Randomized Block-Diagonal Preconditioning for Parallel Learning

Celestine Mendler-Dünner [1] **Aurelien Lucchi** [2]

Abstract

We study preconditioned gradient-based optimization methods where the preconditioning matrix has block-diagonal form. Such a structural constraint comes with the advantage that the update computation can be parallelized across multiple independent tasks. Our main contribution is to demonstrate that the convergence of these methods can significantly be improved by a randomization technique which corresponds to repartitioning coordinates across tasks during the optimization procedure. We provide a theoretical analysis that accurately characterizes the expected convergence gains of repartitioning and validate our findings empirically on various traditional machine learning tasks. From an implementation perspective, block-separable models are well suited for parallelization and, when shared memory is available, randomization can be implemented on top of existing methods very efficiently to improve convergence.

1. Introduction

We focus on the task of parallel learning where we want to solve the convex and smooth optimization problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) \tag{1}$$

on a multi-core machine with shared memory. In this context we study iterative optimization methods where the repeated computation of the incremental update

$$\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t + \Delta\mathbf{x} \tag{2}$$

is parallelized across cores. Such methods traditionally build on one of the following three approaches: i) they implement a mini-batch algorithm (Dekel et al., 2012), where a finite sample approximation to f is used to compute the update $\Delta\mathbf{x}$, ii) they implement asynchronous updates (Niu et al., 2011; Liu et al., 2015), where stochastic updates are interleaved, or iii) they compute block updates (Richtárik & Takáč, 2016), where multiple coordinates of $\mathbf{x}$ are updated independently and in parallel. The primary goal of all these methods is to introduce parallel computations into an otherwise stochastic algorithm in order to better utilize the number of available cores.

Ioannou et al. (2019) argue that these methods are often not able to utilize the full potential of parallel systems because they make simplified modeling assumptions of the underlying hardware: They treat a multi-core machine as a uniform collection of cores whereas in fact it is a more elaborate system with complex data access patterns and cache structures. As a consequence, memory contention issues and false sharing can significantly impede their performance.

To resolve this the authors have proposed a novel approach to *parallel* learning that relies on block-separable models, so far solely used for distributed learning. Such models have the advantage that, in addition to computational parallelism, they implement a stricter separability between computational tasks which enables more efficient implementations. This potentially comes at the cost of slower convergence.

Interestingly, this new application area of block-separable models in a single machine setting with shared memory opens the door to previously unstudied algorithmic optimization techniques. Namely, we can relax strong communication constraints, as long as we do not compromise the desired separability between computational tasks.

One such algorithmic technique that preserves separability and can help convergence is *repartitioning*. It refers to randomly assigning coordinates to tasks for each update step. In a distributed setting repartitioning would involve expensive communication of large amounts of data across the network and has thus not been considered as an option. But in a single machine setting we can reassign coordinates to cores without incurring significant overheads. This has been verified empirically by Ioannou et al. (2019) who showed that for the specific example of training a logistic regression classifier using the CoCoA method (Smith et al., 2018) the convergence gains of repartitioning can significantly prevail the overheads of reassign coordinates to cores in a shared-memory setting.

[1]University of California, Berkeley [2]ETH Zürich. Correspondence to: Celestine Mendler-Dünner <mendler@berkeley.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

In this work we follow up on this interesting finding and our main contribution is to provide the first theoretical study of repartitioning. In particular, we frame repartitioning as a radomization step applied to a block-diagonal preconditioning matrix. This allows us to quantify the gain of repartitioning over static partitioning in a general setting, which covers a broad class of existing distributed methods, including the CoCoA method. We further validate our theoretical findings empirically for both ridge regression and logistic regression on a variety of datasets with different sizes and dimensions. Both our theoretical and empirical results indicate that repartitioning can significantly improve the sample efficiency of a broad class of distributed algorithms and thereby turn them into interesting new candidates for parallel learning.

2. Background

We begin by providing some background on distributed methods. This helps us set up a unified framework for analyzing repartitioning in later sections.

2.1. Distributed Optimization

Distributed optimization methods are designed for the scenario where the training data is too large to fit into the memory of a single machine and thus needs to be stored in a distributed fashion across multiple nodes in a cluster. The main objective when designing a distributed algorithm is to define an optimization procedure such that each node can compute its part of the update (2) independently. In addition, this computation should only require access to local data and rely on minimal interaction with other workers.

There are different approaches to achieve this computational separability. They all rely on a second-order approximation to the objective f around the current iterate $\mathbf{x}_t$:

$$f(\mathbf{x}_t + \Delta\mathbf{x}) \approx \tilde{f}_{\mathbf{x}_t}(\Delta\mathbf{x}; Q_t) \tag{3}$$
$$:= f(\mathbf{x}_t) + \nabla f(\mathbf{x}_t)^\top \Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^\top Q_t \Delta\mathbf{x}.$$

Note that $Q_t \in \mathbb{R}^{n\times n}$ can be a function of the iterate $\mathbf{x}_t$. In a single machine case the optimal choice for Q_t would be the Hessian matrix $\nabla^2 f(\mathbf{x}_t)$. But in a distributed setting we can not, or do not want to compute and store the full Hessian matrix across the entire dataset.

One approach to nevertheless benefit from second-order information is to locally use a finite sample approximation to $\nabla^2 f(\mathbf{x}_t)$ for computing $\Delta\mathbf{x}^k$ on each machine $k \in [K]$, before aggregating these updates to get a global update $\Delta\mathbf{x}$. This strategy has been exploited in methods such as DANE (Shamir et al., 2014), GIANT (Wang et al., 2017), AIDE (Reddi et al., 2016) and DISCO (Zhang & Lin, 2015). The convergence of these methods typically relies on concentration results. Hence, they require the data to be distributed uniformly across the machines, but are otherwise indifferent to the specific partitioning of the data.

For studying repartitioning we focus on an orthogonal approach, where the computation of the individual coordinates of $\Delta\mathbf{x}$ is distributed across nodes. This includes methods such as CoCoA (Smith et al., 2018), ADN (Dünner et al., 2018) and other distributed block coordinate descent methods such as (Lee & Chang, 2017; Hsieh et al., 2016; Mahajan et al., 2017; Lee & Chang, 2017). All these methods construct a separable auxiliary model of the objective function by enforcing a block-diagonal structure on Q_t in (3). As illustrated in Figure 1, this renders the computation of the individual coordinate blocks of $\Delta\mathbf{x}$ independent.

The partitioning of the coordinates across nodes determines which elements of Q_t are being ignored. While not all elements of Q_t might be equally important, each partitioning inevitably ignores a large subset of elements which can hurt convergence. Ideally, we would like to maintain as much information about Q_t as possible. Therefore, repartitioning offers an interesting alternative. It considers different elements of Q_t for each update step and over the course of the algorithm it gets information from all elements of Q_t with non-zero probability.

To gain intuition how repartitioning helps convergence, let us investigate the specific structure of the matrix Q_t at the example of generalized linear models (GLMs).

2.1.1. GLM Training

One attract of GLMs is the simple linear dependency on the data imposed by the model. This makes them particularly appealing in distributed settings where tasks can be separated across data partitions. It most likely also explains why so many distributed methods found in the literature have been specifically designed for GLMs.

For GLMs, the objective f depends linearly on the data matrix $A \in \mathbb{R}^{m\times n}$:

$$f(\mathbf{x}) = \ell(A\mathbf{x}), \tag{4}$$

where ℓ in general denotes the loss function. Let $\mathbf{v} := A\mathbf{x}$ be the information that is periodically shared across nodes, then the second-order model $\tilde{f}_{\mathbf{x}_t}(\Delta\mathbf{x}; Q_t)$ can be written as

$$\ell(\mathbf{v}_t) + \nabla\ell(\mathbf{v}_t)^\top A\Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^\top A^\top \hat{Q}_t A\Delta\mathbf{x}. \tag{5}$$

In this case $Q_t = A^\top \hat{Q}_t A$ which makes the dependence of $\tilde{f}_{\mathbf{x}_t}$ on the data more explicit. It is not hard to see that separability of (5) across coordinate blocks and corresponding columns of A follows by making Q_t block-diagonal and setting elements outside the diagonal blocks to zero. This is particularly easy to realize if $\hat{Q}_t$ has diagonal form. This

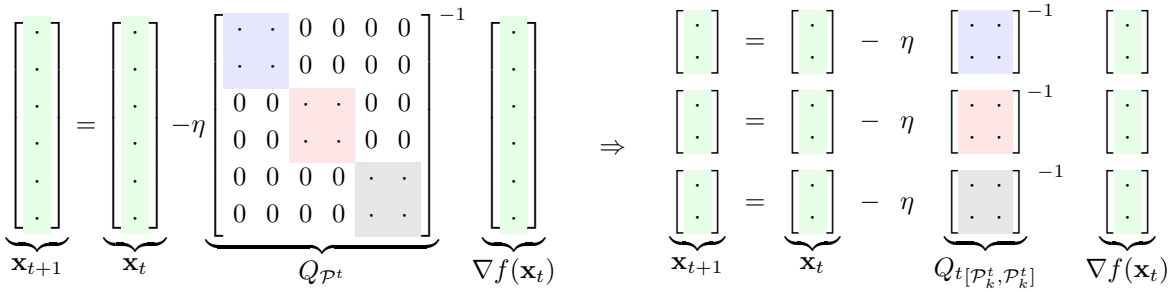


Figure 1. Parallelism in the update computation induced by block-diagonal preconditioning as described in Algorithm 1.

observation has been the basis for many distributed methods such as CoCoA (Smith et al., 2018; Jaggi et al., 2014), ADN (Dünner et al., 2018) and other block-separable methods such as (Lee & Chang, 2017). In CoCoA, $\hat{Q}_t$ is set to $\gamma_\ell I$ – where γ_ℓ denotes the smoothness parameter of ℓ – thus forming an over-approximation to $\nabla^2\ell$. In (Dünner et al., 2018) and (Lee & Chang, 2017), it was observed that for popular loss functions used in machine learning, $\nabla^2\ell(\mathbf{x}_t)$ is a diagonal matrix. Hence, they keep $\hat{Q}_t = \nabla^2\ell(\mathbf{x}_t)$ and directly enforce the block-diagonal structure on $A^\top\nabla^2\ell(\mathbf{x}_t)A$ to preserve additional local second-order information. We note that methods of the latter form are augmented by a line-search strategy or a trust-region like approach (Nesterov & Polyak, 2006) to guarantee sufficient function decrease and ensure convergence. We will come back to this condition in Section 4.3.

2.1.2. Static Partitioning

All the distributed methods we found in the literature assume a static partitioning of data across nodes. In that way, expensive communication of data across the network can be avoided. When distributing the computation of $\Delta\mathbf{x}$ coordinate-wise across nodes, this implies a static allocation of data columns to nodes (hence coordinates to blocks) throughout the optimization.

In this work, motivated by the recent trend in parallel learning, we take a different approach. We study the setting where one can randomly reassign coordinates to blocks for each repeated computation of $\Delta\mathbf{x}$. To the best of our knowledge, a formal study of such a repartitioning approach in the context of block-separable methods does not yet exist in the literature. This is likely due to the fact that block-diagonal approximations to Q_t have only been studied in the context of distributed learning where data repartitioning seems unreasonable. In addition, the theoretical analysis of existing methods can not readily be extended to explain the effect of repartitioning because they look at the function decrease in each individual iteration in isolation. Therefore, we will resort to analysis tools from the literature on preconditioned gradient descent methods.

2.2. Preconditioned Gradient Methods

Any optimization method that relies on a second-order auxiliary model of the form (3) can be interpreted as a preconditioned gradient descent method where

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \eta Q_t^{-1}\nabla f(\mathbf{x}_t) \tag{6}$$

and $\eta > 0$ denotes the step size. Various choices for the matrix Q_t have been discussed in the literature on preconditioning (Nocedal & Wright, 1999). The simplest example is standard gradient descent where Q_t is equal to the identity matrix and η is chosen to be inversely proportional to the smoothness parameter of f. On the other side of the spectrum, the classical Newton method defines Q_t via the Hessian matrix $\nabla^2 f(\mathbf{x}_t)$. Since the computation of the exact Hessian is often too computationally expensive, even in a single machine setting, various approximation methods have been developed. Such methods typically rely on finite sample approximations to the Hessian using sketching techniques (Pilanci & Wainwright, 2016), sub-sampling (Erdogdu & Montanari, 2015) or some quasi-Newton approximation (Dennis & Moré, 1977). They can also be combined with various line-search or trust-region-like strategies as in (Blanchet et al., 2016; Kohler & Lucchi, 2017). However, all these methods are either first-order methods, or they do not induce a block-diagonal structure on Q_t. Hence, our approach has also not been studied in the context of preconditioning methods until now.

3. Method

We introduce a general framework for studying repartitioning with the goal to cover the different distributed methods introduced in Section 2.1. The common starting point in all these methods is a second-order approximations $\tilde{f}_{\mathbf{x}_t}(\,\cdot\,; Q_t)$ to f as defined in (3). The methods then differ in their choice of Q_t and the mechanisms they implement to guarantee sufficient function decrease when preconditioning on a block-diagonal version of Q_t. To focus on repartitioning in isolation we abstract these technicalities into assumptions in Section 4. For now, let us assume a *good* local second-order model $\tilde{f}_{\mathbf{x}_t}(\,\cdot\,; Q_t)$ is given and walk through

the block-diagonal preconditioning method outlined in Algorithm 1. We first need to introduce some notation.

3.1. Notation

We write $i \in [n]$ for $i = 1, 2, ..., n$ and we denote $\mathbf{x}^\star = \arg\min_{\mathbf{x}} f(\mathbf{x})$ to refer to the minimizer of f which is written as $f^\star = f(\mathbf{x}^\star)$.

Definition 1 (Partitioning)**.** *We denote the* partitioning *of the indices $[n]$ into K disjoint subsets as $\mathcal{P} := \{\mathcal{P}_i\}_{i\in[K]}$ where $\cup_{i\in[K]}\mathcal{P}_i = [n]$ and $\mathcal{P}_i \cap \mathcal{P}_j = \emptyset,\ \forall i \neq j$. If the partitioning is randomized throughout the algorithm we use the superscripts $\mathcal{P}^t$ to refer to the partitioning at iteration t.*

Further, we write $\mathbf{x}_{[\mathcal{P}_k]} \in \mathbb{R}^n$ to refer to the vector with elements $(x_{[\mathcal{P}_k]})_i = x_i$ for $i \in \mathcal{P}_k$ and zero otherwise. Similarly, we use $M_{[\mathcal{P}_i,\mathcal{P}_j]} \in \mathbb{R}^{n\times n}$ to denote the masked version of the matrix $M \in \mathbb{R}^{n\times n}$, with only non-zero elements for $M_{k,\ell}$ with $k \in \mathcal{P}_i$ and $\ell \in \mathcal{P}_j$.

3.2. Block-Diagonal Preconditioning

In each step $t \geq 0$ of Algorithm 1 we select a partitioning $\mathcal{P}^t$ and construct a block-diagonal version $Q_{\mathcal{P}^t}$ from Q_t according to $\mathcal{P}^t$:

$$Q_{\mathcal{P}^t} := \sum_{k\in[K]} Q_{t[\mathcal{P}^t_k,\mathcal{P}^t_k]}. \tag{7}$$

This matrix then serves as a preconditioning matrix in the update step (line 7) of Algorithm 1. Note that we will for illustration purposes, and without loss of generality, refer to $Q_{\mathcal{P}^t}$ as a block-diagonal matrix. Although $Q_{\mathcal{P}^t}$ is not necessarily block-diagonal for all $\mathcal{P}^t$, it can be brought into block-diagonal form by permuting the rows and columns of the matrix.

3.3. Dynamic Partitioning

In a classical distributed method, the partitioning $\mathcal{P}^t$ is fixed throughout the entire algorithm, as discussed in Section 2.1.2. This corresponds to option (i) in Algorithm 1. The novel feature in our study is to allow for a different random partitioning in each iteration t and use the induced block-diagonal preconditioning matrix to perform the update step. This randomized procedure, also referred to as *repartitioning*, is summarized as option (ii) in Algorithm 1.

4. Convergence Analysis

We now turn to the main contribution of our work which consists in analyzing and contrasting the convergence rate of Algorithm 1 for the two different partitioning techniques. To convey our main message in the most transparent way, we start by analyzing a quadratic function where the second-order model $\tilde{f}_{\mathbf{x}_t}$ in (3) is exact and no additional assumptions are needed. We then extend this result to GLMs and to more general second-order auxiliary models. All proofs can be found in the appendix.

Algorithm 1 *Block-Diagonal Preconditioning for* (1) with (i) static and (ii) dynamic partitioning

1: **Input:** $\tilde{f}_{\mathbf{x}_t}(\cdot, Q_t)$, step size η, partitioning $\mathcal{P}$
2: **Initialize:** $\mathbf{x}_0 \in \mathbb{R}^n$
3: **for** $t = 0$ **to** $T-1$ **do**
4: (i) use default partitioning $\mathcal{P}^t = \mathcal{P}$
5: (ii) choose a random partitioning $\mathcal{P}^t$ of the $i \in [n]$
6: **for** $k \in [K]$ on each processor in parallel **do**
7: $\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t - \eta Q^{-1}_{[\mathcal{P}^t_k,\mathcal{P}^t_k]}\nabla f(\mathbf{x}_t)$
8: **end for**
9: **end for**
10: **Return:** $\mathbf{x}_T$

4.1. Quadratic Functions

Let us consider the setting where $f : \mathbb{R}^n \to \mathbb{R}$ is a quadratic function of the following form:

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top H\mathbf{x} - \mathbf{c}^\top\mathbf{x}, \tag{8}$$

where $H \in \mathbb{R}^{n\times n}$ is a symmetric matrix and $\mathbf{c} \in \mathbb{R}^n$. The natural choice is to define $Q_t = H$ when working with the auxiliary model (3). Obviously, using the full matrix H for preconditioning would yield convergence in a single step. But under the constraint of Algorithm 1 that the preconditioning matrix $Q_{\mathcal{P}^t}$ has block-diagonal structure this in general does not hold true.

Theorem 1. *Assume f is defined as (8), and $Q_t := H$. Then Algorithm 1 with a fixed step size $\eta = \frac{1}{K}$ converges at a linear rate*

$$\mathbb{E}\left[f(\mathbf{x}_{t+1}) - f(\mathbf{x}^\star)\right] \leq (1-\rho)^t\left[f(\mathbf{x}_0) - f(\mathbf{x}^\star)\right]$$

with

$$\rho := \frac{1}{K}\lambda_{\min}\left(\mathbb{E}[H^{-1}_{\mathcal{P}^t}]H\right). \tag{9}$$

The expectations are taken over the randomness of the partitioning $\mathcal{P}^t$.

One of the key insights is that *the convergence rate of Algorithm 1 depends on the partitioning scheme through the term $\mathbb{E}[H^{-1}_{\mathcal{P}^t}]$*. Ideally, for optimal convergence we want ρ to be as large as possible, and hence $\mathbb{E}[H^{-1}_{\mathcal{P}^t}] \approx H^{-1}$. How well $\mathbb{E}[H^{-1}_{\mathcal{P}^t}]$ is able to approximate H^{-1} is measured by the spectrum of $\mathbb{E}[H^{-1}_{\mathcal{P}^t}]H$ which determines the convergence rate of the respective partitioning scheme. For fixed partitioning the expectation $\mathbb{E}[H^{-1}_{\mathcal{P}^t}]$ reduces to $H^{-1}_{\mathcal{P}}$ induced by the default partitioning $\mathcal{P}$, whereas for repartitioning it is an average over all possible partitionings. As a

consequence the convergence of repartitioning is superior to static partitioning whenever the average of $H_{\mathcal{P}}^{-1}$ over all partitionings is able to better approximates H^{-1} compared to any individual term.

Note that even in cases where there exists a single fixed partitioning that is better than repartitioning, it could still be combinatorially hard to discover it and repartitioning provides an appealing alternative. We will provide several empirical results that support this claim in Section 6 and investigate the properties of the matrix $\mathbb{E}[H_{\mathcal{P}^t}^{-1}]H$ analytically for some particular H in Section 5.

4.2. Smoothness Upper-bound for GLMs

As a second case study we focus on GLMs, as defined in (4), where ℓ is a γ_ℓ-smooth loss function. In this setting we analyze Algorithm 1 for the second-order model $\tilde{f}_{\mathbf{x}_t}(\,\cdot\,; Q_t)$ defined through

$$Q_t := \gamma_\ell A^\top A. \tag{10}$$

This model forms a global upper-bound on the objective function f. It is used in (Smith et al., 2018) and related algorithms. It can intuitively be understood that in this case the quality of the auxiliary model depends on the correlation between data columns residing in different partitions; these are the coordinates of Q_t that are being ignored when enforcing a block-diagonal structure for achieving separability. Let us for simplicity denote $M := A^\top A$. Then, the expected function decrease in each iteration of Algorithm 1 can be bounded as:

Lemma 2. *Assume f is γ-smooth, has the form (4), and Q_t is chosen as in (10). Then, in each step $t \geq 0$ of Algorithm 1 with a fixed step size $\eta = \frac{1}{K}$ the objective decreases as*

$$\mathbb{E}\left[f(\mathbf{x}_t) - f(\mathbf{x}_{t+1})\right] \geq \frac{\gamma}{K}\lambda_{\min}(A\mathbb{E}[M_{\mathcal{P}^t}^{-1}]A^\top)\|\nabla f(\mathbf{x}_t)\|^2$$

where expectation are taken over the randomness of the partitioning $\mathcal{P}^t$.

The dependence on the partitioning is captured by the term $\lambda_{\min}(A\mathbb{E}[M_{\mathcal{P}^t}^{-1}]A^\top)$, which is optimized for $\mathbb{E}[M_{\mathcal{P}^t}^{-1}] \approx M^{-1}$ and simplifies to $\lambda_{\min}(\mathbb{E}[M_{\mathcal{P}^t}^{-1}]M)$ for symmetric A.

To translate Lemma 2 into a convergence rate for Algorithm 1 we need a lower bound on the curvature of f in order to relate the gradient norm to the suboptimality. The following standard assumption (Polyak, 1963) allows us to do this:

Assumption 1 (Polyak Lojasiewicz)**.** *Assume the function f satisfies the following inequality for some $\mu > 0$*

$$\frac{1}{2}\|\nabla f(\mathbf{x})\|^2 \geq \mu(f(\mathbf{x}) - f(\mathbf{x}^\star)). \tag{11}$$

Note that this assumption is weaker than strong-convexity as shown by Karimi et al. (2016). The following linear convergence rate for Algorithm 1 follows:

Theorem 3. *Consider the same setup as in Lemma 2 where f in addition satisfies Assumption 1 with constant $\mu > 0$. Then, Algorithm 1 with a fixed step size $\eta = \frac{1}{K}$ converges as*

$$\mathbb{E}\left[f(\mathbf{x}_{t+1}) - f(\mathbf{x}^\star)\right] \leq (1-\rho)^t\left[f(\mathbf{x}_0) - f(\mathbf{x}^\star)\right].$$

with

$$\rho := \frac{2\mu}{K\gamma}\lambda_{\min}\left(A\mathbb{E}[M_{\mathcal{P}^t}^{-1}]A^\top\right), \tag{12}$$

where $M_{\mathcal{P}^t}$ denotes the masked version of $M := A^\top A$ given by the partitioning $\mathcal{P}^t$ and expectations are taken over the randomness of the partitioning.

4.3. General Auxiliary Model

For the most general case we do not pose any structural assumption on f. We only assume the auxiliary model $\tilde{f}_{\mathbf{x}_t}(\,\cdot\,, Q_t)$ is a reasonably good approximation to the function f and satisfies the following assumption.

Assumption 2. *$\tilde{f}_{\mathbf{x}_t}(\,\cdot\,; Q_t)$ is such that $\forall \Delta\mathbf{x}$ and some $\xi \in (0, 1]$ it holds that*

$$f(\mathbf{x}_t + \Delta\mathbf{x}) \leq \xi\tilde{f}_{\mathbf{x}_t}(\Delta\mathbf{x}; Q_t) + (1-\xi)f(\mathbf{x}_t). \tag{13}$$

Approximations $\tilde{f}_{\mathbf{x}_t}(\,\cdot\,; Q_t)$ that satisfy (13) can easily be obtained for smooth functions by taking a Taylor approximation truncated at order p and combined with a bound on the p-th derivative, see e.g. (Birgin et al., 2017; Nesterov & Polyak, 2006). In our case where we want a quadratic model we choose $p = 2$ and, because smoothness gives us an upper-bound, the inequality (13) holds for $\xi = 1$. Another popular approach to guarantee sufficient function decrease in the spirit of (13) are backtracking line-search methods, such as used in (Lee & Chang, 2017). Here ξ directly maps to the control parameter in the Armijo-Goldstein condition (Armijo, 1966) if Q_t is PSD. In the appendix we discuss these connections further and explain how our setting could be extended to also cover trust region like approaches (Cartis et al., 2011) such as used in ADN (Dünner et al., 2018).

For methods that build on auxiliary models that satisfy (13) we can quantify the dependence of the function decrease on the partitioning scheme using the following lemma.

Lemma 4. *Consider a convex objective f and a quadratic approximation $\tilde{f}_{\mathbf{x}_t}(\,\cdot\,; Q_t)$ satisfying (13). Then, in each step of Algorithm 1 the function value decreases as*

$$\mathbb{E}[f(\mathbf{x}_t) - f(\mathbf{x}_{t+1})] \geq \rho_t\|\Delta\tilde{\mathbf{x}}_t^\star\|^2$$

where

$$\rho_t := \frac{\xi}{2K}\lambda_{\min}(Q_t^\top\mathbb{E}[Q_{\mathcal{P}^t}^{-1}]Q_t), \tag{14}$$

with $\Delta\tilde{\mathbf{x}}_t^\star := \arg\min_{\mathbf{x}} \tilde{f}_{\mathbf{x}_t}(\mathbf{x}, Q_t)$ denoting the optimal next iterate according to $\tilde{f}_{\mathbf{x}_t}$.

Hence, even in the most general case, the dependency of the convergence rate on the partitioning scheme can be explained through a simple quantity involving the expected block-diagonal preconditioning matrix $\mathbb{E}[Q_{\mathcal{P}^t}^{-1}]$: $\lambda_{\min}(Q_t^\top \mathbb{E}[Q_{\mathcal{P}^t}^{-1}]Q_t)$.

The auxiliary model $\tilde{f}_{\mathbf{x}_t}$, on the other hand, and hence its minimizer $\tilde{\mathbf{x}}_t^\star$ are independent of the partitioning. Hence, how we translate Lemma 4 into a convergence results solely depends on the distributed method we deploy. Here, we make the following assumption.

Assumption 3 (Sufficient function decrease)**.** *The method defines Q_t such that sufficient function decrease of the optimal update $\Delta\tilde{\mathbf{x}}_t^\star$ can be guaranteed:*

$$f(\mathbf{x}_t + \Delta\tilde{\mathbf{x}}_t^\star) - f(\mathbf{x}^\star) \leq \alpha[f(\mathbf{x}_t) - f(\mathbf{x}^\star)] \tag{15}$$

for some $\alpha \in [0, 1)$.

This assumption can be satisfied with a preconditioned gradient descent step and appropriate rescaling of Q_t for any PSD matrix Q_t. Importantly, such a rescaling affects every partitioning scheme equally.

We note that alternative assumptions would also lead to convergence results. For example, techniques found in the trust-region and cubic regularization literature (see e.g. (Cartis et al., 2011; Dünner et al., 2018)) have proposed to adapt the optimization algorithm instead to guarantee sufficient function decrease in the spirit of (15).

Building on Assumption 3 we get the following rate of convergence for Algorithm 1.

Theorem 5. *Assume f is γ-smooth and L-Lipschitz. Then, Algorithm 1 with $\tilde{f}_{\mathbf{x}_t}(\,\cdot\,; Q_t)$ satisfying Assumption 3 and a fixed step size $\eta = \frac{1}{K}$ converges as*

$$\mathbb{E}[f(\mathbf{x}_{t+1}) - f(\mathbf{x}^\star)] \leq \left(1 - \min_t \rho_t \tfrac{(1-\alpha)}{L}\right)^t \varepsilon_0$$

where $\varepsilon_0 = f(\mathbf{x}_0) - f(\mathbf{x}^\star)$ and ρ_t defined in (14).

Note that the step size $\eta = \frac{1}{K}$ is required throughout our analysis because we pose assumption (13) on $\tilde{f}_{\mathbf{x}}$ and need to guarantee convergence uniformly across partitionings for a method that uses a block-diagonal version of Q_t. To dynamically adapt to each partitioning Algorithm 1 could be augmented with a line-search procedure. We omitted this to preserve clarity of our presentation.

Similarly, all our results from this section can readily be extended to the case where the local subproblem (3) is not necessarily solved exactly but only θ-approximately (in the sense of Assumption 1 used by Smith et al. (2018)). This provides additional freedom to trade-off overheads of repartitioning and sample efficiency for optimal performance.

5. Effect of Randomization

Let us return to the quadratic case where the auxiliary model $\tilde{f}_{\mathbf{x}_t}$ is exact and focus on the dependence of ρ on the partitioning scheme. We recall that the value of ρ as defined in (9) is determined by the smallest eigenvalue of

$$\Lambda_{\mathcal{P}} := Q_{\mathcal{P}}^{-1} Q. \tag{16}$$

In the following we will evaluate $\lambda_{\min}(\mathbb{E}[\Lambda_{\mathcal{P}}])$ analytically for some particular choices of Q to quantify the gain of repartitioning over static partitioning predicted by Theorem 1.

For simplicity, we assume Q does not depend on t and the partitioning $\mathcal{P}$ is uniform, such that $|\mathcal{P}_i| = |\mathcal{P}_j| = n_k \,\forall j, i \in [K]$ and $n_k = \frac{n}{K}$ denotes the number of coordinates assigned to each partition.

5.1. Uniform Correlations

We start with the special case where all off-diagonal elements of Q are equal to $\alpha \in [0, 1)$. Thus, for $n = 4$ the matrix Q would look as follows:

$$Q = \begin{bmatrix} 1 & \alpha & \alpha & \alpha \\ \alpha & 1 & \alpha & \alpha \\ \alpha & \alpha & 1 & \alpha \\ \alpha & \alpha & \alpha & 1 \end{bmatrix}.$$

Such a structure of Q would, for example, appear in a linear regression problem, where all columns of the data matrix A are equally correlated. In such a scenario it does not matter which elements of Q we ignore and all fixed partitionings are equivalent from an algorithmic perspective. We refer the reader to Figure 6 in the appendix for an illustration of all matrices involved in this example. Let us note that

$$\Lambda_{\mathcal{P}} = Q_{\mathcal{P}}^{-1} Q = Q_{\mathcal{P}}^{-1}(Q_{\mathcal{P}} + Q_{\mathcal{P}}^c) = I + Q_{\mathcal{P}}^{-1} Q_{\mathcal{P}}^c$$

where $Q_{\mathcal{P}}^c := Q - Q_{\mathcal{P}}$. Given the specific structure of Q considered in this example, the inverse $Q_{\mathcal{P}}^{-1}$ can be computed from the individual blocks of $Q_{\mathcal{P}}$ and is again symmetric and block-diagonal. As a consequence the diagonal blocks of $Q_{\mathcal{P}}^{-1} Q_{\mathcal{P}}^c$ are zero and by symmetry all other elements are equal. We denote the value of these elements by ϵ, where an exact derivation as a function of α can be found in Appendix C.1. In the following we will evaluate

$$\lambda_{\min}(\mathbb{E}[\Lambda_{\mathcal{P}}]) = 1 + \lambda_{\min}(\mathbb{E}[Q_{\mathcal{P}}^{-1} Q_{\mathcal{P}}^c])$$

for the case of static as well as dynamic partitioning.

(i) Static Partitioning. We have $\mathbb{E}[\Lambda_{\mathcal{P}}] = \Lambda_{\mathcal{P}}$ and we compute $\lambda_{\min}(\Lambda_{\mathcal{P}})$ by exploiting the symmetry of the matrix $Q_{\mathcal{P}}^{-1} Q_{\mathcal{P}}^c$. The eigenvector corresponding to the smallest eigenvalue will be $\mathbf{v} = \mathbf{e}_{\mathcal{P}_i} - \mathbf{e}_{\mathcal{P}_j}$ for any $i \neq j$ and the corresponding eigenvalue with multiplicity $K - 1$ is

$$\lambda_{\min}(Q_{\mathcal{P}}^{-1} Q_{\mathcal{P}}^c) = -\epsilon n_k \;\Rightarrow\; \lambda_{\min}(\Lambda_{\mathcal{P}}) = 1 - \epsilon n_k.$$

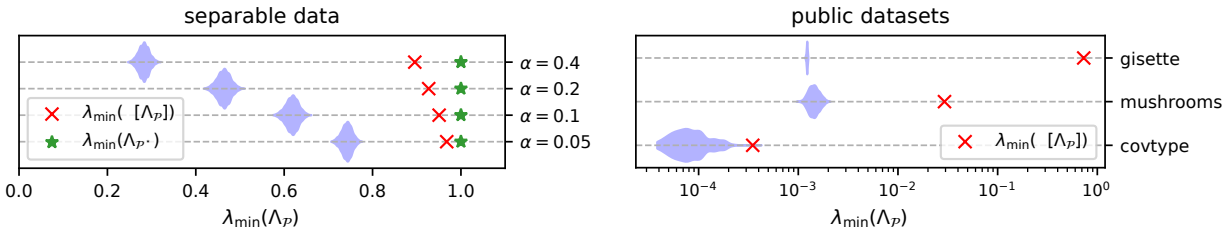


Figure 2. Violin plot of the distribution of $\lambda_{\min}(\Lambda_{\mathcal{P}})$ across 1000 random partitions on different datasets for $K = 5$ partitions. We compare $\lambda_{\min}(\Lambda_{\mathcal{P}})$ that determines the rate of static partitioning to $\lambda_{\min}(\mathbb{E}[\Lambda_{\mathcal{P}}])$ that governs the rate of dynamic partitioning. In the case of synthetic data where the best partitioning $\mathcal{P}^\star$ is known, we also show $\lambda_{\min}(\Lambda_{\mathcal{P}^\star})$.

(ii) Dynamic Partitioning. The matrix $\mathbb{E}[Q_{\mathcal{P}}^{-1}Q_{\mathcal{P}}^c]$ is an expectation over the block-diagonal matrices arising from different partitionings. The probability that a particular off-diagonal element is non-zero for any random partitioning is $p = n_k(K-1)/(n-1)$. This yields a matrix where the diagonal elements are zero and all off-diagonal elements are equal to $p\epsilon$. Hence, again, by symmetry, the eigenvector corresponding to the smallest eigenvalue will be $\mathbf{v} = \mathbf{e}_i - \mathbf{e}_j$ for any $i \neq j$ and the corresponding eigenvalue is

$$\lambda_{\min}(\mathbb{E}[Q_{\mathcal{P}}^{-1}]Q_{\mathcal{P}}^c) = -\epsilon p \;\Rightarrow\; \lambda_{\min}(\mathbb{E}[\Lambda_{\mathcal{P}}]) = 1 - \epsilon p.$$

We conclude that for $K > 1$ and $n_k > 1$ we have

$$0 < \lambda_{\min}(\Lambda_{\mathcal{P}}) \leq \lambda_{\min}(\mathbb{E}[\Lambda_{\mathcal{P}}]) \leq 1$$

where the inequality is strict for any $\alpha > 0$. Hence, repartitioning moves the smallest eigenvalue by a factor of $\frac{K-1}{n-1} \approx \frac{1}{n_k}$ closer to 1 compared to any static partitioning. By inspecting ϵ we also see that the potential convergence gain of repartitioning increases as the weight α in the off-diagonal elements gets larger. This directly translates into a significantly better convergence rate as by Theorem 3. We later verify this empirically in Section 6. For an illustration of the sensitivity of the eigenvalues w.r.t α and K we refer to Figure 8 in Appendix D.

5.2. Separable Data

Let us consider a second extreme case, where Q has block-diagonal structure by definition. We again assume that all non-zero off-diagonal elements are equal to $\alpha \in [0, 1)$. For $n = 4, K = 2$ the matrix Q would look as follows:

$$Q = \begin{bmatrix} 1 & \alpha & 0 & 0 \\ \alpha & 1 & 0 & 0 \\ 0 & 0 & 1 & \alpha \\ 0 & 0 & \alpha & 1 \end{bmatrix}$$

This could, for example, correspond to a linear regression setting where the data is perfectly separable and data columns within partitions are equally correlated. In this case, the best static partitioning $\mathcal{P}^\star$ is aligned with the block structure of the matrix. In this case $Q = Q_{\mathcal{P}^\star}$, and hence $\lambda_{\min}(\Lambda_{\mathcal{P}^\star}) = 1$. We can show that for $K > 1$ it holds that

$$\min_{\mathcal{P}} \lambda_{\min}(\Lambda_{\mathcal{P}}) \leq \lambda_{\min}(\mathbb{E}[\Lambda_{\mathcal{P}}]) \leq \max_{\mathcal{P}} \lambda_{\min}(\Lambda_{\mathcal{P}}) \quad (17)$$

and equality is achieved for $\alpha = 0$. Recall that the convergence rate of Algorithm 1 as by Theorem 1 is proportional to $\mathbb{E}[\Lambda_{\mathcal{P}}]$. Hence, the order in (17) implies that the convergence rate of repartitoning lies between the best and the worst static partitioning.

To investigate where on this spectrum the convergence of repartitiong actually is, we compute the distribution of $\lambda_{\min}(\Lambda_{\mathcal{P}})$ and the corresponding value of $\lambda_{\min}(\mathbb{E}[\Lambda_{\mathcal{P}}])$ numerically for different values of α. Results are illustrated in the left plot of Figure 2. The violin plot suggests that even in the perfectly separable case, repartitioning achieves a significantly better convergence rate than static partitioning with probability close to 1. Hence, if we do not know the best partitioning a priori repartitioning might be the best choice.

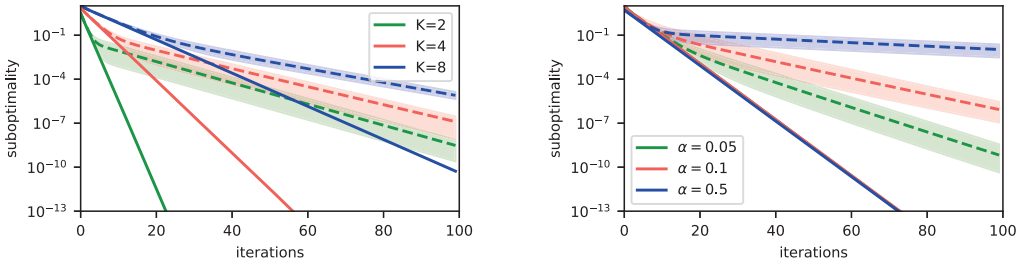
5.3. Real Datasets

We conclude this section by considering a more practical choice of Q. Therefore, we consider a ridge regression problem with $Q = A^\top A + \lambda I$. We choose $\lambda = 1$ and we evaluate $\lambda_{\min}(\Lambda_{\mathcal{P}})$ numerically for some popular datasets. We have chosen the gisette, the mushroom and the covtype dataset that can be downloaded from (Dua & Graff, 2017) and whose statistics are reported in Table 1.

In the right plot of Figure 2 we compare the distribution of $\lambda_{\min}(\Lambda_{\mathcal{P}})$ for the three datsets across random partitionings $\mathcal{P}$ with $\lambda_{\min}(\mathbb{E}[\Lambda_{\mathcal{P}}])$. We see that across all datasets ρ for random partitioning is higher than for any fixed partitioning with very high probability which implies superior convergence of repartitioning as by our theory. This observation is also consistent across different choices of regularizer as shown in Figure 12 in the appendix for completeness.

$\alpha = 0.1$	$\frac{1}{K}\lambda_{\min}(\mathbb{E}[\Lambda_{\mathcal{P}}])$	$\frac{1}{K}\lambda_{\min}(\Lambda_{\mathcal{P}})$
$K = 2$	0.498	0.040
$K = 4$	0.247	0.038
$K = 8$	0.122	0.034

$K = 5$	$\frac{1}{K}\lambda_{\min}(\mathbb{E}[\Lambda_{\mathcal{P}}])$	$\frac{1}{K}\lambda_{\min}(\Lambda_{\mathcal{P}})$
$\alpha = 0.01$	0.199	0.142
$\alpha = 0.1$	0.197	0.037
$\alpha = 0.5$	0.196	0.005



(a) sensitivity w.r.t. K ($\alpha = 0.1$)

(b) sensitivity w.r.t. α ($K = 5$)

Figure 3. Linear regression on synthetic data ($n = 200$) with uniform correlations of strength α. We compare the empirical convergence of Algorithm 1 for static (dashed) and dynamic (solid) partitioning to the corresponding values of $\rho = \frac{1}{K}\lambda_{\min}(\mathbb{E}[\Lambda_{\mathcal{P}}])$ that determine the respective theoretical convergence rate (see Theorem 1) across different values of K and α. Confidence intervals show min-max intervals over 100 runs and the regularization parameter is $\lambda = 1$.

6. Performance Results

Finally, we compare the convergence gains of repartitioning predicted by our theory, with the actual convergence of Algorithm 1 with and without repartitioning. We consider two popular machine learning problems. First, we consider linear regression where

$$f(\mathbf{x}) = \frac{1}{2}\|A\mathbf{x} - \mathbf{y}\|^2 + \frac{\lambda}{2}\|\mathbf{x}\|^2$$

and the second-order model $\tilde{f}_{\mathbf{x}}$ is exact with $Q_t = A^\top A + \lambda I$. This allows us to analyse the scenarios discussed in Section 5 and the gains predicted by Theorem 1. As a second application we consider L_2-regularized logistic regression

$$f(\mathbf{x}) = \sum_{i\in[n]} \log\left(1 + \exp(-y_i A_{i,:}\mathbf{x})\right) + \frac{\lambda}{2}\|\mathbf{x}\|^2$$

with $y_i \in \{\pm 1\}$ where we use the second-order Taylor expansion for defining $\tilde{f}_{\mathbf{x}}(\,\cdot\,, Q_t)$. This corresponds to the general case analyzed in Theorem 5 where Q_t depends on the model $\mathbf{x}_t$ and changes across iterations. If not stated otherwise we use $\lambda = 1$

6.1. Validation of Convergence Rates

Let us revisit the synthetic examples from Section 5 and verify the convergence of Algorithm 1 empirically. We start with the uniform correlation example from Section 5.1 and generate a synthetic data matrix $A = Q^{1/2}$ together with random labels $\mathbf{y}$. We then train a linear regression model and investigate the convergence of Algorithm 1 for (i) static and (ii) dynamic partitioning. In Figure 3 we contrast the convergence results to the theoretical rate predicted by Theorem 1 which we can evaluate using the expressions derived in Section 5.1. We perform this experiment for different values of K and α. The tables on the top contain the values of the convergence rate $\rho = \frac{1}{K}\lambda_{\min}(\mathbb{E}[\Lambda_{\mathcal{P}}])$ for the corresponding figures at the bottom. We observe a very close match between the relative gain of repartitioning over static partitioning predicted by the theory and the empirical behavior. This supports that $\lambda_{\min}(\mathbb{E}[\Lambda_{\mathcal{P}}])$ indeed captures the effect of repartitioning accurately.

Table 1. Size of the datasets used in our experimental results.

Dataset	# datapoints	# features
mushroom	8124	112
covtype	581012	54
gisette	6000	5000
rcv1	20'242	677399
url	2396130	3231961
synthetic	200	200

We further verify the empirical convergence for the separable data from Section 5.2 as well as the ridge regression setting from Section 5.3. The convergence results of Algorithm 1 are depicted in Figure 4 for a subset of the parameter settings. Again, we observe a strong correlation between the empirical convergence gains and the values of $\lambda_{\min}(\Lambda_{\mathcal{P}})$ evaluated numerically in Figure 2 across all datasets.

To be consistent with the assumptions used in our theorems, we have implemented Algorithm 1 with a fixed step size η. Alternatively, the algorithm could also be augmented with backtracking line search (Armijo, 1966). This would potentially improve the performance of good partitionings even further, but it is not expected to significantly change

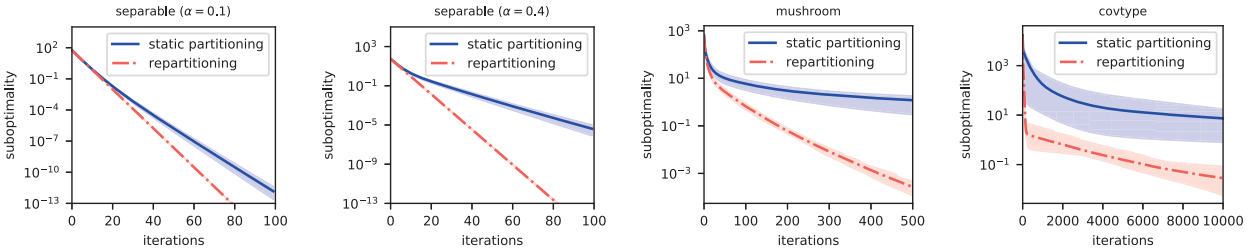
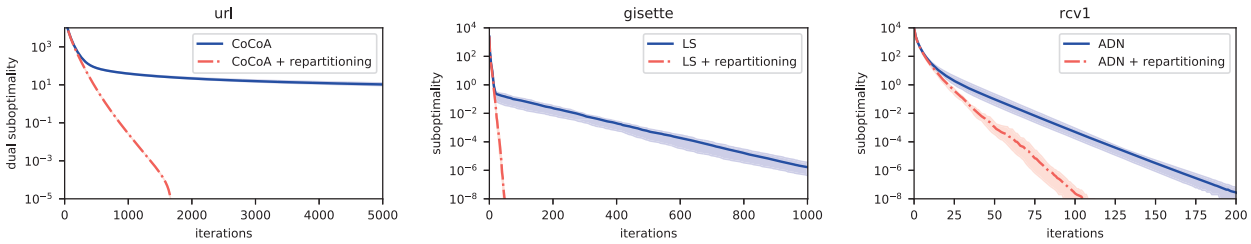


Figure 4. Empirical performance of Algorithm 1 for linear regression on a selection of the datasets analyzed in Figure 2. The relative convergence of Alg 1 with and without repartitioning closely match the values predicted by our theory as given through $\lambda_{\min}(\Lambda_{\mathcal{P}})$ (see Theorem 1) whose values are illustrated for the respective datasets in Figure 2. Confidence intervals show min-max intervals over 10 runs.



(a) CoCoA–dual (Smith et al., 2018) (b) LS (Lee & Chang, 2017) (c) ADN (Dünner et al., 2018)

Figure 5. Combining existing distributed methods with repartitioning: Training of a logistic regression classifier with $\lambda = 1$ using three different algorithms and datasets for $K = 8$. Confidence intervals show min-max intervals over 10 runs. Experiments on additional datasets and different values of K can be found in Figure 10 in the appendix.

the relative behavior of static versus dynamic partitioning which is the main study of this paper. To support this claim we compare the convergence of Algorithm 1 for the two partitioning schemes with and without backtracking line-search on three synthetic examples in Figure 9 in the appendix.

6.2. Existing Algorithms

To complete our study we have implemented three popular existing distributed methods and combined them with repartitioning. These are CoCoA (Smith et al., 2018) with a dual solver, ADN (Dünner et al., 2018) and the line-search-based approach by Lee & Chang (2017), referred to as LS. We have trained a logistic regression classifier using all three algorithms on three different datasets and illustrate the respective convergence with and without repartitioning in Figure 5. Additional results for ADN on two more datasets can be found in Figure 10 in the appendix. Overall, we see a consistent and significant gain of repartitioning for all three optimization methods. We find that the potential convergence gain of repartitioning mostly depends on the statistics of the datasets (which defines Q_t) and it is similarly large for all methods. For the url data repartitioning with a dual solver reduces sample complexity by several orders of magnitude, for gisette the gain is around $30\times$ and for the rcv1 dataset it is $2\times$. When inspecting the properties of the datasets we find that the gain of repartitioning grows with the density and the dimension of the columns of the data matrix A. This is expected, because it implies stronger correlations and hence more weight in the off-diagonal elements of Q_t.

7. Conclusion

We have demonstrated theoretically, as well as empirically, that repartitioning can improve the sample complexity of existing block-separable optimization methods by potentially several orders of magnitude. The gain crucially depends on the problem at hand and is accurately captured by a simple analytical quantity identified in our theoretical analysis. The repartitioning technique discussed in this manuscript is versatile and our analysis is intentionally kept general to cover different types of algorithms and preconditioning matrices.

Together with prior work (Ioannou et al., 2019) that emphasized the implementation efficiency of block-separable models on modern hardware, our results highlight that repartitioning turns existing distributed methods into promising candidates for parallel learning. In addition. these methods have the important benefit that they come with convergence guarantees for arbitrary degrees of parallelism without prior assumptions on the data. This allows them to be scaled to any number of available cores.

Acknowledgements

We wish to acknowledge support from the Swiss National Science Foundation Early Postdoc.Mobility Fellowship Program.

References

Armijo, L. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific J. Math.*, 16(1): 1–3, 1966.

Birgin, E. G., Gardenghi, J., Martínez, J. M., Santos, S. A., and Toint, P. L. Worst-case evaluation complexity for unconstrained nonlinear optimization using high-order regularized models. *Mathematical Programming*, 163 (1-2):359–368, 2017.

Blanchet, J., Cartis, C., Menickelly, M., and Scheinberg, K. Convergence rate analysis of a stochastic trust region method for nonconvex optimization. *arXiv preprint arXiv:1609.07428*, 5, 2016.

Cartis, C., Gould, N. I. M., and Toint, P. L. Adaptive cubic regularisation methods for unconstrained optimization. part i: motivation, convergence and numerical results. *Mathematical Programming*, 127(2):245–295, Apr 2011. ISSN 1436-4646.

Dekel, O., Gilad-Bachrach, R., Shamir, O., and Xiao, L. Optimal distributed online prediction using mini-batches. *J. Mach. Learn. Res.*, 13:165–202, January 2012. ISSN 1532-4435.

Dennis, Jr, J. E. and Moré, J. J. Quasi-newton methods, motivation and theory. *SIAM review*, 19(1):46–89, 1977.

Dua, D. and Graff, C. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Dünner, C., Lucchi, A., Gargiani, M., Bian, A., Hofmann, T., and Jaggi, M. A distributed second-order algorithm you can trust. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pp. 1358–1366, 2018.

Erdogdu, M. A. and Montanari, A. Convergence rates of sub-sampled newton methods. *arXiv preprint arXiv:1508.02810*, 2015.

Hsieh, C.-J., Si, S., and Dhillon, I. S. Communication-Efficient Parallel Block Minimization for Kernel Machines. *arXiv*, August 2016.

Ioannou, N., Dünner, C., and Parnell, T. Syscd: A system-aware parallel coorindate descent algorithm. *NeurIPS*, 2019.

Jaggi, M., Smith, V., Takáč, M., Terhorst, J., Krishnan, S., Hofmann, T., and Jordan, M. I. Communication-efficient distributed dual coordinate ascent. In *Neural Information Processing Systems*, 2014.

Karimi, H., Nutini, J., and Schmidt, M. Linear convergence of gradient and proximal-gradient methods under the polyak-łojasiewicz condition. In Frasconi, P., Landwehr, N., Manco, G., and Vreeken, J. (eds.), *Machine Learning and Knowledge Discovery in Databases*, pp. 795–811, Cham, 2016. Springer International Publishing.

Kohler, J. M. and Lucchi, A. Sub-sampled cubic regularization for non-convex optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1895–1904. JMLR. org, 2017.

Lee, C.-p. and Chang, K.-W. Distributed block-diagonal approximation methods for regularized empirical risk minimization. *arXiv preprint arXiv:1709.03043*, 2017.

Liu, J., Wright, S. J., Ré, C., Bittorf, V., and Sridhar, S. An Asynchronous Parallel Stochastic Coordinate Descent Algorithm. *Journal of Machine Learning Research*, 16: 285–322, 2015.

Ma, C., Konečný, J., Jaggi, M., Smith, V., Jordan, M., Richtárik, P., and Takáč, M. Distributed optimization with arbitrary local solvers. *arXiv.org*, 2015a.

Ma, C., Smith, V., Jaggi, M., Jordan, M. I., Richtárik, P., and Takáč, M. Adding vs. averaging in distributed primal-dual optimization. *International Conference on Machine Learning*, 2015b.

Mahajan, D., Keerthi, S. S., and Sundararajan, S. A distributed block coordinate descent method for training l 1 regularized linear classifiers. *Journal of Machine Learning Research*, 18(91):1–35, 2017.

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

Niu, F., Recht, B., Ré, C., and Wright, S. J. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *Neural Information Processing Systems*, 2011.

Nocedal, J. and Wright, S. Numerical optimization. 1999.

Pilanci, M. and Wainwright, M. J. Iterative hessian sketch: Fast and accurate solution approximation for constrained least-squares. *The Journal of Machine Learning Research*, 17(1):1842–1879, 2016.

Polyak, B. T. Gradient methods for minimizing functionals (in russian). *Zh. Vychisl. Mat. Mat. Fiz*, 1963.

Reddi, S. J., Konečnỳ, J., Richtárik, P., Póczós, B., and Smola, A. Aide: Fast and communication efficient distributed optimization. *arXiv preprint arXiv:1608.06879*, 2016.

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

Shamir, O., Srebro, N., and Zhang, T. Communication-efficient distributed optimization using an approximate newton-type method. In *International conference on machine learning*, pp. 1000–1008, 2014.

Smith, V., Forte, S., Ma, C., Takáč, M., Jordan, M. I., and Jaggi, M. CoCoA: A General Framework for Communication-Efficient Distributed Optimization. *Journal of Machine Learning Research (and arXiv:1611.02189)*, 2018.

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

Zhang, Y. and Lin, X. Disco: Distributed optimization for self-concordant empirical loss. In *International conference on machine learning*, pp. 362–370, 2015.