	✓	✓	✓	Shared mem.
5	ISAGA	I+ SAGA [10]	✗	✓	✓	
6	ISGD	I + SGD [35]	✓	✗	✓	+ Non-convex
7	IASGD	I + ASGD [39]	✓	✗	✓	Accel.
8	IASGD	I + ASGD [32]	✓	✗	✓	Asynch.

Table 1: Summary of all algorithms proposed in the paper.

highly efficient 'reduce' and 'broadcast'. Clearly, the communication to the server will be much faster with $\tau = 1/n$, so the main question is how to make the communication back fast as well. Hopefully, the parameter server can copy the current vector and send it using non-blocking communication, such as *isend()* in MPI4PY [8]. Then, the communication back will not prevent the server from receiving the new updates. We combine the IBCD approach with asynchronous updates, which leads to a new method: IASGD (Algorithm 8).

Distributed sparse learning. Large datasets, such as binary classification data from LibSVM, often have sparse gradients. In this case, the 'reduce' operation is not efficient and one needs to communicate data by sending positions of nonzeros and their values. Moreover, as we prove later, one can use independent sampling with ℓ_1-penalty, which makes the problem solution sparse. In that case, only communication from a worker to the parameter server is slow, so both synchronous and asynchronous methods gain in performance.

Methods with local subproblems. One can also try to extend our analysis to methods with exact block-coordinate minimization or primal-dual and proximal methods such as Point-SAGA [9], PDHG [5], DANE [37], etc. By restricting ourselves to a subset of coordinates, we may obtain a subproblem that is easier to solve by orders of magnitude.

Block-separable problems within machines. Given that the local problem on each machine is block coordinate-wise separable, partial derivative blocks can be evaluated $1/\tau$ times cheaper than the gradients. Thus, independent sampling improves scalability at no cost. Such problems can be obtained considering the dual problem, as is done in [22], for example.

For a comprehensive list of frequently used notation, see Table 2 in the supplementary material.

3 Independent Block Coordinate Descent

Technical assumptions We present the most common technical assumptions required in order to derive convergence rates.

Definition 1. *Function F is L smooth if for all $x, y \in \mathbb{R}^d$ we have:*

$$F(x) \leq F(y) + \langle \nabla F(y), x - y \rangle + \frac{L}{2}\|x - y\|_2^2. \quad (4)$$

Similarly, F is μ strongly convex if for all $x, y \in \mathbb{R}^d$:

$$F(x) \geq F(y) + \langle \nabla F(y), x - y \rangle + \frac{\mu}{2}\|x - y\|_2^2. \quad (5)$$

In most results we present, functions f_i are required to be smooth and convex, and f strongly convex.

Assumption 1. *For every i, function f_i is convex, L smooth and function f is μ strongly convex.*

Block structure of $\mathbb{R}^d$. Let $\mathbb{R}^d$ be partitioned into m blocks $u_1, \ldots, u_m$ of arbitrary sizes, so that the parameter space is $\mathbb{R}^{|u_1|} \times \cdots \mathbb{R}^{|u_m|}$. For any vector $x \in \mathbb{R}^d$ and a set of blocks U we denote by x_U the vector that has the same coordinate as x in the set of blocks U and zeros elsewhere.

3.1 IBCD

In order to provide a quick taste of our results, we first present the IBCD method described in the introduction and formalized as Algorithm 1. Note that this is the simplest algorithm analyzed under a rather strong assumptions where it is the easiest to demonstrate our results; we present more practical methods later on.

A key parameter of the method is $1/m \leq \tau \leq 1$ (chosen so that τm is an integer), representing a fraction of blocks to be sampled by each worker. At iteration t, each machine independently samples a subset of τm blocks $U_i^t \subseteq \{u_1, \ldots, u_m\}$, uniformly at random. The ith worker then performs a subspace gradient step of the form $x_i^{t+1} = x^t - \gamma(\nabla f_i(x^t))_{U_i^t}$, where $\gamma > 0$ is a

Algorithm 1 Independent Block Coordinate Descent (IBCD)

Input: $x^0 \in \mathbb{R}^d$, partition of $\mathbb{R}^d$ into m blocks $u_1, \dots, u_m$, ratio of blocks to be sampled τ, stepsize γ, # of parallel units n
for $t = 0, 1, \dots$ **do**
  for $i = 1, \dots, n$ in parallel **do**
    Sample independently and uniformly a subset of τm blocks $U_i^t \subseteq \{u_1, \dots, u_m\}$
    $x_i^{t+1} = x^t - \gamma(\nabla f_i(x^t))_{U_i^t}$
  end for
  $x^{t+1} = \frac{1}{n}\sum_{i=1}^n x_i^{t+1}$
end for

stepsize. Note that only coordinates of x^t belonging to U_i^t get updated. This is then followed by aggregating all n gradient updates: $x^{t+1} = \frac{1}{n}\sum_i x_i^{t+1}$.

Convergence of IBCD

Theorem 1 provides a convergence rate for Algorithm 1. Admittedly, the assumptions of Theorem 1 are somewhat restrictive; in particular, we require $\nabla f_i(x^*) = 0$ for all i. However, this is necessary. Indeed, in general one can not expect to have $\sum_{i=1}^n (\nabla f_i(x^*))_{U_i} = 0$ (which would be required for the method to converge to x^*) for independently sampled sets of blocks U_i unless $\nabla f_i(x^*) = 0$ for all i. As mentioned, the issue is resolved in Sec 5 using the SEGA trick [15].

Theorem 1. *Suppose that Assumptions 1 holds and $\nabla f_i(x^*) = 0$ for all i.*[4] *For Algorithm 1 with $\gamma = \frac{n}{\tau n + 2(1-\tau)} \frac{1}{2L}$ we have*

$$\mathbb{E}\left[\|x^t - x^*\|_2^2\right] \leq \left(1 - \frac{\mu}{2L}\frac{\tau n}{\tau n + 2(1-\tau)}\right)^t \|x^0 - x^*\|_2^2.$$

As a consequence of Theorem 1, we can choose τ as small as $1/n$ and get, up to a constant factor, the same convergence rate as gradient descent, as described next.

Corollary 1. *If $\tau = 1/n$, the iteration complexity*[5] *of Algorithm 1 is $\mathcal{O}(L/\mu \log 1/\epsilon)$.*

[4] The requirement of $\nabla f_i(x^*) = 0$ is only necessary for the plainest results; which we present to better explain the main idea of the paper; and there are ways to go around it. In particular, in Sec 4 we show that it can be dropped once the memory is shared among the machines. Further, in Sec 5 we show that $\nabla f_i(x^*) = 0$ can be dropped even in the fully distributed setup using the SEGA trick. Lastly, $\nabla f_i(x^*) = 0$ is naturally satisfied in many applications. For example, in the least squares setting $\min \|Ax - b\|_2^2$, it is equivalent to existence of x^* such that $Ax^* = b$. On the other hand, current state-of-the-art deep learning models are often overparameterized so that they allow zero training loss, which is again equivalent to $\nabla f_i(x^*) = 0$ for all i (however, such problems are typically non-convex).

[5] Number of iterations to reach ϵ accurate solution.

Optimal block sizes. If we naively use coordinates as blocks, i.e. all blocks have size equal 1, the update will be very sparse and the efficient way to send it is by providing positions of nonzeros and the corresponding values. If, however, we partition $\mathbb{R}^d$ into blocks of size approximately equal d/n, then on average, only one block will be updated by each worker. This means that it will be just enough for each worker to communicate the block number and its entries, which is twice fewer data sent than when using coordinates as blocks.

4 Variance Reduction

As the first extension of IBCD, we inject independent coordinate sampling into SAGA[6] [10], resulting in a new method which we call ISAGA. We consider two different settings for ISAGA. The first one is standard distributed setup (1), where each f_i is of the fine-sum form (3). The idea is to run SAGA with independent coordinate sampling locally on each worker, followed by aggregating the updates. However, as for IBCD, we require $\nabla f_i(x^*) = 0$ for all i. The second setting is a *shared data/memory* setup; i.e., we assume that all workers have access to all functions from the finite sum. Due to space limitations, we present distributed ISAGA in Sec C of the supplementary.

4.1 Shared data ISAGA

We now present a different setup for ISAGA in which the requirement $\nabla f_i(x^*) = 0$ is not needed. Instead of (1), we rather solve the problem

$$\min_{x \in \mathbb{R}^d} f(x) \triangleq \frac{1}{N}\sum_{j=1}^N \psi_j(x) \tag{6}$$

with n workers all of which have access to all data describing f. Therefore, all workers can evaluate $\nabla \psi_j(x)$ for any $1 \leq j \leq N$. Similarly to plain SAGA, we remember the freshest gradient information in vectors α_j, and update them as

$$\alpha_{j_i^t}^{t+1} = \alpha_{j_i^t}^t + (\nabla \psi_{j_i^t}(x^t) - \alpha_{j_i^t}^t)_{U_i^t}, \quad \alpha_{j'}^{t+1} = \alpha_{j'}^t, \tag{7}$$

where j_i^t is the index sampled at iteration t by machine i, and j' refers to all indices that were not sampled at iteration t by any machine. The iterate updates within each machine are taken only on a sampled set of coordinates, i.e., $x_i^{t+1} = x^t - \gamma(\nabla \psi_{j_i^t}(x^t) - \alpha_{j_i^t}^t + \overline{\alpha}^t)_{U_i^t}$. where $\overline{\alpha}^t$ stands for the average of all α, and thus it is a delayed estimate of $\nabla f(x^t)$. Lastly, we set the next iterate as the average of proposed iterates by each machine

[6] Independent coordinate sampling is not limited to SAGA and can be similarly applied to other variance reduction techniques.

$x^{t+1} = \frac{1}{n}\sum_{i=1}^n x_i^{t+1}$. The formal statement of the algorithm is given in the supplementary as Algorithm 2.

Algorithm 2 ISAGA with shared data

Input: $x^0 \in \mathbb{R}^d$, $\alpha_1^0, \ldots, \alpha_N^0$ partition of $\mathbb{R}^d$ into m blocks $u_1, \ldots, u_m$, ratio of blocks to be sampled τ, stepsize γ, # parallel units n
Set $\overline{\alpha}^0 \triangleq \frac{1}{N}\sum_{j=1}^N \alpha_j^0$
for $t = 0, 1, \ldots$ **do**
  Sample uniformly set of indices $\{j_1^t, \ldots, j_n^t\} \subseteq \{1, \ldots, N\}$ without replacement
  for $i = 1, \ldots, n$ in parallel **do**
    Sample independently and uniformly a subset of τm blocks U_t^i
    $x_i^{t+1} = x^t - \gamma(\nabla\psi_{j_i^t}(x^t) - \alpha_{j_i^t}^t + \overline{\alpha}^t)_{U_i^t}$
    $(\alpha_{j_i^t}^{t+1})_{U_i^t} = \alpha_{j_i^t}^t + (\nabla\psi_{j_i^t}(x^t) - \alpha_{j_i^t}^t)_{U_i^t}$
  end for
  For $j \notin \{j_1^t, \ldots, j_n^t\}$ set $\alpha_j^{t+1} = \alpha_j^t$
  $x^{t+1} = \frac{1}{n}\sum_{i=1}^n x_i^{t+1}$
  $\overline{\alpha}^{t+1} = \frac{1}{n}\sum_{j=1}^N \alpha_j^{t+1}$
end for

Theorem 2. *Suppose that function f is μ strongly convex and each ψ_i is L smooth and convex. If $\gamma \leq \frac{1}{L\left(\frac{3}{n}+\tau\right)}$, then for iterates of Algorithm 2 we have*

$$\mathbb{E}\|x^t - x^*\|_2^2 \leq (1-\vartheta)^t \left(\|x^0 - x^*\|_2^2 + c\gamma^2\Psi^0\right),$$

where $\Psi^0 \triangleq \sum_j \|\alpha_j^0 - \nabla\psi_j(x^*)\|_2^2$, $\vartheta \triangleq \tau\min\left\{\gamma\mu, \frac{n}{N} - \frac{2}{nNc}\right\} \geq 0$ *and* $c \triangleq \frac{1}{n}(\frac{1}{\gamma L} - \frac{1}{n} - \tau) > 0$.

As in Sec C, the choice $\tau = 1/n$ yields a convergence rate which is, up to a constant factor, the same as the convergence rate of SAGA. Therefore, Algorithm 2 enjoys the desired parallel linear scaling without extra assumptions. Corollary 2 formalizes the claim.

Corollary 2. *Consider the setting from Theorem 2. Set $\tau = 1/n$ and $\gamma = n/5L$. Then $c = 3/n^2$, $\rho = \min\{\mu/5L, 1/3N\}$ and the complexity of Algorithm 2 is $\mathcal{O}\left(\max\{L/\mu, N\}\log 1/\varepsilon\right)$.*

5 Beyond $\nabla f_i(x^*) = 0$ without Shared Data and Regularization

For this section only, let us consider a regularized objective of the form

$$\min_{x\in\mathbb{R}^d} f(x) \triangleq \frac{1}{n}\sum_{i=1}^n f_i(x) + R(x), \qquad (8)$$

where R is a closed convex regularizer such that its proximal operator is computable: $\text{prox}_{\gamma R}(x) \triangleq \text{argmin}_y \left\{R(y) + \frac{1}{2\gamma}\|y - x\|_2^2\right\}$. In this section we propose ISEGA: an independent sampling variant of SEGA [15]. We do this in order to both i) avoid assuming $\nabla f_i(x^*) = 0$ (while keeping linear convergence) and ii) allow for R. Original SEGA learns gradients $\nabla f(x^t)$ from sketched gradient information via the so called sketch-and-project process [12], constructing a vector sequence h^t. In ISEGA on each machine i we iteratively construct a sequence of vectors h_i^t which play the role of estimates of $\nabla f_i(x^t)$. This is done via the following rule:

$$h_i^{t+1} = h_i^t + (\nabla f_i(x^t) - h_i^t)_{U_i^t}. \qquad (9)$$

The key idea is again that these vectors are created from random blocks independently sampled on each machine. Next, using h^t, SEGA builds an unbiased gradient estimator g_i^t of $\nabla f_i(x^t)$ as follows:

$$g_i^t = h_i^t + \frac{1}{\tau}(\nabla f_i(x^t) - h_i^t)_{U_i^t}. \qquad (10)$$

Then, we average the vectors g_i^t and take a proximal step.

Unlike coordinate descent, SEGA (or ISEGA) is not limited to separable proximal operators since, as follows from our analysis , $h_i^t \to \nabla f_i(x^*)$. Therefore, ISEGA can be seen as a variance reduced version of IBCD for problems with non-separable regularizers.

In order to be consistent with the rest of the paper, we only develop a simple variant of ISEGA (Algorithm 3) in which we consider block coordinate sketches with uniform probabilities and non-weighted Euclidean metric (i.e. $B = I$ in notation of [15]). It is possible to develop the theory in full generality as in [15]. However, we do not do this for the sake of simplicity.

Algorithm 3 ISEGA

Input: $x^0 \in \mathbb{R}^d$, initial gradient estimates $h_1^0, \ldots, h_n^0 \in \mathbb{R}^d$, partition of $\mathbb{R}^d$ into m blocks $u_1, \ldots, i_m$, ratio of blocks to be sampled τ, stepsize γ, # parallel units n
for $t = 0, 1, \ldots$ **do**
  for $i = 1, \ldots, n$ in parallel **do**
    Sample independently and uniformly a subset of τm blocks U_i^t
    $g_i^t = h_i^t + \frac{1}{\tau}(\nabla f_i(x^t) - h_i^t)_{U_i^t}$
    $h_i^{t+1} = h_i^t + \tau(g_i^t - h_i^t)$
  end for
  $x^{t+1} = \text{prox}_{\gamma R}\left(x^t - \gamma\frac{1}{n}\sum_{i=1}^n g_i^t\right)$
end for

We next present the convergence rate of ISEGA (Algorithm 3).

Theorem 3. *Suppose Assumption 1 holds. Algorithm 3 with $\gamma = \min\{\frac{1}{4L\left(1+\frac{1}{n\tau}\right)}, \frac{1}{\frac{\mu}{\tau}+\frac{4L}{n\tau}}\}$ satisfies $\mathbb{E}[\|x^t - x^*\|_2^2] \leq (1-\gamma\mu)^t\Phi^0$, where $\Phi^0 = \|x^0 - x^*\|_2^2 + \frac{\gamma}{2L\tau n}\sum_{i=1}^n \|h^0 - \nabla f(x^*)\|_2^2$.*

Note that if the condition number of the problem is not too small so that $n = \mathcal{O}(L/\mu)$ (which is usually the case in practice), ISEGA scales linearly in the parallel setting. In particular, when doubling the number of workers, each worker can afford to evaluate only half of the block partial derivatives while keeping the same convergence speed. Moreover, setting $\tau = 1/n$, the rate corresponds, up to a constant factor, to the rate of gradient descent. Corollary 3 states the result.

Corollary 3. *Consider the setting from Theorem 3. Suppose that $L/\mu \geq n$ and choose $\tau = 1/n$. Then, complexity of Algorithm 3 is $\mathcal{O}(L/\mu \log 1/\epsilon)$.*

Remark 1. *Parallel implementation Algorithm 3 would be to always send $(\nabla f_i(x^k))_{U_i^t}$ to the server; which keeps updating vector h^t and takes the prox step.*

6 Experiments

In this section, we numerically verify our theoretical claims. Recall that there are various settings where it is possible to make practical experiments (see Sec 2), however, we do not restrain ourselves to any of them in order to deliver as clear a message as possible.

Due to space limitations, we only present a small fraction of the experiments here. A full and exhaustive comparison, together with the complete experiment setup description, is presented in Sec G of the appendix.

In the first experiment presented here, we compare SAGA against ISAGA in a shared data setup (Algorithm 2) for various values of n with $\tau = 1/n$ in order to demonstrate linear scaling. We consider logistic regression problem on LibSVM data [6]. The results (Figure 1) corroborate our theory: indeed, setting $n\tau = 1$ does not lead to a decrease in the convergence rate when compared to the original SAGA.

The next experiment (Figure 2) supports an analogous claim for ISEGA (Algorithm 3). We run the method for several (n, τ) pairs for which $n\tau = 1$; on logistic regression problems and LibSVM data. We also plot convergence of gradient descent with the analogous stepsize. As our theory predicts, all the methods exhibit almost the same convergence rate.[7] Note that for $n = 100$, Algorithm 3 throws away 99% of partial derivatives while keeping the same convergence speed as GD, which justifies the title of the paper.

[7] We have chosen the stepsize $\gamma = 1/2L$ for GD, as this is the baseline to Algorithm 3 with zero variance. One can in fact set $\gamma = 1/L$ for GD and get $2\times$ faster convergence. However, this is only a constant factor.

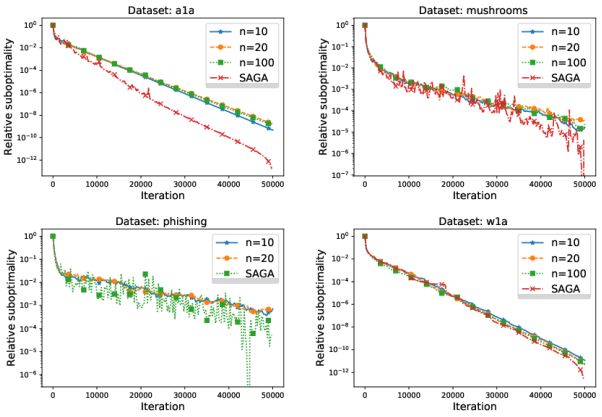

Figure 1: SAGA vs. Alg. 2 for various values of n (number of workers) and $\tau = n^{-1}$ on LibSVM datasets. Stepsize $\gamma = \frac{1}{L(3n^{-1}+\tau)}$ is chosen in each case.

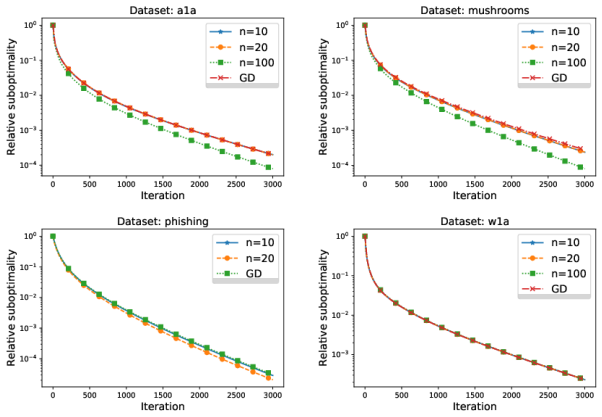

Figure 2: Comparison of Alg. 3 for various (n, τ) such that $n\tau = 1$ and GD on LibSVM datasets. Stepsize $1/(L\left(1 + \frac{1}{n\tau}\right))$ was chosen for Alg. 3 and $\frac{1}{2L}$ for GD.

7 Future Work

We sketch several possible extensions of this work.

• Combining the tricks from the paper. Distributed ISAGA requires $\nabla f_i(x) = 0$. We believe it would be possible to develop SEGA approach on top of it in order to drop the requirement. We also believe that one can accelerate the combination of SEGA and ISAGA.

• Convergence in the asynchronous setup. We have provided theoretical results for parallel algorithms in the synchronous setting and the asynchronous theory is a very natural future step. Moreover, as we mentioned before, it has very direct practical implications. We believe that it possible to extend the works [25, 24] to design a method with proximable regularizer (for ℓ_1 penalty) that would communicate little both sides.

• Importance sampling. Standard coordinate descent is able to exploit a complex smoothness structure of objective in order to sample coordinates non-uniformly [30, 7, 16]. It would be interesting to derive an importance sampling in our setting in order to converge even faster.

References

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

[2] D. Alistarh, T. Hoefler, M. Johansson, N. Konstantinov, S. Khirirat, and C. Renggli. The convergence of sparsified gradient methods. In *Advances in Neural Information Processing Systems*, pages 5973–5983, 2018.

[3] Z. Allen-Zhu. Katyusha: The first direct acceleration of stochastic gradient methods. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1200–1205. ACM, 2017.

[4] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar. SignSGD: Compressed optimisation for non-convex problems. *arXiv preprint arXiv:1802.04434*, 2018.

[5] A. Chambolle and T. Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40(1):120–145, 2011.

[6] C.-C. Chang and C.-J. Lin. LibSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[7] D. Csiba and P. Richtárik. Importance sampling for minibatches. *The Journal of Machine Learning Research*, 19(1):962–982, 2018.

[8] L. D. Dalcin, R. R. Paz, P. A. Kler, and A. Cosimo. Parallel distributed computing using Python. *Advances in Water Resources*, 34(9):1124–1139, 2011.

[9] A. Defazio. A simple practical accelerated method for finite sums. In *Advances in Neural Information Processing Systems*, pages 676–684, 2016.

[10] A. Defazio, F. Bach, and S. Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems*, pages 1646–1654, 2014.

[11] O. Fercoq and P. Richtárik. Accelerated, parallel, and proximal coordinate descent. *SIAM Journal on Optimization*, 25(4):1997–2023, 2015.

[12] R. M. Gower and P. Richtárik. Randomized iterative methods for linear systems. *SIAM Journal on Matrix Analysis and Applications*, 36(4):1660–1690, 2015.

[13] R. M. Gower, P. Richtárik, and F. Bach. Stochastic quasi-gradient methods: Variance reduction via Jacobian sketching. *arXiv preprint arXiv:1805.02632*, 2018.

[14] D. Grishchenko, F. Iutzeler, J. Malick, and M.-R. Amini. Asynchronous distributed learning with sparse communications and identification. *arXiv preprint arXiv:1812.03871*, 2018.

[15] F. Hanzely, K. Mishchenko, and P. Richtárik. SEGA: Variance reduction via gradient sketching. In *Advances in Neural Information Processing Systems*, pages 2083–2094, 2018.

[16] F. Hanzely and P. Richtárik. Accelerated coordinate descent with arbitrary sampling and best rates for minibatches. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 304–312, 2019.

[17] S. Horváth and P. Richtárik. Nonconvex variance reduced optimization with arbitrary sampling. In *International Conference on Machine Learning*, pages 2781–2789, 2019.

[18] R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*, pages 315–323, 2013.

[19] R. Leblond, F. Pedregosa, and S. Lacoste-Julien. Asaga: Asynchronous parallel saga. In *Artificial Intelligence and Statistics*, pages 46–54, 2017.

[20] R. Leblond, F. Pedregosa, and S. Lacoste-Julien. Improved asynchronous parallel optimization analysis for stochastic incremental methods. *The Journal of Machine Learning Research*, 19(1):3140–3207, 2018.

[21] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*, 2017.

[22] C. Ma, V. Smith, M. Jaggi, M. I. Jordan, P. Richtárik, and M. Takáč. Adding vs. averaging in distributed primal-dual optimization. In *The 32nd International Conference on Machine Learning*, pages 1973–1982, 2015.

[23] K. Mishchenko, E. Gorbunov, M. Takáč, and P. Richtárik. Distributed learning with compressed gradient differences. *arXiv preprint arXiv:1901.09269*, 2019.

[24] K. Mishchenko, F. Iutzeler, and J. Malick. A distributed flexible delay-tolerant proximal gradient algorithm. *SIAM Journal on Optimization*, 30(1):933–959, 2020.

[25] K. Mishchenko, F. Iutzeler, J. Malick, and M.-R. Amini. A delay-tolerant proximal-gradient algorithm for distributed learning. In *International Conference on Machine Learning*, pages 3584–3592, 2018.

[26] Y. Nesterov. A method for solving the convex programming problem with convergence rate O(1/kˆ2). In *Dokl. Akad. Nauk SSSR*, volume 269, pages 543–547, 1983.

[27] Y. Nesterov. *Introductory lectures on convex optimization: A basic course*. Kluwer Academic Publishers, 2004.

[28] Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

[29] Z. Qu and P. Richtárik. Coordinate descent with arbitrary sampling I: Algorithms and complexity. *Optimization Methods and Software*, 31(5):829–857, 2016.

[30] Z. Qu and P. Richtárik. Coordinate descent with arbitrary sampling II: Expected separable overapproximation. *Optimization Methods and Software*, 31(5):858–884, 2016.

[31] Z. Qu, P. Richtárik, and T. Zhang. Quartz: Randomized dual coordinate ascent with arbitrary sampling. In *Advances in Neural Information Processing Systems 28*, pages 865–873, 2015.

[32] B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 693–701. Curran Associates, Inc., 2011.

[33] P. Richtárik and M. Takáč. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, 144(1-2):1–38, 2014.

[34] P. Richtárik and M. Takáč. Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, 156(1-2):433–484, 2016.

[35] H. Robbins and S. Monro. A stochastic approximation method. In *Herbert Robbins Selected Papers*, pages 102–109. Springer, 1985.

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

[37] O. Shamir, N. Srebro, and T. Zhang. Communication-efficient distributed optimization using an approximate Newton-type method. In *International Conference on Machine Learning*, pages 1000–1008, 2014.

[38] S. U. Stich, J.-B. Cordonnier, and M. Jaggi. Sparsified sgd with memory. In *Advances in Neural Information Processing Systems*, pages 4447–4458, 2018.

[39] S. Vaswani, F. Bach, and M. Schmidt. Fast and faster convergence of SGD for over-parameterized models and an accelerated perceptron. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1195–1204, 2019.

[40] H. Wang, S. Sievert, S. Liu, Z. Charles, D. Papailiopoulos, and S. Wright. ATOMO: Communication-efficient learning via atomic sparsification. In *Advances in Neural Information Processing Systems*, pages 9872–9883, 2018.

[41] J. Wangni, J. Wang, J. Liu, and T. Zhang. Gradient sparsification for communication-efficient distributed optimization. In *Advances in Neural Information Processing Systems*, pages 1306–1316, 2018.

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

[43] S. J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

[44] T. Zhao, M. Yu, Y. Wang, R. Arora, and H. Liu. Accelerated mini-batch randomized block coordinate descent method. In *Advances in neural information processing systems*, pages 3329–3337, 2014.