Federated Optimization: Distributed Machine Learning for On-Device Intelligence

Federated Optimization: Distributed Machine Learning for On-Device Intelligence

Jakub Konečný
University of Edinburgh
kubo.konecny@gmail.com

H. Brendan McMahan
Google
mcmahan@google.com

Daniel Ramage
Google
dramage@google.com

Peter Richtárik
University of Edinburgh
peter.richtarik@ed.ac.uk

October 11, 2016

Abstract

We introduce a new and increasingly relevant setting for distributed optimization in machine learning, where the data defining the optimization are unevenly distributed over an extremely large number of nodes. The goal is to train a high-quality centralized model. We refer to this setting as *Federated Optimization*. In this setting, communication efficiency is of the utmost importance and minimizing the number of rounds of communication is the principal goal.

A motivating example arises when we keep the training data locally on users' mobile devices instead of logging it to a data center for training. In federated optimization, the devices are used as compute nodes performing computation on their local data in order to update a global model. We suppose that we have extremely large number of devices in the network — as many as the number of users of a given service, each of which has only a tiny fraction of the total data available. In particular, we expect the number of data points available locally to be much smaller than the number of devices. Additionally, since different users generate data with different patterns, it is reasonable to assume that no device has a representative sample of the overall distribution.

We show that existing algorithms are not suitable for this setting, and propose a new algorithm which shows encouraging experimental results for sparse convex problems. This work also sets a path for future research needed in the context of federated optimization.

1 Introduction

Mobile phones and tablets are now the primary computing devices for many people. In many cases, these devices are rarely separated from their owners [19], and the combination of rich user interactions and powerful sensors means they have access to an unprecedented amount of data, much of it private in nature. Models learned on such data hold the promise of greatly improving usability by powering more intelligent applications, but the sensitive nature of the data means there are risks and responsibilities to storing it in a centralized location.

We advocate an alternative — *federated learning* — that leaves the training data distributed on the mobile devices, and learns a shared model by aggregating locally computed updates via a central coordinating server. This is a direct application of the principle of focused collection or data minimization proposed by the 2012 White House report on the privacy of consumer data [98]. Since these updates are specific to improving the current model, they can be purely ephemeral — there is no reason to store them on the server once they have been applied. Further, they will never contain more information than the raw training data (by the data processing inequality), and will generally contain much less. A principal advantage of this approach is the decoupling of model training from the need for direct access to the raw training data. Clearly, some trust of the server coordinating the training is still required, and depending on the details of the model and algorithm, the updates may still contain private information. However, for applications where the training objective can be specified on the basis of data available on each client, federated learning can significantly reduce privacy and security risks by limiting the attack surface to only the device, rather than the device and the cloud.

If additional privacy is needed, randomization techniques from differential privacy can be used. The centralized algorithm could be modified to produce a differentially private model [17, 33, 1], which allows the model to be released while protecting the privacy of the individuals contributing updates to the training process. If protection from even a malicious (or compromised) coordinating server is needed, techniques from local differential privacy can be applied to privatize the individual updates [32]. Details of this are beyond the scope of the current work, but it is a promising direction for future research.

A more complete discussion of applications of federated learning as well as privacy ramifications can be found in [62]. Our focus in this work will be on federated optimization, the optimization problem that must be solved in order to make federated learning a practical alternative to current approaches.

1.1 Problem Formulation

The optimization community has seen an explosion of interest in solving problems with finite-sum structure in recent years. In general, the objective is formulated as

$$\min_{w \in \mathbb{R}^d} f(w) \qquad \text{where} \qquad f(w) \overset{\text{def}}{=} \frac{1}{n} \sum_{i=1}^{n} f_i(w). \tag{1}$$

The main source of motivation are problems arising in machine learning. The problem structure (1) covers linear or logistic regressions, support vector machines, but also more complicated models such as conditional random fields or neural networks.

We suppose we have a set of input-output pairs $\{x_i, y_i\}_{i=1}^n$, and a loss function, giving rise to the functions f_i. Typically, $x_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$ or $y_i \in \{-1, 1\}$. Simple examples include

- linear regression: $f_i(w) = \frac{1}{2}(x_i^T w - y_i)^2$, $y_i \in \mathbb{R}$
- logistic regression: $f_i(w) = -\log(1 + \exp(-y_i x_i^T w))$, $y_i \in \{-1, 1\}$
- support vector machines: $f_i(w) = \max\{0, 1 - y_i x_i^T w\}$, $y_i \in \{-1, 1\}$

More complicated non-convex problems arise in the context of neural networks, where rather than via the linear-in-the-features mapping $x_i^T w$, the network makes prediction through a non-convex function of the feature vector x_i. However, the resulting loss can still be written as $f_i(w)$, and gradients can be computed efficiently using backpropagation.

The amount of data that businesses, governments and academic projects collect is rapidly increasing. Consequently, solving problem (1) arising in practice is often impossible on a single node, as merely storing the whole dataset on a single node becomes infeasible. This necessitates the use of a distributed computational framework, in which the training data describing the problem is stored in a distributed fashion across a number of interconnected nodes and the optimization problem is solved collectively by the cluster of nodes.

Loosely speaking, one can use any network of nodes to simulate a single powerful node, on which one can run any algorithm. The practical issue is that the time it takes to communicate between a processor and memory on the same node is normally many orders of magnitude smaller than the time needed for two nodes to communicate; similar conclusions hold for the energy required [89]. Further, in order to take advantage of parallel computing power on each node, it is necessary to subdivide the problem into subproblems suitable for independent/parallel computation.

State-of-the-art optimization algorithms are typically inherently sequential. Moreover, they usually rely on performing a large number of very fast iterations. The problem stems from the fact that if one needs to perform a round of communication after each iteration, practical performance drops down dramatically, as the round of communication is much more time-consuming than a single iteration of the algorithm.

These considerations have lead to the development of novel algorithms specialized for distributed optimization (we defer thorough review until Section 2). For now, we note that most of the results in literature work in the setting where the data is evenly distributed, and further suppose that $K \ll n/K$ where K is the number of nodes. This is indeed often close to reality when data is stored in a large data center. Additionally, an important subfield of the field of distributed learning relies on the assumption that each machine has a representative sample of the data available locally. That is, it is assumed that each machine has an IID sample from the underlying distribution. However, this assumption is often too strong; in fact, even in the data center paradigm this is often not the case since the data on a single node can be close to each other on a temporal scale, or clustered by its geographical origin. Since the patterns in the data can change over time, a feature might be present frequently on one node, while not appear on another at all.

The federated optimization setting describes a novel optimization scenario where none of the above assumptions hold. We outline this setting in more detail in the following section.

1.2 The Setting of Federated Optimization

The main purpose of this paper is to bring to the attention of the machine learning and optimization communities a new and increasingly practically relevant setting for distributed optimization, where none of the typical assumptions are satisfied, and communication efficiency is of utmost importance. In particular, algorithms for federated optimization must handle training data with the following characteristics:

- **Massively Distributed**: Data points are stored across a large number of nodes K. In particular, the number of nodes can be much bigger than the average number of training examples stored on a given node (n/K).

- **Non-IID**: Data on each node may be drawn from a different distribution; that is, the data points available locally are far from being a representative sample of the overall distribution.

- **Unbalanced**: Different nodes may vary by orders of magnitude in the number of training examples they hold.

In this work, we are particularly concerned with **sparse** data, where some features occur on a small subset of nodes or data points only. Although this is not necessary characteristic of the setting of federated optimization, we will show that the sparsity structure can be used to develop an effective algorithm for federated optimization. Note that data arising in the largest machine learning problems being solved nowadays, ad click-through rate predictions, are extremely sparse.

We are particularly interested in the setting where training data lives on users' mobile devices (phones and tablets), and the data may be privacy sensitive. The data $\{x_i, y_i\}$ is generated through device usage, e.g., via interaction with apps. Examples include predicting the next word a user will type (language modelling for smarter keyboard apps), predicting which photos a user is most likely to share, or predicting which notifications are most important.

To train such models using traditional distributed learning algorithms, one would collect the training examples in a centralized location (data center) where it could be shuffled and distributed evenly over proprietary compute nodes. In this paper we propose and study an alternative model: the training examples are not sent to a centralized location, potentially saving significant network bandwidth and providing additional privacy protection. In exchange, users allow some use of their devices' computing power, which shall be used to train the model.

In the communication model of this paper, in each round we send an update $\delta \in \mathbb{R}^d$ to a centralized server, where d is the dimension of the model being computed/improved. The update δ could be a gradient vector, for example. While it is certainly possible that in some applications the δ may encode some private information of the user, it is likely much less sensitive (and orders of magnitude smaller) than the original data itself. For example, consider the case where the raw training data is a large collection of video files on a mobile device. The size of the update δ will be *independent* of the size of this local training data corpus. We show that a global model can be trained using a small number of communication rounds, and so this also reduces the network bandwidth needed for training by orders of magnitude compared to copying the data to the datacenter.

Further, informally, we choose δ to be the minimum piece of information necessary to improve the global model; its utility for other uses is significantly reduced compared to the original data. Thus, it is natural to design a system that does not store these δ's longer than necessary to update the model, again increasing privacy and reducing liability on the part of the centralized model trainer. This setting, in which a single vector $\delta \in \mathbb{R}^d$ is communicated in each round, covers most existing first-order methods, including dual methods such as CoCoA+ [57].

Communication constraints arise naturally in the massively distributed setting, as network connectivity may be limited (e.g., we may wish to deffer all communication until the mobile device is charging and connected to a wi-fi network). Thus, in realistic scenarios we may be limited to only a single round of communication per day. This implies that, within reasonable bounds, we have access to essentially unlimited local computational power. Consequently, the practical objective is solely to minimize the number of communication rounds.

The main purpose of this work is initiate research into, and design a first practical implementation of federated optimization. Our results suggest that with suitable optimization algorithms, very little is lost by not having an IID sample of the data available, and that even in the presence of a large number of nodes, we can still achieve convergence in relatively few rounds of communication.

2 Related Work

In this section we provide a detailed overview of the relevant literature. We particularly focus on algorithms that can be used to solve problem (1) in various contexts. First, in Sections 2.1 and 2.2 we look at algorithms designed to be run on a single computer. In Section 2.3 we follow with a discussion of the distributed setting, where no single node has direct access to all data describing f. We describe a paradigm for measuring the efficiency of distributed methods, followed by overview of existing methods and commentary on whether they were designed with communication efficiency in mind or not.

2.1 Baseline Algorithms

In this section we shall describe several fundamental baseline algorithms which can be used to solve problems of the form (1).

Gradient Descent. A trivial benchmark for solving problems of structure (1) is *Gradient Descent* (GD) in the case when functions f_i are smooth (or Subgradient Descent for non-smooth functions) [69]. The GD algorithm performs the iteration

$$w^{t+1} = w^t - h_t \nabla f(w^t),$$

where $h_t > 0$ is a stepsize parameter. As we mentioned earlier, the number of functions, or equivalently, the number of training data pairs, n, is typically very large. This makes GD impractical, as it needs to process the whole dataset in order to evaluate a single gradient and update the model.

Gradient descent can be substantially accelerated, in theory and practice, via the addition of a momentum term. Acceleration ideas for gradient methods in convex optimization can be traced back to the work of Polyak [73] and Nesterov [68, 69]. While accelerated GD methods have a substantially better convergence rate, in each iteration they still need to do at least one pass over all data. As a result, they are not practical for problems where n very large.

Stochastic Gradient Descent. At present a basic, albeit in practice extremely popular, alternative to GD is *Stochastic Gradient Descent* (SGD), dating back to the seminal work of Robbins and Monro [82]. In the context of (1), SGD samples a random function (i.e., a random data-label pair) $i_t \in \{1, 2, \dots, n\}$ in iteration t, and performs the update

$$w^{t+1} = w^t - h_t \nabla f_{i_t}(w^t),$$

where $h_t > 0$ is a stepsize parameter. Intuitively speaking, this method works because if i_t is sampled uniformly at random from indices 1 to n, the update direction is an unbiased estimate of the gradient — $\mathbb{E}[\nabla f_{i_t}(w)] = \nabla f(w)$. However, noise introduced by sampling slows down the convergence, and a diminsihing sequence of stepsizes h_k is necessary for convergence. For a theoretical analysis for convex functions we refer the reader to [66, 64, 65] and [87, 93] for SVM problems. In a recent review [12], the authors outline further research directions. For a more practically-focused discussion, see [11]. In the context of neural networks, computation of stochastic gradients is referred to as *backpropagation* [49]. Instead of specifying the functions f_i and its gradients explicitly, backpropagation is a general way of computing the gradient. Performance of several competitive algorithms for training deep neural networks has been compared in [70].

One common trick that has been practically observed to provide superior performance, is to replace random sampling in each iteration by going through all the functions in a random order. This ordering is replaced by another random order after each such cycle [10]. Theoretical understanding of this phenomenon had been a long standing open problem, understood recently in [40].

The core differences between GD and SGD can be summarized as follows. GD has a fast convergence rate, but each iteration in the context of (1) is potentially very slow, as it needs to process the entire dataset in each iteration. On the other hand, SGD has slower convergence rate, but each iteration is fast, as the work needed is independent of number of data points n. For the problem structure of (1), SGD is usually better, as for practical purposes relatively low accuracy is required, which SGD can in extreme cases achieve after single pass through data, while GD would make just a single update. However, if a high accuracy was needed, GD or its faster variants would prevail.

2.2 A Novel Breed of Randomized Algorithms

Recent years have seen an explosion of new randomized methods which, in a first approximation, combine the benefits of cheap iterations of SGD with fast convergence of GD. Most of these methods can be said to belong to one of two classes — dual methods of the randomized coordinate descent variety, and primal methods of the stochastic gradient descent with variance reduction variety.

Randomized Coordinate Descent. Although the idea of coordinate descent has been around for several decades in various contexts (and for quadratic functions dates back even much further, to works on the Gauss-Seidel methods), it came to prominence in machine learning and optimization with the work of Nesterov [67] which equipped the method with a randomization strategy. Nesterov's work on *Randomized Coordinate Descent* (RCD) popularized the method and demonstrated that randomization can be very useful for problems of structure (1).

The RCD algorithm in each iteration chooses a random coordinate $j_t \in \{1, \dots, d\}$ and performs the update

$$w^{t+1} = w^t - h_{j_t} \nabla_{j_t} f(w^t) e_{j_t},$$

where $h_{j_t} > 0$ is a stepsize parameter, $\nabla_j f(w)$ denotes the j^{th} partial derivative of function f, and e_j is the j^{th} unit standard basis vector in $\mathbb{R}^d$. For the case of generalized linear models, when the data exhibits certain sparsity structure, it is possible to evaluate the partial derivative $\nabla_j f(w)$ efficiently, i.e., without need to process the entire dataset, leading to a practically efficient algorithm, see for instance [79, Section 6].

Numerous follow-up works extended the concept to proximal setting [79], single processor parallelism [15, 80] and develop efficiently implementable acceleration [51]. All of these three properties were connected in a single algorithm in [35], to which we refer the reader for a review of the early developments in the area of RCD, particularly to overview in Table 1 therein.

Stochastic Dual Coordinate Ascent. When an explicit strongly convex, but not necessarily smooth, regularizer is added to the average loss (1), it is possible to write down its (Fenchel) dual and the dual variables live in n-dimensional space. Applying RCD leads to an algorithm for solving (1) known under the name *Stochastic Dual Coordinate Ascent* [88]. This method has gained broad popularity with practicioners, likely due to the fact that for a number of loss functions, the method comes without the need to tune any hyper-parameters. The work [88] was first to show that by applying RCD [79] to the dual problem, one also solves the primal problem (1). For a theoretical and computational comparison of applying RCD to the primal versus the dual problems, see [21].

A directly primal-dual randomized coordinate descent method called Quartz, was developed in [75]. It has been recently shown in SDNA [74] that incorporating curvature information contained in random low dimensional subspaces spanned by a few coordinates can sometimes lead to dramatic speedups. Recent works [86, 20] interpret the SDCA method in primal-only setting, shedding light onto why this method works as a SGD method with a version of variance reduction property.

We now move the the second class of novel randomized algorithms which can be generally interpreted as variants of SGD, with an attempt to reduce variance inherent in the process of gradient estimation.

Stochastic Average Gradient. The first notable algorithm from this class is the *Stochastic Average Gradient* (SAG) [83, 85]. The SAG algorithm stores an average of n gradients of functions f_i evalueated at different points in the history of the algorithm. In each iteration, the algotithm, updates randomly chosen gradient out of this average, and makes a step in the direction of the average. This way, complexity of each iteration is independent of n, and the algorithm enjoys a fast convergence. The drawback of this algorithm is that it needs to store n gradients in memory because of the update operation. In the case of generalized linear models, this memory requirement can be reduced to the need of n scalars, as the gradient is a scalar multiple of the data point. This methods has been recently extended for use in Conditional Random Fields [84]. Nevertheless, the memory requirement makes the algorithm infeasible for application even in relatively small neural networks.

A followup algorithm SAGA [26] and its simplification [25], modifies the SAG algorithm to achieve unbiased estimate of the gradients. The memory requirement is still present, but the method significantly simplifies theoretical analysis, and yields a slightly stronger convergence guarantee.

Stochastic Variance Reduced Gradient. Another algorithm from the SGD class of methods is *Stochastic Variance Reduced Gradient*[1] (SVRG) [43] and [47, 100, 44]. The SVRG algorithm runs in two nested loops. In the outer loop, it computes full gradient of the whole function, $\nabla f(w^t)$, the expensive operation one tries to avoid in general. In the inner loop, the update step is iteratively computed as

$$w = w - h[\nabla f_i(w) - \nabla f_i(w^t) + \nabla f(w^t)].$$

The core idea is that the stochastic gradients are used to estimate the change of the gradient between point w^t and w, as opposed to estimating the gradient directly. We return to more detailed description of this algorithm in Section 3.2.

The SVRG has the advantage that it does not have the additional memory requirements of SAG/SAGA, but it needs to process the whole dataset every now and then. Indeed, comparing to SGD, which typically makes significant progress in the first pass through data, SVRG does not make any update whatsoever, as it needs to compute the full gradient. This and several other practical issues have been recently addressed in [41], making the algorithm competitive with SGD early on, and superior in later iterations. Although there is nothing that prevents one from applying SVRG and its variants in deep learning, we are not aware of any systematic assessment of

[1]The same algorithm was simultaneously introduced as Semi-Stochastic Gradient Descent (S2GD) [47]. Since the former work gained more attention, we will for clarity use the name SVRG throughout this paper.

its performance in this setting. Vanilla experiments in [43, 77] suggest that SVRG matches basic SGD, and even outperforms in the sense that variance of the iterates seems to be significantly smaller for SVRG. However, in order to draw any meaningful conclusions, one would need to perform extensive experiments and compare with state-of-the-art methods usually equipped with numerous heuristics.

There already exist attempts at combining SVRG type algorithms with randomized coordinate descent [46, 97]. Although these works highlight some interesting theoretical properties, the algorithms do not seem to be practical at the moment; more work is needed in this area. The first attempt to unify algorithms such as SVRG and SAG/SAGA already appeared in the SAGA paper [26], where the authors interpret SAGA as a midpoint between SAG and SVRG. Recent work [76] presents a general algorithm, which recovers SVRG, SAGA, SAG and GD as special cases, and obtains an asynchronous variant of these algorithms as a byproduct of the formulation. SVRG can be equipped with momentum (and negative momentum), leading to a new accelerated SVRG method known as Katyusha [3]. SVRG can be further accelerated via a raw clustering mechanism [4].

Stochastic Quasi-Newton Methods. A third class of new algorithms are the *Stochastic quasi-Newton* methods [16, 9]. These algorithms in general try to mimic the limited memory BFGS method (L-BFGS) [54], but model the local curvature information using inexact gradients — coming from the SGD procedure. A recent attempt at combining these methods with SVRG can be found in [63]. In [38], the authors utilize recent progress in the area of stochastic matrix inversion [39] revealing new connections with quasi-Newton methods, and devise a new stochastic limited memory BFGS method working in tandem with SVRG. The fact that the theoretical understanding of this branch of research is the least understood and having several details making the implementation more difficult compared to the methods above may limit its wider use. However, this approach could be most promising for deep learning once understood better.

One important aspect of machine learning is that the Empirical Risk Minimization problem (1) we are solving is just a proxy for the Expected Risk we are ultimately interested in. When one can find exact minimum of the empirical risk, everything reduces to balancing approximation–estimation tradeoff that is the object of abundant literature — see for instance [96]. An assessment of asymptotic performance of some optimization algorithms as *learning* algorithms in large-scale learning problems[2] has been introduced in [13]. Recent extension in [41] has shown that the variance reduced algorithms (SAG, SVRG, . . .) can in certain setting be better *learning* algorithms than SGD, not just better optimization algorithms.

Further Remarks. A general method, referred to as Universal Catalyst [53, 37], effectively enables conversion of a number of the algorithms mentioned in the previous sections to their 'accelerated' variants. The resulting convergence guarantees nearly match lower bounds in a number of cases. However, the need to tune additional parameter makes the method rather impractical.

Recently, lower and upper bounds for complexity of stochastic methods on problems of the form (1) were recently obtained in [99].

[2]See [13, Section 2.3] for their definition of large scale learning problem.

2.3 Distributed Setting

In this section we review the literature concerning algorithms for solving (1) in the distributed setting. When we speak about distributed setting, we refer to the case when the data describing the functions f_i are not stored on any single storage device. This can include setting where one's data just don't fit into a single RAM/computer/node, but two is enough. This also covers the case where data are distributed across several datacenters around the world, and across many nodes in those datacenters. The point is that in the system, there is no single processing unit that would have direct access to all the data. Thus, the distributed setting does not include single processor parallelism[3]. Compared with local computation on any single node, the cost of communication between nodes is much higher both in terms of speed and energy consumption [6, 89], introducing new computational challenges, not only for optimization procedures.

We first reveiew a theoretical decision rule for determining the practically best algorithm for a given problem in Section 2.3.1, followed by overview of distributed algorithms in Section 2.3.2, and communication efficient algorithms in Section 2.3.3. The following paradigm highlights why the class of communication efficient algorithms are not only preferable choice in the trivial sense. The communication efficient algorithms provide us with much more flexible tools for designing overall optimization procedure, which can make the algorithms inherently adaptive to differences in computing resources and architectures.

2.3.1 A Paradigm for Measuring Distributed Optimization Efficiency

This section reviews a paradigm for comparing efficency of distributed algorithms. Let us suppose we have many algorithms $\mathcal{A}$ readily available to solve the problem (1). The question is: "How do we decide which algorithm is the best for our purpose?" Initial version of this reasoning already appeared in [57], and applies also to [78].

First, consider the basic setting on a single machine. Let us define $\mathcal{I}_{\mathcal{A}}(\epsilon)$ as the number of iterations algorithm $\mathcal{A}$ needs to converge to some fixed ϵ accuracy. Let $\mathcal{T}_{\mathcal{A}}$ be the time needed for a single iteration. Then, in practice, the best algorithm is one that minimizes the following quantity.[4]

$$\text{TIME} = \mathcal{I}_{\mathcal{A}}(\epsilon) \times \mathcal{T}_{\mathcal{A}}. \tag{2}$$

The number of iterations $\mathcal{I}_{\mathcal{A}}(\epsilon)$ is usually given by theoretical guarantees or observed from experience. The $\mathcal{T}_{\mathcal{A}}$ can be empirically observed, or one can have idea of how the time needed per iteration varies between different algorithms in question. The main point of this simplified setting is to highlight key issue with extending algorithms to the distributed setting.

The natural extension to distributed setting is the following. Let c be time needed for communication during a single iteration of the algorithm $\mathcal{A}$. For sake of clarity, we suppose we consider only algorithms that need to communicate a single vector in $\mathbb{R}^d$ per round of communication. Note that essentially all first-order algorithms fall into this category, so this is not a restrictive assumption, which effectively sets c to be a constant, given any particular distributed architecture one has at disposal.

$$\text{TIME} = \mathcal{I}_{\mathcal{A}}(\epsilon) \times (c + \mathcal{T}_{\mathcal{A}}) \tag{3}$$

[3]It should be noted that some of the works presented in this section were originally presented as parallel algorithms. We include them anyway as many of the general ideas in carry over to the distributed setting.

[4]Considering only algorithms that can be run on a given machine.

The communication cost c does not only consist of actual exchange of the data, but also many other things like setting up and closing a connection between nodes. Consequently, even if we need to communicate very small amount of information, c always remains above a nontrivial threshold.

Most, if not all, of the current state-of-the-art algorithms that are the best in setting of (2), are stochastic and rely on doing very large number (big $\mathcal{I}_{\mathcal{A}}(\epsilon)$) of very fast (small $\mathcal{T}_{\mathcal{A}}$) iterations. As a result, even relatively small c can cause the practical performance of those algorithms drop down dramatically, because $c \gg \mathcal{T}_{\mathcal{A}}$.

This has been indeed observed in practice, and motivated development of new methods, designed with this fact in mind from scratch, which we review in Section 2.3.2. Although this is a good development for academia — motivation to explore new setting, it is not necessarily a good news for the industry.

Many companies have spent significant resources to build excellent algorithms to tackle their problems of form (1), fine tuned to the specific patterns arising in their data and side applications required. When the data companies collect grows too large to be processed on a single machine, it is understandable that they would be reluctant to throw away their fine tuned algorithms. This issue was first time explicitly addressed in CoCoA [57], which is rather framework than a algorithm, which works as follows (more detailed description follows in Section 2.3.3).

The CoCoA framework formulates a general way to form a specific subproblem on each node, based on data available locally and a single shared vector that needs to be distributed to all nodes. Within a iteration of the framework, each node uses *any* optimization algorithm $\mathcal{A}$, to reach a relative Θ accuracy on the local subproblem. Updates from all nodes are then aggregated to form an update to the global model.

The efficiency paradigm changes as follows:

$$\text{TIME} = \mathcal{I}(\epsilon, \Theta) \times (c + \mathcal{T}_{\mathcal{A}}(\Theta)) \tag{4}$$

The number of iterations $\mathcal{I}(\epsilon, \Theta)$ is independent of choice of the algorithm $\mathcal{A}$ used as a local solver, because there is theory predicting how many iterations of the CoCoA framework are needed to achieve ϵ accuracy, if we solve the local subproblems to relative Θ accuracy. Here, $\Theta = 0$ would mean we require the subproblem to be solved to optimality, and $\Theta = 1$ that we don't need any progress whatsoever. The general upper bound on number of iterations of the CoCoA framework is $\mathcal{I}(\epsilon, \Theta) = \frac{\mathcal{O}(\log(1/\epsilon))}{1-\Theta}$ [42, 58, 57] for strongly convex objectives. From the inverse dependence on $1 - \Theta$, we can see that there is a fundamental limit to the number of communication rounds needed. Hence, it will probably not be efficient to spend excessive resources to attain very high local accuracy (small Θ). Time per iteration $\mathcal{T}_{\mathcal{A}}(\Theta)$ denotes the time algorithm $\mathcal{A}$ needs to reach the relative Θ accuracy on the local subproblem.

This efficiency paradigm is more powerful for a number of reasons.

1. It allows practicioners to continue using their fine-tuned solvers, that can run only on single machine, instead of having to implement completely new algorithms from scratch.

2. The actual performance in terms of number of rounds of communication is independent from the choice of optimization algorithm, making it much easier to optimize the overall performance.

3. Since the constant c is architecture dependent, running optimal algorithm on one node network does not have to be optimal on another. In the setting (3), this could mean moving from

one cluster to another, a completely different algorithm is optimal, which is a major change. In the setting (4), this can be improved by simply changing Θ, which is typically implicitly determined by number of iterations algorithm $\mathcal{A}$ runs for.

In this work we propose a different way to formulate the local subproblems, which does not rely on duality as in the case of CoCoA. We also highlight that some algorithms seem to be particularly suitable to solve those local subproblems, effectively leading to novel algorithms for distributed optimization.

2.3.2 Distributed Algorithms

As discussed below in Section 2.3.1, this setting creates unique challenges. Distributed optimization algorithms typically require a small number (1–4) of communication rounds per iteration. By communication round we typically understand a single MapReduce operation [24], implemented efficiently for iterative procedures [36], such as optimization algorithms. Spark [102] has been established as a popular open source framework for implementing distributed iterative algorithms, and includes several of the algorithms mentioned in this section.

Optimization in distributed setting has been studied for decades, tracing back to at least works of Bertsekas and Tsitsiklis [8, 7, 95]. Recent decade has seen an explosion of interest in this area, greatly motivated by rapid increase of data availability in machine learning applications.

Much of the recent effort was focused on creating new optimization algorithms, by building variants of popular algorithms suitable for running on a single processor (See Section 2.1). A relatively common feature of many of these efforts is a) The computation overhead in the case of synchronous algorithms, and b) The difficulty of analysing asynchronous algorithms without restrictive assumptions. By computation overhead we mean that if optimization program runs in a compute-communicate-update cycle, the update part cannot start until all nodes finish their computation. This causes some of the nodes be idle, while remaining nodes finish their part of computation, clearly an inefficient use of computational resources. This pattern often diminishes or completely reverts potential speed-ups from distributed computation. In the asynchronous setting in general, an update can be applied to a parameter vector, followed by computation done based on a now-outdated version of that parameter vector. Formally grasping this pattern, while keeping the setting realistic is often quite challenging. Consequently, this is very open area, and optimal choice of algorithm in any particular case is often heavily dependent on the problem size, details in its structure, computing architecture available, and above all, expertise of the practitioner.

This general issue is best exhibited with numerous attempts at parallelizing the Stochastic Gradient Descent and its variants. As an example, [27, 29] provide theoretically linear speedup with number of nodes, but are difficult to implement efficiently, as the nodes need to synchronize frequently in order to compute reasonable gradient averages. As an alternative, no synchronization between workers is assumed in [71, 2, 31]. Consequently, each worker reads w^t from memory, parameter vector w at time point t, computes a stochastic gradient $\nabla f_i(w^t)$ and applies it to already changed state of the parameter vector $w^{t+\tau}$. The above mentioned methods assume that the delay τ is bounded by a constant, which is not necessarily realistic assumption[5]. Some of the

[5]A bound on the delay τ can be deterministic or probabilistic. However, in practice, the delays are mostly about the number of nodes in the network, and there rare very long delays, when a variety of operating system-related events can temporarily postpone computation of a single node. To the best of our knowledge, no formal assumptions reflect this setting well. In fact, two recent works [60, 48] highlight subtle but important issue with labelling of iterates in the presence of asynchrony, rendering most of the existing analyses of asynchronous optimization algorithms incorrect.

works also introduce assumptions on the sparsity structures or conditioning of the Hessian of f. Asymptotically optimal convergent rates were proven in [30] with considerably milder assumptions. Improved analysis of asynchronous SGD was also presented in [22], simultaneously with a version that uses lower-precision arithmetic was introduced without sacrificing performance, which is a trend that might find use in other parts of machine learning in the following years.

The negative effect of asynchronous distributed implementations of SGD seem to be negligible, when applied to the task of training very large deep networks — which is the ultimate industrial application of today. The practical usefulness has been demonstrated for instance by Google's Downpour SGD [23] and Microsoft's Project Adam [18].

The first distributed versions of Coordinate Descent algorithms were the Hydra and its accelerated variant, Hydra^2, [81, 34], which has been demonstrated to be very efficient on large sparse problems implemented on a computing cluster. An extended version with description of implementation details is presented in [61]. Effect of asynchrony has been explored and partially theoretically understood in the works of [56, 55]. Another asynchronous, rather framework than an algorithm, for coordinate updates, applicable to wider class of objectives is presented in [72].

The data are assumed to be partitioned to nodes by features/coordinates in the above algorithms. This setting can be restrictive if one is not able to distribute the data beforehand, but instead the data are distributed "as is" — in which case the data are most commonly distributed by data points. This does not need to be an issue, if a dual version of coordinate descent is used — in which the distribution is done by data points [94] followed by works on Communication Efficient Dual Cooridante Ascent, described in next section. The use of duality however requires usage of additional explicit strongly convex regularization term, hence can be used to solve smaller class of problems. Despite the apparent practical disadvantages, variants of distributed coordinate descent algorithms are among the most widely used methods in practice.

Moving to variance reduced methods, distributed versions of SAG/SAGA algorithms have not been proposed yet. However, several distributed versions of the SVRG algorithm already exist. A scheme for replicating data to simulate iid sampling in distributed environment was proposed in [50]. Although the performance is well analysed, the setting requires significantly stronger control of data distribution which is rarely practicaly feasible. A relatively similar method to Algorithm 3 presented here has been proposed in [78], which was analysed, and in [59], a largely experimental work that can be also cast as communication efficient — described in detail in Section 2.3.3.

Another class of algorithms relevant for this work is Alternating Direction Method of Multipliers (ADMM) [14, 28]. These algorithms are in general applicable to much broader class of problems, and hasn't been observed to perform better than other algorithms presented in this section, in the machine learning setting of (1).

2.3.3 Communication-Efficient Algorithms

In this Section, we describe algorithms that can be cast as "communication efficient". The common theme of the algorithms presented here, is that in order to perform better in the sense of (3), one should design algorithms with high $\mathcal{T}_{\mathcal{A}}$, in order to make the cost of communcation c negligible.

Before moving onto specific methods, it is worth the noting some of the core limits concerning the problem we are trying to solve in distributed setting. Fundamental limitations of stochastic versions of the problem (1) in terms of runtime, communication costs and number of samples used are studied in [90]. Efficient algorithms and lower bounds for distributed statistical estimation are established in [104, 103].

However, these works do not fit into our framework, because they assume that each node has access to data generated IID from a single distribution. In the case of [104, 103] also $K \ll n/K$, that the number of nodes K is much smaller than the number of data point on each node is also assumed. As we stress in the Introduction, these assumptions are far from being satisfied in our setting. Intuitively, relaxing these assumptions should make the problem harder. However, it is not as straightforward to conclude this, as there are certainly particular non-iid data distributions that simplify the problem — for instance if data are distributed according to separability structure of the objective. Lower bounds on communication complexity of distributed convex optimization of (1) are presented in [5], concluding that for IID data distributions, existing algorithms already achieve optimal complexity in specific settings.

Probably first, rather extreme, work [107] proposed to parallelize SGD in a single round of communication. Each node simply runs SGD on the data available locally, and their outputs are averaged to form a final result. This approach is however not very robust to differences in data distributions available locally, and it has been shown [91, Appendix A] that in general it cannot perform better than using output of a single machine, ignoring all the other data.

Shamir et al. proposed the DANE algorithm, Distributed Approximate Newton [91], to exactly solve a general subproblem available locally, before averaging their solutions. The method relies on similarity of Hessians of local objectives, representing their iterations as an average of inexact Newton steps. We describe the algorithm in greater detail in Section 3.4 as our proposed work builds on it. A quite similar approach was proposed in [59], with richer class class of subproblems that can be formulated locally, and solved approximately. An analysis of inexact version of DANE and its accelerated variant, AIDE, appeared recently in [78]. Inexact DANE is closely related to the algorithms presented in this paper. We, however, continue in different direction shaped by the setting of federated optimization.

The DiSCO algorithm [105] of Zhang and Xiao is based on inexact damped Newton method. The core idea is that the inexact Newton steps are computed by distributed preconditioned conjugate gradient, which can be very fast, if the data are distributed in an IID fashion, enabling a good preconditioner to be computed locally. The theoretical upper bound on number of rounds of communication improves upon DANE and other methods, and in certain settings matches the lower bound presented in [5]. The DiSCO algorithm is related to [52, 106], a distributed truncated Newton method. Although it was reported to perform well in practice, the total number of conjugate gradient iterations may still be high to be considered a communication efficient algorithm.

Common to the above algorithms is the assumption that each node has access to data points sampled IID from the same distribution. This assumption is not required only in theory, but can cause the algorithms to converge significantly slower or even diverge (as reported for instance in [91, Table 3]). Thus, these algorithms, at least in their default form, are not suitable for the setting of Federated Optimization presented here.

An algorithm that bypasses the need for IID data assumption is CoCoA, which provably converges under any distribution of the data, while the convergence rate does depend on properties of the data distribution. The first version of the algorithm was proposed as DisDCA in [101], without convergence guarantees. First analysis was introduced in [42], with further improvements in [58], and a more general version in [57]. Recently, its variant for L1-regularized objectives was introduced in [92].

The CoCoA framework formulates general local subproblems based on the dual form of (1) (See for instance [57, Eq. (2)]). Data points are distributed to nodes, along with corresponding dual variables. Arbitrary optimization algorithm is used to attain a relative Θ accuracy on the local subproblem — by changing only local dual variables. These updates have their corresponding updates to primal variable w, which are synchronously aggregated (could be averaging, adding up, or anything in between; depending on the local subproblem formulation).

From the description in this section it appears that the CoCoA framework is the only usable tool for the setting of Federated Optimization. However, the theoretical bound on number of rounds of communications for ill-conditioned problems scales with the number of nodes K. Indeed, as we will show in Section 4 on real data, CoCoA framework does converge very slowly.

3 Algorithms for Federated Optimization

In this section we introduce the first algorithm that was designed with the unique challenges of federated optimization in mind. Before proceeding with the explanation, we first revisit two important and at first sight unrelated algorithms. The connection between these algorithms helped to motivate our research. Namely, the algorithms are the Stochastic Variance Reduced Gradient (SVRG) [43, 47], a stochastic method with explicit variance reduction, and the Distributed Approximate Newton (DANE) [91] for distributed optimization.

The descriptions are followed by their connection, giving rise to a new distributed optimization algorithm, at first sight almost identical to the SVRG algorithm, which we call Federated SVRG (FSVRG).

Although this algorithm seems to work well in practice in simple circumstances, its performance is still unsatisfactory in the general setting we specify in Section 3.3. We proceed by making the FSVRG algorithm adaptive to different local data sizes, general sparsity patterns and significant differences in patterns in data available locally, and those present in the entire data set.

3.1 Desirable Algorithmic Properties

It is a useful thought experiment to consider the properties one would hope to find in an algorithm for the non-IID, unbalanced, and massively-distributed setting we consider. In particular:

(A) If the algorithm is initialized to the optimal solution, it stays there.

(B) If all the data is on a single node, the algorithm should converge in $\mathcal{O}(1)$ rounds of communication.

(C) If each feature occurs on a single node, so the problems are fully decomposable (each machine is essentially learning a disjoint block of parameters), then the algorithm should converge in $\mathcal{O}(1)$ rounds of communication[6].

(D) If each node contains an identical dataset, then the algorithm should converge in $\mathcal{O}(1)$ rounds of communication.

For convex problems, “converges” has the usual technical meaning of finding a solution sufficiently close to the global minimum, but these properties also make sense for non-convex problems where “converge” can be read as “finds a solution of sufficient quality”. In these statements, $\mathcal{O}(1)$ round is ideally exactly one round of communication.

[6]This is valid only for generalized linear models.

Property (A) is valuable in any optimization setting. Properties (B) and (C) are extreme cases of the federated optimization setting (non-IID, unbalanced, and sparse), whereas (D) is an extreme case of the classic distributed optimization setting (large amounts of IID data per machine). Thus, (D) is the least important property for algorithms in the federated optimization setting.

3.2 SVRG

The SVRG algorithm [43, 47] is a stochastic method designed to solve problem (1) on a single node. We present it as Algorithm 1 in a slightly simplified form.

Algorithm 1 SVRG

```
 1: parameters: $m$ = number of stochastic steps per epoch, $h$ = stepsize
 2: for $s = 0, 1, 2, \dots$ do
 3:     Compute and store $\nabla f(w^t) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(w^t)$        ▷ Full pass through data
 4:     Set $w = w^t$
 5:     for $t = 1$ to $m$ do
 6:         Pick $i \in \{1, 2, \dots, n\}$, uniformly at random
 7:         $w = w - h\left(\nabla f_i(w) - \nabla f_i(w^t) + \nabla f(w^t)\right)$        ▷ Stochastic update
 8:     end for
 9:     $w^{t+1} = w$
10: end for
```

The algorithm runs in two nested loops. In the outer loop, it computes gradient of the entire function f (Line 3). This constitutes for a full pass through data — in general expensive operation one tries to avoid unless necessary. This is followed by an inner loop, where m fast stochastic updates are performed. In practice, m is typically set to be a small multiple (1–5) of n. Although the theoretically optimal choice for m is a small multiple of a condition number [47, Theorem 6], this is often of the same order as n in practice.

The central idea of the algorithm is to avoid using the stochastic gradients to estimate the entire gradient $\nabla f(w)$ directly. Instead, in the stochastic update in Line 7, the algorithm evaluates two stochastic gradients, $\nabla f_i(w)$ and $\nabla f_i(w^t)$. These gradients are used to estimate the change of the gradient of the entire function between points w^t and w, namely $\nabla f(w) - \nabla f(w^t)$. Using this estimate together with $\nabla f(w^t)$ pre-computed in the outer loop, yields an unbiased estimate of $\nabla f(w)$.

Apart from being an unbiased estimate, it could be intuitively clear that if w and w^t are close to each other, the variance of the estimate $\nabla f_i(w) - \nabla f_i(w^t)$ should be small, resulting in estimate of $\nabla f(w)$ with small variance. As the inner iterate w goes further, variance grows, and the algorithm starts a new outer loop to compute new full gradient $\nabla f(w^{t+1})$ and reset the variance.

The performance is well understood in theory. For λ-strongly convex f and L-smooth functions f_i, convergence results are in the form

$$\mathbb{E}[f(w^t) - f(w^*)] \le c^t [f(w^0) - f(w^*)], \tag{5}$$

where w^* is the optimal solution, and $c = \Theta\left(\frac{1}{mh}\right) + \Theta(h)$.[7]

[7]See [47, Theorem 4] and [43, Theorem 1] for details.

It is possible to show [47, Theorem 6] that for appropriate choice of parameters m and h, the convergence rate (5) translates to the need of

$$(n + \mathcal{O}(L/\lambda))\log(1/\epsilon)$$

evaluations of ∇f_i for some i to achieve $\mathbb{E}[f(w) - f(w^*)] < \epsilon$.

3.3 Distributed Problem Formulation

In this section, we introduce notation and specify the structure of the distributed version of the problem we consider (1), focusing on the case where the f_i are convex. We assume the data $\{x_i, y_i\}_{i=1}^n$, describing functions f_i are stored across a large number of nodes.

Let K be the number of nodes. Let $\mathcal{P}_k$ for $k \in \{1, \dots, K\}$ denote a partition of data point indices $\{1, \dots, n\}$, so $\mathcal{P}_k$ is the set stored on node k, and define $n_k = |\mathcal{P}_k|$. That is, we assume that $\mathcal{P}_k \cap \mathcal{P}_l = \emptyset$ whenever $k \neq l$, and $\sum_{k=1}^K n_k = n$. We then define local empirical loss as

$$F_k(w) \overset{\text{def}}{=} \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w), \tag{6}$$

which is the local objective based on the data stored on machine k. We can then rephrase the objective (1) as

$$f(w) = \sum_{k=1}^K \frac{n_k}{n} F_k(w) = \sum_{k=1}^K \frac{n_k}{n} \cdot \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w). \tag{7}$$

The way to interpret this structure is to see the empirical loss $f(w) = \frac{1}{n}\sum_{i=1}^n f_i(w)$ as a convex combination of the local empirical losses $F_k(w)$, available locally to node k. Problem (1) then takes the simplified form

$$\min_{w \in \mathbb{R}^d} f(w) \equiv \sum_{k=1}^K \frac{n_k}{n} F_k(w). \tag{8}$$

3.4 DANE

In this section, we introduce a general reasoning providing stronger intuitive support for the DANE algorithm [91], which we describe in detail below. We will follow up on this reasoning in Appendix A and draw a connection between two existing methods that was not known in the literature.

If we wanted to design a distributed algorithm for solving the above problem (8), where node k contains the data describing function F_k. The first, and as we shall see, a rather naive idea is to ask each node to minimize their local functions, and average the results (a variant of this idea appeared in [107]):

$$w_k^{t+1} = \arg\min_{w \in \mathbb{R}^d} F_k(w), \qquad w^{t+1} = \sum_{k=1}^K \frac{n_k}{n} w_k^{t+1}.$$

Clearly, it does not make sense to run this algorithm for more than one iteration as the output w will always be the same. This is simply because w_k^{t+1} does not depend on t. In other words, this method effectively performs just a single round of communication. While the simplicity is appealing, the drawback of this method is that it can't work. Indeed, there is no reason to expect that in general the solution of (8) will be a weighted average of the local solutions, unless the local functions are all the same — in which case we do not need a distributed algorithm in the first place and can instead solve the much simpler problem $\min_{w\in\mathbb{R}^d} F_1(w)$. This intuitive reasoning can be also formally supported, see for instance [91, Appendix A].

One remedy to the above issue is to modify the local problems before each aggregation step. One of the simplest strategies would be to perturb the local function F_k in iteration t by a quadratic term of the form: $-(a_k^t)^T w + \frac{\mu}{2}\|w - w^t\|^2$ and to ask each node to solve the perturbed problem instead. With this change, the improved method then takes the form

$$w_k^{t+1} = \arg\min_{w\in\mathbb{R}^d} F_k(w) - (a_k^t)^T w + \frac{\mu}{2}\|w - w^t\|^2, \qquad w^{t+1} = \frac{1}{K}\sum_{k=1}^{K} w_k^{t+1}. \tag{9}$$

The idea behind iterations of this form is the following. We would like each node $k \in [K]$ to use as much curvature information stored in F_k as possible. By keeping the function F_k in the subproblem in its entirety, we are keeping the curvature information nearly intact — the Hessian of the subproblem is $\nabla^2 F_k + \mu I$, and we can even choose $\mu = 0$.

As described, the method is not yet well defined, since we have not described how the vectors a_k^t would change from iteration to iteration, and how one should choose μ. In order to get some insight into how such a method might work, let us examine the optimality conditions. Asymptotically as $t \to \infty$, we would like a_k^t to be such that the minimum of each subproblem is equal to w^*; the minimizer of (8). Hence, we would wish for w^* to be the solution of

$$\nabla F_k(w) - a_k^t + \mu(w - w^t) = 0.$$

Hence, in the limit, we would ideally like to choose $a_k^t = \nabla F_k(w^*) + \mu(w^* - w^t) \approx \nabla F_k(w^*)$, since $w^* \approx w^t$. Not knowing w^* however, we cannot hope to be able to simply set a_k^t to this value. Hence, the second option is to come up with an update rule which would guarantee that a_k^t converges to $\nabla F_k(w^*)$ as $t \to \infty$. Notice at this point that it has been long known in the optimization community that the gradient of the objective at the optimal point is intimately related to the optimal solution of a dual problem. Here the situation is further complicated by the fact that we need to learn K such gradients. In the following, we show that DANE is in fact a particular instantiation of the scheme above.

DANE. We present the Distributed Approximate Newton algorithm (DANE) [91], as Algorithm 2. The algorithm was originally analysed for solving the problem of structure (7), with n_k being identical for each k — i.e., each computer has the same number of data points. Nothing prevents us from running it in our more general setting though.

As alluded to earlier, the main idea of DANE is to form a local subproblem, dependent only on local data, and gradient of the entire function — which can be computed in a single round of communication (Line 3). The subproblem is then solved exactly (Line 4), and updates from individual nodes are averaged to form a new iterate (Line 5). This approach allows any algorithm to be used to solve the local subproblem (10). As a result, it often achieves communication efficiency in the sense of requiring expensive local computation between rounds of communication, hopefully rendering the time needed for communication insignificant (see Section 2.3.1). Further, note that DANE belongs to the family of distributed method that operate via the quadratic perturbation trick (9) with

$$a_k^t = \nabla F_k(w^t) - \eta \nabla f(w^t).$$

Algorithm 2 Distributed Approximate Newton (DANE)

```
1: Input: regularizer $\mu \geq 0$, parameter $\eta$ (default: $\mu = 0, \eta = 1$)
2: for $s = 0, 1, 2, \dots$ do
3:     Compute $\nabla f(w^t) = \frac{1}{n}\sum_{i=1}^n \nabla f_i(w^t)$ and distribute to all machines
4:     For each node $k \in \{1, \dots, K\}$, solve
           $w_k = \arg\min_{w \in \mathbb{R}^d} \left\{ F_k(w) - \left(\nabla F_k(w^t) - \eta \nabla f(w^t)\right)^T w + \frac{\mu}{2}\|w - w^t\|^2 \right\}$   (10)
5:     Compute $w^{t+1} = \frac{1}{K}\sum_{k=1}^K w_k$
6: end for
```

If we assumed that the method works, i.e., that $w^t \to w^*$ and hence $\nabla f(w^t) \to \nabla f(w^*) = 0$, then $a_k^t \to \nabla F_k(w^*)$, which agrees with the earlier discussion.

In the default setting when $\mu = 0$ and $\eta = 1$, DANE achieves desirable property (D) (immediate convergence when all local datasets are identical), since in this case $\nabla F_k(w^t) - \eta \nabla f(w^t) = 0$, and so we exactly minimize $F_k(w) = f(w)$ on each machine. For any choice of μ and η, DANE also achieves property (A), since in this case $\nabla f(w^t) = 0$, and w^t is a minimizer of $F_k(w) - \nabla F_k(w^t) \cdot w$ as well as of the regularization term. Unfortunately, DANE does not achieve the more federated optimization-specific desirable properties (B) and (C).

The convergence analysis for DANE assumes that the functions are twice differentiable, and relies on the assumption that each node has access to IID samples from the same underlying distribution. This implies that that the Hessians of $\nabla^2 F_k(w)$ are similar to each other [91, Lemma 1]. In case of linear regression, with $\lambda = \mathcal{O}(1/\sqrt{n})$-strongly convex functions, the number of DANE iterations needed to achieve ϵ-accuracy is $\mathcal{O}(K \log(1/\epsilon))$. However, for general L-smooth loss, the theory is significantly worse, and does not match its practical performance.

The practical performance also depends on the additional local regularization parameter μ. For small number of nodes K, the algorithm converges quickly with $\mu = 0$. However, as reported [91, Figure 3], it can diverge quickly with growing K. Bigger μ makes the algorithm more stable at the cost of slower convergence. Practical choice of μ remains an open question.

3.5 SVRG meets DANE

As we mentioned above, the DANE algorithm can perform poorly in certain settings, even without the challenging aspects of federated optimization. Another point that is seen as drawback of DANE is the need to find the *exact* minimum of (10) — this can be feasible for quadratics with relatively small dimension, but infeasible or extremely expensive to achieve for other problems. We adapt the idea from the CoCoA algorithm [57], in which an arbitrary optimization algorithm is used to obtain relative Θ accuracy on a locally defined subproblem. We replace the exact optimization with an approximate solution obtained by using any optimization algorithm.

Considering all the algorithms one could use to solve (10), the SVRG algorithm seems to be a particularly good candidate. Starting the local optimization of (10) from point w^t, the algorithm automatically has access to the derivative at w^t, which is identical for each node — $\nabla f(w^t)$. Hence, the SVRG algorithm can skip the initial expensive operation, evaluation of the entire gradient (Line 3, Algorithm 1), and proceed only with the stochastic updates in the inner loop.

It turns out that this modified version of the DANE algorithm is equivalent to a distributed version of SVRG.

Proposition 1. *Consider the following two algorithms.*

1. *Run the DANE algorithm (Algorithm 2) with $\eta = 1$ and $\mu = 0$, and use SVRG (Algorithm 1) as a local solver for* (10)*, running it for a single iteration, initialized at point w^t.*

2. *Run a distributed variant of the SVRG algorithm, described in Algorithm 3.*

The algorithms are equivalent in the following sense. If both start from the same point w^t, they generate identical sequence of iterates $\{w^t\}$.

Proof. We construct the proof by showing that single step of the SVRG algorithm applied to the problem (10) on computer k is identical to the update on Line 8 in Algorithm 3.

The way to obtain a stochastic gradient of (10) is to sample one of the functions composing $F_k(w) = \frac{1}{n_k}\sum_{i\in\mathcal{P}_k} f_i(w)$, and add the linear term $\nabla F_k(w^t) - \eta f(w^t)$, which is known and does not need to be estimated. Upon sampling an index $i \in \mathcal{P}_k$, the update direction follows as

$$\left[\nabla f_i(w) - \nabla F_k(w^t) - f(w^t)\right] - \left[\nabla f_i(w^t) - \nabla F_k(w^t) - f(w^t)\right] + \nabla f(w^t) = \nabla f_i(w) - \nabla f_i(w^t) + \nabla f(w^t),$$

which is identical to the direction in Line 8 in Algorithm 3. The claim follows by chaining the identical updates to form identical iterate w^{t+1}. □

Algorithm 3 naive Federated SVRG (FSVRG)

```
 1: parameters: $m$ = # of stochastic steps per epoch, $h$ = stepsize, data partition $\{\mathcal{P}_k\}_{k=1}^K$
 2: for $s = 0, 1, 2, \dots$ do                                    ▷ Overall iterations
 3:     Compute $\nabla f(w^t) = \frac{1}{n}\sum_{i=1}^n \nabla f_i(w^t)$
 4:     for $k = 1$ to $K$ do in parallel over nodes $k$               ▷ Distributed loop
 5:         Initialize: $w_k = w^t$
 6:         for $t = 1$ to $m$ do                                     ▷ Actual update loop
 7:             Sample $i \in \mathcal{P}_k$ uniformly at random
 8:             $w_k = w_k - h\left(\nabla f_i(w_k) - \nabla f_i(w^t) + \nabla f(w^t)\right)$
 9:         end for
10:     end for
11:     $w^{t+1} = w^t + \frac{1}{K}\sum_{k=1}^K (w_k - w^t)$                       ▷ Aggregate
12: end for
```

Remark 2. *The algorithms considered in Proposition 1 are inherently stochastic. The statement of the proposition is valid under the assumption that in both cases, identical sequence of samples $i \in \mathcal{P}_k$ would be generated by all nodes $k \in \{1, 2, \dots, K\}$.*

Remark 3. *In the Proposition 1 we consider the DANE algorithm with particular values of η and μ. The Algorithm 3 and the Proposition can be easily gereralized, but we present only the default version for the sake of clarity.*

Since the first version of this paper, this connection has been mentioned in [78], which analyses an inexact version of the DANE algorithm. We proceed by adapting the above algorithm to other challenges arising in the context of federated optimization.

3.6 Federated SVRG

Empirically, the Algorithm 3 fits in the model of distributed optimization efficiency described in Section 2.3.1, since we can balance how many stochastic iterations should be performed locally against communication costs. However, several modifications are necessary to achieve good performance in the full federated optimization setting (Section 3.3). Very important aspect that needs to be addressed is that the number of data points available to a given node can differ greatly from the average number of data points available to any single node. Furthermore, this setting always comes with the data available locally being clustered around a specific pattern, and thus not being a representative sample of the overall distribution we are trying to learn. In the Experiments section we focus on the case of L2 regularized logistic regression, but the ideas carry over to other generalized linear prediction problems.

3.6.1 Notation

Note that in large scale generalized linear prediction problems, the data arising are almost always sparse, for example due to bag-of-words style feature representations. This means that only a small subset of d elements of vector x_i have nonzero values. In this class of problems, the gradient $\nabla f_i(w)$ is a multiple of the data vector x_i. This creates additional complications, but also potential for exploitation of the problem structure and thus faster algorithms. Before continuing, let us summarize and denote a number of quantities needed to describe the algorithm.

- n — number of data points / training examples / functions.
- $\mathcal{P}_k$ — set of indices, corresponding to data points stored on device k.
- $n_k = |\mathcal{P}_k|$ — number of data points stored on device k.
- $n^j = \left|\{i \in \{1, \dots, n\} : x_i^T e_j \neq 0\}\right|$ — the number of data points with nonzero j^{th} coordinate
- $n_k^j = \left|\{i \in \mathcal{P}_k : x_i^T e_j \neq 0\}\right|$ — the number of data points stored on node k with nonzero j^{th} coordinate
- $\phi^j = n^j/n$ — frequency of appearance of nonzero elements in j^{th} coordinate
- $\phi_k^j = n_k^j/n_k$ — frequency of appearance of nonzero elements in j^{th} coordinate on node k
- $s_k^j = \phi^j/\phi_k^j$ — ratio of global and local appearance frequencies on node k in j^{th} coordinate
- $S_k = \text{Diag}(s_k^j)$ — diagonal matrix, composed of s_k^j as j^{th} diagonal element
- $\omega^j = \left|\{\mathcal{P}_k : n_k^j \neq 0\}\right|$ — Number of nodes that contain data point with nonzero j^{th} coordinate
- $a^j = K/\omega^j$ — aggregation parameter for coordinate j
- $A = \text{Diag}(a_j)$ — diagonal matrix composed of a_j as j^{th} diagonal element

With these quantities defined, we can state our proposed algorithm as Algorithm 4. Our experiments show that this algorithm works very well in practice, but the motivation for the particular scaling of the updates may not be immediately clear. In the following section we provide the intuition that lead to the development of this algorithm.

3.6.2 Intuition Behind FSVRG Updates

The difference between the Algorithm 4 and Algorithm 3 is in the introduction of the following properties.

1. Local stepsize — $h_k = h/n_k$.

Algorithm 4 Federated SVRG (FSVRG)

```
1: parameters: $h$ = stepsize, data partition $\{\mathcal{P}_k\}_{k=1}^K$,
               diagonal matrices $A, S_k \in \mathbb{R}^{d \times d}$ for $k \in \{1, \dots, K\}$
2: for $s = 0, 1, 2, \dots$ do                                          ▷ Overall iterations
3:     Compute $\nabla f(w^t) = \frac{1}{n}\sum_{i=1}^n \nabla f_i(w^t)$
4:     for $k = 1$ to $K$ do in parallel over nodes $k$                ▷ Distributed loop
5:         Initialize: $w_k = w^t$ and $h_k = h/n_k$
6:         Let $\{i_t\}_{t=1}^{n_k}$ be random permutation of $\mathcal{P}_k$
7:         for $t = 1, \dots, n_k$ do                                    ▷ Actual update loop
8:             $w_k = w_k - h_k\left(S_k\left[\nabla f_{i_t}(w_k) - \nabla f_{i_t}(w^t)\right] + \nabla f(w^t)\right)$
9:         end for
10:    end for
11:    $w^t = w^t + A\sum_{k=1}^K \frac{n_k}{n}(w_k - w^t)$                  ▷ Aggregate
12: end for
```

2. Aggregation of updates proportional to partition sizes — $\frac{n_k}{n}(w_k - w^t)$
3. Scaling stochastic gradients by diagonal matrix — S_k
4. Per-coordinate scaling of aggregated updates — $A(w_k - w^t)$

Let us now explain what motivated us to get this particular implementation.

As a simplification, assume that at some point in time, we have for some w, $w_k = w$ for all $k \in [K]$. In other words, all the nodes have the same local iterate. Although this is not exactly the case in practice, thinking about the issue in this simplified setting will give us insight into what would be meaningful to do if it was true. Further, we can hope that the reality is not too far from the simplification and it will still work in practice. Indeed, all nodes do start from the same point, and adding the linear term $\nabla F_k(w^t) - \nabla f(w^t)$ to the local objective forces all nodes to move in the same direction, at least initially.

Suppose the nodes are about to make a single step synchronously. Denote the update direction on node k as $G_k = \nabla f_i(w) - \nabla f_i(w^t) + \nabla f(w^t)$, where i is sampled uniformly at random from $\mathcal{P}_k$.

If we had only one node, i.e., $K = 1$, it is clear that we would have $\mathbb{E}[G_1] = \nabla f(w^t)$. If K is more than 1, the values of G_k are in general biased estimates of $\nabla f(w^t)$. We would like to achieve the following: $\mathbb{E}\left[\sum_{k=1}^K \alpha_k G_k\right] = \nabla f(w^t)$, for some choice of α_k. This is motivated by the general desire to make stochastic first-order methods to make a gradient step in expectation.

We have

$$\mathbb{E}\left[\sum_{k=1}^K \alpha_k G_k\right] = \sum_{k=1}^K \alpha_k \frac{1}{n_k}\sum_{i \in \mathcal{P}_k}\left[\nabla f_i(w) - \nabla f_i(w^t) + \nabla f(w^t)\right].$$

By setting $\alpha_k = \frac{n_k}{n}$, we get

$$\mathbb{E}\left[\sum_{k=1}^K \alpha_k G_k\right] = \frac{1}{n}\sum_{k=1}^K \sum_{i \in \mathcal{P}_k}\left[\nabla f_i(w) - \nabla f_i(w^t) + \nabla f(w^t)\right] = \nabla f(w).$$

This motivates the aggregation of updates from nodes proportional to n_k, the number of data points available locally (Point 2).

Next, we realize that if the local data sizes, n_k, are not identical, we likely don't want to do the same number of local iterations on each node k. Intuitively, doing one pass through data (or a fixed number of passes) makes sense. As a result, the aggregation motivated above does not make perfect sense anymore. Nevertheless, we can even it out, by setting the stepsize h_k inversely proportional to n_k, making sure each node makes progress of roughly the same magnitude overall. Hence, $h_k = h/n_k$ (Point 1).

To motivate the Point 3, scaling of stochastic gradients by diagonal matrix S_k, consider the following example. We have $1,000,000$ data points, distributed across $K = 1,000$ nodes. When we look at a particular feature of the data points, we observe it is non-zero only in $1,000$ of them. Moreover, all of them happen to be stored on a single node, that stores only these $1,000$ data points. Sampling a data point from this node and evaluating the corresponding gradient, will clearly yield an estimate of the gradient $\nabla f(w)$ with 1000-times larger magnitude. This would not necessarily be a problem if done only once. However, repeatedly sampling and overshooting the magnitude of the gradient will likely cause the iterative process to diverge quickly.

Hence, we scale the stochastic gradients by a diagonal matrix. This can be seen as an attempt to enforce the estimates of the gradient to be of the correct magnitude, conditioned on us, algorithm designers, being aware of the structure of distribution of the sparsity pattern.

Let us now highlight some properties of the modification in Point 4. Without any extra information, or in the case of fully dense data, averaging the local updates is the only way that actually makes sense — because each node outputs approximate solution of a proxy to the overall objective, and there is no induced separability structure in the outputs such as in CoCoA [57]. However, we could do much more in the other extreme. If the sparsity structure is such that each data point only depends on one of disjoint groups of variables, and the data were distributed according to this structure, we would efficiently have several disjoint problems. Solving each of them locally, and adding up the results would solve the problem in single iteration — desired algorithm property (C).

What we propose is an interpolation between these two settings, on a per-variable basis. If a variable appears in data on each node, we are going to take average. However, the less nodes a particular variable appear on, the more we want to trust those few nodes in informing us about the meaningful update to this variable — or alternatively, take a longer step. Hence the per-variable scaling of aggregated updates.

3.7 Further Notes

Looking at the Proposition 1, we identify equivalence of two algorithms, take the second one and try modify it to make it suitable for the setting of federated optimization. A question naturally arise: Is it possible to achieve the same by modifying the first algorithm suitable for federated optimization — by only altering the local optimization objective?

We indeed tried to experiment with idea, but we don't report the details for two reasons. First, the requirement of exact solution of the local subproblem is often impractical. Relaxing it gradually moves us to the setting we presented in the previous sections. But more importantly, using this approach we have only managed to get results significantly inferior to those reported later in the Experiments section.

4 Experiments

In this section we present the first experimental results in the setting of federated optimization. In particular, we provide results on a dataset based on public Google+ posts[8], clustered by user — simulating each user as a independent node. This preliminary experiment demonstrates why none of the existing algorithms are suitable for federated optimization, and the robustness of our proposed method to challenges arising there.

4.1 Predicting Comments on Public Google+ Posts

The dataset presented here was generated based on public Google+ posts. We randomly picked $10,000$ authors that have at least 100 public posts in English, and try to predict whether a post will receive at least one comment (that is, a binary classification task).

We split the data chronologically on a per-author basis, taking the earlier 75% for training and the following 25% for testing. The total number of training examples is $n = 2,166,693$. We created a simple bag-of-words language model, based on the $20,000$ most frequent words in dictionary based on all Google+ data. This results in a problem with dimension $d = 20,002$. The extra two features represent a bias term and variable for unknown word. We then use a logistic regression model to make a prediction based on these features.

We shape the distributed optimization problem as follows. Suppose that each user corresponds to one node, resulting in $K = 10,000$. The average n_k, number of data points on node k is thus roughly 216. However, the actual numbers n_k range from 75 to $9,000$, showing the data is in fact substantially unbalanced.

It is natural to expect that different users can exhibit very different patterns in the data generated. This is indeed the case, and hence the distribution to nodes cannot be considered an IID sample from the overall distribution. Since we have a bag-of-words model, our data are very sparse — most posts contain only small fraction of all the words in the dictionary. This, together with the fact that the data are naturally clustered on a per-user basis, creates additional challenge that is not present in the traditional distributed setting.

Figure 1 shows the frequency of different features across nodes. Some features are present everywhere, such as the bias term, while most features are relatively rare. In particular, over 88% of features are present on fewer than $1,000$ nodes. However, this distribution does not necessarily resemble the overall appearance of the features in data examples. For instance, while an unknown word is present in data of almost every user, it is far from being contained in every data point.

Naive prediction properties. Before presenting the results, it is useful to look at some of the important basic prediction properties of the data. We use L2-regularized logistic regression, with regularization parameter $\lambda = 1/n$. We chose λ to be the best in terms of test error in the optimal solution.

- If one chooses to predict -1 (no comment), classification error is **33.16%**.
- The optimal solution of the global logistic regression problem yields **26.27%** test set error.
- Predicting the per-author majority from the training data yields **17.14%** test error. That is, predict $+1$ or -1 for all the posts of an author, based on which label was more common in

[8]The posts were public at the time the experiment was performed, but since a user may decide to delete the post or make it non-public, we cannot release (or even permanently store) any copies of the data.

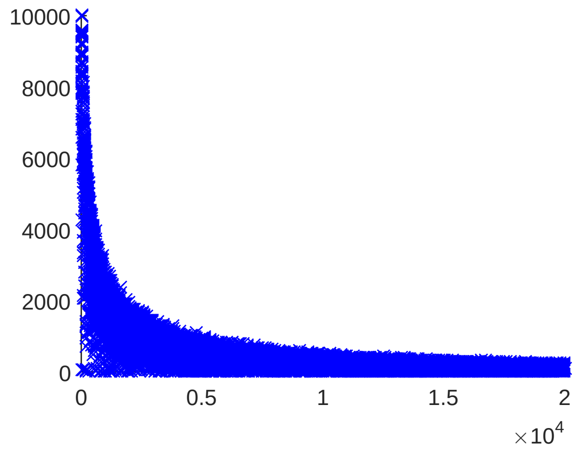

Figure 1: Features vs. appearance on nodes. The x-axis is a feature index, and the y-axis represents the number of nodes where a given feature is present.

that author's training data. This indicates that knowing the author is actually more useful than knowing what they said, which is perhaps not surprising.

In summary, this data is representative for our motivating application in federated optimization. It is possible to improve upon naive baseline using a fixed global model. Further, the per-author majority result suggests it is possible to improve further by adapting the global model to each user individually. Model personalization is common practice in industrial applications, and the techniques used to do this are orthogonal to the challenges of federated optimization. Exploring its performance is a natural next step, but beyond the scope of this work.

While we do not provide experiments for per user personalized models, we remark that this could be a good descriptor of how far from IID the data is distributed. Indeed, if each node has access to an IID sample, any adaptation to local data is merely over-fitting. However, if we can significantly improve upon the global model by per user/node adaptation, this means that the data available locally exhibit patterns specific to the particular node.

The performance of the Algorithm 4 is presented below. The only parameter that remains to be chosen by user is the stepsize h. We tried a set of stepsizes, and retrospectively choose one that works best — a typical practice in machine learning.

In Figure 2, we compare the following optimization algorithms[9]:

- The blue squares (OPT) represent the best possible offline value (the optimal value of the optimization task in the first plot, and the test error corresponding to the optimum in the second plot).
- The teal diamonds (GD) correspond to a simple distributed gradient descent.
- The purple triangles (COCOA) are for the CoCoA+ algorithm [57].
- The green circles (FSVRG) give values for our proposed algorithm.
- The red stars (FSVRGR) correspond to the same algorithm applied to the same problem with randomly reshuffled data. That is, we keep the unbalanced number of examples per node, but populate each node with randomly selected examples.

[9]We thank Mark Schmidt for his `prettyPlot` function, available on his website.

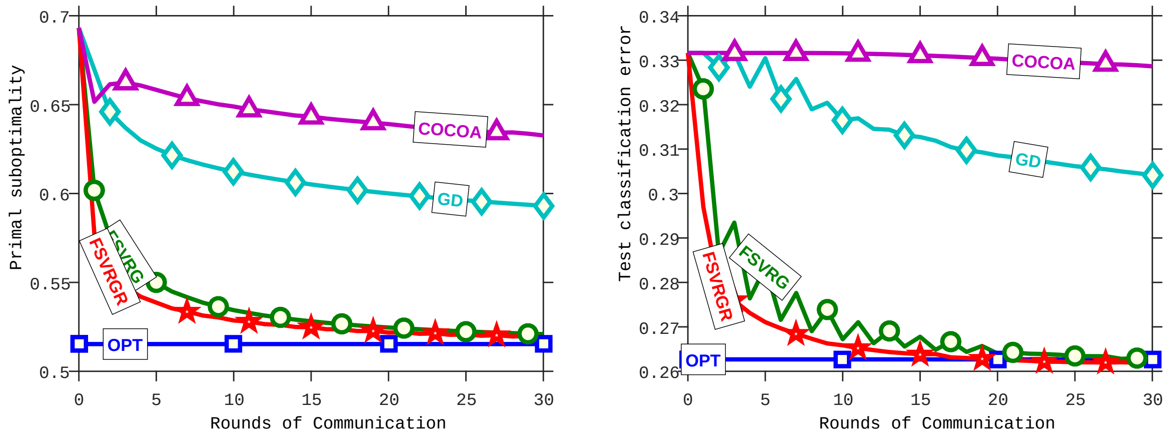

Figure 2: Rounds of communication vs. objective function (left) and test prediction error (right).

The first thing to notice is that CoCoA+ seems to be worse than trivial benchmark — distributed gradient descent. This behaviour can be predicted from theory, as the overall convergence rate directly depends on the best choice of aggregation parameter σ'. For sparse problems, it is upperbounded by the maximum of the values reported in Figure 1, which is K, and it is close to it also in practice. Althought it is expected that the algorithm could be modified to depend on average of these quantities (which could be orders of magnitude smaller), akin to coordinate descent algorithms [79], it has not been done yet. Note that other communication efficient algorithms fail to converge altogether.

The algorithm we propose, FSVRG, converges to optimal test classification accuracy in just 30 iterations. Recall that in the setting of federated optimization we introduced in Section 1.2, minimization of rounds of communication is the principal goal. However, concluding that the approach is stunningly superior to existing methods would not be completely fair nor correct. The conclusion is that the *FSVRG is the first algorithm to tackle federated optimization*, a problem that existing methods fail to generalize to. It is important to stress that none of the existing methods were designed with these particular challenges in mind, and we formulate the first benchmark.

Since the core reason other methods fail to converge is the non-IID data distribution, we test our method on the same problem, with data randomly reshuffled among the same number of nodes (FSVRGR; red stars). Since the difference in convergence is subtle, we can conclude that the techniques described in Section 3.6.2 serve its purpose and make the algorithm robust to challenges present in federated optimization.

This experiment demonstrates that learning from massively decentralized data, clustered on a per-user basis is indeed problem we can tackle in practice. Since the first version of this paper [45], additional experimental results were presented in [62]. We refer the reader to this paper for experiments in more challenging setting of deep learning, and a further discussion on how such system would be implemented in practice.

5 Conclusions and Future Challenges

We have introduced a new setting for distributed optimization, which we call *federated optimization*. This setting is motivated by the outlined vision, in which users do not send the data they generate to companies at all, but rather provide part of their computational power to be used to solve optimization problems. This comes with a unique set of challenges for distributed optimization. In particular, we argue that the massively distributed, non-IID, unbalanced, and sparse properties of federated optimization problems need to be addressed by the optimization community.

We explain why existing methods are not applicable or effective in this setting. Even the distributed algorithms that can be applied converge very slowly in the presence of large number of nodes on which the data are stored. We demonstrate that in practice, it is possible to design algorithms that work surprisingly efficiently in the challenging setting of federated optimization, which makes the vision conceptually feasible.

We realize that it is important to scale stochastic gradients on a per-coordinate basis, differently on each node to improve performance. To the best of our knowledge, this is the first time such per-node scaling has been used in distributed optimization. Additionally, we use per-coordinate aggregation of updates from each node, based on distribution of the sparsity patterns in the data.

Even though our results are encouraging, there is a lot of room for future work. One natural direction is to consider fully asynchronous versions of our algorithms, where the updates are applied as soon as they arrive. Another is developing a better theoretical understanding of our algorithm, as we believe that development of a strong understanding of the convergence properties will drive further research in this area.

Study of the federated optimization problem for non-convex objectives is another important avenue of research. In particular, neural networks are the most important example of a machine learning tool that yields non-convex functions f_i, without any convenient general structure. Consequently, there are no useful results describing convergence guarantees of optimization algorithms. Despite the lack of theoretical understanding, neural networks are now state-of-the-art in many application areas, ranging from natural language understanding to visual object detection. Such applications arise naturally in federated optimization settings, and so extending our work to such problems is an important direction.

The non-IID data distribution assumed in federated optimization, and mobile applications in particular, suggest that one should consider the problem of training a *personalized* model together with that of learning a global model. That is, if there is enough data available on a given node, and we assume that data is drawn from the same distribution as future test examples for that node, it may be preferable to make predictions based on a personalized model that is biased toward good performance on the local data, rather than simply using the global model.

References

[1] Martín Abadi, Andy Chu, Ian Goodfellow, Brendan H McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. *arXiv:1607.00133*, 2016.

[2] Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. In *Advances in Neural Information Processing Systems*, pages 873–881, 2011.

[3] Zeyuan Allen-Zhu. Katyusha: The first direct acceleration of stochastic gradient methods. *arXiv:1603.05953*, 2016.

[4] Zeyuan Allen-Zhu, Yang Yuan, and Karthik Sridharan. Exploiting the structure: Stochastic gradient methods using raw clusters. *arXiv:1602.02151*, 2016.

[5] Yossi Arjevani and Ohad Shamir. Communication complexity of distributed convex learning and optimization. In *Advances in Neural Information Processing Systems*, pages 1756–1764, 2015.

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

[7] Dimitri P Bertsekas. Distributed asynchronous computation of fixed points. *Mathematical Programming*, 27(1):107–120, 1983.

[8] Dimitri P Bertsekas and John N Tsitsiklis. *Parallel and distributed computation: numerical methods*. Prentice-Hall, Inc., 1989.

[9] Antoine Bordes, Léon Bottou, and Patrick Gallinari. Sgd-qn: Careful quasi-Newton stochastic gradient descent. *The Journal of Machine Learning Research*, 10:1737–1754, 2009.

[10] Léon Bottou. Curiously fast convergence of some stochastic gradient descent algorithms. In *Proceedings of the symposium on learning and data science*, 2009.

[11] Léon Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, pages 421–436. Springer, 2012.

[12] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *arXiv:1606.04838*, 2016.

[13] Olivier Bousquet and Léon Bottou. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems*, pages 161–168, 2008.

[14] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[15] Joseph Bradley, Aapo Kyrola, Daniel Bickson, and Carlos Guestrin. Parallel coordinate descent for l1-regularized loss minimization. In *Proceedings of the 28th International Conference on Machine Learning*, pages 321–328, 2011.

[16] Richard H Byrd, Samantha L Hansen, Jorge Nocedal, and Yoram Singer. A stochastic quasi-Newton method for large-scale optimization. *SIAM Journal on Optimization*, 26(2):1008–1031, 2016.

[17] Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12:1069–1109, 2011.

[18] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 571–582, 2014.

[19] CNN. Where (and when) do you use your smartphone: Bedroom? Church? News article `http://edition.cnn.com/2013/07/13/tech/smartphone-use-survey/`, 2013.

[20] Dominik Csiba and Peter Richtárik. Primal method for ERM with flexible mini-batching schemes and non-convex losses. *arXiv:1506.02227*, 2015.

[21] Dominik Csiba and Peter Richtárik. Coordinate descent face-off: primal or dual? *arXiv:1605.08982*, 2016.

[22] Christopher M De Sa, Ce Zhang, Kunle Olukotun, and Christopher Ré. Taming the wild: A unified analysis of hogwild-style algorithms. In *Advances in Neural Information Processing Systems*, pages 2656–2664, 2015.

[23] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2012.

[24] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[25] Aaron Defazio. A simple practical accelerated method for finite sums. *arXiv preprint arXiv:1602.02442*, 2016.

[26] Aaron Defazio, Francis Bach, and Simon 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.

[27] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal distributed online prediction using mini-batches. *The Journal of Machine Learning Research*, 13(1):165–202, 2012.

[28] Wei Deng and Wotao Yin. On the global and linear convergence of the generalized alternating direction method of multipliers. *Journal of Scientific Computing*, 66(3):889–916, 2016.

[29] John C Duchi, Alekh Agarwal, and Martin J Wainwright. Dual averaging for distributed optimization: convergence analysis and network scaling. *Automatic control, IEEE Transactions on*, 57(3):592–606, 2012.

[30] John C Duchi, Sorathan Chaturapruek, and Christopher Ré. Asynchronous stochastic convex optimization. *arXiv:1508.00882*, 2015.

[31] John C Duchi, Michael I Jordan, and Brendan H McMahan. Estimation, optimization, and parallelism when data is sparse. In *Advances in Neural Information Processing Systems*, pages 2832–2840, 2013.

[32] John C Duchi, Michael I Jordan, and Martin J Wainwright. Privacy aware learning. *Journal of the Association for Computing Machinery*, 2014.

[33] Cynthia Dwork and Aaron Roth. *The Algorithmic Foundations of Differential Privacy*. Foundations and Trends in Theoretical Computer Science. Now Publishers, 2014.

[34] Olivier Fercoq, Zheng Qu, Peter Richtárik, and Martin Takáč. Fast distributed coordinate descent for non-strongly convex losses. In *Machine Learning for Signal Processing (MLSP), 2014 IEEE International Workshop on*, pages 1–6. IEEE, 2014.

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

[36] The MPI Forum. MPI: A message passing interface standard, Version 3.1. Document available at `http://www.mpi-forum.org/`, 2015.

[37] Roy Frostig, Rong Ge, Sham M Kakade, and Aaron Sidford. Un-regularizing: approximate proximal point and faster stochastic algorithms for empirical risk minimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015.

[38] Robert Mansel Gower, Donald Goldfarb, and Peter Richtárik. Stochastic block BFGS: squeezing more curvature out of data. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1869–1878, 2016.

[39] Robert Mansel Gower and Peter Richtárik. Randomized quasi-Newton updates are linearly convergent matrix inversion algorithms. *arXiv:1602.01768*, pages 1869–1878, 2016.

[40] Mert Gürbüzbalaban, Asu Ozdaglar, and Pablo Parrilo. Why random reshuffling beats stochastic gradient descent. *arXiv:1510.08560*, 2015.

[41] Reza Harikandeh, Mohamed Osama Ahmed, Alim Virani, Mark Schmidt, Jakub Konečný, and Scott Sallinen. Stop wasting my gradients: Practical SVRG. In *Advances in Neural Information Processing Systems*, pages 2251–2259, 2015.

[42] Martin Jaggi, Virginia Smith, Martin Takáč, Jonathan Terhorst, Sanjay Krishnan, Thomas Hofmann, and Michael I Jordan. Communication-efficient distributed dual coordinate ascent. In *Advances in Neural Information Processing Systems*, pages 3068–3076, 2014.

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

[44] Jakub Konečný, Jie Liu, Peter Richtárik, and Martin Takáč. Mini-batch semi-stochastic gradient descent in the proximal setting. *IEEE Journal of Selected Topics in Signal Processing*, 10(2):242–255, 2016.

[45] Jakub Konečný, Brendan H McMahan, and Daniel Ramage. Federated optimization: Distributed optimization beyond the datacenter. *arXiv:1511.03575*, 2015.

[46] Jakub Konečný, Zheng Qu, and Peter Richtárik. Semi-stochastic coordinate descent. *arXiv:1412.6293*, 2014.

[47] Jakub Konečný and Peter Richtárik. Semi-stochastic gradient descent methods. *arXiv:1312.1666*, 2013.

[48] Rémi Leblond, Fabian Pedregosa, and Simon Lacoste-Julien. ASAGA: Asynchronous parallel saga. *arXiv:1606.04809*, 2016.

[49] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[50] Jason Lee, Tengyu Ma, and Qihang Lin. Distributed stochastic variance reduced gradient methods. *arXiv:1507.07595*, 2015.

[51] Yin Tat Lee and Aaron Sidford. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 147–156. IEEE, 2013.

[52] Chieh-Yen Lin, Cheng-Hao Tsai, Ching-Pei Lee, and Chih-Jen Lin. Large-scale logistic regression and linear support vector machines using spark. In *Big Data (Big Data), 2014 IEEE International Conference on*, pages 519–528. IEEE, 2014.

[53] Hongzhou Lin, Julien Mairal, and Zaid Harchaoui. A universal catalyst for first-order optimization. In *Advances in Neural Information Processing Systems*, pages 3366–3374, 2015.

[54] Dong C Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[55] Ji Liu and Stephen J Wright. Asynchronous stochastic coordinate descent: Parallelism and convergence properties. *SIAM Journal on Optimization*, 25(1):351–376, 2015.

[56] Ji Liu, Stephen J Wright, Christopher Ré, Victor Bittorf, and Srikrishna Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. *Journal of Machine Learning Research*, 16:285–322, 2015.

[57] Chenxin Ma, Jakub Konečný, Martin Jaggi, Virginia Smith, Michael I Jordan, Peter Richtárik, and Martin Takáč. Distributed optimization with arbitrary local solvers. *arXiv:1512.04039*, 2015.

[58] Chenxin Ma, Virginia Smith, Martin Jaggi, Michael Jordan, Peter Richtárik, and Martin Takáč. Adding vs. averaging in distributed primal-dual optimization. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 1973–1982, 2015.

[59] Dhruv Mahajan, Nikunj Agrawal, S Sathiya Keerthi, S Sundararajan, and Leon Bottou. An efficient distributed learning algorithm based on effective local functional approximations. *arXiv preprint arXiv:1310.8418*, 2013.

[60] Horia Mania, Xinghao Pan, Dimitris Papailiopoulos, Benjamin Recht, Kannan Ramchandran, and Michael I Jordan. Perturbed iterate analysis for asynchronous stochastic optimization. *arXiv:1507.06970*, 2015.

[61] Jakub Mareček, Peter Richtárik, and Martin Takáč. Distributed block coordinate descent for minimizing partially separable functions. In *Numerical Analysis and Optimization*, pages 261–288. Springer, 2015.

[62] Brendan H McMahan, Eider Moore, Daniel Ramage, and Blaise Aguera y Arcas. Federated learning of deep networks using model averaging. *arXiv:1602.05629*, 2016.

[63] Philipp Moritz, Robert Nishihara, and Michael Jordan. A linearly-convergent stochastic l-bfgs algorithm. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 249–258, 2016.

[64] Eric Moulines and Francis R Bach. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *Advances in Neural Information Processing Systems*, pages 451–459, 2011.

[65] Deanna Needell, Rachel Ward, and Nati Srebro. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. In *Advances in Neural Information Processing Systems*, pages 1017–1025, 2014.

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

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

[68] Yurii Nesterov. A method of solving a convex programming problem with convergence rate $o(1/k^2)$. *Soviet Mathematics Doklady*, 27(2):372–376, 1983.

[69] Yurii Nesterov. *Introductory Lectures on Convex Optimization. A Basic Course*. Kluwer, 2004.

[70] Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, Quoc V Le, and Andrew Y Ng. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on Machine Learning*, pages 265–272, 2011.

[71] Feng Niu, Benjamin Recht, Christopher Re, and Stephen Wright. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011.

[72] Zhimin Peng, Yangyang Xu, Ming Yan, and Wotao Yin. Arock: an algorithmic framework for asynchronous parallel coordinate updates. *arXiv:1506.02396*, 2015.

[73] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[74] Zheng Qu, Peter Richtárik, Martin Takáč, and Olivier Fercoq. SDNA: Stochastic dual newton ascent for empirical risk minimization. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1823–1832, 2016.

[75] Zheng Qu, Peter Richtárik, and Tong Zhang. Quartz: Randomized dual coordinate ascent with arbitrary sampling. In *Advances in Neural Information Processing Systems*, volume 28, pages 865–873, 2015.

[76] Sashank J Reddi, Ahmed Hefny, Suvrit Sra, Barnabás Póczós, and Alex Smola. On variance reduction in stochastic gradient descent and its asynchronous variants. In *Advances in Neural Information Processing Systems*, pages 2647–2655, 2015.

[77] Sashank J Reddi, Ahmed Hefny, Suvrit Sra, Barnabás Póczós, and Alex Smola. Stochastic variance reduction for nonconvex optimization. *arXiv:1603.06160*, 2016.

[78] Sashank J Reddi, Jakub Konečný, Peter Richtárik, Barnabás Póczós, and Alex Smola. AIDE: Fast and communication efficient distributed optimization. *arXiv:1608.06879*, 2016.

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

[80] Peter Richtárik and Martin Takáč. Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, 156(1):433–484, 2016.

[81] Peter Richtárik and Martin Takáč. Distributed coordinate descent method for learning with big data. *Journal of Machine Learning Research*, 17(75):1–25, 2016.

[82] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, pages 400–407, 1951.

[83] Nicolas Le Roux, Mark Schmidt, and Francis Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *Advances in Neural Information Processing Systems*, pages 2663–2671, 2012.

[84] Mark Schmidt, Reza Babanezhad, Mohamed Ahmed, Aaron Defazio, Ann Clifton, and Anoop Sarkar. Non-uniform stochastic average gradient method for training conditional random fields. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, pages 819–828, 2015.

[85] Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *arXiv:1309.2388*, 2013.

[86] Shai Shalev-Shwartz. SDCA without duality, regularization, and individual convexity. *arXiv:1602.01582*, 2016.

[87] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.

[88] Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss. *The Journal of Machine Learning Research*, 14(1):567–599, 2013.

[89] John Shalf, Sudip Dosanjh, and John Morrison. Exascale computing technology challenges. In *High Performance Computing for Computational Science–VECPAR 2010*, pages 1–25. Springer, 2011.

[90] Ohad Shamir and Nathan Srebro. Distributed stochastic optimization and learning. In *Communication, Control and Computing (Allerton), 2014 52nd Annual Allerton Conference on*, pages 850–857. IEEE, 2014.

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

[92] Virginia Smith, Simone Forte, Michael I Jordan, and Martin Jaggi. L1-regularized distributed optimization: A communication-efficient primal-dual framework. *arXiv:1512.04011*, 2015.

[93] Martin Takáč, Avleen Bijral, Peter Richtárik, and Nathan Srebro. Mini-batch primal and dual methods for SVMs. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.

[94] Martin Takáč, Peter Richtárik, and Nathan Srebro. Distributed mini-batch SDCA. *arXiv:1507.08322*, 2015.

[95] John Nikolas Tsitsiklis. Problems in decentralized decision making and computation. Technical report, DTIC Document, 1984.

[96] Vladimir N Vapnik. An overview of statistical learning theory. *Neural Networks, IEEE Transactions on*, 10(5):988–999, 1999.

[97] Huahua Wang and Arindam Banerjee. Randomized block coordinate descent for online and stochastic optimization. *arXiv:1407.0107*, 2014.

[98] White House Report. Consumer data privacy in a networked world: A framework for protecting privacy and promoting innovation in the global digital economy. *Journal of Privacy and Confidentiality*, 2013.

[99] Blake Woodworth and Nathan Srebro. Tight complexity bounds for optimizing composite objectives. *arXiv:1605.08003*, 2016.

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

[101] Tianbao Yang. Trading computation for communication: Distributed stochastic dual coordinate ascent. In *Advances in Neural Information Processing Systems*, pages 629–637, 2013.

[102] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, volume 10, page 10, 2010.

[103] Yuchen Zhang, John Duchi, Michael I Jordan, and Martin J Wainwright. Information-theoretic lower bounds for distributed statistical estimation with communication constraints. In *Advances in Neural Information Processing Systems*, pages 2328–2336, 2013.

[104] Yuchen Zhang, John C Duchi, and Martin J Wainwright. Communication-efficient algorithms for statistical optimization. *Journal of Machine Learning Research*, 14:3321–3363, 2013.

[105] Yuchen Zhang and Xiao Lin. DiSCO: Distributed optimization for self-concordant empirical loss. In *Proceedings of The 32th International Conference on Machine Learning*, pages 362–370, 2015.

[106] Yong Zhuang, Wei-Sheng Chin, Yu-Chin Juan, and Chih-Jen Lin. Distributed newton methods for regularized logistic regression. In *Advances in Knowledge Discovery and Data Mining*, pages 690–703. Springer, 2015.

[107] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 2595–2603, 2010.

A Distributed Optimization via Quadratic Perturbations

This appendix follows from the discussion motivating DANE algorithm by a general algorithmic perturbation template (9) for λ-strongly convex objectives. We use this to propose a similar but new method, which unlike DANE converges under arbitrary data partitioning $\{\mathcal{P}_k\}_{k=1}^K$, and we highlight its relation to the dual CoCoA algorithm for distributed optimization.

For simplicity and ease of drawing the above connections we assume that n_k is identical for all $k \in \{1, 2, \dots, K\}$ throughout the appendix. All the arguments can be simply extended, but would unnecessarily complicate the notation for current purpose.

A.1 New Method

We now present a new method (Algorithm 5), which also belongs to the family of quadratic perturbation methods (9). However, the perturbation vectors a_k^t are different from those of DANE. In particular, we set

$$a_k^t \overset{\text{def}}{=} \nabla F_k(w^t) - (\eta \nabla F_k(w^t) + g_k^t),$$

where $\eta > 0$ is a parameter, and the vectors g_k^t are maintained by the method. As we show in Lemma 4, Algorithm 5 satisfies

$$\sum_{k=1}^K g_k^t = 0$$

for all iterations t. This implies that $\frac{1}{K}\sum_{k=1}^K a_k^t = (1-\eta)\nabla f(w^t)$. That is, both DANE and the new method use a linear perturbation which, when averaged over the nodes, involves the gradient of the objective function f at the latest iterate w^t. Therefore, the methods have one more property in common beyond both being of the form (9). However, as we shall see in the rest of this section, Algorithm 5 allows an insightful dual interpretation. Moreover, while DANE may not converge for arbitrary problems (even when restricted to ridge regression)—and is only known to converge under the assumption that the data stored on each node are in some precise way similar, Algorithm 5 converges for any ridge regression problem and any data partitioning.

Let us denote by X_k the matrix obtained by stacking the data points x_i as column vectors for all $i \in \mathcal{P}_k$. We have the following Lemma.

Lemma 4. *For all $t \geq 0$ we have $\sum_{k=1}^K g_k^t = 0$.*

Proof. The statement holds for $t = 0$. Indeed,

$$\sum_{k=1}^K g_k^t = \eta \sum_{k=1}^K \left(\frac{K}{n} X_k \alpha_k^0 - \lambda w^0 \right) = 0,$$

where the last step follows from the definition of w^0. Assume now that the statement hold for t. Then

$$\sum_{k=1}^K g_k^{t+1} = \sum_{k=1}^K \left(g_k^t + \eta\lambda(w_k^{t+1} - w^{t+1}) \right) = \eta\lambda \sum_{k=1}^K (w_k^{t+1} - w^{t+1}).$$

The first equation follows from the way g_k is updated in the algorithm. The second equation follows from the inductive assumption, and the last equation follows from the definition of w^{t+1} in the algorithm. □

Algorithm 5 Primal Method

```
1: Input: $\sigma \in [1, K]$
2: Choose: $\alpha_k^0 \in \mathbb{R}^{|\mathcal{P}_k|}$ for $k = 1, 2, \ldots, K$
3: Set: $\eta = \frac{K}{\sigma}$, $\mu = \lambda(\eta - 1)$
4: Set: $w^0 = \frac{1}{\lambda n} \sum_{k=1}^K X_k \alpha_k^0$
5: Set: $g_k^0 = \eta(\frac{K}{n} X_k \alpha_k^0 - \lambda w^0)$ for $k = 1, 2, \ldots, K$
6: for $t = 0, 1, 2, \ldots$ do
7:     for $k = 1$ to $K$ do
8:         $w_k^{t+1} = \arg\min_{w \in \mathbb{R}^d} F_k(w) - \left(\nabla F_k(w^t) - (\eta \nabla F_k(w^t) + g_k^t)\right)^T w + \frac{\mu}{2}\|w - w^t\|^2$
9:     end for
10:    $w^{t+1} = \frac{1}{K} \sum_{k=1}^K w_k^{t+1}$
11:    for $k = 1$ to $K$ do
12:        $g_k^{t+1} = g_k^t + \lambda\eta(w_k^{t+1} - w^{t+1})$
13:    end for
14:    return $w^t$
15: end for
```

A.2 L2-Regularized Linear Predictors

In the rest of this section we consider the case of L2-regularized *linear predictors*. That is, we focus on problem (1) with f_i of the form

$$f_i(w) = \phi_i(x_i^T w) + \frac{\lambda}{2}\|w\|^2,$$

where $\lambda > 0$ is a regularization parameter. This leads to L2 regularized empirical risk minimization (ERM) problem

$$\min_{w \in \mathbb{R}^d} \left\{ f(w) \overset{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n \phi_i(x_i^T w) + \frac{\lambda}{2}\|w\|^2 \right\}. \tag{11}$$

We assume that the loss functions $\phi_i : \mathbb{R} \to \mathbb{R}$ are convex and $1/\gamma$-smooth for some $\gamma > 0$; these are standard assumptions. As usual, we allow the loss function ϕ_i to depend on the label y_i. For instance, we may choose the quadratic loss: $\phi_i(t) = \frac{1}{2}(t - y_i)^2$ (for which $\gamma = 1$).

Let $X = [x_1, \ldots, x_n] \in \mathbb{R}^{d \times n}$. As described in Section 3.3, we assume that the data $(x_i, y_i)_{i=1}^n$ is distributed among K nodes of a computer cluster as follows: node $k = 1, 2, \ldots, K$ contains pairs (x_i, y_i) for $i \in \mathcal{P}_k$, where $\mathcal{P}_1, \ldots, \mathcal{P}_K$ forms a partition of the set $[n] = \{1, 2, \ldots, n\}$. Letting $X = [X_1, \ldots, X_K]$, where $X_k \in \mathbb{R}^{d \times |\mathcal{P}_k|}$ is a submatrix of A corresponding to columns $i \in \mathcal{P}_k$, and $y_k \in \mathbb{R}^{|\mathcal{P}_k|}$ is the subvector of y corresponding to entries $i \in \mathcal{P}_k$. Hence, node k contains the pair (X_k, y_k). With this notation, we can write the problem in the form (8), where

$$F_k(w) = \frac{K}{n} \sum_{i \in \mathcal{P}_k} \phi_i(x_i^T w) + \frac{\lambda}{2}\|w\|^2. \tag{12}$$

A.3 A Dual Method: Dual Block Proximal Gradient Ascent

The dual of (11) is the problem

$$\max_{\alpha\in\mathbb{R}^n}\left\{D(\alpha)\overset{\text{def}}{=}-\frac{1}{2\lambda n^2}\|X\alpha\|^2-\frac{1}{n}\sum_{i=1}^n\phi_i^*(-\alpha_i)\right\}, \tag{13}$$

where ϕ_i^* is the convex conjugate of ϕ_i. Since we assume that ϕ_i is $1/\gamma$ smooth, it follows that ϕ_i^* is γ strongly convex. Therefore, D is a strongly concave function.

From dual solution to a primal solution. It is well known that if α^* is the optimal solution of the dual problem (11), then $w^*\overset{\text{def}}{=}\frac{1}{\lambda n}X\alpha^*$ is the optimal solution of the primal problem. Therefore, for any dual algorithm producing a sequence of iterates α^t, we can define a corresponding primal algorithm via the linear mapping

$$w^t\overset{\text{def}}{=}\frac{1}{\lambda n}X\alpha^t. \tag{14}$$

Clearly, if $\alpha^t\to\alpha^*$, then $w^t\to w^*$. We shall now design a method for maximizing the dual function D and then in Theorem 5 we claim that for quadratic loss functions, Algorithm 5 arises as an image, defined via (14), of dual iterations of this dual ascent method.

Design of the dual gradient ascent method. Let $\xi(\alpha)\overset{\text{def}}{=}\frac{1}{2}\|X\alpha\|^2$. Since ξ is a convex quadratic, we have

$$\xi(\alpha+h)=\xi(\alpha)+\langle\nabla\xi(\alpha),h\rangle+\frac{1}{2}h^T\nabla^2\xi(\alpha)h,\ \le\xi(\alpha)+\langle\nabla\xi(\alpha),h\rangle+\frac{\sigma}{2}\|h\|_B^2,$$

where $\nabla\xi(\alpha)=X^TX\alpha$ and $\nabla^2\xi(\alpha)=X^TX$. Further, we define the block-diagonal matrix $B\overset{\text{def}}{=}Diag(X_1^TX_1,\dots,X_K^TX_K)$, and a norm associate with this matrix:

$$\|h\|_B^2\overset{\text{def}}{=}\sum_{k=1}^K\|X_kh_k\|^2.$$

By σ we refer to a large enough constant for which $X^TX\preceq\sigma B$. In order to avoid unnecessary technicalities, we shall assume that the matrices $X_k^TX_k$ are positive definite, which implies that $\|\cdot\|_B$ is a norm. It can be shown that $1\le\sigma\le K$. Clearly, ξ is σ-smooth with respect to the norm $\|\cdot\|_B$. In view of the above, for all $h\in\mathbb{R}^n$ we can estimate D from below as follows:

$$\begin{aligned}D(\alpha^t+h)&\ge-\frac{1}{\lambda n^2}\left(\xi(\alpha^t)+\langle\nabla\xi(\alpha^t),h\rangle+\frac{\sigma}{2}\sum_{k=1}^K\|X_kh_k\|^2\right)-\frac{1}{n}\sum_{i=1}^n\phi_i^*(-\alpha_i^t-h_i)\\&=-\frac{1}{\lambda n^2}\xi(\alpha^t)-\sum_{k=1}^K\left[\frac{1}{\lambda n^2}\langle\nabla_k\xi(\alpha^t),h_k\rangle+\frac{\sigma}{2\lambda n^2}\|X_kh_k\|^2+\frac{1}{n}\sum_{i\in\mathcal{P}_k}\phi_i^*(-\alpha_i^t-h_i)\right],\end{aligned}$$

where $\nabla_k\xi(\alpha^t)$ corresponds to the subvector of $\nabla\xi(\alpha^t)$ formed by entries $i\in\mathcal{P}_k$.

We now let $h^t=(h_1^t,\dots,h_K^t)$ be the maximizer of this lower bound. Since the lower bound is separable in the blocks $\{h_k^t\}_k$, we can simply set

$$h_k^t:=\arg\min_{u\in\mathbb{R}^{|\mathcal{P}_k|}}\left\{D_k^t(u)\overset{\text{def}}{=}\frac{1}{\lambda n^2}\langle\nabla_k\xi(\alpha^t),u\rangle+\frac{\sigma}{2\lambda n^2}\|X_ku\|^2+\frac{1}{n}\sum_{i\in\mathcal{P}_k}\phi_i^*(-\alpha_i^t-u_i)\right\}. \tag{15}$$

Algorithm 6 Dual Method

```
1: Input: $\sigma \in [1, K]$
2: Choose: $\alpha_k^0 \in \mathbb{R}^{|\mathcal{P}_k|}$ for $k = 1, 2, \dots, K$
3: for $t = 0, 1, 2, \dots$ do
4:     for $k = 1$ to $K$ do
5:         $h_k^{t+1} = \arg\min_{u \in \mathbb{R}^{|\mathcal{P}_k|}} D_k^t(u)$        ▷ See (15)
6:     end for
7:     $\alpha^{t+1} = \alpha^t + h^t$
8: end for
9: return $w^t$
```

Having computed h_k^t for all k, we can set $\alpha_k^{t+1} = \alpha_k^t + h_k^t$ for all k, or equivalently, $\alpha^{t+1} = \alpha^t + h^t$. This is formalized as Algorithm 6. Algorithm 6 is a proximal gradient ascent method applied to the dual problem, with smoothness being measured using the block norm $\|h\|_B$. It is known that gradient ascent converges at a linear rate for smooth and strongly convex (for minimization problems) objectives.

One of the main insights of this section is the following equivalence result.

Theorem 5 (Equivalence of Algorithms 5 and 6 for Quadratic Loss)**.** *Consider the ridge regression problem. That is, set $\phi_i(t) = \frac{1}{2}(t - y_i)^2$ for all i. Assume $\alpha_1^0, \dots, \alpha_K^0$ is chosen in the same way in Algorithms 5 and 6. Then the dual iterates α^t and the primal iterates w^t produced by the two algorithms are related via* (14) *for all $t \geq 0$.*

Since the dual method converges linearly, in view of the above theorem, so does the primal method. Here we only remark that the popular algorithm CoCoA+ [57] arises if Step 5 in Algorithm 6 is done inexactly. Hence, we show that duality provides a deep relationship between the CoCoA+ and DANE algorithms, which were previously considered completely different.

A.4 Proof of Theorem 5

In this part we prove the theorem.

Primal and Dual Problems. Since $\phi_i(t) = \frac{1}{2}(t - y_i)^2$, the primal problem (11) is a ridge regression problem of the form

$$\min_{w \in \mathbb{R}^d} f(w) = \frac{1}{2n}\|X^T w - y\|^2 + \frac{\lambda}{2}\|w\|^2, \tag{16}$$

where $X \in \mathbb{R}^{d \times n}$ and $y \in \mathbb{R}^n$. In view of (13), the dual of (16) is

$$\min_{\alpha \in \mathbb{R}^n} D(\alpha) = \frac{1}{2\lambda n^2}\|X\alpha\|^2 + \frac{1}{2n}\|\alpha\|^2 - \frac{1}{n} y^T \alpha. \tag{17}$$

Primal Problem: Distributed Setup. The primal objective function is of the form (8), where in view of (12), we have $F_k(w) = \frac{K}{2n}\|X_k^T w - y_k\|^2 + \frac{\lambda}{2}\|w\|^2$. Therefore,

$$\nabla F_k(w) = \frac{K}{n} X_k (X_k^T w - y_k) + \lambda w \tag{18}$$

and $\nabla f(w) = \frac{1}{K}\sum_k \nabla F_k(w) = \frac{1}{K}\sum_k \left(\frac{K}{n} X_k (X_k^T w - y_k) + \lambda w\right)$.

Dual Method. Since D is a quadratic, we have

$$D(\alpha^t + h) \;\;=\;\; D(\alpha^t) + \nabla D(\alpha^t)^T h + \frac{1}{2} h^T \nabla^2 D(\alpha^t) h,$$

with

$$\nabla D(\alpha^t) = \frac{1}{\lambda n^2} X^T X \alpha^t + \frac{1}{n}(\alpha^t - y), \qquad \nabla^2 D(\alpha^t) = \frac{1}{\lambda n^2} X^T X + \frac{1}{n} I.$$

We know that $X^T X \preceq \sigma Diag(X_1^T X_1, \ldots, X_K^T X_K)$. With this approximation, for all $h \in \mathbb{R}^n$ we can estimate D from above by a node-separable quadratic function as follows:

$$\begin{aligned} D(\alpha^t + h) \;&\leq\; D(\alpha^t) + \left(\frac{1}{\lambda n^2} X^T X \alpha^t + \frac{1}{n}(\alpha^t - y)\right)^T h + \frac{1}{2n}\|h\|^2 + \frac{\sigma}{2\lambda n^2} \sum_{k=1}^K \|X_k h_k\|^2 \\ &=\; D(\alpha^t) + \frac{1}{n}\left[\frac{1}{\lambda n}(X\alpha^t)^T X h + (\alpha^t - y)^T h + \frac{1}{2}\|h\|^2 + \frac{\sigma}{2\lambda n}\sum_{k=1}^K \|X_k h_k\|^2\right] \\ &=\; D(\alpha^t) + \frac{1}{n}\sum_{k=1}^K \left((w^t)^T X_k h_k + (\alpha_k^t - y_k)^T h_k + \frac{1}{2}\|h_k\|^2 + \frac{\sigma}{2\lambda n}\|X_k h_k\|^2\right). \end{aligned}$$

Next, we shall define

$$h_k^t \overset{\text{def}}{=} \arg \min_{h_k \in \mathbb{R}^{|\mathcal{P}_k|}} \frac{\sigma}{2\lambda n}\|X_k h_k\|^2 + \frac{1}{2}\|h_k\|^2 - (y_k - X_k^T w^t - \alpha_k^t)^T h_k \tag{19}$$

for $k = 1, 2, \ldots, K$ and then set

$$\alpha^{t+1} = \alpha^t + h^t. \tag{20}$$

Primal Version of the Dual Method. Note that (19) has the same form as (17), with X replaced by X_k, λ replaced by λ/σ and y replaced by $c_k := y_k - X_k^T w^t - \alpha_k^t$. Hence, we know that

$$s_k^t \overset{\text{def}}{=} \frac{1}{(\lambda/\sigma) n} X_k h_k^t \tag{21}$$

is the optimal solution of the primal problem of (22):

$$s_k^t = \arg \min_{s \in \mathbb{R}^d} \frac{1}{2n}\|X_k^T s - c_k\|^2 + \frac{\lambda/\sigma}{2}\|s\|^2. \tag{22}$$

Hence, the primal version of method (20) is given by

$$\begin{aligned} w^{t+1} \;&\overset{(14)}{=}\; \frac{1}{\lambda n} X \alpha^{t+1} \overset{(20)}{=} \frac{1}{\lambda n} X(\alpha^t + h^t) \overset{(14)}{=} w^t + \frac{1}{\lambda n}\sum_{k=1}^K X_k h_k^t \\ &=\; \frac{1}{K}\sum_{k=1}^K \left(w^t + \frac{K}{\sigma}\frac{\sigma}{\lambda n} X_k h_k^t\right) \overset{(21)}{=} \frac{1}{K}\sum_{k=1}^K \left(w^t + \frac{K}{\sigma} s_k^t\right). \end{aligned}$$

With the change of variables $w := w^t + \frac{K}{\sigma}s$ (i.e., $s = \frac{\sigma}{K}(w - w^t)$), from (22) we know that $w_k^{t+1} := w^t + \frac{K}{\sigma}s_k^t$ solves

$$w_k^{t+1} = \arg \min_{w \in \mathbb{R}^d} \left\{ L_k(w) \overset{\text{def}}{=} \frac{1}{2n}\left\|X_k^T \frac{\sigma}{K}(w - w^t) - c_k\right\|^2 + \frac{\lambda/\sigma}{2}\left\|\frac{\sigma}{K}(w - w^t)\right\|^2 \right\} \tag{23}$$

and $w^{t+1} = \frac{1}{K}\sum_{k=1}^K w_k^{t+1}$.

Let us now rewrite the function in (23) so as to connect it to Algorithm 5:

$$\begin{aligned}
L_k(w) &= \frac{1}{2n}\left\|X_k^T\frac{\sigma}{K}(w-w^t)-c_k\right\|^2+\frac{\lambda/\sigma}{2}\left\|\frac{\sigma}{K}(w-w^t)\right\|^2\\
&= \frac{1}{2n}\frac{\sigma^2}{K^2}\left\|(X_k^Tw-y_k)-\underbrace{\left(X_k^Tw^t-y_k+\frac{K}{\sigma}c_k\right)}_{d_k}\right\|^2+\frac{\lambda\sigma^2}{2K^3}\|w\|^2-\frac{\lambda\sigma^2}{2K^3}\|w\|^2+\frac{\lambda/\sigma}{2}\left\|\frac{\sigma}{K}(w-w^t)\right\|^2\\
&= \frac{1}{2n}\frac{\sigma^2}{K^2}\left(\left\|X_k^Tw-y_k\right\|^2+\|d_k\|^2-2(X_k^Tw-y_k)^Td_k\right)+\frac{\lambda\sigma^2}{2K^3}\|w\|^2-\frac{\lambda\sigma^2}{2K^3}\|w\|^2+\frac{\lambda\sigma}{2K^2}\left\|w-w^t\right\|^2\\
&= \frac{\sigma^2}{K^3}\left(\frac{K}{2n}\left\|X_k^Tw-y_k\right\|^2+\frac{K}{2n}\|d_k\|^2-\frac{K}{n}(X_k^Tw-y_k)^Td_k\right)+\frac{\lambda\sigma^2}{2K^3}\|w\|^2\\
&\qquad -\frac{\lambda\sigma^2}{2K^3}\|w\|^2+\frac{\lambda\sigma}{2K^2}\left\|w-w^t\right\|^2\\
&= \frac{\sigma^2}{K^3}\underbrace{\left(\frac{K}{2n}\left\|X_k^Tw-y_k\right\|^2+\frac{\lambda}{2}\|w\|^2\right)}_{F_k(w)}+\frac{\sigma^2}{K^3}\left(\frac{K}{2n}\|d_k\|^2-\frac{K}{n}(X_k^Tw-y_k)^Td_k\right)\\
&\qquad -\frac{\lambda\sigma^2}{2K^3}\|w\|^2+\frac{\lambda\sigma}{2K^2}\left\|w-w^t\right\|^2\\
&= \frac{\sigma^2}{K^3}F_k(w)-\frac{\sigma^2}{K^2n}(X_k^Tw-y_k)^Td_k+\frac{\sigma^2}{2nK^2}\|d_k\|^2-\frac{\lambda\sigma^2}{2K^3}\|w\|^2+\frac{\lambda\sigma}{2K^2}\left\|w-w^t\right\|^2\\
&= \frac{\sigma^2}{K^3}F_k(w)-\frac{\sigma^2}{K^2n}(X_kd_k)^Tw-\frac{\lambda\sigma^2}{2K^3}\|w\|^2+\frac{\lambda\sigma}{2K^2}\left\|w-w^t\right\|^2+\underbrace{\left(\frac{\sigma^2}{2nK^2}\|d_k\|^2+\frac{\sigma^2}{K^2n}y_k^Td_k\right)}_{\beta_1}.
\end{aligned}$$

Next, since $\|w\|^2 = \|w-w^t\|^2-\|w^t\|^2+2(w^t)^Tw$, we can further write

$$\begin{aligned}
L_k(w) &= \frac{\sigma^2}{K^3}F_k(w) - \frac{\sigma^2}{K^2 n}(X_k d_k)^T w - \frac{\lambda\sigma^2}{2K^3}(\|w-w^t\|^2 - \|w^t\|^2 + 2(w^t)^T w) + \frac{\lambda\sigma}{2K^2}\left\|w-w^t\right\|^2 + \beta_1 \\
&= \frac{\sigma^2}{K^3}F_k(w) - \frac{\sigma^2}{K^2 n}(X_k d_k)^T w - \frac{\lambda\sigma^2}{K^3}(w^t)^T w + \left(\frac{\lambda\sigma}{2K^2} - \frac{\lambda\sigma^2}{2K^3}\right)\left\|w-w^t\right\|^2 + \underbrace{\frac{\lambda\sigma^2}{2K^3}\|w^t\|^2 + \beta_1}_{\beta_2} \\
&= \frac{\sigma^2}{K^3}\left(F_k(w) - \left(\frac{K}{n}X_k d_k + \lambda w^t\right)^T w + \frac{\lambda}{2}\left(\frac{K}{\sigma} - 1\right)\|w-w^t\|^2\right) + \beta_2 \\
&= \frac{\sigma^2}{K^3}\left(F_k(w) - \left(\nabla F_k(w^t) - \frac{K^2}{\sigma n}X_k(X_k^T w^t - y_k + \alpha_k^t)\right)^T w + \frac{\mu}{2}\|w-w^t\|^2\right) + \beta_2 \\
&= \frac{\sigma^2}{K^3}\left(F_k(w) - \left(\nabla F_k(w^t) - \frac{K}{\sigma}\underbrace{\frac{K}{n}X_k(X_k^T w^t - y_k + \alpha_k^t)}_{z_k^t}\right)^T w + \frac{\mu}{2}\|w-w^t\|^2\right) + \beta_2 \\
&= \frac{\sigma^2}{K^3}\left(F_k(w) - \left(\nabla F_k(w^t) - (\eta\nabla F_k(w^t) + g_k^t)\right)^T w + \frac{\mu}{2}\|w-w^t\|^2\right) + \beta_2,
\end{aligned}$$

where the last step follows from the claim that $\eta z_k^t = \eta\nabla F_k(w^t) + g_k^t$. We now prove the claim. First, we have

$$\begin{aligned}
\eta z_k^t &= \eta\frac{K}{n}X_k(X_k^T w^t - y_k + \alpha_k^t) \\
&= \eta\frac{K}{n}X_k(X_k^T w^t - y_k) + \eta\frac{K}{n}X_k\alpha_k^t \\
&= \eta\left(\frac{K}{n}X_k(X_k^T w^t - y_k) + \lambda w^t\right) + \eta\left(\frac{K}{n}X_k\alpha_k^t - \lambda w^t\right) \\
&\overset{(18)}{=} \eta\nabla F_k(w^t) + \eta\left(\frac{K}{n}X_k\alpha_k^t - \lambda w^t\right).
\end{aligned}$$

Due to the definition of g_k^0 in Step 5 of Algorithm 5 as $g_k^0 = \eta(\frac{K}{n}X_k\alpha_k^0 - \lambda w^0)$, we observe that the claim holds for $t = 0$. If we show that

$$g_k^t = \eta\left(\frac{K}{n}X_k\alpha_k^t - \lambda w^t\right)$$

for all $t \geq 0$, then we are done. This can be shown by induction. This finishes the proof of Theorem 5.