System-Aware Optimization for Machine Learning at Scale

Virginia Smith

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2017-140
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-140.html

August 9, 2017

System-Aware Optimization for Machine Learning at Scale

by

Virginia Smith

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

and the Designated Emphasis

in

Communication, Computation, and Statistics

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Michael I. Jordan, Co-chair
Professor David Culler, Co-chair
Professor Benjamin Recht
Assistant Professor Joan Bruna

Summer 2017

System-Aware Optimization for Machine Learning at Scale

Abstract

System-Aware Optimization for Machine Learning at Scale

by

Virginia Smith

Doctor of Philosophy in Computer Science
and the Designated Emphasis in
Communication, Computation, and Statistics

University of California, Berkeley

Professor Michael I. Jordan, Co-chair

Professor David Culler, Co-chair

New computing systems have emerged in response to the increasing size and complexity of modern datasets. For best performance, machine learning methods must be designed to closely align with the underlying properties of these systems.

In this thesis, we illustrate the impact of system-aware machine learning through the lens of *optimization*, a crucial component in formulating and solving most machine learning problems. Classically, the performance of an optimization method is measured in terms of accuracy (i.e., *does it realize the correct machine learning model?*) and convergence rate (*after how many iterations?*). In modern computing regimes, however, it becomes critical to additionally consider a number of systems-related aspects for best overall performance. These aspects can range from low-level details, such as data structures or machine specifications, to higher-level concepts, such as the tradeoff between communication and computation.

We propose a general optimization framework for machine learning, CoCoA, that gives careful consideration to systems parameters, often incorporating them directly into the method and theory. We illustrate the impact of CoCoA in two popular distributed regimes: the traditional cluster-computing environment, and the increasingly common setting of on-device (federated) learning. Our results indicate that by marrying systems-level parameters and optimization techniques, we can achieve orders-of-magnitude speedups for solving modern machine learning problems at scale. We corroborate these empirical results by providing theoretical guarantees that expose systems parameters to give further insight into empirical performance.

To my family

Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I would like to thank my advisors, Michael Jordan and David Culler. They have been exemplars of curious and committed researchers, enthusiastic teachers, and encouraging mentors. I am extremely grateful for the many opportunities they have afforded me and for the exceptional research environment they cultivated during my graduate studies. I would also like to thank the remaining members of my committee, Benjamin Recht and Joan Bruna, for their helpful advice and support.

It has been a pleasure to be part of several labs at Berkeley, including the AMPLab, SAIL, and LoCal/SDB. Within these groups, I have had the fortune of working with many great collaborators, including but not limited to: Sanjay Krishnan, Xinghao Pan, Evan Sparks, and Jay Taneja. Special thanks to Evan and Shivaram Venkataraman for their friendship and indispensable help with Spark, and to Jonathan Terhorst, Adam Bloniarz, and Sara Alspaugh for their cookies, friendship, and humor. I am particularly grateful to Isabelle Stanton for her mentorship during my internship at Google. Additional thanks to Kattt Atchley, Jon Kuroda, Audrey Sillers, Shirley Salanio, and all of the amazing EECS staff who were always there to help.

I have also had the opportunity to work with many excellent collaborators beyond Berkeley, including Martin Jaggi, Martin Takáč, Chenxin Ma, Simone Forte, Jakub Konečný, Peter Richtárik, Thomas Hofmann, Ameet Talwalkar, Chao-Kai Chiang, and Maziar Sanjabi. The work in this thesis is a product of these fruitful collaborations.

My involvements with WICSE and the Women in Technology Round Table have been some of the most rewarding in my PhD. Thank you to Gitanjali Swamy, Tsu-Jae King Liu, Camille Crittenden, Jo Yuen, and the members of WICSE for your support, collaboration, and continued commitment to strengthening the presence of women in tech. Thank you especially to Sheila Humphreys for your unwavering resolve, quick wit, and warm encouragement.

Finally, I would like to thank my family and friends, who are a constant source of support and inspiration. I am especially grateful to my parents, my brothers and their families, CHCM, and my wonderful husband Ameet. This thesis was possible only as a result of their love and encouragement.

Chapter 1

Introduction

Distributed computing architectures have come to the fore in modern machine learning, in response to the challenges arising from a wide range of large-scale learning applications. Distributed architectures offer the promise of scalability by increasing both computational and storage capacities. A critical challenge in realizing this promise of scalability is to develop efficient methods for communicating and coordinating information between distributed machines, taking into account the specific needs of machine-learning algorithms.

On most distributed systems, the communication of data between machines is vastly more expensive than reading data from main memory and performing local computation. Moreover, the optimal trade-off between communication and computation can vary widely depending on the dataset being processed, the system being used, and the objective being optimized. It is therefore essential for distributed methods to accommodate flexible communication-computation profiles while still providing convergence guarantees.

Although numerous distributed optimization methods have been proposed, the mini-batch optimization approach has emerged as one of the most popular paradigms for tackling this communication-computation tradeoff [cf. 72, 79, 83, 90]. Mini-batch methods are often developed by generalizing classical stochastic methods to process multiple data points at a time, which helps to alleviate the communication bottleneck by enabling more distributed computation per round of communication. However, while the need to reduce communication would suggest large mini-batch sizes, the theoretical convergence rates of these methods degrade with increased mini-batch size, reverting to the rates of classical (batch) gradient methods. Empirical results corroborate these theoretical rates, and in practice, mini-batch methods have limited flexibility to adapt to the communication-computation tradeoffs that would maximally leverage parallel execution. Moreover, because mini-batch methods are typically derived from a specific single-machine solver, these methods and their associated analyses are often tailored to specific problem instances and can suffer both theoretically and practically when applied outside of their restricted problem setting.

In this thesis, we propose a framework, CoCoA, that addresses these two fundamental limitations. First, we allow arbitrary local solvers to be used on each machine in parallel. This allows our framework to directly incorporate state-of-the-art, application-specific single-

machine solvers in the distributed setting. Second, we share information between machines in our framework with a highly flexible communication scheme. This allows the amount of communication to be easily tailored to the problem and system at hand, in particular allowing for the case of significantly reduced communication in the distributed environment.

A key step in providing these features in our framework is to first define meaningful subproblems for each machine to solve in parallel, and to then combine updates from the subproblems in an efficient manner. Our method and convergence results rely on noting that, depending on the distribution of the data (e.g., by feature or by training point), and whether we solve the problem in the primal or the dual, certain machine learning objectives can be more easily decomposed into subproblems in the distributed setting. In particular, we categorize common machine learning objectives into several cases, and use duality to help decompose these objectives. As we demonstrate, using primal-dual information in this manner not only allows for highly efficient methods (achieving, e.g., up to 50x speedups compared to state-of-the-art distributed methods), but also allows for strong primal-dual convergence guarantees and practical benefits such as computation of the duality gap for use as an accuracy certificate and stopping criterion.

1.1 Contributions

General framework. We develop a communication-efficient primal-dual framework that is applicable to a broad class of convex optimization problems. Notably, in contrast to earlier work of [38, 51, 100]; and [53], our generalized, cohesive framework: (1) specifically incorporates difficult cases of L_1 regularization and other non-strongly convex regularizers; (2) allows for the flexibility of distributing the data by either feature or training point; and (3) can be run on either a primal or dual formulation, which we show to have significant theoretical and practical implications.

Flexible communication and local solvers. Two key advantages of the proposed framework are its communication efficiency and ability to employ off-the-shelf single-machine solvers internally. On real-world systems, the cost of communication versus computation can vary widely, and it is thus advantageous to permit a flexible amount of communication depending on the setting at hand. Our framework provides exactly such control. Moreover, we allow arbitrary solvers to be used on each machine, which permits the reuse of existing code and the benefits from multi-core or other optimizations therein.

Primal-dual rates. We derive convergence rates for our framework, leveraging a novel approach in the analysis of primal-dual rates for non-strongly convex regularizers. The proposed technique is a significant improvement over simple smoothing techniques used in, e.g., [67, 84] and [110] that enforce strong convexity by adding a small L_2 term to the objective. Our results include primal-dual rates and certificates for the general class of linear regularized loss minimization, and we show how earlier work can be derived as a special case of our more general approach.

Experimental comparison. The proposed framework yields order-of-magnitude speedups (as much as 50× faster) compared to state-of-the-art methods for large-scale machine learning. We demonstrate these performance gains with an extensive experimental comparison on real-world distributed datasets. We additionally explore properties of the framework itself, including the effect of running the framework in the primal or the dual. All algorithms for comparison are implemented in Apache Spark and run on Amazon EC2 clusters. Our code is open-source and publicly available at: `github.com/gingsmith/proxcocoa`.

Federated learning. Finally, we explore the framework in various distributed computing settings, including the nascent area of *federated learning*, in which the aim is to perform optimization over large networks of low-powered devices. We propose an extension to CoCoA, Mocha, which is ideally suited to handle the unique systems and statistical challenges of the federated setting. We demonstrate both the superior statistical performance and empirical speedups of this method through simulations on real-world federated datasets, and provide a careful theoretical analysis that explores the effect of systems challenges such as stragglers and fault tolerance on our convergence guarantees.

1.2 Related Work

Single-machine coordinate solvers. For strongly convex regularizers, the current state-of-the-art for empirical loss minimization is randomized coordinate ascent on the dual (SDCA) [85] and its accelerated variants [cf. 84]. In contrast to primal stochastic gradient descent (SGD) methods, the SDCA family is often preferred as it is free of learning-rate parameters and has faster (geometric) convergence guarantees. Interestingly, a similar trend in coordinate solvers has been observed in the recent literature on the lasso, but with the roles of primal and dual reversed. For those problems, coordinate descent methods on the primal have become state-of-the-art, as in glmnet [28] and extensions [105] (cf. the overview in [106]). However, primal-dual convergence rates for unmodified coordinate algorithms have to our knowledge only been obtained for strongly convex regularizers to date [84, 110].

Coordinate descent on L_1-regularized problems (i.e., (A) with $g(\cdot) = \lambda\|\cdot\|_1$) can be interpreted as the iterative minimization of a quadratic approximation of the smooth part of the objective (as in a one-dimensional Newton step), followed by a shrinkage step resulting from the L_1 part. In the single-coordinate update case, this is at the core of glmnet [28, 106], and widely used in solvers based on the primal formulation of L_1-regularized objectives [cf. 12, 27, 82, 91, 105]. When changing more than one coordinate at a time, again employing a quadratic upper bound on the smooth part, this results in a two-loop method as in glmnet for the special case of logistic regression. This idea is crucial for the distributed setting. When the set of active coordinates coincides with the ones on the local machine, these single-machine approaches closely resemble the distributed framework proposed here.

Parallel methods. For the general regularized loss minimization problems of interest, methods based on stochastic subgradient descent (SGD) are well-established. Several variants of SGD have been proposed for parallel computing, many of which build on the idea of asynchronous communication [23, 68]. Despite their simplicity and competitive performance on shared-memory systems, the downside of this approach in the distributed environment is that the amount of required communication is equal to the amount of data read locally, since one data point is accessed per machine per round (e.g., mini-batch SGD with a batch size of one per worker). These variants are in practice not competitive with the more communication-efficient methods considered in this work, which allow more local updates per communication round.

For the specific case of L_1-regularized objectives, parallel coordinate descent (with and without using mini-batches) was proposed in [17] (Shotgun) and generalized in [12]; it is among the best performing solvers in the parallel setting. Our framework reduces to Shotgun as a special case when the internal solver is a single-coordinate update on the subproblem (2.10), $\gamma = 1$, and for a suitable σ'. However, Shotgun is not covered by our convergence theory, since it uses a potentially unsafe upper bound of β instead of σ', which is not guaranteed to satisfy our condition for convergence (2.11). We compare empirically with Shotgun in Chapter 4 to highlight the detrimental effects of running this high-communication method in the distributed environment.

One-shot communication schemes. At the other extreme, there are methods that use only a single round of communication [cf. 57, 62, 109, 112, 32]. These methods require additional assumptions on the partitioning of the data, which are usually not satisfied in practice if the data are distributed "as is", i.e., if we do not have the opportunity to distribute the data in a specific way beforehand. Furthermore, some cannot guarantee convergence rates beyond what could be achieved if we ignored data residing on all but a single computer, as shown in [86]. Additional relevant lower bounds on the minimum number of communication rounds necessary for a given approximation quality are presented in [8] and [7].

Mini-batch methods. Mini-batch methods (which use updates from several training points or features per round) are more flexible and lie within the two extremes of parallel and one-shot communication schemes. However, mini-batch versions of both SGD and coordinate descent (CD) (e.g., [72, 79, 83, 90]) suffer from their convergence rate degrading towards the rate of batch gradient descent as the size of the mini-batch is increased. This follows because mini-batch updates are made based on the outdated previous parameter vector $\mathbf{w}$, in contrast to methods that allow immediate local updates like CoCoA.

Another disadvantage of mini-batch methods is that the aggregation parameter is more difficult to tune, as it can lie anywhere in the order of mini-batch size. The optimal choice is often either unknown or too challenging to compute in practice. In the CoCoA framework there is no need to tune parameters, as the aggregation parameter and subproblem parameters can be set directly using the safe bound discussed in Chapter 2 (Definition 5).

Batch solvers. ADMM [16], gradient descent, and quasi-Newton methods such as L-BFGS and are also often used in distributed environments because of their relatively low communication requirements. However, they require at least a full (distributed) batch gradient computation at each round, and therefore do not allow the gradual trade-off between communication and computation provided by CoCoA. In Chapter 4, we include experimental comparisons with ADMM, gradient descent, and L-BFGS variants, including orthant-wise limited memory quasi-Newton (OWL-QN) for the L_1 setting [3].

Finally, we note that while the convergence rates provided for CoCoA mirror the convergence class of classical batch gradient methods in terms of the number of outer rounds, existing batch gradient methods come with a weaker theory, as they do not allow general inexactness Θ for the local subproblem (2.10). In contrast, our convergence rates incorporate this approximation directly, and, moreover, hold for arbitrary local solvers of much cheaper cost than batch methods (where in each round, every machine has to process exactly a full pass through the local data). This makes CoCoA more flexible in the distributed setting, as it can adapt to varied communication costs on real systems. We have seen in Chapter 4 that this flexibility results in significant performance gains over the competing methods.

Distributed solvers. By making use of the primal-dual structure in the line of work of [70, 100, 101, 103] and [48], the CoCoA-v1 and CoCoA$^+$ frameworks (which are special cases of the presented framework, CoCoA) are the first to allow the use of any local solver—of weak local approximation quality—in each round in the distributed setting. The practical variant of the DisDCA [100], called DisDCA-p, allows for additive updates in a similar manner to CoCoA, but is restricted to coordinate decent (CD) being the local solver, and was initially proposed without convergence guarantees. DisDCA-p, CoCoA-v1, and CoCoA$^+$ are all limited to strongly convex regularizers, and therefore are not as general as the CoCoA framework discussed in this work.

In the L_1-regularized setting, an approach related to our framework includes distributed variants of GLMNET as in [56]. Inspired by GLMNET and [105], the works of [12] and [56] introduced the idea of a block-diagonal Hessian upper approximation in the distributed L_1 context. The later work of [92] specialized this approach to sparse logistic regression.

If hypothetically each of our quadratic subproblems $\mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]})$ as defined in (2.10) were to be minimized exactly, the resulting steps could be interpreted as block-wise Newton-type steps on each coordinate block k, where the Newton-subproblem is modified to also contain the L_1-regularizer [56, 73, 105]. While [56] allows a fixed accuracy for these subproblems, but not arbitrary approximation quality Θ as in our framework, the works of [92, 105]; and [102] assume that the quadratic subproblems are solved exactly. Therefore, these methods are not able to freely trade off communication and computation. Also, they do not allow the re-use of arbitrary local solvers. On the theoretical side, the convergence rate results provided by [56, 92]; and [105] are not explicit convergence rates but only asymptotic, as the quadratic upper bounds are not explicitly controlled for safety as with our σ' (2.11).

1.3 Organization

The remainder of this thesis is organized as follows. Chapter 2 presents the CoCoA framework for distributed optimization. We begin by providing necessary background, including standard definitions from optimization and duality. Using the presented primal-dual structure, we then state problem cases for our framework which we elucidate with a set of running examples. Next, we detail CoCoA and explain how the framework may be run in either its primal or dual form in the distributed setting. Two critical components of the method are its updating scheme and subproblem formulation; we end the chapter by exploring several interpretations of these components and comparing the method to related work.

The proposed method is applicable to many common problems in machine learning and signal processing. Chapter 3 details several example applications that can be realized via the general CoCoA framework. For each application, we describe the primal-dual setup and algorithmic details; discuss the convergence properties of our framework for the application; and include practical concerns such as information on relevant local solvers.

Chapters 4 and 5 provide our empirical and theoretical results, respectively. In Chapter 4 we compare CoCoA to other state-of-the-art solvers in the distributed data center setting. Our results demonstrate order-of-magnitude speedups over competitors in solving common machine learning problems on real-world distributed datasets. To supplement these comparisons, we explore various properties of the framework, including the tradeoffs of primal vs. dual distributed optimization, the impact of communication on the framework, and the effect of the subproblem formulation. In Chapter 5, we derive our convergence guarantees for the framework and prove all other results given in the prior chapters. Our convergence results include primal-dual rates and certificates for the general class of linear regularized loss minimization. A key contribution in these results is our ability to abstract the performance of a local solver as an approximate solution of the subproblem defined on each distributed machine. Our convergence guarantees are derived by relating this local convergence to improvement made towards the global solution.

Finally, in Chapter 6, we explore an extension of CoCoA to federated learning, an increasingly common scenario in which the training of a machine learning model takes place directly on distributed devices. This setting presents new *statistical* challenges in our modeling approach, as well as new *systems* challenges in the training of these models. In particular, issues such as non-IID data, stragglers, and fault tolerance are much more prevalent than in a typical data center setting. To handle these challenges, we propose Mocha, a method that one (1) leverages multi-task learning, (2) performs alternating minimization of the objective, and (3) extends CoCoA to perform federated MTL updates. The resulting framework has superior statistical performance and a highly flexible optimization scheme relative to competitors. We demonstrate the empirical performance of this method through simulations on real-world federated datasets, and provide a careful theoretical analysis that explores the effect of the challenges prevalent in federated learning on our convergence guarantees.

Chapter 2

CoCoA Framework

In this work, we develop a general framework for minimizing problems of the following form:

$$\ell(\mathbf{u}) + r(\mathbf{u})\,, \tag{I}$$

for convex functions ℓ and r. Frequently the first term ℓ is an empirical loss over the data, taking the form $\sum_i \ell_i(\mathbf{u})$, and the second term r is a regularizer, e.g., $r(\mathbf{u}) = \lambda\|\mathbf{u}\|_p$. This formulation includes many popular methods in machine learning and signal processing, such as support vector machines, linear and logistic regression, lasso and sparse logistic regression, and many others.

2.1 Notation

The following standard definitions will be used throughout the thesis.

Definition 1 (L-Lipschitz Continuity)**.** *A function $h : \mathbb{R}^m \to \mathbb{R}$ is* L-Lipschitz continuous *if $\forall \boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^m$, we have*

$$|h(\boldsymbol{u}) - h(\boldsymbol{v})| \leq L\|\boldsymbol{u} - \boldsymbol{v}\|\,. \tag{2.1}$$

Definition 2 (L-Bounded Support)**.** *A function $h : \mathbb{R}^m \to \mathbb{R} \cup \{+\infty\}$ has* L-bounded support *if its effective domain is bounded by L, i.e.,*

$$h(\boldsymbol{u}) < +\infty \;\Rightarrow\; \|\boldsymbol{u}\| \leq L\,. \tag{2.2}$$

Definition 3 ($(1/\mu)$-Smoothness)**.** *A function $h : \mathbb{R}^m \to \mathbb{R}$ is* $(1/\mu)$-smooth *if it is differentiable and its derivative is $(1/\mu)$-Lipschitz continuous, or equivalently*

$$h(\boldsymbol{u}) \leq h(\boldsymbol{v}) + \langle \nabla h(\boldsymbol{v}), \boldsymbol{u} - \boldsymbol{v}\rangle + \frac{1}{2\mu}\|\boldsymbol{u} - \boldsymbol{v}\|^2 \qquad \forall \boldsymbol{u}, \boldsymbol{w} \in \mathbb{R}^m\,. \tag{2.3}$$

Definition 4 (μ-Strong Convexity)**.** *A function $h : \mathbb{R}^m \to \mathbb{R}$ is* μ-strongly convex *for $\mu \geq 0$ if*

$$h(\boldsymbol{u}) \geq h(\boldsymbol{v}) + \langle s, \boldsymbol{u} - \boldsymbol{v}\rangle + \frac{\mu}{2}\|\boldsymbol{u} - \boldsymbol{v}\|^2 \qquad \forall \boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^m\,, \tag{2.4}$$

for any $s \in \partial h(\boldsymbol{v})$, where $\partial h(\boldsymbol{v})$ denotes the subdifferential of h at $\boldsymbol{v}$.

2.2 Duality

Numerous methods have been proposed to solve (I), and these methods generally fall into two categories: *primal methods*, which run directly on the primal objective, and *dual methods*, which instead run on the dual formulation of the primal objective. In developing our framework, we present an abstraction that allows for either a primal or a dual variant of our framework to be run. In particular, to solve the input problem (I), we consider mapping the problem to one of the following two general problems:

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \left[\mathcal{O}_A(\boldsymbol{\alpha}) := f(A\boldsymbol{\alpha}) + g(\boldsymbol{\alpha}) \right] \tag{A}$$

$$\min_{\mathbf{w}\in\mathbb{R}^d} \left[\mathcal{O}_B(\mathbf{w}) := f^*(\mathbf{w}) + g^*(-A^\top \mathbf{w}) \right] \tag{B}$$

Here $\boldsymbol{\alpha} \in \mathbb{R}^n$ and $\mathbf{w} \in \mathbb{R}^d$ are parameter vectors, $A := [\mathbf{x}_1; \ldots; \mathbf{x}_n] \in \mathbb{R}^{d\times n}$ is a data matrix with column vectors $\mathbf{x}_i \in \mathbb{R}^d$, $i \in \{1, \ldots, n\}$, and the functions f^* and g_i^* are the *convex conjugates* of f and g_i, respectively.

The dual relationship in problems (A) and (B) is known as Fenchel-Rockafellar duality [14, Theorem 4.4.2]. We provide a self-contained derivation of the duality in Section 5.1.2. Note that while dual problems are typically presented as a pair of (min, max) problems, we have equivalently reformulated (A) and (B) to both be minimization problems in accordance with their roles in our framework.

Given $\boldsymbol{\alpha} \in \mathbb{R}^n$ in the context of (A), a corresponding vector $\mathbf{w} \in \mathbb{R}^d$ for problem (B) is obtained by:

$$\mathbf{w} = \mathbf{w}(\boldsymbol{\alpha}) := \nabla f(A\boldsymbol{\alpha}) \,. \tag{2.5}$$

This mapping arises from first-order optimality conditions on the f-part of the objective. The duality gap, given by:

$$G(\boldsymbol{\alpha}) := \mathcal{O}_A(\boldsymbol{\alpha}) - [-\mathcal{O}_B(\mathbf{w}(\boldsymbol{\alpha}))] \tag{2.6}$$

is always non-negative, and under strong duality, the gap will reach zero only for an optimal pair $(\boldsymbol{\alpha}^\star, \mathbf{w}^\star)$. The duality gap at any point provides a practically computable upper bound on the unknown primal as well as dual optimization error (suboptimality), since

$$\mathcal{O}_A(\boldsymbol{\alpha}) \geq \mathcal{O}_A(\boldsymbol{\alpha}^\star) \geq -\mathcal{O}_B(\mathbf{w}^\star) \geq -\mathcal{O}_B(\mathbf{w}(\boldsymbol{\alpha})) \,.$$

In developing the proposed framework, noting the duality between (A) and (B) has many benefits, including the ability to compute the duality gap, which acts as a certificate of the approximation quality. It is also useful as an analysis tool, helping us to present a cohesive framework and relate this work to the prior work of [100, 38]; and [53, 51]. As a word of caution, note that we avoid prescribing the name "primal" or "dual" directly to either of the problems (A) or (B), as we demonstrate below that their role as primal or dual can change depending on the application problem of interest.

2.3 Assumptions and Problem Cases

Our main assumptions on problem (A) are that f is $(1/\tau)$-smooth, and the function g is separable, i.e., $g(\boldsymbol{\alpha}) = \sum_i g_i(\alpha_i)$, with each g_i having L-bounded support. Given the duality between the problems (A) and (B), this can be equivalently stated as assuming that in problem (B), f^* is τ-strongly convex, and the function $g^*(-A^\top \mathbf{w}) = \sum_i g_i^*(-\mathbf{x}_i^\top \mathbf{w})$ is separable with each g_i^* being L-Lipschitz.

For clarity, in Table 2.1 we relate our assumptions on objectives (A) and (B) to the general input problem (I). Suppose, as in equation (I), we would like to find a minimizer of the general objective $\ell(\mathbf{u}) + r(\mathbf{u})$. Depending on the smoothness of the function ℓ and the strong convexity of the function r, we will be able to map the input function (I) to one (or both) of the objectives (A) and (B) based on our assumptions.

In particular, we outline three separate cases: Case I, in which the function ℓ is smooth and the function r is strongly convex; case II, in which ℓ is smooth, and r is non-strongly convex and separable; and case III, in which ℓ is non-smooth and separable, and r is strongly convex. The union of these cases will capture most commonly-used applications of linear regularized loss minimization problems. In Section 2.9, we will see that different variants of our framework may be realized depending on which of these three cases we consider when solving the input problem (I).

Table 2.1: Criteria for objectives (A) and (B).

	Smooth ℓ	Non-smooth, separable ℓ
Strongly convex r	Case I: Obj (A) or (B)	Case III: Obj (B)
Non-strongly convex, separable r	Case II: Obj (A)	–

2.4 Running Examples

To illustrate the three cases in Table 2.1, we consider several examples below. These applications will serve as running examples throughout this thesis, and we will revisit them in our experiments (Chapter 4). Further applications and details are provided in Chapter 3.

1. *Elastic Net Regression (Case I: map to either (A) or (B)).* We can map elastic-net regularized least squares regression,

$$\min_{\mathbf{u}\in\mathbb{R}^p} \ \tfrac{1}{2}\|A\mathbf{u} - \mathbf{b}\|_2^2 + \eta\lambda\|\mathbf{u}\|_1 + (1-\eta)\frac{\lambda}{2}\|\mathbf{u}\|_2^2\,, \tag{2.7}$$

to either objective (A) or (B). To map to objective (A), we let: $f(A\boldsymbol{\alpha}) = \frac{1}{2}\|A\boldsymbol{\alpha} - \mathbf{b}\|_2^2$ and $g(\boldsymbol{\alpha}) = \sum_i g_i(\alpha_i) = \sum_i \eta\lambda|\alpha_i| + (1-\eta)\frac{\lambda}{2}\alpha_i^2$, setting n to be the number of features and d the number of training points. To map to (B), we let: $g(-A^\top\mathbf{w}) = \sum_i g_i^*(-\mathbf{x}_i^\top\mathbf{w}) = \sum_i \frac{1}{2}(\mathbf{x}_i^\top\mathbf{w} - \mathbf{b}_i)^2$ and $f^*(\mathbf{w}) = \eta\lambda\|\mathbf{w}\|_1 + (1-\eta)\frac{\lambda}{2}\|\mathbf{w}\|_2^2$, setting d to be the number of features and n the number of training points. We discuss in Section 2.9 how the choice of mapping elastic net regression to either (A) or to (B) will result in one of two variants of our framework, and can have implications on the distribution scheme and overall performance of the method.

2. *Lasso (Case II: map to (A)).* We can represent L_1-regularized least squares regression by mapping the model:

$$\min_{\mathbf{u}\in\mathbb{R}^p} \ \frac{1}{2}\|A\mathbf{u} - \mathbf{b}\|_2^2 + \lambda\|\mathbf{u}\|_1 \tag{2.8}$$

to objective (A), letting $f(A\boldsymbol{\alpha}) = \frac{1}{2}\|A\boldsymbol{\alpha} - \mathbf{b}\|_2^2$ and $g(\boldsymbol{\alpha}) = \sum_i g_i(\alpha_i) = \sum_i \lambda|\alpha_i|$. In this mapping, n represents the number of features, and d the number of training points. Note that we cannot map the lasso objective to (B) directly, as f^* must be τ-strongly convex and the L_1-norm is non-strongly convex.

3. *Support Vector Machine (Case III: map to (B)).* We can represent a hinge loss support vector machine (SVM) by mapping the model:

$$\min_{\mathbf{u}\in\mathbb{R}^p} \ \frac{1}{m}\sum_{i=1}^{m} \max\left\{0, 1 - y_i(\mathbf{x}_i^\top\mathbf{u})\right\} + \tfrac{\lambda}{2}\|\mathbf{u}\|_2^2\,, \tag{2.9}$$

to objective (B), letting $g^*(-A^\top\mathbf{w}) = \sum_i g_i^*(-\mathbf{x}_i^\top\mathbf{w}) = \sum_i \frac{1}{n}\max\{0, 1 - y_i\mathbf{x}_i^\top\mathbf{w}\}$ and $f^*(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|_2^2$. In this mapping, d represents the number of features, and n the number of training points. Note that we cannot map the hinge loss SVM primal to objective (A) directly, as f must be $(1/\tau)$-smooth and the hinge loss is non-smooth.

2.5 Data Partitioning

To view our setup in the distributed environment, we suppose that the dataset A is distributed over K machines according to a partition $\{\mathcal{P}_k\}_{k=1}^K$ of the *columns* of $A \in \mathbb{R}^{d\times n}$. We denote the size of the partition on machine k by $n_k = |\mathcal{P}_k|$. For machine $k \in \{1, \ldots, K\}$ and weight vector $\boldsymbol{\alpha} \in \mathbb{R}^n$, we define $\boldsymbol{\alpha}_{[k]} \in \mathbb{R}^n$ as the n-vector with elements $(\boldsymbol{\alpha}_{[k]})_i := \alpha_i$ if $i \in \mathcal{P}_k$ and $(\boldsymbol{\alpha}_{[k]})_i := 0$ otherwise. Analogously, we write $A_{[k]}$ for the corresponding group of columns of A, and zeros elsewhere (note that columns can correspond to either training examples or features, depending on the application). We discuss these distribution schemes in greater detail in Section 2.9.

Algorithm 1 Generalized CoCoA Distributed Framework

Input: Data matrix A distributed column-wise according to partition $\{\mathcal{P}_k\}_{k=1}^K$, aggregation parameter $\gamma \in (0, 1]$, and parameter σ' for the local subproblems $\mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]}; \mathbf{v}, \boldsymbol{\alpha}_{[k]})$.
Starting point $\boldsymbol{\alpha}^{(0)} := \mathbf{0} \in \mathbb{R}^n$, $\mathbf{v}^{(0)} := \mathbf{0} \in \mathbb{R}^d$.
for $t = 0, 1, 2, \ldots$ **do**
  for $k \in \{1, 2, \ldots, K\}$ **in parallel over computers do**
    compute Θ-approximate solution $\Delta\boldsymbol{\alpha}_{[k]}$ of local subproblem (2.10)
    update local variables $\boldsymbol{\alpha}_{[k]}^{(t+1)} := \boldsymbol{\alpha}_{[k]}^{(t)} + \gamma\, \Delta\boldsymbol{\alpha}_{[k]}$
    return updates to shared state $\Delta\mathbf{v}_k := A_{[k]}\Delta\boldsymbol{\alpha}_{[k]}$
  reduce $\mathbf{v}^{(t+1)} := \mathbf{v}^{(t)} + \gamma \sum_{k=1}^K \Delta\mathbf{v}_k$

2.6 Method

The goal of our framework is to find a global minimizer of the objective (A), while distributing computation based on the partitioning of the dataset A across machines (Section 2.5). As a first step, note that distributing the update to the function g in objective (A) is straightforward, as we have required that this term is separable according to the partitioning of our data, i.e., $g(\boldsymbol{\alpha}) = \sum_{i=1}^n g_i(\alpha_i)$. However, the same does not hold for the term $f(A\boldsymbol{\alpha})$. To minimize this part of the objective in a distributed fashion, we propose minimizing a quadratic approximation of the function, which allows the minimization to separate across machines. We make this approximation precise in the following subsection.

Data-local quadratic subproblems. In the general CoCoA framework (Algorithm 1), we distribute computation by defining a data-local subproblem of the optimization problem (A) for each machine. This simpler problem can be solved on machine k and only requires accessing data which is already available locally, i.e., the columns $A_{[k]}$. More formally, each machine k is assigned the following local subproblem, which depends only on the previous shared vector $\mathbf{v} := A\boldsymbol{\alpha} \in \mathbb{R}^d$, and the local data $A_{[k]}$:

$$\min_{\Delta\boldsymbol{\alpha}_{[k]} \in \mathbb{R}^n} \; \mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]}; \mathbf{v}, \boldsymbol{\alpha}_{[k]})\,, \tag{2.10}$$

where

$$\mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]}; \mathbf{v}, \boldsymbol{\alpha}_{[k]}) := \frac{1}{K} f(\mathbf{v}) + \mathbf{w}^\top A_{[k]}\Delta\boldsymbol{\alpha}_{[k]} + \frac{\sigma'}{2\tau}\left\| A_{[k]}\Delta\boldsymbol{\alpha}_{[k]} \right\|^2 + \sum_{i \in \mathcal{P}_k} g_i(\alpha_i + \Delta\alpha_{[k]_i}),$$

and $\mathbf{w} := \nabla f(\mathbf{v})$. Here we let $\Delta\boldsymbol{\alpha}_{[k]}$ denote the change of local variables α_i for indices $i \in \mathcal{P}_k$, and we set $(\Delta\boldsymbol{\alpha}_{[k]})_i := 0$ for all $i \notin \mathcal{P}_k$. It is important to note that the subproblem (2.10) is simple in the sense that it is always a quadratic objective (apart from the g_i term). The subproblem does not depend on the function f itself, but only its linearization at the fixed shared vector $\mathbf{v}$. This property additionally simplifies the task of the local solver, especially for cases of complex functions f.

Framework parameters γ and σ'. There are two parameters that must be set in our framework: γ, the aggregation parameter, which controls how the updates from each machine are combined, and σ', the subproblem parameter, which is a data-dependent term measuring the difficulty of the data partitioning $\{\mathcal{P}_k\}_{k=1}^K$. These terms play a crucial role in the convergence of the method, as we demonstrate in Chapter 5. In practice, we provide a simple and robust way to set these parameters: For a given aggregation parameter $\gamma \in (0, 1]$, the subproblem parameter σ' will be set as $\sigma' := \gamma K$, but can also be improved in a data-dependent way as we discuss below. In general, as we show in Chapter 5, setting $\gamma := 1$ and $\sigma' := K$ will guarantee convergence while delivering our fastest convergence rates.

Definition 5 (Data-dependent aggregation parameter)**.** *In Algorithm 1, the aggregation parameter γ controls the level of adding ($\gamma := 1$) versus averaging ($\gamma := \frac{1}{K}$) of the partial solutions from all machines. For our convergence results (Chapter 5) to hold, the subproblem parameter σ' must be chosen not smaller than*

$$\sigma' \geq \sigma'_{min} := \gamma \max_{\boldsymbol{\alpha} \in \mathbb{R}^n} \frac{\|A\boldsymbol{\alpha}\|^2}{\sum_{k=1}^K \|A_{[k]}\boldsymbol{\alpha}_{[k]}\|^2} \,. \tag{2.11}$$

The simple choice of $\sigma' := \gamma K$ is valid for (2.11), i.e.,

$$\gamma K \geq \sigma'_{min} \,.$$

In some cases, it will be possible to give a better (data-dependent) choice for σ', closer to the actual bound given in σ'_{min}.

Subproblem Interpretation. Here we provide further intuition behind the data-local subproblems (2.10). The local objective functions $\mathcal{G}_k^{\sigma'}$ are defined to closely approximate the global objective in (A) as the "local" variable $\Delta\boldsymbol{\alpha}_{[k]}$ varies, which we will see in the analysis (Chapter 5, Lemma 15). In fact, if the subproblem were solved exactly, this could be interpreted as a data-dependent, block-separable proximal step, applied to the f part of the objective (A) as follows:

$$\sum_{k=1}^K \mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]}; \mathbf{v}, \boldsymbol{\alpha}_{[k]}) \;=\; R + f(\mathbf{v}) + \nabla f(\mathbf{v})^\top A\Delta\boldsymbol{\alpha} + \frac{\sigma'}{2\tau}\Delta\boldsymbol{\alpha}^\top \begin{bmatrix} A_{[1]}^\top A_{[1]} & & 0 \\ & \ddots & \\ 0 & & A_{[K]}^\top A_{[K]} \end{bmatrix} \Delta\boldsymbol{\alpha} \,,$$

where $R = \sum_{i \in [n]} g_i(-\alpha_i - \Delta\alpha_i)$.

However, note that in contrast to traditional proximal methods, our algorithm does *not* assume that this subproblem is solved to high accuracy, as we instead allow the use of local solvers of any approximation quality Θ.

Reusability of existing single-machine solvers. Our local subproblems (2.10) have the appealing property of being very similar in structure to the global problem (A), with the main difference being that they are defined on a smaller (local) subset of the data, and are simpler because they are not dependent on the shape of f. For a user of CoCoA, this presents a major advantage in that existing single machine-solvers can be directly re-used in our distributed framework (Algorithm 1) by employing them on the subproblems $\mathcal{G}_k^{\sigma'}$.

Therefore, problem-specific tuned solvers which have already been developed, along with associated speed improvements (such as multi-core implementations), can be easily leveraged in the distributed setting. We quantify the dependence on local solver performance with the following assumption and remark, and relate this performance to our global convergence rates in Chapter 5.

Assumption 1 (Θ-approximate solution)**.** *We assume that there exists $\Theta \in [0,1)$ such that $\forall k \in [K]$, the local solver at any outer iteration t produces a (possibly) randomized approximate solution $\Delta\boldsymbol{\alpha}_{[k]}$, which satisfies*

$$\mathbb{E}\left[\mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]}; \boldsymbol{v}, \boldsymbol{\alpha}_{[k]}) - \mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]}^\star; \boldsymbol{v}, \boldsymbol{\alpha}_{[k]})\right] \leq \Theta\left(\mathcal{G}_k^{\sigma'}(\boldsymbol{0}; \boldsymbol{v}, \boldsymbol{\alpha}_{[k]}) - \mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]}^\star; \boldsymbol{v}, \boldsymbol{\alpha}_{[k]})\right), \quad (2.12)$$

where

$$\Delta\boldsymbol{\alpha}_{[k]}^\star \in \arg\min_{\Delta\boldsymbol{\alpha}\in\mathbb{R}^n} \mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]}; \boldsymbol{v}, \boldsymbol{\alpha}_{[k]}), \quad \forall k \in [K]\,. \quad (2.13)$$

Remark 1. *In practice, the time spent solving the local subproblems in parallel should be chosen comparable to the required time of a communication round, for best overall efficiency on a given system. We study this trade-off both in theory (Chapter 5) and experiments (Chapter 4).*

Remark 2. *Note that the accuracy parameter Θ does not have to be chosen a priori: Our convergence results (Chapter 5) are valid if Θ is an upper bound on the actual empirical values Θ in the rounds of Algorithm 1. This allows for some of the K machines to at times deliver better or worse accuracy (e.g., if a slow local machine is stopped early during a specific round, to avoid the others needing to wait).*

With this general framework in place, we next discuss two variants of our framework, CoCoA-Primal and CoCoA-Dual. In running either the primal or dual variant of our framework, the goal will always be to solve objective (A) in a distributed fashion. The main difference will be whether this objective is viewed as the primal or dual of the input problem (I). If we map the input (I) to objective (A), then (A) will be viewed as the primal. If we map (I) to (B), the objective (A) will be viewed as the dual. We make this mapping technique precise and discuss its implications in the following sections (Sections 2.7–2.9).

2.7 CoCoA in the Primal

In the primal distributed version of the framework (Algorithm 2), the framework is run by mapping the initial problem (I) directly to objective (A) and then applying the generalized CoCoA framework described in Algorithm 1. In other words, we view problem (A) as the primal objective, and solve this problem directly.

From a theoretical perspective, viewing (A) as the primal will allow us to consider non-strongly convex regularizers, since we allow the terms g_i to be non-strongly convex. This setting was not covered in earlier work of [38, 53, 100]; and [51], and we discuss it in detail in Chapter 5, as additional machinery must be introduced to develop primal-dual rates for this setting.

Running the primal version of the framework has important practical implications in the distributed setting, as it typically implies that the data is distributed by feature rather than by training point. In this setting, the amount of communication at every outer iteration will be $O(\#$ of training points). When the number of features is high (as is common when using sparsity-inducing regularizers) this can help to reduce communication and improve overall performance, as we demonstrate in Chapter 4.

Algorithm 2 CoCoA-Primal (Mapping Problem (I) to (A))

1: **Map:** Input problem (I) to objective (A)
2: **Distribute:** Dataset A by columns (here typically features) according to partition $\{\mathcal{P}_k\}_{k=1}^K$
3: **Run:** Algorithm 1 with aggregation parameter γ and subproblem parameter σ'

2.8 CoCoA in the Dual

In the dual distributed version of the framework (Algorithm 3), we run the framework by mapping the original problem (I) to objective (B), and then solve the problem by running Algorithm 1 on the dual (A). In other words, we view problem (B) as the primal, and solve this problem via the dual (A).

This version of the framework will allow us to consider non-smooth losses, such as the hinge loss or absolute deviation loss, since the terms g_i^* can be non-smooth. From a practical perspective, this version of the framework will typically imply that the data is distributed by training point, and for a vector $O(\#$ of features) to be communicated at every outer iteration. This variant may therefore be preferable when the number of training points exceeds the number of features.

Algorithm 3 CoCoA-Dual (Mapping Problem (I) to (B))

1: **Map:** Input problem (I) to objective (B)
2: **Distribute:** Dataset A by columns (here typically training points) according to partition $\{\mathcal{P}_k\}_{k=1}^K$
3: **Run:** Algorithm 1 with aggregation parameter γ and subproblem parameter σ'

2.9 Primal vs. Dual

In Table 2.2, we revisit the three cases from Section 2.3 showing how the primal and dual variants of CoCoA can be applied to various input problems $\ell(\mathbf{u}) + r(\mathbf{u})$, depending on properties of the functions ℓ and r. In particular, in the setting where ℓ is smooth and r is strongly convex, the user may choose whether to run the framework in the primal (Algorithm 2), or in the dual (Algorithm 3). Intuitively, Algorithm 2 will be preferable as r loses strong convexity, and Algorithm 3 will be preferable as ℓ loses smoothness. However, there are also systems-related aspects to consider. In Algorithm 2, we typically distribute the data by feature, and in Algorithm 3, by training point (this distribution depends on how the terms n and d are defined in our mapping, see Chapter 3). Depending on whether the number of features or number of training points is the dominating term, we may chose to run Algorithm 2 or Algorithm 3, respectively, in order to reduce communication costs. We validate these ideas empirically in Chapter 4 by comparing the performance of each variant (primal vs. dual) on real distributed datasets.

Table 2.2: Criteria for running Algorithm 2 vs. Algorithm 3.

	Smooth ℓ	Non-smooth and separable ℓ
Strongly convex r	Case I: Alg.2 or 3	Case III: Alg.3
Non-strongly convex and separable r	Case II: Alg.2	–

In the following two sections, we provide greater insight into the form of the generalized CoCoA framework and its relation to prior work. An extended discussion on related work is available in Section 1.2.

2.10 Interpretation

There are numerous methods that have been developed to solve (A) and (B) in parallel and distributed environments. We describe related work in detail in Section 1.2, and here briefly highlight a major algorithmic difference between CoCoA and other widely-used parallelized

methods. In particular, we contrast CoCoA with mini-batch and batch methods commonly used in distributed computing environments, such as mini-batch stochastic gradient descent or coordinate descent, gradient descent, and quasi-Newton methods.

CoCoA is similar to these methods in that they are all *iterative*, i.e., they make progress towards the optimal solution by updating the parameter vector $\boldsymbol{\alpha}$ according to some function $h : \mathbb{R}^n \to \mathbb{R}^n$ at each iteration t:

$$\boldsymbol{\alpha}^{(t+1)} = h(\boldsymbol{\alpha}^{(t)}) \quad t = 0, 1, \ldots,$$

until convergence is reached. From a coordinate-wise perspective, two approaches for updating the parameter vector $\boldsymbol{\alpha}$ in an iterative fashion include the Jacobi method, in which updates made to coordinates of $\boldsymbol{\alpha}$ do not take into account the most recent updates to the other coordinates, and Gauss-Seidel, in which the most recent information is used [11]. In particular, these two paradigms make the following updates to a coordinate i at iteration $t + 1$:

$$\begin{aligned} \text{Jacobi:} \quad & \alpha_i^{(t+1)} = h_i(\alpha_1^{(t)}, \ldots, \alpha_n^{(t)}), \quad i = 1, \ldots, n, \\ \text{Gauss-Seidel:} \quad & \alpha_i^{(t+1)} = h_i(\alpha_1^{(t+1)}, \ldots, \alpha_{i-1}^{(t+1)}, \alpha_i^{(t)}, \ldots, \alpha_n^{(t)}), \quad i = 1, \ldots, n. \end{aligned}$$

The Jacobi method does not require information from the other coordinates to update coordinate i, which makes this style of method well-suited for parallelization. However, the Gauss-Seidel style method tends to converge faster in terms of iterations, since it is able to incorporate information from the other coordinates more quickly. This difference is well-known and evident in single machine solvers, where stochastic methods (benefiting from fresh updates) tend to outperform their batch counterparts.

Typical mini-batch methods, e.g., mini-batch coordinate descent, perform a Jacobi-style update on a subset of the coordinates at each iteration. This makes these methods amenable to high levels of parallelization. However, they are unable to incorporate information as quickly as their serial counterparts in terms of number of data points accessed, because they must wait for a synchronization step to update the coordinates. As the size of the mini-batch grows, this can slow them down in terms of overall runtime, and can even lead to divergence in practice [58, 79, 89, 90].

CoCoA instead attempts to combine attractive properties of both of these update paradigms. It performs Jacobi-style parallel updates to *blocks* of the coordinates of $\boldsymbol{\alpha}$ to parallelize the method, while allowing for (though not necessarily requiring) faster Gauss-Seidel style updates on each machine. This change in parallelization scheme is one of the major reasons for improved performance over simpler mini-batch or batch style methods.

CoCoA incorporates an additional level of flexibility by allowing an *arbitrary number* of Gauss-Seidel iterations (or any other local solver for that matter) to be performed on each machine, which lets the framework scale from very low-communication environments,

where more iterations will be made before communicating, to higher communication environments, where fewer internal iterations are necessary. We will see in Chapter 4 that this communication flexibility also greatly improves the overall runtime in practice.

2.11 Comparison to ADMM

Finally, in this section we provide a direct comparison between CoCoA and ADMM [16]. Alternating direction method of multipliers (ADMM) is a well-established framework for distributed optimization. Similar to CoCoA, ADMM differs from the methods discussed in the previous section in that it defines a subproblem for each problem to solve in parallel, rather than parallelizing a global batch or mini-batch update. It also leverages duality structure, similar to that presented in Section 2.2.

For consensus ADMM, the objective (B) is decomposed with a re-parameterization:

$$\begin{aligned} \max_{\mathbf{w}_1,\dots\mathbf{w}_K,\mathbf{w}} \quad & \sum_{k=1}^{K}\sum_{i\in\mathcal{P}_k} g^*(-\mathbf{x}_i^\top \mathbf{w}_k) + f^*(\mathbf{w}) \\ s.t. \quad & \mathbf{w}_k = \mathbf{w},\ k = 1,\dots,K. \end{aligned}$$

This problem is then solved by constructing the augmented Lagrangian, which yields the following decomposable updates:

$$\begin{aligned} \mathbf{w}_k^{(t)} &= \arg\min_{\mathbf{w}_k} \sum_{i\in\mathcal{P}_k} g^*(-\mathbf{x}_i^\top \mathbf{w}_k) + \frac{\rho}{2}\|\mathbf{w}_k - \left(\mathbf{w}^{(t-1)} - \mathbf{u}_k^{(t-1)}\right)\|^2, \qquad (2.14) \\ \mathbf{w}^{(t)} &= \arg\min_{\mathbf{w}} f^*(\mathbf{w}) + \rho\sum_{k=1}^{K}\mathbf{u}_k^\top(\mathbf{w}_k - \mathbf{w}) + \frac{\rho}{2}\sum_{k=1}^{K}\|\mathbf{w}_k - \mathbf{w}\|^2, \\ \mathbf{u}_k^{(t)} &= \mathbf{u}_k^{(t-1)} + \mathbf{w}_k^{(t)} - \mathbf{w}^{(t)}, \end{aligned}$$

where ρ is a penalty parameter that must be tuned for best performance. When running CoCoA in the dual (Algorithm 3) and setting $f(\cdot) = \frac{1}{2}\|\cdot\|_2^2$, we can derive a similar subproblem for updating $\mathbf{w}_k$ in the CoCoA framework. In particular, the following subproblem can be found by unrolling the CoCoA update and viewing the dual subproblem in its primal formulation:

$$\min_{\mathbf{w}_k} \sum_{i\in\mathcal{P}_k} g_i^*(-\mathbf{x}_i^\top \mathbf{w}_k) + \frac{\tau}{2\sigma'}\left\|\mathbf{w}_k - \left(\mathbf{w}^{(t-1)} + \gamma\Delta\mathbf{v}^{(t-1)}\right)\right\|^2. \qquad (2.15)$$

Comparing (2.14) and (2.15) we can see that in the specific case where $f(\cdot) = \frac{1}{2}\|\cdot\|_2^2$ and we solve the problem in the dual (according to Algorithm 3), ADMM and CoCoA consider a similar subproblem on each machine, but where the parameter ρ is explicitly set in CoCoA as $\frac{\tau}{\sigma'}$. However, there are major differences between the methods even in this setting. First, CoCoA has a more direct and simplified scheme for updating the global weight vector $\mathbf{w}$. Second, and most importantly, in the CoCoA method and theory, we allow for the subproblem to be solved approximately, rather than requiring a full batch update as in ADMM. We will see in our experiments that these differences have a large impact in practice (Chapter 4). We provide a full derivation of the comparison to ADMM for reference in Section 5.1.3.

Chapter 3

Applications

In this chapter, we provide a detailed treatment of example applications that can be cast within the general CoCoA framework. For each example, we describe the primal-dual setup and algorithmic details, discuss the convergence properties our framework for the application, and include practical concerns such as information on state-of-the-art local solvers. We discuss examples according to the three cases defined in Table 2.1 of Chapter 2 for finding a minimizer of the general objective $\ell(\mathbf{u}) + r(\mathbf{u})$, and provide a summary of these common examples in Table 3.1.

Table 3.1: Common losses and regularizers.

(i) Losses

Loss	Obj	f / g^*
Least Squares	(A)	$f{=}\frac{1}{2}\|A\boldsymbol{\alpha}-\mathbf{b}\|_2^2$
	(B)	$g^*{=}\frac{1}{2}\|A^\top\mathbf{w}-\mathbf{b}\|_2^2$
Logistic Reg.	(A)	$f{=}\frac{1}{d}\sum_j\log(1{+}\exp(b_j\mathbf{x}_j^\top\boldsymbol{\alpha}))$
	(B)	$g^*{=}\frac{1}{n}\sum_i\log(1{+}\exp(b_i\mathbf{x}_i^\top\mathbf{w}))$
SVM	(B)	$g^*{=}\frac{1}{n}\sum_i\max(0,1{-}y_i\mathbf{x}_i^\top\mathbf{w})$
Absolute Dev.	(B)	$g^*=\frac{1}{n}\sum_i\|\mathbf{x}_i^\top\mathbf{w}-y_i\|$

(ii) Regularizers

Regularizer	Obj	g / f^*
Elastic Net	(A)	$g{=}\lambda(\eta\|\boldsymbol{\alpha}\|_1{+}\frac{1-\eta}{2}\|\boldsymbol{\alpha}\|_2^2)$
	(B)	$f^*{=}\lambda(\eta\|\mathbf{w}\|_1{+}\frac{1-\eta}{2}\|\mathbf{w}\|_2^2)$
L_2	(A)	$g{=}\frac{\lambda}{2}\|\boldsymbol{\alpha}\|_2^2$
	(B)	$f^*{=}\frac{\lambda}{2}\|\mathbf{w}\|_2^2$
L_1	(A)	$g{=}\lambda\|\boldsymbol{\alpha}\|_1$
Group Lasso	(A)	$g{=}\lambda\sum_p\|\boldsymbol{\alpha}_{\mathcal{I}_p}\|_2$, $\mathcal{I}_p\subseteq[n]$

3.1 Smooth ℓ, Strongly Convex r

For input problems (I) with smooth ℓ and strongly convex r, Theorem 6 from Chapter 5 gives a global linear (geometric) convergence rate. Smooth loss functions can be mapped either to the function f in objective (A), or g^* in (B). Similarly, strongly convex regularizers can be mapped either to function g in objective (A), or f^* in (B). To illustrate the role of f as a smooth loss function and g as a strongly convex regularizer in objective (A), contrasting with

their traditional roles in prior work [38, 51, 53, 100], we consider the following examples. Note that mapping to objective (B) instead will follow trivially assuming that the loss is separable across training points (see Table 3.1).

For the examples in this section, we use nonstandard definitions of the number of training points as d and the number of features as n. These definitions are intentionally used so that we can present both the primal and dual variations of our framework (Algorithms 2 and 3) with a single abstracted method (Algorithm 1).

Smooth ℓ: least squares loss. Let $\mathbf{b} \in \mathbb{R}^d$ be labels or response values, and consider the least squares objective, $f(\mathbf{v}) := \frac{1}{2}\|\mathbf{v} - \mathbf{b}\|_2^2$, which is 1-smooth. We obtain the familiar least-squares regression objective in our optimization problem (A), using

$$f(A\boldsymbol{\alpha}) := \tfrac{1}{2}\|A\boldsymbol{\alpha} - \mathbf{b}\|_2^2 \,. \tag{3.1}$$

Observing that the gradient of f is $\nabla f(\mathbf{v}) = \mathbf{v} - \mathbf{b}$, the primal-dual mapping is given by: $\mathbf{w}(\boldsymbol{\alpha}) := \nabla f(\mathbf{v}(\boldsymbol{\alpha})) = A\boldsymbol{\alpha} - \mathbf{b}$, which is well known as the *residual vector* in least-squares regression.

Smooth ℓ: logistic regression loss. For classification problems, we consider a logistic regression model with d training examples, $\mathbf{y}_j \in \mathbb{R}^n$ for $j \in [d]$, collected as the rows of the data matrix A. For each training example, we are given a binary label, which we collect in the vector $\mathbf{b} \in \{-1, 1\}^d$. Formally, the objective is defined as $f(\mathbf{v}) := \sum_{j=1}^{d} \log\left(1 + \exp\left(-b_j v_j\right)\right)$, which is again a separable function. The classifier loss is given by

$$f(A\boldsymbol{\alpha}) := \sum_{j=1}^{d} \log\left(1 + \exp\left(-b_j \mathbf{y}_j^\top \boldsymbol{\alpha}\right)\right), \tag{3.2}$$

where $\boldsymbol{\alpha} \in \mathbb{R}^n$ is the parameter vector. It is not hard to show that f is 1-smooth if the labels satisfy $b_j \in [-1, 1]$. The primal-dual mapping $\mathbf{w}(\boldsymbol{\alpha}) := \nabla f(\mathbf{v}(\boldsymbol{\alpha})) = \nabla f(A\boldsymbol{\alpha})$ is given by $w_j(\boldsymbol{\alpha}) := \frac{-b_j}{1+\exp\left(b_j \mathbf{y}_j^\top \boldsymbol{\alpha}\right)}$.

Strongly convex r: elastic net regularizer. An application we can consider for a strongly convex regularizer, g in (A) or f^* in (B), is elastic net regularization, $\eta\lambda\|\mathbf{u}\|_1 + (1-\eta)\frac{\lambda}{2}\|\mathbf{u}\|_2^2$, for fixed parameter $\eta \in (0, 1]$. This can be obtained in (A) by setting

$$g(\boldsymbol{\alpha}) = \sum_{i=1}^{n} g_i(\alpha_i) := \sum_{i=1}^{n} \eta\lambda|\alpha_i| + (1-\eta)\tfrac{\lambda}{2}\alpha_i^2. \tag{3.3}$$

For the special case $\eta = 1$, we obtain the L_1-norm, and for $\eta = 0$, we obtain the L_2-norm. The conjugate of g_i is given by: $g_i^*(x) := \frac{1}{2(1-\eta)}\left(\left[|x| - \eta\right]_+\right)^2$, where $[.]_+$ is the positive part operator, $[s]_+ = s$ for $s > 0$, and zero otherwise.

3.2 Smooth ℓ, Non-strongly Convex r

In case II, we consider mapping the input problem (I) to objective (A), where ℓ is assumed to be smooth, and r non-strongly convex and separable. For smooth losses in (A), we can consider as examples those provided in Section 3.1, e.g., the least squares loss or logistic loss. For an example of a non-strongly convex regularizer, we consider the important case of L_1 regularization below. Again, we note that this application cannot be realized by objective (B), where it is assumed that the regularization term f^* is strongly convex.

Non-strongly convex r: L_1 regularizer. L_1 regularization is obtained in objective (A) by letting $g_i(\cdot) := \lambda|\cdot|$. However, an additional modification is necessary to obtain primal-dual convergence and certificates for this setting. In particular, we employ the modification introduced in Chapter 5, which will guarantee L-bounded support. Formally, we replace $g_i(\cdot) = |\cdot|$ by

$$\bar{g}(\alpha) := \begin{cases} |\alpha| & : \alpha \in [-B, B], \\ +\infty & : \text{otherwise.} \end{cases}$$

For large enough B, this problem yields the same solution as the original L_1-objective. Note that this only affects convergence theory, in that it allows us to present a strong primal-dual rate (Theorem 5 for $L{=}B$). With this modified L_1-regularizer, the optimization problem (A) with regularization parameter λ becomes

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \; f(A\boldsymbol{\alpha}) + \lambda \sum_{i=1}^{n} \bar{g}(\alpha_i)\,. \tag{3.4}$$

For large enough choice of the value B, this problems yields the same solution as the original objective:

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \quad \left\{ \; \mathcal{O}_A(\boldsymbol{\alpha}) := \; f(A\boldsymbol{\alpha}) + \lambda \sum_{i=1}^{n} |\alpha_i| \; \right\}. \tag{3.5}$$

The modified $\bar{g}$ is simply a constrained version of the absolute value to the interval $[-B, B]$. Therefore by setting B to a large enough value that the values of α_i will never reach it, $\bar{g}^*$ will be continuous and at the same time make (3.4) equivalent to (3.5).

Formally, a simple way to obtain a large enough value of B, so that all solutions of (3.5) are unaffected, is the following: If we start the algorithm at $\boldsymbol{\alpha} = \mathbf{0}$, for every solution encountered during execution, the objective values will never become worse than $\mathcal{O}_A(\mathbf{0})$. Formally, under the assumption that f is non-negative, we will have that (for each i):

$$\lambda|\alpha_i| \leq f(\mathbf{0}) = \mathcal{O}_A(\mathbf{0}) \implies |\alpha_i| \leq \frac{f(\mathbf{0})}{\lambda}.$$

We can therefore safely set the value of B as $\frac{f(\mathbf{0})}{\lambda}$. For the modified $\bar{g}_i$, the conjugate $\bar{g}_i^*$ is given by:

$$\bar{g}_i^*(x) := \begin{cases} 0 & : x \in [-1, 1], \\ B(|x| - 1) & : \text{otherwise.} \end{cases}$$

We provide a proof of this in Section 5.1.2 (Lemma 3).

Non-strongly convex r: group lasso. The group lasso penalty can be mapped to objective (A), with:

$$g(\boldsymbol{\alpha}) := \lambda \sum_{p=1}^{P} \|\boldsymbol{\alpha}_{\mathcal{I}_p}\|_2 \quad \text{with} \quad \bigcup_{p=1}^{P} \mathcal{I}_p = \{1, \ldots, n\}, \tag{3.6}$$

where the disjoint sets $\mathcal{I}_p \subseteq \{1, \ldots, n\}$ represent a partitioning of the total set of variables. This penalty can be viewed as an intermediate between a pure L_1 or L_2 penalty, performing variable selection only at the group level. The term $\boldsymbol{\alpha}_{\mathcal{I}_p} \in \mathbb{R}^{|\mathcal{I}_p|}$ denotes part of the vector $\boldsymbol{\alpha}$ with indices $\mathcal{I}_p$. The conjugate is given by:

$$g^*(\mathbf{w}) = I_{\{\mathbf{w} | \max_{\mathcal{I}_p \in [n]} \|\boldsymbol{\alpha}_{\mathcal{I}_p}\|_2 \leq \lambda\}}(\mathbf{w}).$$

For details, see, e.g., [24] or Boyd and Vandenberghe [15, Example 3.26].

3.3 Non-smooth ℓ, Strongly Convex r

Finally, in case III, we consider mapping the input problem (I) to objective (B), where ℓ is assumed to be non-smooth and separable, and r strongly convex. We discuss two common cases of general non-smooth losses ℓ, including the the hinge loss for classification and absolute deviation loss for regression. When paired with a strongly convex regularizer, the regularizer via f gives rise to the primal-dual mapping, and Theorem 5 provides a sublinear convergence rate for objectives of this form. We note that these losses cannot be realized directly by objective (A), where it is assumed that the data fit term f is smooth.

Non-smooth ℓ: hinge loss. For classification problems, we can consider a hinge loss support vector machine model, on n training points in $\mathbb{R}^d$, given with the loss:

$$g^*(-A^\top \mathbf{w}) = \sum_{i=1}^{n} g_i^*(-\mathbf{x}_i^\top \mathbf{w}) := \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i \mathbf{x}_i^\top \mathbf{w}\}. \tag{3.7}$$

The conjugate function of the hinge loss $\phi(a) = \max\{0, 1-b\}$ is given by $\phi^*(b) = \{b$ if $b \in [-1, 0]$, else ∞ .$\}$. When using the L_2 norm for regularization in this problem: $f^*(\mathbf{w}) := \lambda\|\mathbf{w}\|_2^2$, a primal-dual mapping is given by: $\mathbf{w}(\boldsymbol{\alpha}) := \frac{1}{\lambda n} A\boldsymbol{\alpha}$.

Non-smooth ℓ: absolute deviation loss. The absolute deviation loss, used, e.g., in quantile regression or least absolute deviation regression, can be realized in objective (B) by setting:

$$g^*(-A^\top \mathbf{w}) = \sum_{i=1}^{n} g_i^*(-\mathbf{x}_i^\top \mathbf{w}) := \frac{1}{n}\sum_{i=1}^{n} \left|\mathbf{x}_i^\top \mathbf{w} - y_i\right|. \tag{3.8}$$

The conjugate function of the absolute deviation loss $\phi(a) = |a - y_i|$ is given by $\phi^*(-b) = -by_i$, with $b \in [-1, 1]$.

3.4 Local Solvers

As discussed in Chapter 2, the subproblems solved on each machine in the CoCoA framework are appealing in that they are very similar in structure to the global problem (A), with the main difference being that they are defined on a smaller (local) subset of the data, and have a simpler dependence on the term f. Therefore, solvers which have already proven their value in the single machine or multicore setting can be easily leveraged within the framework. We discuss some specific examples of local solvers below, and point the reader to [51] for an empirical exploration of these choices.

In the primal setting (Algorithm 2), the local subproblem (2.10) becomes a simple quadratic problem on the local data, with regularization applied only to local variables $\boldsymbol{\alpha}_{[k]}$. For the L_1 examples discussed, existing fast L_1-solvers for the single-machine case, such as GLMNET variants [28] or BLITZ [40] can be directly applied to each local subproblem $\mathcal{G}_k^{\sigma'}(\,\cdot\,; \mathbf{v}, \boldsymbol{\alpha}_{[k]})$ within Algorithm 1. The sparsity induced on the subproblem solutions of each machine naturally translates into the sparsity of the global solution, since the local variables $\boldsymbol{\alpha}_{[k]}$ will be concatenated.

In terms of the approximation quality parameter Θ for the local problems (Assumption 1), we can apply existing recent convergence results from the single machine case. For example, for randomized coordinate descent (as part of GLMNET), Lu and Xiao [50, Theorem 1] gives a $O(1/t)$ approximation quality for any separable regularizer, including L_1 and elastic net; see also [91] and [82].

In the dual setting (Algorithm 3) for the discussed examples, the losses are applied only to local variables $\boldsymbol{\alpha}_{[k]}$, and the regularizer is approximated via a quadratic term. Current state of the art for the problems of the form in (B) are variants of randomized coordinate ascent—Stochastic Dual Coordinate Ascent (SDCA) [85]. This algorithm and its variants are increasingly used in practice [98], and extensions such as accelerated and parallel versions can directly be applied [26, 84] in our framework. For non-smooth losses such as SVMs, the analysis of [85] provides a $O(1/t)$ rate, and for smooth losses, a faster linear rate. There have also been recent efforts to derive a linear convergence rate for problems like the hinge-loss SVM that could be applied, e.g., by using error bound conditions [66, 97], weak strong convexity conditions [52, 65] or by considering Polyak-Łojasiewicz conditions [41].

Chapter 4

Evaluation

In this chapter we demonstrate the empirical performance of CoCoA in the distributed data center setting. We first compare CoCoA to competing methods for two common machine learning applications: lasso regression (Section 4.2.1) and support vector machine (SVM) classification (Section 4.2.2). We then explore the performance of CoCoA in the primal versus the dual directly by solving an elastic net regression model with both variants (Section 4.3.1). Finally, we illustrate general properties of the CoCoA method empirically in Section 4.3.2.

4.1 Details and Setup

We compare CoCoA to numerous state-of-the-art general-purpose methods for large-scale optimization, including:

- Mb-SGD: Mini-batch stochastic gradient. For our experiments with lasso, we compare against Mb-SGD with an L_1-prox.
- GD: Full gradient descent. For lasso we use the proximal version, Prox-GD.
- L-BFGS: Limited-memory quasi-Newton method. For lasso, we use OWL-QN (orthant-wise limited quasi-Newton).
- ADMM: Alternating direction method of multipliers. We use conjugate gradient internally for the lasso experiments, and SDCA for SVM experiments.
- Mb-CD: Mini-batch parallel coordinate descent. For SVM experiments, we implement Mb-SDCA (mini-batch stochastic dual coordinate ascent).

The first three methods are optimized and implemented in Apache Spark's MLlib (v1.5.0) [63]. We test the performance of each method in large-scale experiments fitting lasso, elastic net regression, and SVM models to the datasets shown in Table 4.1. In comparing to other methods, we plot the distance to the optimal primal solution. This optimal value is calculated by running all methods for a large number of iterations (until progress has stalled), and then

selecting the smallest primal value amongst the results. All code is written in Apache Spark and experiments are run on public-cloud Amazon EC2 m3.xlarge machines with one core per machine. Our code is publicly available at `github.com/gingsmith/proxcocoa`.

Table 4.1: Datasets for empirical study.

Dataset	**Training Size**	**Feature Size**	**Sparsity**
url	2 M	3 M	3.5e-5
epsilon	400 K	2 K	1.0
kddb	19 M	29 M	9.8e-7
webspam	350 K	16 M	2.0e-4

We carefully tune each competing method in our experiments for best performance. ADMM requires the most tuning, both in selecting the penalty parameter ρ and in solving the subproblems. Solving the subproblems to completion for ADMM is prohibitively slow, and we thus use an iterative method internally and improve performance by allowing early stopping. We also use a varying penalty parameter ρ — practices described in Boyd et al. [16, Sections 4.3, 8.2.3, 3.4.1]. For Mb-SGD, we tune the step size and mini-batch size parameters. For Mb-CD and Mb-SDCA, we scale the updates at each round by $\frac{\beta}{b}$ for mini-batch size b and $\beta \in [1, b]$, and tune both parameters b and β. Further implementation details for all methods are given in Section 4.1.1.

For simplicity of presentation and comparison, in all of the following experiments, we restrict CoCoA to only use simple coordinate descent as the local solver. We note that even stronger empirical results for CoCoA could be obtained by plugging in state of the art local solvers for each application at hand.

4.1.1 Methods for Comparison

In this section we provide thorough details on the experimental setup and methods used in our comparison. All experiments are run on Amazon EC2 clusters of m3.xlarge machines, with one core per machine. The code for each method is written in Apache Spark, v1.5.0. Our code is open source and publicly available at `github.com/gingsmith/proxcocoa`.

ADMM. Alternating Direction Method of Multipliers (ADMM) [16] is a popular method that lends itself naturally to the distributed environment. For lasso regression, implementing ADMM for the problems of interest requires solving a large linear system $C\mathbf{x} = \mathbf{d}$ on each machine, where $C \in \mathbb{R}^{n \times n}$ with n scaling beyond 10^7 for the datasets in Table 4.1, and with C being possibly dense. It is prohibitively slow to solve this directly on each machine, and we therefore employ the iterative method of conjugate gradient with early stopping [cf. 16, Section 4.3]. For SVM classification, we use stochastic dual coordinate ascent as an

internal optimizer, which is shown in [107] to have superior performance. We further improve performance by using a varying rather than constant penalty parameter, as suggested in Boyd et al. [16, Section 3.4.1].

Mini-batch SGD and proximal GD. Mini-batch SGD is a standard and widely used method for parallel and distributed optimization. We use the optimized code provided in Spark's machine learning library, MLlib, v1.5.0 [63]. We tune both the size of the mini-batch and the SGD step size using grid search. For lasso, we use the proximal version of the method. Full gradient descent can be seen as a specific setting of mini-batch SGD, where the mini-batch size is equal to the total number of training points. We thus also use the implementation in MLlib for full GD, and tune the step size parameter using grid search.

Mini-batch CD and SDCA. Mini-batch CD (for lasso) and SDCA (for SVM) aim to improve mini-batch SGD by employing coordinate descent, which has theoretical and practical justifications [27, 82, 89, 90, 91]. We implement mini-batch CD and SDCA in Spark and scale the updates made at each round by $\frac{\beta}{b}$ for mini-batch size b and $\beta \in [1, b]$, tuning both parameters b and β via grid search. For the case of lasso regression, we implement Shotgun [17], which is a popular method for parallel optimization. Shotgun can be seen an extreme case of mini-batch CD where the mini-batch is set to K, i.e., there is a single update made by each machine per round. We see in the experiments that communicating this frequently becomes prohibitively slow in the distributed environment.

OWL-QN. OWN-QN [104] is a quasi-Newton method optimized in Spark's spark.ml package [63]. Outer iterations of OWL-QN make significant progress towards convergence, but the iterations themselves can be slow because they require processing the entire dataset. CoCoA, the mini-batch methods, and ADMM with early stopping all improve on this by allowing the flexibility of only a subset of the dataset to be processed at each iteration. CoCoA and ADMM have even greater flexibility by allowing internal methods to process the dataset more than once. CoCoA makes this approximation quality explicit, both in theoretical convergence rates and by providing general guidelines for setting the parameter.

CoCoA. We implement CoCoA with coordinate descent as the local solver. We note that since the framework and theory allow any internal solver to be used, CoCoA could benefit even beyond the results shown, e.g., by using existing fast L_1-solvers for the single-machine case, such as GLMNET variants [28] or BLITZ [40] or SVM solvers like LIBLINEAR [26]. The only parameter necessary to tune for CoCoA is the level of approximation quality, which we parameterize in the experiments through H, the number of local iterations of the iterative method run locally. Our theory relates local approximation quality to global convergence (Chapter 5), and we provide a guideline for how to choose this value in practice that links the parameter to the systems environment at hand (Remark 1).

4.2 Comparison to Other Methods

In the following sections we explore the performance of CoCoA in the primal and CoCoA in the dual compared to other distributed optimization methods.

4.2.1 CoCoA in the Primal

We demonstrate the performance of CoCoA in the primal (Algorithm 2) by fitting a lasso regression model (2.8) to the distributed datasets in Table 4.1. We use stochastic coordinate descent as a local solver for CoCoA, and select the number of local iterations H (a proxy for subproblem approximation quality, Θ) from several options with best performance.

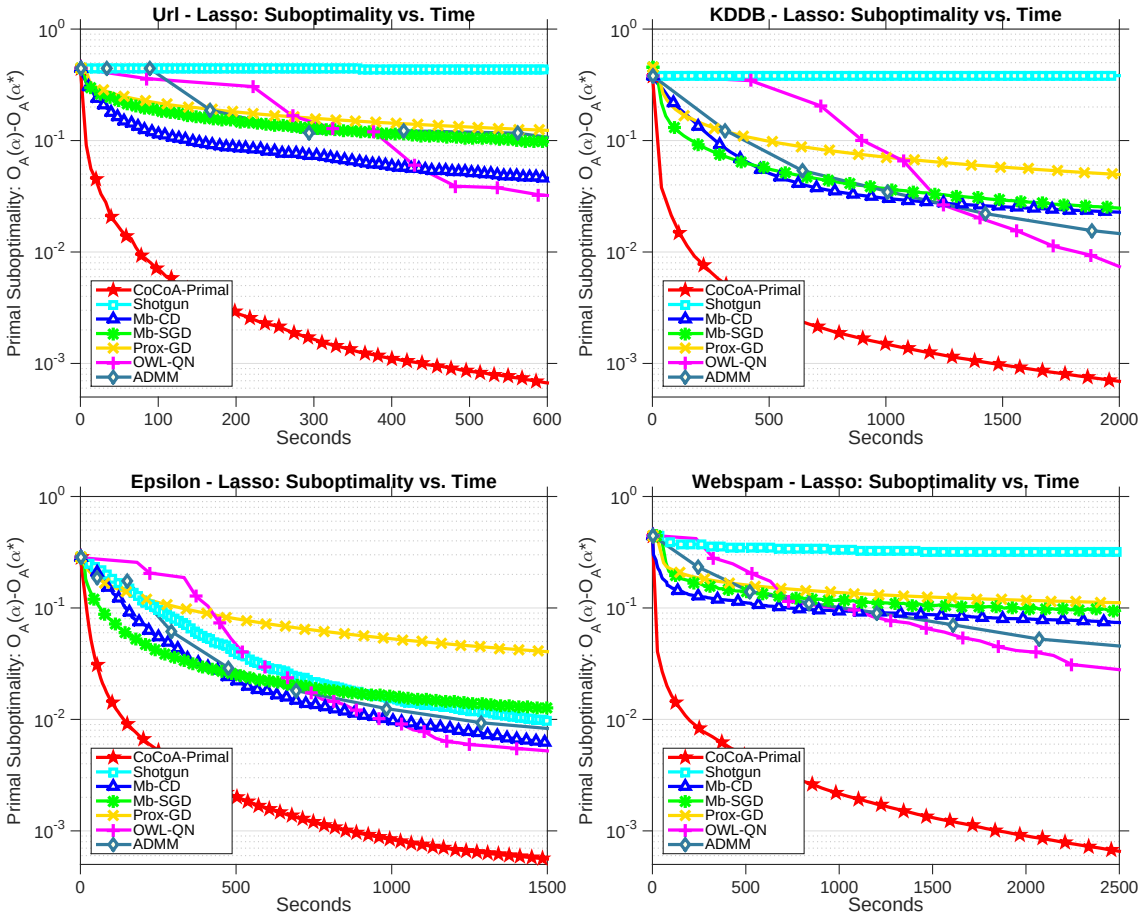

Figure 4.1: Suboptimality in terms of $\mathcal{O}_A(\boldsymbol{\alpha})$ for fitting a lasso regression model to four datasets: url (K=4, λ=1E-4), kddb (K=4, λ=1E-6), epsilon (K=8, λ=1E-5), and webspam (K=16, λ=1E-5) datasets. CoCoA applied to the primal formulation converges more quickly than all other compared methods in terms of the time in seconds.

We compare CoCoA to the general methods listed above, including Mb-SGD with an L_1-prox, Prox-GD, OWL-QN, ADMM and Mb-CD. A comparison with Shotgun [17], a popular method for solving L_1-regularized problems in the multicore environment, is provided as an extreme case to highlight the detrimental effects of frequent communication in the distributed environment. For Mb-CD, Shotgun, and CoCoA in the primal, datasets are distributed by feature, whereas for Mb-SGD, Prox-GD, OWL-QN and ADMM they are distributed by training point.

In analyzing the performance of each algorithm (Figure 4.1), we measure the improvement to the primal objective given in (A) ($\mathcal{O}_A(\boldsymbol{\alpha})$) in terms of wall-clock time in seconds. We see that both Mb-SGD and Mb-CD are slow to converge, and come with the additional burden of having to tune extra parameters (though Mb-CD makes clear improvements over Mb-SGD). As expected, naively distributing Shotgun (single coordinate updates per machine) does not perform well, as it is tailored to shared-memory systems and requires communicating too frequently. OWL-QN performs the best of all compared methods, but is still much slower to converge than CoCoA, and converges, e.g., 50× more slowly for the webspam dataset. The optimal performance of CoCoA is particularly evident in datasets with large numbers of features (e.g., url, kddb, webspam), which are exactly the datasets of interest for L_1 regularization.

Results are shown for regularization parameters λ such that the resulting weight vector $\boldsymbol{\alpha}$ is sparse. However, our results are robust to varying values of λ as well as to various problem settings, as we illustrate in Figure 4.2.

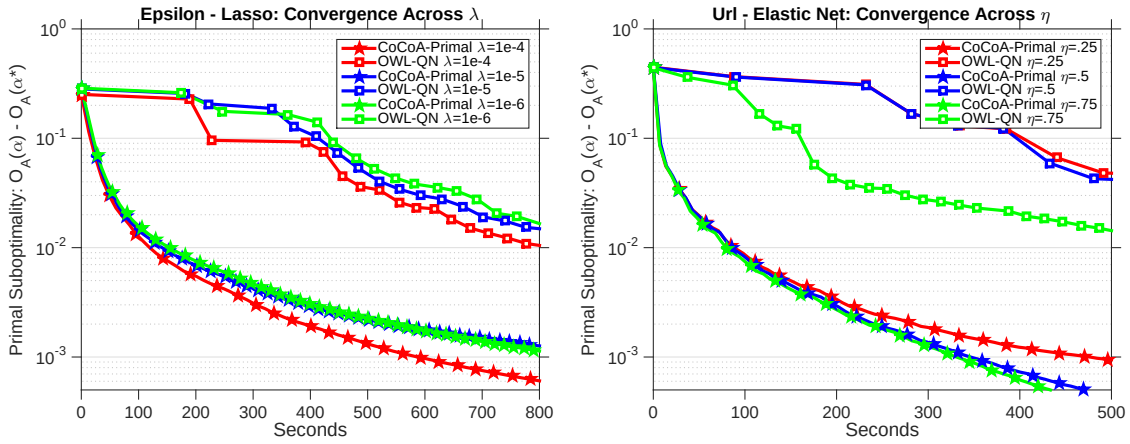

Figure 4.2: Suboptimality in terms of $\mathcal{O}_A(\boldsymbol{\alpha})$ for fitting a lasso regression model to the epsilon dataset (left, K=8) and an elastic net regression model to the url dataset, (right, K=4, λ=1e-4). Speedups are robust over different regularizers λ (left), and across problem settings, including varying η parameters of elastic net regularization (right).

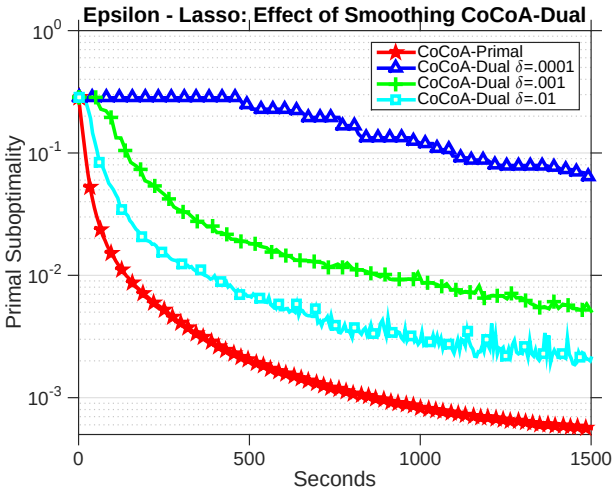

Figure 4.3: For pure L_1 regularization, Nesterov smoothing is not an effective option for CoCoA in the dual. It either slows convergence (as shown in the plot above), or modifies the solution (as shown in Table 4.2). This motivates running CoCoA instead on the primal for these problems.

Method	**Sparsity**
CoCoA-Primal	0.6030
CoCoA-Dual: $\delta = 0.0001$	0.6035
CoCoA-Dual: $\delta = 0.001$	0.6240
CoCoA-Dual: $\delta = 0.01$	0.6465

Table 4.2: The sparsity of the final iterate is affected by Nesterov smoothing (i.e., adding a small amount of strong convexity $\delta\|\boldsymbol{\alpha}\|_2^2$ to the objective for lasso regression). As δ increases, the convergence improves (as shown in Figure 4.3), but the final sparsity does not match that of pure L_1-regularized regression.

A case against smoothing. We additionally motivate the use of CoCoA in the primal by showing how it improves upon CoCoA in the dual [38, 51, 53, 100] for non-strongly convex regularizers. First, CoCoA in the dual cannot be included in the set of experiments in Figure 4.1 because it cannot be directly applied to the lasso objective (recall that Algorithm 3 only allows for strongly convex regularizers).

To get around this requirement, previous work has suggested implementing the Nesterov smoothing technique used in, e.g., [84, 110] — adding a small amount of strong convexity $\delta\|\boldsymbol{\alpha}\|_2^2$ to the objective for lasso regression. In Figure 4.3 and Table 4.2, we demonstrate the issues with this approach, comparing CoCoA in the primal on a pure L_1-regularized regression problem to CoCoA in the dual for decreasing levels of δ. The smaller we set δ, the less smooth the problem becomes. As δ decreases, the final sparsity of running CoCoA in the dual starts to match that of running pure L_1 (Table 4.2), but the performance also degrades (Figure 4.3). We note that by using CoCoA in the primal with the modification presented in Chapter 5, we can deliver strong rates without having to make these fundamental alterations to the problem of interest.

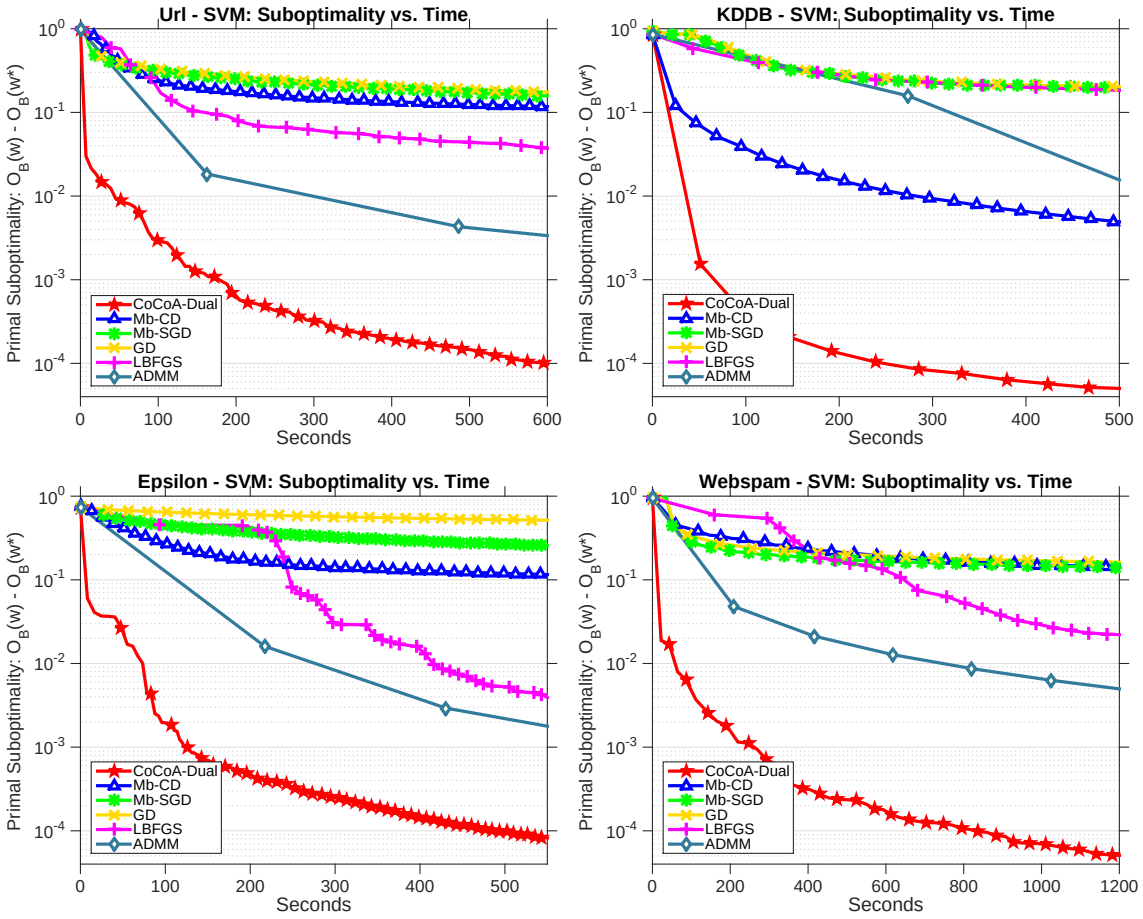

Figure 4.4: Suboptimality in terms of $\mathcal{O}_B(\mathbf{w})$ for fitting a hinge-loss support vector machine model to various datasets: url (K=4, λ=1E-4), kddb (K=4, λ=1E-6), epsilon (K=8, λ=1E-5), and webspam (K=16, λ=1E-5). CoCoA applied to the dual formulation converges more quickly than all other compared methods in terms of the time in seconds.

4.2.2 CoCoA in the Dual

Next we present results on CoCoA in the dual against competing methods, for an SVM model (2.9) on the datasets in Table 4.1. We use stochastic dual coordinate ascent (SDCA) as a local solver for CoCoA in this setting, again selecting the number of local iterations H from several options with best performance. We compare CoCoA to the general methods listed above, including Mb-SGD, GD, L-BFGS, ADMM, and Mb-SDCA. All datasets are distributed by training point for these methods.

In analyzing the performance the methods in this setting (Figure 4.4), we measure the improvement to the primal objective given in (B) ($\mathcal{O}_B(\mathbf{w})$) in terms of wall-clock time in seconds. We see again that Mb-SGD and Mb-CD are slow to converge, and come with

the additional burden of having to tune extra parameters. ADMM performs the best of the methods other than CoCoA, followed by L-BFGS. However, both are still much slower to converge than CoCoA in the dual. ADMM was in particular affected by the fact that many internal iterations of SDCA were necessary in order to guarantee convergence. In contrast, CoCoA is able to incorporate arbitrary amounts of work locally and still converge. We note that although CoCoA, ADMM and Mb-SDCA run in the dual, the plots in Figure 4.4 mark progress towards the primal objective, $\mathcal{O}_B(\mathbf{w})$.

4.3 Properties

Finally, we explore several properties of CoCoA, including the tradeoff between primal vs. dual distributed optimization, the effect of communication on the distributed method, and the impact of the subproblem parameter, σ', on overall convergence.

4.3.1 Primal vs. Dual

To understand the effect of primal versus dual optimization for CoCoA, we compare the performance of both variants by fitting an elastic net regression model (2.7) to two datasets. For comparability of the methods, we use coordinate descent (with closed-form updates) as the local solver in both variants. From the results in Figure 4.5, we see that CoCoA in the

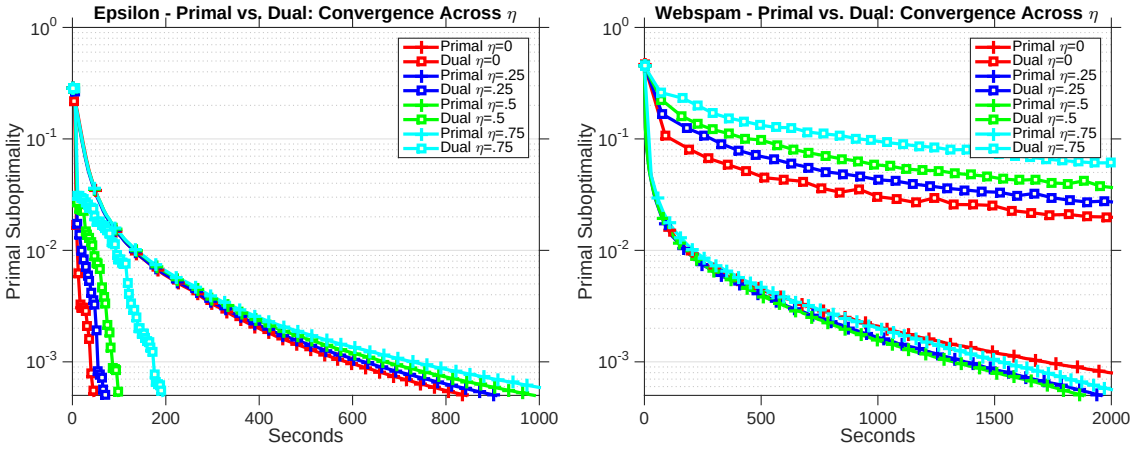

Figure 4.5: The convergence of CoCoA in the primal versus dual for various values of η in an elastic net regression model. CoCoA in dual performs better on the epsilon dataset, where the training point size is the dominating term, and CoCoA in the primal performs better on the webspam dataset, where the feature size is the dominating term. In both datasets, CoCoA in the dual performs better as the problem becomes more strongly convex ($\eta \to 0$), whereas CoCoA in the primal is robust to changes in strong convexity.

dual tends to perform better on datasets with a large number of training points (relative to the number of features), and that as expected, the performance deteriorates as the strong convexity in the problem disappears. In contrast, CoCoA in the primal performs well on datasets with a large number of features relative to training points, and is robust to changes in strong convexity. These changes in performance are to be expected, as we have already discussed that CoCoA in the primal is more suited for non-strongly convex regularizers (Section 4.2.1), and that the feature size dominates communication for CoCoA in the dual, as compared to the training point size for CoCoA in the primal (Section 2.9).

4.3.2 Effect of Communication

In contrast to the compared methods from Sections 4.2.1 and 4.2.2, CoCoA comes with the benefit of having only a single parameter to tune: the subproblem approximation quality, Θ, which we control in our experiments via the number of local subproblem iterations, H, for the example of local coordinate descent. We further explore the effect of this parameter in Figure 4.6, and provide a general guideline for choosing it in practice (see Remark 1). In particular, we see that while increasing H always results in better performance in terms of the number of communication rounds, smaller or larger values of H may result in better performance in terms of wall-clock time, depending on the cost of communication and computation. The flexibility to fine-tune H is one of the reasons for CoCoA's significant performance gains.

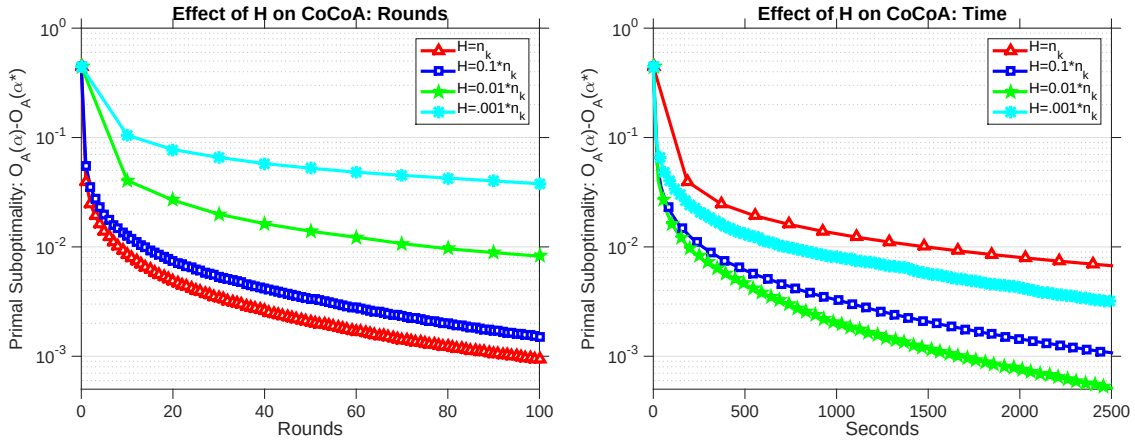

Figure 4.6: Suboptimality in terms of $\mathcal{O}_A(\boldsymbol{\alpha})$ for fitting a lasso regression model to the webspam dataset (K=16, λ=1E-5). Here we illustrate how the work spent in the local subproblem (given by H) influences the total performance of CoCoA in terms of number of rounds as well as wall time.

4.3.3 Subproblem Parameter

Finally, in Figure 4.7, we consider the effect of the choice of the subproblem parameter σ' on convergence. We plot both the number of communications and clock time on a log-log scale for the RCV1[1] dataset with K=8 and H=1e4. For CoCoA in the dual, we solve an SVM model and consider several different values of σ', ranging from 1 to 8. The value σ'=8 represents the safe upper bound of γK. The optimal convergence occurs around σ'=4, and diverges for $\sigma' \leq 2$. Notably, we see that the easy to calculate upper bound of $\sigma' := \gamma K$ (as given by Definition 5) has only slightly worse performance than best possible subproblem parameter in our setting. This indicates that, even though stronger performance is possible, the bound can be used effectively in practice.

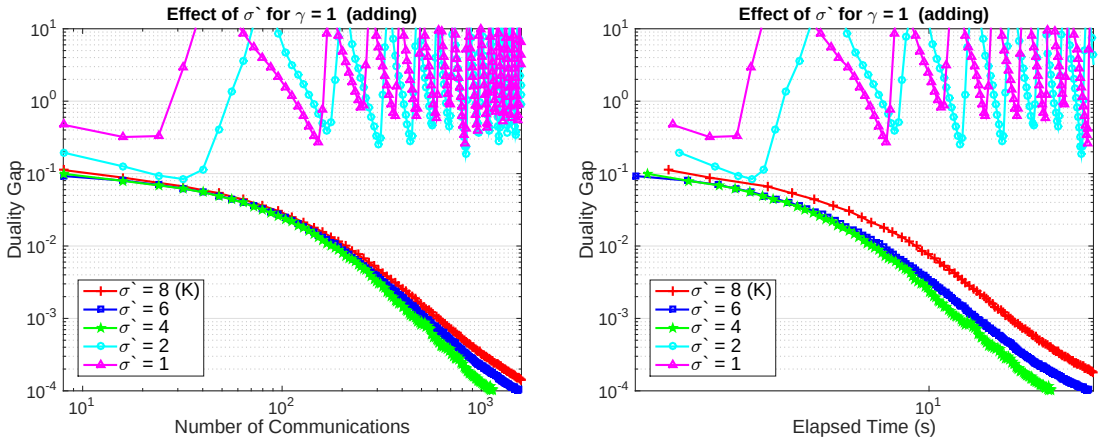

Figure 4.7: The effect of the subproblem parameter σ' on convergence of CoCoA for the RCV1 dataset distributed across K=8 machines. Decreasing σ' improves performance in terms of communication and overall run time until a certain point, after which the algorithm diverges. The "safe" upper bound of σ':=K=8 has only slightly worse performance than the practically best "un-safe" value of σ'.

[1] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#rcv1.binary

Chapter 5

Theoretical Analysis

In this chapter, we derive our convergence guarantees for the CoCoA framework and prove all other results given in the prior chapters. We begin by providing several important results and definitions needed for our primal-dual analysis.

5.1 Preliminaries

5.1.1 Conjugates

The convex conjugate of a function $f : \mathbb{R}^d \to \mathbb{R}$ is defined as

$$f^*(\mathbf{v}) := \max_{\mathbf{u} \in \mathbb{R}^d} \mathbf{v}^\top \mathbf{u} - f(\mathbf{u}) \,. \tag{5.1}$$

Below we list several useful properties of conjugates [cf. 15, Section 3.3.2]:

- Double conjugate: $(f^*)^* = f$ if f is closed and convex.
- Value Scaling: (for $\alpha > 0$) $f(\mathbf{v}) = \alpha g(\mathbf{v}) \quad \Rightarrow \quad f^*(\mathbf{w}) = \alpha g^*(\mathbf{w}/\alpha)$.
- Argument Scaling: (for $\alpha \neq 0$) $f(\mathbf{v}) = g(\alpha \mathbf{v}) \quad \Rightarrow \quad f^*(\mathbf{w}) = g^*(\mathbf{w}/\alpha)$.
- Conjugate of a separable sum: $f(\mathbf{v}) = \sum_i \phi_i(v_i) \quad \Rightarrow \quad f^*(\mathbf{w}) = \sum_i \phi_i^*(w_i)$.

Lemma 1 (Duality between Lipschitzness and L-Bounded Support, [80, Corollary 13.3.3])**.** *Given a proper convex function f, it holds that f is L-Lipschitz if and only if f^* has L-bounded support.*

Lemma 2 (Duality between Smoothness and Strong Convexity, [33, Theorem 4.2.2])**.** *Given a closed convex function f, it holds that f is μ strongly convex w.r.t. the norm $\|\cdot\|$ if and only if f^* is $(1/\mu)$-smooth w.r.t. the dual norm $\|\cdot\|_*$.*

5.1.2 Primal-Dual Relationship

In the following sections we provide derivations of the primal-dual relationship of the general objectives (A) and (B), and then show how to derive the conjugate of the modified L_1-norm as an example of the bounded-support modification introduced in Section 5.2.3.

5.1.3 Primal-Dual Relationship

The relation of our original formulation (A) to its dual formulation (B) is standard in convex analysis, and is a special case of the concept of Fenchel Duality. Using the combination with the linear map A as in our case, the relationship is called Fenchel-Rockafellar Duality (cf. Borwein and Zhu [14, Theorem 4.4.2] or Bauschke and Combettes [9, Proposition 15.18]). For completeness, we illustrate this correspondence with a self-contained derivation of the duality.

Starting with the original formulation (A), we introduce an auxiliary vector $\mathbf{v} \in \mathbb{R}^d$ representing $\mathbf{v} = A\boldsymbol{\alpha}$. Then optimization problem (A) becomes:

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \quad f(\mathbf{v}) + g(\boldsymbol{\alpha}) \quad \text{such that } \mathbf{v} = A\boldsymbol{\alpha}\,. \tag{5.2}$$

Introducing Lagrange multipliers $\mathbf{w} \in \mathbb{R}^d$, the Lagrangian is given by:

$$L(\boldsymbol{\alpha}, \mathbf{v}; \mathbf{w}) := f(\mathbf{v}) + g(\boldsymbol{\alpha}) + \mathbf{w}^\top (A\boldsymbol{\alpha} - \mathbf{v})\ .$$

The dual problem of (A) follows by taking the infimum with respect to both $\boldsymbol{\alpha}$ and $\mathbf{v}$:

$$\begin{aligned}
\inf_{\boldsymbol{\alpha},\mathbf{v}} L(\mathbf{w}, \boldsymbol{\alpha}, \mathbf{v}) &= \inf_{\mathbf{v}} \left\{ f(\mathbf{v}) - \mathbf{w}^\top \mathbf{v} \right\} + \inf_{\boldsymbol{\alpha}} \left\{ g(\boldsymbol{\alpha}) + \mathbf{w}^\top A\boldsymbol{\alpha} \right\} \\
&= -\sup_{\mathbf{v}} \left\{ \mathbf{w}^\top \mathbf{v} - f(\mathbf{v}) \right\} - \sup_{\boldsymbol{\alpha}} \left\{ (-\mathbf{w}^\top A)\boldsymbol{\alpha} - g(\boldsymbol{\alpha}) \right\} \\
&= -f^*(\mathbf{w}) - g^*(-A^\top \mathbf{w})\,.
\end{aligned} \tag{5.3}$$

We change signs and turn the maximization of the dual problem (5.3) into a minimization, thereby arriving at the dual formulation (B) as claimed:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \quad \Big[\ \mathcal{O}_B(\mathbf{w}) := g^*(-A^\top \mathbf{w}) + f^*(\mathbf{w})\ \Big]\,.$$

Continuous Conjugate Modification for Indicator Functions

Lemma 3 (Conjugate of the modified L_1-norm)**.** *The convex conjugate of the bounded support modification of the L_1-norm, as defined in* (5.7)*, is:*

$$\bar{g}_i^*(x) := \begin{cases} 0 & : x \in [-1, 1], \\ B(|x| - 1) & : \textit{otherwise,} \end{cases}$$

and is B-Lipschitz.

Proof. We start by applying the definition of convex conjugate:

$$\bar{g}_i(\alpha) = \sup_{x \in \mathbb{R}} \left[\alpha x - \bar{g}_i^*(x) \right] .$$

We begin by looking at the case in which $\alpha \geq B$; in this case it's easy to see that when $x \to +\infty$, we have:

$$\alpha x - B(|x| - 1) = (\alpha - B)x - B \to +\infty ,$$

as $\alpha - B \geq 0$. The case $\alpha \leq -B$ holds analogously. We now look at the case $\alpha \in [0, B]$; in this case it is clear we must have $x^\star \geq 0$. It also must hold that $x^\star \leq 1$, since

$$\alpha x - B(x - 1) < \alpha x ,$$

for every $x > 1$. Therefore the maximization becomes

$$\bar{g}_i(\alpha) = \sup_{x \in [0,1]} \alpha x ,$$

which has maximum α at $x = 1$. The remaining $\alpha \in [-B, 0]$ case follows in similar fashion.

Lipschitz continuity of $\bar{g}_i^*$ follows directly, or alternatively also from the general result that g_i^* is L-Lipschitz if and only if g_i has L-bounded support [80, Corollary 13.3.3] or [24, Lemma 5]. □

Comparison to ADMM. Here we derive the comparison of ADMM and CoCoA discussed in Section 2.11, following the line of reasoning in [100]. For consensus ADMM, the objective (B) is decomposed using the following re-parameterization:

$$\max_{\mathbf{w}_1, \ldots \mathbf{w}_K, \mathbf{w}} \quad \sum_{k=1}^{K} \sum_{i \in \mathcal{P}_k} g^*(-\mathbf{x}_i^\top \mathbf{w}_k) + f^*(\mathbf{w})$$

$$s.t. \quad \mathbf{w}_k = \mathbf{w}, \ k = 1, \ldots, K.$$

To solve this problem, we construct the augmented Lagrangian:

$$L_\rho(\mathbf{w}_1, \ldots, \mathbf{w}_k, \mathbf{u}_1, \ldots, \mathbf{u}_k, \mathbf{w}) := \sum_{k=1}^{K} \sum_{i \in \mathcal{P}_k} g^*(-\mathbf{x}_i^\top \mathbf{w}_k)$$

$$+ f^*(\mathbf{w}) + \rho \sum_{k=1}^{K} \mathbf{u}_k^\top (\mathbf{w}_k - \mathbf{w}) + \frac{\rho}{2} \sum_{k=1}^{K} \|\mathbf{w}_k - \mathbf{w}\|^2 ,$$

which yields the following decomposable updates:

$$\mathbf{w}_k^{(t)} = \arg\min_{\mathbf{w}_k} \sum_{i \in \mathcal{P}_k} g^*(-\mathbf{x}_i^\top \mathbf{w}_k) + \frac{\rho}{2} \|\mathbf{w}_k - \mathbf{w}^{(t-1)} + \mathbf{u}_k^{(t-1)}\|^2,$$

$$\mathbf{w}^{(t)} = \arg\min_{\mathbf{w}} \; f^*(\mathbf{w}) + \rho \sum_{k=1}^{K} \mathbf{u}_k^\top (\mathbf{w}_k - \mathbf{w}) + \frac{\rho}{2} \sum_{k=1}^{K} \|\mathbf{w}_k - \mathbf{w}\|^2,$$

$$\mathbf{u}_k^{(t)} = \mathbf{u}_k^{(t-1)} + \mathbf{w}_k^{(t)} - \mathbf{w}^{(t)}.$$

To compare this to the proposed framework, recall that the subproblem (2.10) (excluding the extraneous term $f(\mathbf{v})$) can be written as:

$$\min_{\boldsymbol{\alpha}_{[k]} \in \mathbb{R}^n} \sum_{i \in \mathcal{P}_k} g_i((\boldsymbol{\alpha}_{[k]})_i) + \mathbf{w}^\top A \boldsymbol{\alpha}_{[k]} + \frac{\sigma'}{2\tau} \left\| A_{[k]} \boldsymbol{\alpha}_{[k]} \right\|^2.$$

We can further reformulate by completing the square:

$$\min_{\boldsymbol{\alpha}_{[k]} \in \mathbb{R}^n} \sum_{i \in \mathcal{P}_k} g_i((\boldsymbol{\alpha}_{[k]})_i) + \frac{\tau}{2\sigma'} \left\| \; \mathbf{w} + \frac{\sigma'}{\tau} A_{[k]} \boldsymbol{\alpha}_{[k]} \right\|^2.$$

Assuming for the time being that $f(\cdot) = \frac{1}{2}\|\cdot\|_2^2$ such that $\mathbf{w} = \nabla f(\mathbf{v}) = \mathbf{v}$, we can unroll the update as follows, using $\gamma \Delta \mathbf{v}^{(t-1)} = \gamma \sum_{i=1}^{K} \Delta \mathbf{v}_k^{(t-1)}$:

$$\min_{\boldsymbol{\alpha}_{[k]} \in \mathbb{R}^n} \sum_{i \in \mathcal{P}_k} g_i((\boldsymbol{\alpha}_{[k]})_i) + \frac{\tau}{2\sigma'} \left\| \; \mathbf{w}^{(t-1)} + \gamma \Delta \mathbf{v}^{(t-1)} + \frac{\sigma'}{\tau} A_{[k]} \boldsymbol{\alpha}_{[k]} \right\|^2.$$

We will show that the above objective has the following primal form for each machine k:

$$\min_{\mathbf{w}} \sum_{i \in \mathcal{P}_k} g_i^*(-\mathbf{x}_i^\top \mathbf{w}) + \frac{\tau}{2\sigma'} \left\| \mathbf{w} - \left(\mathbf{w}^{(t-1)} + \gamma \Delta \mathbf{v}^{(t-1)} \right) \right\|^2. \tag{5.4}$$

Indeed, suppressing the subscript k for simplicity, we have:

$$\min_{\mathbf{w}} \sum_i g_i^*(-\mathbf{x}_i^\top \mathbf{w}) + \frac{\tau}{2\sigma'} \left\| \mathbf{w} - \left(\mathbf{w}^{(t-1)} + \gamma \Delta \mathbf{v}^{(t-1)} \right) \right\|^2$$

$$= \min_{\mathbf{w}} \sum_i \max_{\alpha_i} -\mathbf{x}_i^\top \mathbf{w} \alpha_i - g_i(\alpha_i) + \frac{\tau}{2\sigma'} \left\| \mathbf{w} - \left(\mathbf{w}^{(t-1)} + \gamma \Delta \mathbf{v}^{(t-1)} \right) \right\|^2$$

$$= \max_{\boldsymbol{\alpha}} \min_{\mathbf{w}} \sum_i -\mathbf{x}_i^\top \mathbf{w} \alpha_i - g_i(\alpha_i) + \frac{\tau}{2\sigma'} \left\| \mathbf{w} - \left(\mathbf{w}^{(t-1)} + \gamma \Delta \mathbf{v}^{(t-1)} \right) \right\|^2.$$

Solving the minimization yields: $\mathbf{w} = \mathbf{w}^{t-1} + \gamma\Delta\mathbf{v}^{(t-1)} + \frac{\sigma'}{\tau}A\boldsymbol{\alpha}$. Plugging this back in yields:

$$= \max_{\alpha} \sum_i -g_i(\alpha_i) - (A\boldsymbol{\alpha})^\top \mathbf{w}^{(t-1)} - (A\boldsymbol{\alpha})^\top \gamma\Delta\mathbf{v}^{(t-1)} - \frac{\sigma'}{\tau}\|A\boldsymbol{\alpha}\|^2 + \frac{\tau}{2\sigma'}\left\|\frac{\sigma'}{\tau}A\boldsymbol{\alpha}\right\|^2$$

$$= \max_{\boldsymbol{\alpha}} \sum_i -g_i(\alpha_i) - (A\boldsymbol{\alpha})^\top \mathbf{w}^{(t-1)} - (A\boldsymbol{\alpha})^\top \gamma\Delta\mathbf{v}^{(t-1)} - \frac{\sigma'}{2\tau}\|A\boldsymbol{\alpha}\|^2$$

$$= \min_{\boldsymbol{\alpha}} \sum_i g_i(\alpha_i) + (A\boldsymbol{\alpha})^\top \mathbf{w}^{(t-1)} + (A\boldsymbol{\alpha})^\top \gamma\Delta\mathbf{v}^{(t-1)} + \frac{\sigma'}{2\tau}\|A\boldsymbol{\alpha}\|^2$$

$$= \min_{\boldsymbol{\alpha}} \sum_i g_i(\alpha_i) + \frac{\tau}{2\sigma'}\left\| \mathbf{w}^{(t-1)} + \gamma\Delta\mathbf{v}^{(t-1)} + \frac{\sigma'}{\tau}A\boldsymbol{\alpha}\right\|^2.$$

5.2 Convergence

In this section, we provide convergence rates for the proposed framework and introduce an important theoretical technique in analyzing non-strongly convex terms in the primal-dual setting. For simplicity of presentation, we assume in the analysis that the data partitioning is balanced; i.e., $n_k = n/K$ for all k. Furthermore, we assume that the columns of A satisfy $\|\mathbf{x}_i\| \leq 1$ for all $i \in [n]$. We present rates for the case where $\gamma := 1$ in Algorithm 1, and where the subproblems (2.10) are defined using the corresponding safe bound $\sigma' := K$. This case will guarantee convergence while delivering our fastest rates in the distributed setting, which in particular do not degrade as the number of machines K grows and n remains fixed.

5.2.1 Proof Strategy: Relating Subproblem Approximation to Global Progress

To guarantee convergence, it is critical to show how progress made on the local subproblems (2.10) relates to the global objective $\mathcal{O}_A$. Our first lemma provides exactly this information. In particular, we see that if the aggregation and subproblem parameters are selected according to Definition 5, the sum of the subproblem objectives, $\sum_{k=1}^K \mathcal{G}_k^{\sigma'}$, will form a block-separable upper bound on the global objective $\mathcal{O}_A$.

Lemma 4. *For any weight vector $\boldsymbol{\alpha}, \Delta\boldsymbol{\alpha} \in \mathbb{R}^n$, $\boldsymbol{v} = \boldsymbol{v}(\boldsymbol{\alpha}) := A\boldsymbol{\alpha}$, and real values γ, σ' satisfying* (2.11)*, it holds that*

$$\mathcal{O}_A\Big(\boldsymbol{\alpha} + \gamma\sum_{k=1}^K \Delta\boldsymbol{\alpha}_{[k]}\Big) \leq (1-\gamma)\mathcal{O}_A(\boldsymbol{\alpha}) + \gamma\sum_{k=1}^K \mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]}; \boldsymbol{v}, \boldsymbol{\alpha}_{[k]})\,. \tag{5.5}$$

A proof of Lemma 15 is provided in Section 5.2.8. We use this main lemma, in combination with our assumption on the quality of the subproblem approximations (Assumption 1), to deliver our global convergence rates.

5.2.2 Rates for General Convex g_i, L-Lipschitz g_i^*

Our first main theorem provides convergence guarantees for objectives with general convex g_i (or, equivalently, L-Lipschitz g_i^*), including models with non-strongly convex regularizers such as lasso and sparse logistic regression, or models with non-smooth losses, such as the hinge loss support vector machine.

Providing primal-dual rates and globally defined primal-dual accuracy certificates for these objectives may require an important theoretical technique that we introduce below, in which we show how to satisfy the notion of L-bounded support for g_i, as stated in Definition 2.

Theorem 5. *Consider Algorithm 1 with $\gamma := 1$, and let Θ be the quality of the local solver as in Assumption 1. Let g_i have L-bounded support, and let f be $(1/\tau)$-smooth. Then after T iterations, where*

$$T \geq T_0 + \max\{\left\lceil \frac{1}{1-\Theta} \right\rceil, \frac{4L^2n^2}{\tau\epsilon_G(1-\Theta)}\}\,, \tag{5.6}$$

$$T_0 \geq t_0 + \left[\frac{2}{1-\Theta}\left(\frac{8L^2n^2}{\tau\epsilon_G} - 1\right)\right]_+\,,$$

$$t_0 \geq \max(0, \left\lceil \tfrac{1}{(1-\Theta)} \log\left(\tfrac{\tau(\mathcal{O}_A(\boldsymbol{\alpha}^{(0)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star))}{2L^2Kn}\right)\right\rceil)\,,$$

we have that the expected duality gap satisfies

$$\mathbb{E}\big[\mathcal{O}_A(\overline{\boldsymbol{\alpha}}) - (-\mathcal{O}_B(\boldsymbol{w}(\overline{\boldsymbol{\alpha}})))\big] \leq \epsilon_G\,,$$

where $\overline{\boldsymbol{\alpha}}$ is the averaged iterate: $\frac{1}{T-T_0}\sum_{t=T_0+1}^{T-1}\boldsymbol{\alpha}^{(t)}$.

5.2.3 Bounded support modification

As mentioned earlier, additional work is necessary if Theorem 5 is to be applied to non-strongly convex regularizers such as the L_1 norm, which do not have L-bounded support for each g_i, and thus violate the assumptions of the theorem. Note for example that the conjugate function of $g_i = |\cdot|$, which is the indicator function of an interval, is not defined globally over $\mathbb{R}$, and thus (without further modification) the duality gap $G(\boldsymbol{\alpha}) := \mathcal{O}_A(\boldsymbol{\alpha}) - (-\mathcal{O}_B(\mathbf{w}(\boldsymbol{\alpha})))$ is not even defined at all points $\boldsymbol{\alpha}$.

Smoothing. To address this problem, existing approaches typically use a simple smoothing technique [as in 67, 84]: by adding a small amount of L_2 to the objective g_i, the functions g_i become strongly convex. Followed by this change, the algorithms are then run on the dual of instead of the original primal problem at hand. While this modification satisfies the necessary assumptions for convergence of our framework, this Nesterov smoothing technique is often undesirable in practice, as it changes the iterates, the algorithms at hand, the convergence rate, and the tightness of the resulting duality gap compared to the original objective. Further, the amount of smoothing can be difficult to tune and can have a large influence on the performance of the method at hand. We show practical examples of these difficulties in Chapter 4.

Bounded support modification. In contrast to smoothing, our approach preserves all solutions of the original objective, leaves the iterate sequence unchanged, and allows for direct reusability of existing solvers for the original g_i objectives (such as L_1 solvers). It also removes the need for tuning a smoothing parameter. To achieve this, we modify the function g_i by imposing an additional weak constraint that is inactive in our region of interest. Formally, we replace $g_i(\alpha_i)$ by the following modified function:

$$\bar{g}_i(\alpha_i) := \begin{cases} g_i(\alpha_i) & : \alpha_i \in [-B, B] \\ +\infty & : \text{otherwise.} \end{cases} \tag{5.7}$$

For large enough B, this problem yields the same solution as the original objective. Note also that this only affects convergence theory, in that it allows us to present a strong primal-dual rate (Theorem 5 for $L{=}B$). The modification of g_i does not affect the algorithms for the original problems. Whenever a monotone optimizer is used, we will never leave the level set defined by the objective at the starting point.

Using the resulting modified function will allow us to apply the results of Theorem 5 for general convex functions g_i. This technique can also be thought of as "Lipschitzing" the dual g_i^*, because of the general result that g_i^* is L-Lipschitz if and only if g_i has L-bounded support [80, Corollary 13.3.3]. We derive the conjugate function $\bar{g}_i^*$ for completeness in Section 5.1.2 (Lemma 3). In Chapter 3, we show how to leverage this technique for a variety of application input problems. See also [24] for a follow-up discussion of this technique in the non-distributed case.

5.2.4 Rates for Strongly Convex g_i, Smooth g_i^*

For the case of objectives with strongly convex g_i (or, equivalently, smooth g_i^*), e.g., elastic net regression or logistic regression, we obtain the following faster *linear* convergence rate.

Theorem 6. *Consider Algorithm 1 with $\gamma := 1$, and let Θ be the quality of the local solver as in Assumption 1. Let g_i be μ-strongly convex $\forall i \in [n]$, and let f be $(1/\tau)$-smooth. Then after T iterations where*

$$T \geq \frac{1}{(1-\Theta)} \frac{\mu\tau + n}{\mu\tau} \log \frac{n}{\epsilon_{\mathcal{O}_A}}, \tag{5.8}$$

it holds that

$$\mathbb{E}\big[\mathcal{O}_A(\boldsymbol{\alpha}^{(T)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)\big] \leq \epsilon_{\mathcal{O}_A}\,.$$

Furthermore, after T iterations with

$$T \geq \tfrac{1}{(1-\Theta)}\tfrac{\mu\tau+n}{\mu\tau}\log\left(\tfrac{1}{(1-\Theta)}\tfrac{\mu\tau+n}{\mu\tau}\tfrac{n}{\epsilon_G}\right)\,,$$

we have the expected duality gap

$$\mathbb{E}\big[\mathcal{O}_A(\boldsymbol{\alpha}^{(T)}) - (-\mathcal{O}_B(\boldsymbol{w}(\boldsymbol{\alpha}^{(T)}))\big] \leq \epsilon_G\,.$$

We provide proofs of Theorem 5 and Theorem 6 below.

5.2.5 Convergence Cases

Revisiting Table 2.1 from Chapter 2, we summarize our convergence guarantees for the three cases of input problems (I) in the following table. In particular, we see that for cases II and III, we obtain a sublinear convergence rate, whereas for case I we can obtain a faster linear rate, as provided in Theorem 6.

Table 5.1: Applications of convergence rates.

	Smooth ℓ	Non-smooth, separable ℓ
Strongly convex r	Case I: Theorem 6	Case III: Theorem 5
Non-strongly convex, separable r	Case II: Theorem 5	–

5.2.6 Recovering Earlier Work as a Special Case

As a special case, the proposed framework and rates directly apply to L_2-regularized loss-minimization problems, including those presented in the earlier work of [38] and [53].

Remark 3. *If we run Algorithm 3 (mapping* (I) *to* (B)*), restrict $f^*(\cdot) := \frac{\lambda}{2}\|\cdot\|^2$ (so that $\tau = \lambda$), and let $g_i^* := \frac{1}{n}\ell_i^*$, Theorem 5 recovers as a special case the* CoCoA$^+$ *rates for general L-Lipschitz ℓ_i^* losses [see 53, Corollary 9]. The earlier work of* CoCoA*-v1 [38] did not provide rates for L-Lipschitz ℓ_i^* losses.*

These cases follow since g_i^* is L-Lipschitz if and only if g_i has L-bounded support [80, Corollary 13.3.3].

Remark 4. *If we run Algorithm 3 (mapping* (I) *to* (B)*), restrict $f^*(\cdot) := \frac{\lambda}{2}\|\cdot\|^2$ (so that $\tau = \lambda$), and scale $g_i^* := \frac{1}{n}\ell_i^*$, Theorem 6 recovers as a special case the* CoCoA$^+$ *rates for $(1/\ell_i^*)$-smooth losses [see 53, Corollary 11]. The earlier rates of* CoCoA*-v1 can be obtained by setting $\gamma{:=}\frac{1}{K}$ and $\sigma'{=}1$ in Algorithm 1 [38, Theorem 2].*

These cases follow since g_i^* is μ-strongly convex if and only if g_i is $(1/\mu)$-smooth [33, Theorem 4.2.2].

5.2.7 Local Subproblems

In this section we provide proofs of our main convergence results. The arguments follow the reasoning in [53, 51], but where we have generalized them to be applicable directly to (A). We provide full details of Lemma 15 as a proof of concept, but omit details in later proofs that can be derived using the arguments in [53] or earlier work of [85], and instead outline the proof strategy and highlight sections where the theory deviates.

5.2.8 Approximation of $\mathcal{O}_A(\cdot)$ by the Local Subproblems $\mathcal{G}_k^{\sigma'}(\cdot)$

Our first lemma in the overall proof of convergence helps to relate progress on the local subproblems to the global objective $\mathcal{O}_A(\cdot)$.

Lemma' 15. *For any dual variables $\boldsymbol{\alpha}, \Delta\boldsymbol{\alpha} \in \mathbb{R}^n$, $\boldsymbol{v} = \boldsymbol{v}(\boldsymbol{\alpha}) := A\boldsymbol{\alpha}$, and real values γ, σ' satisfying* (2.11)*, it holds that*

$$\mathcal{O}_A\Big(\boldsymbol{\alpha} + \gamma \sum_{k=1}^{K} \Delta\boldsymbol{\alpha}_{[k]}\Big) \leq (1-\gamma)\mathcal{O}_A(\boldsymbol{\alpha}) + \gamma \sum_{k=1}^{K} \mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]}; \boldsymbol{v}, \boldsymbol{\alpha}_{[k]})\,. \tag{5.9}$$

Proof. In this proof we follow the line of reasoning in Ma et al. [53, Lemma 4] with a more general $(1/\tau)$ smoothness assumption on $f(\cdot)$. An outer iteration of CoCoA performs the following update:

$$\mathcal{O}_A(\boldsymbol{\alpha} + \gamma \sum_{k=1}^{K} \Delta\boldsymbol{\alpha}_{[k]}) = \underbrace{f(\mathbf{v}(\boldsymbol{\alpha} + \gamma \sum_{k=1}^{K} \Delta\boldsymbol{\alpha}_{[k]}))}_{A} + \underbrace{\sum_{i=1}^{n} g_i(\alpha_i + \gamma(\sum_{k=1}^{K} \Delta\boldsymbol{\alpha}_{[k]})_i)}_{B}\,. \tag{5.10}$$

We bound A and B separately. First we bound A using $(1/\tau)$-smoothness of f:

$$A = f\Big(\mathbf{v}(\boldsymbol{\alpha} + \gamma \sum_{k=1}^{K} \Delta\boldsymbol{\alpha}_{[k]})\Big) = f\Big(\mathbf{v}(\boldsymbol{\alpha}) + \gamma \sum_{k=1}^{K} \mathbf{v}(\Delta\boldsymbol{\alpha}_{[k]})\Big)$$

$$\overset{\text{smoothness of } f \text{ as in (2.3)}}{\leq} f(\mathbf{v}(\boldsymbol{\alpha})) + \sum_{k=1}^{K} \gamma \nabla f(\mathbf{v}(\boldsymbol{\alpha}))^\top \mathbf{v}(\Delta\boldsymbol{\alpha}_{[k]}) + \frac{\gamma^2}{2\tau} \| \sum_{k=1}^{K} \mathbf{v}(\boldsymbol{\alpha}_{[k]}) \|^2$$

$$\overset{\text{definition of } \mathbf{w} \text{ as in (2.5)}}{\leq} f(\mathbf{v}(\boldsymbol{\alpha})) + \sum_{k=1}^{K} \gamma \mathbf{v}(\Delta\boldsymbol{\alpha}_{[k]})^\top \mathbf{w}(\boldsymbol{\alpha}) + \frac{\gamma^2}{2\tau} \| \sum_{k=1}^{K} \mathbf{v}(\boldsymbol{\alpha}_{[k]}) \|^2$$

$$\overset{\text{safe choice of } \sigma' \text{ as in (2.11)}}{\leq} f(\mathbf{v}(\boldsymbol{\alpha})) + \sum_{k=1}^{K} \gamma \mathbf{v}(\Delta\boldsymbol{\alpha}_{[k]})^\top \mathbf{w}(\boldsymbol{\alpha}) + \frac{1}{2\tau}\gamma\sigma' \sum_{k=1}^{K} \|\mathbf{v}(\boldsymbol{\alpha}_{[k]})\|^2 .$$

Next we use Jensen's inequality to bound B:

$$B = \sum_{k=1}^{K} \left(\sum_{i\in\mathcal{P}_k} g_i(\alpha_i + \gamma(\Delta\boldsymbol{\alpha}_{[k]})_i) \right) = \sum_{k=1}^{K} \left(\sum_{i\in\mathcal{P}_k} g_i((1-\gamma)\alpha_i + \gamma(\boldsymbol{\alpha} + \Delta\boldsymbol{\alpha}_{[k]})_i) \right)$$

$$\leq \sum_{k=1}^{K} \left(\sum_{i\in\mathcal{P}_k} (1-\gamma) g_i(\alpha_i) + \gamma g_i(\alpha_i + \Delta\alpha_{[k]_i}) \right) .$$

Plugging A and B back into (5.10) yields:

$$\mathcal{O}_A\Big(\boldsymbol{\alpha} + \gamma \sum_{k=1}^{K} \Delta\boldsymbol{\alpha}_{[k]}\Big) \leq f(\mathbf{v}(\boldsymbol{\alpha})) \pm \gamma f(\mathbf{v}(\boldsymbol{\alpha})) + \sum_{k=1}^{K} \gamma \mathbf{v}(\Delta\boldsymbol{\alpha}_{[k]})^\top \mathbf{w}(\boldsymbol{\alpha})$$

$$+ \frac{1}{2\tau}\gamma\sigma' \sum_{k=1}^{K} \|\mathbf{v}(\boldsymbol{\alpha}_{[k]})\|^2 + \sum_{k=1}^{K} \sum_{i\in\mathcal{P}_k} (1-\gamma) g_i(\alpha_i) + \gamma g_i(\alpha_i + \Delta\alpha_{[k]_i})$$

$$= \underbrace{(1-\gamma) f(\mathbf{v}(\boldsymbol{\alpha})) + \sum_{k=1}^{K} \left(\sum_{i\in\mathcal{P}_k} (1-\gamma) g_i(\alpha_i) \right)}_{(1-\gamma)\mathcal{O}_A(\boldsymbol{\alpha})}$$

$$+ \gamma \sum_{k=1}^{K} \Bigg(\frac{1}{K} f(\mathbf{v}(\boldsymbol{\alpha})) + \mathbf{v}(\Delta\boldsymbol{\alpha}_{[k]})^\top \mathbf{w}(\boldsymbol{\alpha}) + \frac{\sigma'}{2\tau} \|\mathbf{v}(\boldsymbol{\alpha}_{[k]})\|^2$$

$$+ \sum_{i\in\mathcal{P}_k} g_i(\alpha_i + \Delta\alpha_{[k]_i}) \Bigg)$$

$$\overset{(2.10)}{=} (1-\gamma)\mathcal{O}_A(\boldsymbol{\alpha}) + \gamma \sum_{k=1}^{K} \mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]}; \mathbf{v}) ,$$

where the last equality is by the definition of the subproblem objective $\mathcal{G}_k^{\sigma'}(.)$ as in (2.10). □

5.2.9 Proof of Convergence Result for General Convex g_i

Before proving the main convergence results, we introduce several useful quantities, and establish the following lemma, which characterizes the effect of iterations of Algorithm 1 on the duality gap for any chosen local solver of approximation quality Θ.

Lemma 7. *Let g_i be strongly convex* [1] *with convexity parameter $\mu \geq 0$ with respect to the norm $\|\cdot\|$, $\forall i \in [n]$. Then at each iteration of Algorithm 1 under Assumption 1, and any $s \in [0,1]$, it holds that*

$$\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^{(t+1)})] \geq \gamma(1-\Theta)\Big(sG(\boldsymbol{\alpha}^{(t)}) - \frac{\sigma' s^2}{2\tau} R^{(t)}\Big)\,, \tag{5.11}$$

where

$$R^{(t)} := -\tfrac{\tau\mu(1-s)}{\sigma' s}\|\boldsymbol{u}^{(t)} - \boldsymbol{\alpha}^{(t)}\|^2 + \textstyle\sum_{k=1}^K \|A_{[k]}(\boldsymbol{u}^{(t)} - \boldsymbol{\alpha}^{(t)})_{[k]}\|^2\,, \tag{5.12}$$

for $\boldsymbol{u}^{(t)} \in \mathbb{R}^n$ with

$$u_i^{(t)} \in \partial g_i^*(-\boldsymbol{x}_i^\top \boldsymbol{w}(\boldsymbol{\alpha}^{(t)}))\,. \tag{5.13}$$

Proof. This proof is motivated by Shalev-Shwartz and Zhang [85, Lemma 19] and follows Ma et al. [53, Lemma 5], with a difference being the extension to our generalized subproblems $\mathcal{G}_k^{\sigma'}(\cdot; \mathbf{v}, \boldsymbol{\alpha}_{[k]})$ along with the mappings $\mathbf{w}(\boldsymbol{\alpha}) := \nabla f(\mathbf{v}(\boldsymbol{\alpha}))$ with $\mathbf{v}(\boldsymbol{\alpha}) := A\boldsymbol{\alpha}$.

For simplicity, we write $\boldsymbol{\alpha}$ instead of $\boldsymbol{\alpha}^{(t)}$, $\mathbf{v}$ instead of $\mathbf{v}(\boldsymbol{\alpha}^{(t)})$, $\mathbf{w}$ instead of $\mathbf{w}(\boldsymbol{\alpha}^{(t)})$ and $\mathbf{u}$ instead of $\mathbf{u}^{(t)}$. We can estimate the expected change of the objective $\mathcal{O}_A(\boldsymbol{\alpha})$ as follows. Starting from the definition of the update $\boldsymbol{\alpha}^{(t+1)} := \boldsymbol{\alpha}^{(t)} + \gamma \sum_k \Delta\boldsymbol{\alpha}_{[k]}$ from Algorithm 1, we apply Lemma 15, which relates the local approximation $\mathcal{G}_k^{\sigma'}(\boldsymbol{\alpha}; \mathbf{v}, \boldsymbol{\alpha}_{[k]})$ to the global objective $\mathcal{O}_A(\boldsymbol{\alpha})$, and then bound this using the notion of quality of the local solver (Θ), as in Assumption 1. This gives us:

$$\begin{aligned}\mathbb{E}\big[\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^{(t+1)})\big] &= \mathbb{E}\Big[\mathcal{O}_A(\boldsymbol{\alpha}) - \mathcal{O}_A\Big(\boldsymbol{\alpha} + \gamma\sum_{k=1}^K \Delta\boldsymbol{\alpha}_{[k]}\Big)\Big] \\ &\geq \gamma(1-\Theta)\left(\underbrace{\mathcal{O}_A(\boldsymbol{\alpha}) - \sum_{k=1}^K \mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_{[k]}^\star; \mathbf{v}, \boldsymbol{\alpha}_{[k]})}_{C}\right)\,. \end{aligned}\tag{5.14}$$

We next upper bound the C term, denoting $\Delta\boldsymbol{\alpha}^\star = \sum_{k=1}^K \Delta\boldsymbol{\alpha}_{[k]}^\star$. We first plug in the definition of the objective $\mathcal{O}_A$ in (A) and the local subproblems (2.10), and then substitute

[1] Note that the case of weakly convex $g_i(.)$ is explicitly allowed here as well, as the Lemma holds for the case $\mu = 0$.

$s(u_i - \alpha_i)$ for $\Delta\boldsymbol{\alpha}_i^\star$ and apply the μ-strong convexity of the g_i terms. This gives us:

$$C = \sum_{i=1}^{n} \left(g_i(\alpha_i) - g_i(\alpha_i + \Delta\alpha_i^\star)\right) - (A\Delta\boldsymbol{\alpha}^\star)^\top \mathbf{w}(\boldsymbol{\alpha}) - \sum_{k=1}^{K} \frac{\sigma'}{2\tau}\left\|A_{[k]}\Delta\boldsymbol{\alpha}_{[k]}^\star\right\|^2$$

$$\geq \sum_{i=1}^{n} \left(s g_i(\alpha_i) - s g_i(u_i) + \frac{\mu}{2}(1-s)s(u_i - \alpha_i)^2\right)$$

$$- A(s(\mathbf{u} - \boldsymbol{\alpha}))^\top \mathbf{w}(\boldsymbol{\alpha}) - \sum_{k=1}^{K} \frac{\sigma'}{2\tau}\left\|A_{[k]}(s(\mathbf{u} - \boldsymbol{\alpha})_{[k]})\right\|^2 . \tag{5.15}$$

From the definition of the optimization problems (A) and (B), and definition of convex conjugates, we can write the duality gap as:

$$G(\boldsymbol{\alpha}) := \mathcal{O}_A(\boldsymbol{\alpha}) - (-\mathcal{O}_B(\mathbf{w}(\boldsymbol{\alpha}))$$

$$\overset{\text{(A),(B)}}{=} \sum_{i=1}^{n} \left(g_i^*(-\mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha})) + g_i(\alpha_i)\right) + f^*(\mathbf{w}(\boldsymbol{\alpha})) + f(A\boldsymbol{\alpha}))$$

$$= \sum_{i=1}^{n} \left(g_i^*(-\mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha})) + g_i(\alpha_i)\right) + f^*(\nabla f(A\boldsymbol{\alpha})) + f(A\boldsymbol{\alpha})$$

$$= \sum_{i=1}^{n} \left(g_i^*(-\mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha})) + g_i(\alpha_i)\right) + (A\boldsymbol{\alpha})^\top \mathbf{w}(\boldsymbol{\alpha})$$

$$= \sum_{i=1}^{n} \left(g_i^*(-\mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha})) + g_i(\alpha_i) + \alpha_i \mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha})\right) . \tag{5.16}$$

The convex conjugate maximal property from (5.13) implies that

$$g_i(u_i) = u_i(-\mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha})) - g_i^*(-\mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha})) . \tag{5.17}$$

Using (5.17) and (5.16), we therefore have:

$$
\begin{aligned}
C &\overset{(5.17)}{\geq} \sum_{i=1}^{n} \Big(s g_i(\alpha_i) - s u_i(-\mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha})) + s g_i^*(-\mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha})) + \frac{\mu}{2}(1-s)s(u_i-\alpha_i)^2 \Big) \\
&\quad - A(s(\mathbf{u}-\boldsymbol{\alpha}))^\top \mathbf{w}(\boldsymbol{\alpha}) - \sum_{k=1}^{K} \frac{\sigma'}{2\tau} \Big\| A_{[k]} (s(\mathbf{u}-\boldsymbol{\alpha})_{[k]}) \Big\|^2 \\
&= \sum_{i=1}^{n} \left[s g_i(\alpha_i) + s g_i^*(-\mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha})) + s \mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha}) \alpha_i \right] \\
&\quad - \sum_{i=1}^{n} \left[s \mathbf{x}_i^\top \mathbf{w}(\boldsymbol{\alpha})(\alpha_i - u_i) - \frac{\mu}{2}(1-s)s(u_i-\alpha_i)^2 \right] \\
&\quad - A(s(\mathbf{u}-\boldsymbol{\alpha}))^\top \mathbf{w}(\boldsymbol{\alpha}) - \sum_{k=1}^{K} \frac{\sigma'}{2\tau} \Big\| A_{[k]} (s(\mathbf{u}-\boldsymbol{\alpha})_{[k]}) \Big\|^2 \\
&\overset{(5.16)}{=} s G(\boldsymbol{\alpha}) + \frac{\mu}{2}(1-s)s \|\mathbf{u}-\boldsymbol{\alpha}\|^2 - \frac{\sigma' s^2}{2\tau} \sum_{k=1}^{K} \|A_{[k]}(\mathbf{u}-\boldsymbol{\alpha})_{[k]}\|^2 . \qquad (5.18)
\end{aligned}
$$

The claimed improvement bound (5.11) then follows by plugging (5.18) into (5.14). □

The following Lemma provides a uniform bound on $R^{(t)}$:

Lemma 8. *If g_i^* are L-Lipschitz continuous for all $i \in [n]$, then*

$$
\forall t : R^{(t)} \leq 4L^2 \underbrace{\sum_{k=1}^{K} \sigma_k n_k}_{=:\sigma} , \qquad (5.19)
$$

where

$$
\sigma_k := \max_{\boldsymbol{\alpha}_{[k]} \in \mathbb{R}^n} \frac{\|A_{[k]} \boldsymbol{\alpha}_{[k]}\|^2}{\|\boldsymbol{\alpha}_{[k]}\|^2} . \qquad (5.20)
$$

Proof. [53, Lemma 6]. For general convex functions, the strong convexity parameter is $\mu = 0$, and hence the definition (5.12) of the complexity constant $R^{(t)}$ becomes

$$
R^{(t)} = \sum_{k=1}^{K} \|A_{[k]}(\mathbf{u}^{(t)} - \boldsymbol{\alpha}^{(t)})_{[k]}\|^2 \overset{(5.20)}{\leq} \sum_{k=1}^{K} \sigma_k \|(\mathbf{u}^{(t)} - \boldsymbol{\alpha}^{(t)})_{[k]}\|^2 \leq \sum_{k=1}^{K} \sigma_k |\mathcal{P}_k| 4L^2 .
$$

Here the last inequality follows from [85, Lemma 21], which shows that for $g_i^* : \mathbb{R} \to \mathbb{R}$ being L-Lipschitz, it holds that for any real value a with $|a| > L$ one has that $g_i(a) = +\infty$. □

Remark 5. *[53, Remark 7] If the data points $\boldsymbol{x}_i$ are normalized such that $\|\boldsymbol{x}_i\| \leq 1$, $\forall i \in [n]$, then $\sigma_k \leq |\mathcal{P}_k| = n_k$. Furthermore, if we assume that the data partition is balanced, i.e., that $n_k = n/K$ for all k, then $\sigma \leq n^2/K$. This can be used to bound the constants $R^{(t)}$, above, as $R^{(t)} \leq \frac{4L^2n^2}{K}$.*

Theorem 9. *Consider Algorithm 1, using a local solver of quality Θ (See Assumption 1). Let $g_i^*(\cdot)$ be L-Lipschitz continuous, and $\epsilon_G > 0$ be the desired duality gap (and hence an upper-bound on suboptimality $\epsilon_{\mathcal{O}_A}$). Then after T iterations, where*

$$T \geq T_0 + \max\{\left\lceil \frac{1}{\gamma(1-\Theta)} \right\rceil, \frac{4L^2\sigma\sigma'}{\tau\epsilon_G\gamma(1-\Theta)}\}\,, \tag{5.21}$$

$$T_0 \geq t_0 + \left[\frac{2}{\gamma(1-\Theta)}\left(\frac{8L^2\sigma\sigma'}{\tau\epsilon_G} - 1\right)\right]_+\,,$$

$$t_0 \geq \max(0, \left\lceil \tfrac{1}{\gamma(1-\Theta)} \log\left(\tfrac{\tau(\mathcal{O}_A(\boldsymbol{\alpha}^{(0)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star))}{2L^2\sigma\sigma'}\right)\right\rceil)\,,$$

we have that the expected duality gap satisfies

$$\mathbb{E}[\mathcal{O}_A(\overline{\boldsymbol{\alpha}}) - (-\mathcal{O}_B(\boldsymbol{w}(\overline{\boldsymbol{\alpha}})))] \leq \epsilon_G$$

at the averaged iterate

$$\overline{\boldsymbol{\alpha}} := \tfrac{1}{T-T_0}\textstyle\sum_{t=T_0+1}^{T-1} \boldsymbol{\alpha}^{(t)}\,. \tag{5.22}$$

Proof. We begin by estimating the expected change of feasibility for $\mathcal{O}_A$. We can bound this above by using Lemma 7 and the fact that the $\mathcal{O}_B(\cdot)$ is always a lower bound for $-\mathcal{O}_A(\cdot)$, and then applying (5.19) to find:

$$\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(t+1)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)] \leq (1 - \gamma(1-\Theta)s)\left(\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)\right)$$

$$+ \gamma(1-\Theta)\tfrac{\sigma' s^2}{2\tau} 4L^2\sigma\,. \tag{5.23}$$

Using (5.23) recursively we have

$$\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)] \leq (1 - \gamma(1-\Theta)s)^t\left(\mathcal{O}_A(\boldsymbol{\alpha}^{(0)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)\right) + s\frac{4L^2\sigma\sigma'}{2\tau}\,. \tag{5.24}$$

Choosing $s = 1$ and $t = t_0 := \max\{0, \lceil \frac{1}{\gamma(1-\Theta)} \log(2(\mathcal{O}_A(\boldsymbol{\alpha}^{(0)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star))/(4L^2\sigma\sigma'))\rceil\}$ leads to

$$\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)] \leq (1 - \gamma(1-\Theta))^{t_0}\left(\mathcal{O}_A(\boldsymbol{\alpha}^{(0)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)\right) + \frac{4L^2\sigma\sigma'}{2\tau}$$

$$\leq \frac{4L^2\sigma\sigma'}{\tau}\,. \tag{5.25}$$

Next, we show inductively that

$$\forall t \geq t_0 : \mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)] \leq \frac{4L^2\sigma\sigma'}{\tau(1+\frac{1}{2}\gamma(1-\Theta)(t-t_0))}\,. \tag{5.26}$$

Clearly, (5.25) implies that (5.26) holds for $t = t_0$. Assuming that it holds for any $t \geq t_0$, we show that it must also hold for $t+1$. Indeed, using

$$s = \frac{1}{1+\frac{1}{2}\gamma(1-\Theta)(t-t_0)} \in [0,1]\,, \tag{5.27}$$

we obtain

$$\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(t+1)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)] \leq \frac{4L^2\sigma\sigma'}{\tau} \underbrace{\left(\frac{1+\frac{1}{2}\gamma(1-\Theta)(t-t_0) - \frac{1}{2}\gamma(1-\Theta)}{(1+\frac{1}{2}\gamma(1-\Theta)(t-t_0))^2}\right)}_{D}$$

by applying the bounds (5.23) and (5.26), plugging in the definition of s (5.27), and simplifying. We upper bound the term D using the fact that geometric mean is less or equal to arithmetic mean:

$$D = \frac{1}{1+\frac{1}{2}\gamma(1-\Theta)(t+1-t_0)} \underbrace{\frac{(1+\frac{1}{2}\gamma(1-\Theta)(t+1-t_0))(1+\frac{1}{2}\gamma(1-\Theta)(t-1-t_0))}{(1+\frac{1}{2}\gamma(1-\Theta)(t-t_0))^2}}_{\leq 1}$$

$$\leq \frac{1}{1+\frac{1}{2}\gamma(1-\Theta)(t+1-t_0)}.$$

If $\overline{\boldsymbol{\alpha}}$ is defined as (5.22), we apply the results of Lemma 7 and Lemma 8 to obtain

$$\mathbb{E}[G(\overline{\boldsymbol{\alpha}})] = \mathbb{E}\left[G\left(\sum_{t=T_0}^{T-1} \tfrac{1}{T-T_0}\boldsymbol{\alpha}^{(t)}\right)\right] \leq \tfrac{1}{T-T_0}\mathbb{E}\left[\sum_{t=T_0}^{T-1} G\left(\boldsymbol{\alpha}^{(t)}\right)\right]$$

$$\leq \frac{1}{\gamma(1-\Theta)s}\frac{1}{T-T_0}\mathbb{E}\left[\mathcal{O}_A(\boldsymbol{\alpha}^{(T_0)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)\right] + \tfrac{4L^2\sigma\sigma' s}{2\tau}\,. \tag{5.28}$$

If $T \geq \lceil \frac{1}{\gamma(1-\Theta)} \rceil + T_0$ such that $T_0 \geq t_0$ we have

$$\mathbb{E}[G(\overline{\boldsymbol{\alpha}})] \overset{(5.28),(5.26)}{\leq} \frac{1}{\gamma(1-\Theta)s}\frac{1}{T-T_0}\left(\frac{4L^2\sigma\sigma'}{\tau(1+\frac{1}{2}\gamma(1-\Theta)(T_0-t_0))}\right) + \frac{4L^2\sigma\sigma' s}{2\tau}$$

$$= \frac{4L^2\sigma\sigma'}{\tau}\left(\frac{1}{\gamma(1-\Theta)s}\frac{1}{T-T_0}\frac{1}{1+\frac{1}{2}\gamma(1-\Theta)(T_0-t_0)} + \frac{s}{2}\right)\,. \tag{5.29}$$

Choosing

$$s = \frac{1}{(T - T_0)\gamma(1-\Theta)} \in [0, 1] \tag{5.30}$$

gives us

$$\mathbb{E}[G(\overline{\boldsymbol{\alpha}})] \overset{(5.29),(5.30)}{\leq} \frac{4L^2\sigma\sigma'}{\tau}\left(\frac{1}{1+\frac{1}{2}\gamma(1-\Theta)(T_0-t_0)} + \frac{1}{(T-T_0)\gamma(1-\Theta)}\frac{1}{2}\right). \tag{5.31}$$

To have right hand side of (5.31) smaller then ϵ_G it is sufficient to choose T_0 and T such that

$$\frac{4L^2\sigma\sigma'}{\tau}\left(\frac{1}{1+\frac{1}{2}\gamma(1-\Theta)(T_0-t_0)}\right) \leq \frac{1}{2}\epsilon_G\,, \tag{5.32}$$

$$\frac{4L^2\sigma\sigma'}{\tau}\left(\frac{1}{(T-T_0)\gamma(1-\Theta)}\frac{1}{2}\right) \leq \frac{1}{2}\epsilon_G\,. \tag{5.33}$$

Hence if $T_0 \geq t_0 + \frac{2}{\gamma(1-\Theta)}\left(\frac{8L^2\sigma\sigma'}{\tau\epsilon_G} - 1\right)$ and $T \geq T_0 + \frac{4L^2\sigma\sigma'}{\tau\epsilon_G\gamma(1-\Theta)}$ then (5.32) and (5.33) are satisfied. □

The following main theorem simplifies the results of Theorem 9 and is a generalization of Ma et al. [53, Corollary 9] for general $f^*(\cdot)$ functions:

Theorem' 5. *Consider Algorithm 1 with $\gamma := 1$, using a local solver of quality Θ (see Assumption 1). Let $g_i^*(\cdot)$ be L-Lipschitz continuous, and assume that the columns of A satisfy $\|\boldsymbol{x}_i\| \leq 1$, $\forall i \in [n]$. Let $\epsilon_G > 0$ be the desired duality gap (and hence an upper-bound on primal sub-optimality). Then after T iterations, where*

$$T \geq T_0 + \max\{\left\lceil\frac{1}{1-\Theta}\right\rceil, \frac{4L^2n^2}{\tau\epsilon_G(1-\Theta)}\}\,, \tag{5.34}$$

$$T_0 \geq t_0 + \left[\frac{2}{1-\Theta}\left(\frac{8L^2n^2}{\tau\epsilon_G} - 1\right)\right]_+\,,$$

$$t_0 \geq \max(0, \left\lceil\tfrac{1}{(1-\Theta)}\log\left(\tfrac{\tau(\mathcal{O}_A(\boldsymbol{\alpha}^{(0)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star))}{2L^2Kn}\right)\right\rceil)\,,$$

we have that the expected duality gap satisfies

$$\mathbb{E}[\mathcal{O}_A(\overline{\boldsymbol{\alpha}}) - (-\mathcal{O}_B(\boldsymbol{w}(\overline{\boldsymbol{\alpha}})))] \leq \epsilon_G\,,$$

where $\overline{\boldsymbol{\alpha}}$ is the averaged iterate returned by Algorithm 1.

Proof. Plug in parameters $\gamma := 1$, $\sigma' := \gamma K = K$ to the results of Theorem 9, and note that for balanced datasets we have $\sigma \leq \frac{n^2}{K}$ (see Remark 5). We can further simplify the rate by noting that $\tau = 1$ for the 1-smooth losses (least squares and logistic) given as examples in this work. □

5.2.10 Proof of Convergence Result for Strongly Convex g_i

Our second main theorem follows reasoning in [85] and is a generalization of Ma et al. [53, Corollary 11]. We first introduce a lemma to simplify the proof.

Lemma 10. *Assume that $g_i(0) \in [0,1]$ for all $i \in [n]$, then for the zero vector $\boldsymbol{\alpha}^{(0)} := \boldsymbol{0} \in \mathbb{R}^n$, we have*

$$\mathcal{O}_A(\boldsymbol{\alpha}^{(0)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star) = \mathcal{O}_A(\boldsymbol{0}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star) \leq n\,. \tag{5.35}$$

Proof. For $\boldsymbol{\alpha} := \mathbf{0} \in \mathbb{R}^n$, we have $\mathbf{w}(\boldsymbol{\alpha}) = A\boldsymbol{\alpha} = \mathbf{0} \in \mathbb{R}^d$. Therefore, since the dual $-\mathcal{O}_A(\cdot)$ is always a lower bound on the primal $\mathcal{O}_B(\cdot)$, and by definition of the objective $\mathcal{O}_A$ given in (A),

$$0 \leq \mathcal{O}_A(\boldsymbol{\alpha}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star) \leq \mathcal{O}_A(\boldsymbol{\alpha}) - (-\mathcal{O}_B(\mathbf{w}(\boldsymbol{\alpha}))) \overset{\text{(A)}}{\leq} n\,. \qquad \square$$

Theorem 11. *Assume that g_i are μ-strongly convex $\forall i \in [n]$. We define $\sigma_{\max} = \max_{k\in[K]} \sigma_k$. Then after T iterations of Algorithm 1, with*

$$T \geq \tfrac{1}{\gamma(1-\Theta)} \tfrac{\mu\tau + \sigma_{\max}\sigma'}{\mu\tau} \log \tfrac{n}{\epsilon_{\mathcal{O}_A}}\,,$$

it holds that

$$\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(T)}) - \mathcal{O}_A(\boldsymbol{\alpha}^\star)] \leq \epsilon_{\mathcal{O}_A}\,.$$

Furthermore, after T iterations with

$$T \geq \tfrac{1}{\gamma(1-\Theta)} \tfrac{\mu\tau + \sigma_{\max}\sigma'}{\mu\tau} \log \left(\tfrac{1}{\gamma(1-\Theta)} \tfrac{\mu\tau + \sigma_{\max}\sigma'}{\mu\tau} \tfrac{n}{\epsilon_G} \right)\,,$$

we have the expected duality gap

$$\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(T)}) - (-\mathcal{O}_B(\boldsymbol{w}(\boldsymbol{\alpha}^{(T)})))] \leq \epsilon_G\,.$$

Proof. Given that $g_i(.)$ is μ-strongly convex with respect to the $\|\cdot\|$ norm, we can apply (5.12) and the definition of σ_k to find:

$$\begin{aligned} R^{(t)} &\leq -\tfrac{\tau\mu(1-s)}{\sigma' s} \|\mathbf{u}^{(t)} - \boldsymbol{\alpha}^{(t)}\|^2 + \sum\nolimits_{k=1}^{K} \sigma_k \|\mathbf{u}^{(t)} - \boldsymbol{\alpha}_{[k]}^{(t)}\|^2 \\ &\leq \left(-\tfrac{\tau\mu(1-s)}{\sigma' s} + \sigma_{\max} \right) \|\mathbf{u}^{(t)} - \boldsymbol{\alpha}^{(t)}\|^2\,, \end{aligned} \tag{5.36}$$

where $\sigma_{\max} = \max_{k\in[K]} \sigma_k$. If we plug the following value of s

$$s = \frac{\tau\mu}{\tau\mu + \sigma_{\max}\sigma'} \in [0,1] \tag{5.37}$$

into (5.36) we obtain that $\forall t : R^{(t)} \leq 0$. Putting the same s into (5.11) will give us

$$\begin{aligned}\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^{(t+1)})] &\overset{(5.11),(5.37)}{\geq} \gamma(1-\Theta)\frac{\tau\mu}{\tau\mu + \sigma_{\max}\sigma'}G(\boldsymbol{\alpha}^{(t)}) \\ &\geq \gamma(1-\Theta)\frac{\tau\mu}{\tau\mu + \sigma_{\max}\sigma'}(\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^{\star}))\,. \end{aligned} \tag{5.38}$$

Using the fact that

$$\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^{(t+1)})] = \mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{\star}) - \mathcal{O}_A(\boldsymbol{\alpha}^{(t+1)})] + \mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^{\star})\ ,$$

we have

$$\begin{aligned}&\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{\star}) - \mathcal{O}_A(\boldsymbol{\alpha}^{(t+1)})] + \mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^{\star}) \\ &\qquad\overset{(5.38)}{\geq} \gamma(1-\Theta)\frac{\tau\mu}{\tau\mu + \sigma_{\max}\sigma'}(\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^{\star}))\,,\end{aligned}$$

which is equivalent to

$$\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(t+1)}) - \mathcal{O}_A(\boldsymbol{\alpha}^{\star})] \leq \left(1 - \gamma(1-\Theta)\frac{\tau\mu}{\tau\mu + \sigma_{\max}\sigma'}\right)(\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^{\star}))\,. \tag{5.39}$$

Therefore if we denote $\epsilon_{\mathcal{O}_A}^{(t)} = \mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^{\star})$, we have recursively that

$$\begin{aligned}\mathbb{E}[\epsilon_{\mathcal{O}_A}^{(t)}] &\overset{(5.39)}{\leq} \left(1 - \gamma(1-\Theta)\frac{\tau\mu}{\tau\mu + \sigma_{\max}\sigma'}\right)^t \epsilon_{\mathcal{O}_A}^{(0)} \\ &\overset{(5.35)}{\leq} \left(1 - \gamma(1-\Theta)\frac{\tau\mu}{\tau\mu + \sigma_{\max}\sigma'}\right)^t n \\ &\leq \exp\left(-t\gamma(1-\Theta)\frac{\tau\mu}{\tau\mu + \sigma_{\max}\sigma'}\right) n\,.\end{aligned}$$

The right hand side will be smaller than some $\epsilon_{\mathcal{O}_A}$ if

$$t \geq \frac{1}{\gamma(1-\Theta)}\frac{\tau\mu + \sigma_{\max}\sigma'}{\tau\mu}\log\frac{n}{\epsilon_{\mathcal{O}_A}}\,.$$

Moreover, to bound the duality gap, we have

$$\begin{aligned}\gamma(1-\Theta)\frac{\tau\mu}{\tau\mu + \sigma_{\max}\sigma'}G(\boldsymbol{\alpha}^{(t)}) &\overset{(5.38)}{\leq} \mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^{(t+1)})] \\ &\leq \mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(t)}) - \mathcal{O}_A(\boldsymbol{\alpha}^{\star})]\,.\end{aligned}$$

Thus, $G(\boldsymbol{\alpha}^{(t)}) \leq \frac{1}{\gamma(1-\Theta)}\frac{\tau\mu+\sigma_{\max}\sigma'}{\tau\mu}\epsilon_{\mathcal{O}_A}^{(t)}$. Hence, if $\epsilon_{\mathcal{O}_A} \leq \gamma(1-\Theta)\frac{\tau\mu}{\tau\mu+\sigma_{\max}\sigma'}\epsilon_G$, then $G(\boldsymbol{\alpha}^{(t)}) \leq \epsilon_G$. Therefore after

$$t \geq \frac{1}{\gamma(1-\Theta)}\frac{\tau\mu+\sigma_{\max}\sigma'}{\tau\mu}\log\left(\frac{1}{\gamma(1-\Theta)}\frac{\tau\mu+\sigma_{\max}\sigma'}{\tau\mu}\frac{n}{\epsilon_G}\right)$$

iterations, we have obtained a duality gap less than ϵ_G. □

Theorem' 6. *Consider Algorithm 1 with $\gamma := 1$, using a local solver of quality Θ (see Assumption 1). Let $g_i(\cdot)$ be μ-strongly convex, $\forall i \in [n]$, and assume that the columns of A satisfy $\|\boldsymbol{x}_i\| \leq 1$ $\forall i \in [n]$. Then we have that T iterations are sufficient for suboptimality $\epsilon_{\mathcal{O}_A}$, with*

$$T \geq \frac{1}{\gamma(1-\Theta)}\frac{\tau\mu+n}{\tau\mu}\log\frac{n}{\epsilon_{\mathcal{O}_A}}\,.$$

Furthermore, after T iterations with

$$T \geq \frac{1}{\gamma(1-\Theta)}\frac{\tau\mu+n}{\tau\mu}\log\left(\frac{1}{\gamma(1-\Theta)}\frac{\tau\mu+n}{\tau\mu}\frac{n}{\epsilon_G}\right)\,,$$

we have the expected duality gap

$$\mathbb{E}[\mathcal{O}_A(\boldsymbol{\alpha}^{(T)}) - (-\mathcal{O}_B(\boldsymbol{w}(\boldsymbol{\alpha}^{(T)})))] \leq \epsilon_G\,.$$

Proof. Plug in parameters $\gamma := 1$, $\sigma' := \gamma K = K$ to the results of Theorem 11 and note that for balanced datasets we have $\sigma_{\max} \leq \frac{n}{K}$ (see Remark 5). We can further simplify the rate by noting that $\tau = 1$ for the 1-smooth losses (least squares and logistic) given as examples in this work. □

Chapter 6

Extension: Federated Learning

Federated learning poses new statistical and systems challenges in training machine learning models over distributed networks of devices. In this chapter, we show that multi-task learning is naturally suited to handle the statistical challenges of this setting, and propose system-aware optimization method, MOCHA, that extends our earlier work on COCOA and is robust to practical systems issues. Our method and theory for the first time consider issues of high communication cost, stragglers, and fault tolerance for distributed multi-task learning. The resulting method achieves significant speedups compared to alternatives in the federated setting, as we demonstrate through simulations on real-world federated datasets.

6.1 Introduction

Mobile phones, wearable devices, and smart homes are just a few of the modern distributed networks generating massive amounts of data each day. Due to the growing storage and computational power of devices in these networks, it is increasingly attractive to store data locally and push more network computation to the edge. The nascent field of *federated learning* explores *training* statistical models directly on devices [60]. Examples of potential applications include: learning sentiment, semantic location, or activities of mobile phone users; predicting health events like low blood sugar or heart attack risk from wearable devices; or detecting burglaries within smart homes [4, 69, 75]. Following [43, 44, 61], we summarize the unique challenges of federated learning below.

1. **Statistical Challenges**: The aim in federated learning is to fit a model to distributed data, $\{\mathbf{X}_1, \ldots, \mathbf{X}_m\}$, generated by m nodes. Each node, $t \in [m]$, collects data in a *non-IID* manner across the network, with data on each node being generated by a distinct distribution $\mathbf{X}_t \sim P_t$. The number of data points on each node, n_t, may also vary significantly, and there may be an underlying structure present that captures the relationship amongst nodes and their associated distributions.

2. **Systems Challenges**: There are typically a large number of nodes, m, in the network, and communication is often a significant bottleneck. Additionally, the storage, computational, and communication capacities of each node may differ due to variability in hardware (CPU, memory), network connection (3G, 4G, WiFi), and power (battery level). These systems challenges, compounded with unbalanced data and statistical heterogeneity, make issues such as stragglers and fault tolerance significantly more prevalent than in typical data center environments.

In this work, we propose a modeling approach that differs significantly from prior work on federated learning, where the aim thus far has been to train a single global model across the network [43, 44, 61]. Instead, we address statistical challenges in the federated setting by learning separate models for each node, $\{\mathbf{w}_1, \ldots, \mathbf{w}_m\}$. This can be naturally captured through a *multi-task learning (MTL)* framework, where the goal is to consider fitting separate but related models simultaneously [2, 25, 46, 108]. Unfortunately, current multi-task learning methods are not suited to handle the systems challenges that arise in federated learning, including high communication cost, stragglers, and fault tolerance. Addressing these challenges is therefore a key component of our work.

6.1.1 Contributions

We make the following contributions. First, we show that MTL is as a natural choice to handle statistical challenges in the federated setting. Second, we develop a novel method, MOCHA, to solve a general MTL framework. Our method generalizes the distributed optimization method COCOA in order to address systems challenges associated with network size and node heterogeneity. Third, we provide convergence guarantees for MOCHA that carefully consider these unique systems challenges and provide insight into practical performance. Finally, we demonstrate the superior empirical performance of MOCHA with a new benchmarking suite of federated datasets.

6.2 Related Work

Learning Beyond the Data Center. Computing SQL-like queries across distributed, low-powered nodes is a decades-long area of research that has been explored under the purview of query processing in sensor networks, computing at the edge, and fog computing [13, 21, 29, 34, 54, 55]. Recent work has also considered training machine learning models centrally but serving and storing them locally, e.g., this is a common approach in mobile user modeling and personalization [45, 76, 77]. However, as the computational power of nodes within distributed networks grows, it is possible to do even more work locally, which has led to recent interest in federated learning.[1] In contrast to our proposed approach, existing

[1]The term *on-device learning* has been used to describe both the task of model training and of model serving. Due to the ambiguity of this phrase, we exclusively use the term federated learning.

federated learning approaches [43, 44, 60, 61] aim to learn a single global model across the data.[2] This limits their ability to deal with non-IID data and structure amongst the nodes. These works also come without convergence guarantees, and have not addressed practical issues of stragglers or fault tolerance, which are important characteristics of the federated setting. The work proposed here is, to the best of our knowledge, the first federated learning framework to consider these challenges, theoretically and in practice.

Multi-Task Learning. In multi-task learning, the goal is to learn models for multiple related tasks simultaneously. While the MTL literature is extensive, most MTL modeling approaches can be broadly categorized into two groups based on how they capture relationships amongst tasks. The first (e.g., [6, 20, 25, 42]) assumes that a clustered, sparse, or low-rank structure between the tasks is known *a priori*. A second group instead assumes that the task relationships are not known beforehand and can be learned directly from the data (e.g., [30, 37, 108]). In this work, we focus our attention on this latter group, as task relationships may not be known beforehand in real-world settings. In comparison to learning a single global model, these MTL approaches can directly capture relationships amongst non-IID and unbalanced data, which makes them particularly well-suited for the statistical challenges of federated learning. We demonstrate this empirically on real-world federated datasets in Section 6.5. However, although MTL is a natural modeling choice to address the statistical challenges of federated learning, currently proposed methods for distributed MTL (discussed below) do not adequately address the systems challenges associated with federated learning.

Distributed Multi-Task Learning. Distributed multi-task learning is a relatively new field, in which the aim is to solve an MTL problem when data for each task is distributed over a network. While several recent works [1, 59, 95, 96] have considered the issue of distributed MTL training, the proposed methods do not allow for flexibility of communication versus computation. As a result, they are unable to efficiently handle concerns of fault tolerance and stragglers, the latter of which stems from both data and system heterogeneity. The works of [39] and [10] allow for asynchronous updates to help mitigate stragglers, but do not address fault tolerance. Moreover, [39] provides no convergence guarantees, and the convergence of [10] relies on a bounded delay assumption that is impractical for the federated setting, where delays may be significant and devices may drop out completely. Finally, [49] proposes a method and setup leveraging the distributed framework CoCoA, which we show in Section 6.3 to be a special case of the more general approach in this work. However, the authors in [49] do not explore the federated setting, and their assumption that the same amount of work is done locally on each node is prohibitive in federated settings, where unbalance is common due to data and system variability.

[2] While not the focus of our work, we note privacy is an important concern in the federated setting, and that the privacy benefits associated with global federated learning (as discussed in [61]) also apply to our approach.

6.3 Federated Multi-Task Learning

In federated learning, the aim is to learn a model over data that resides on, and has been generated by, m distributed nodes. As a running example, consider learning the activities of mobile phone users in a cell network based on their individual sensor, text, or image data. Each node (phone), $t \in [m]$, may generate data via a distinct distribution, and so it is natural to fit separate models, $\{\mathbf{w}_1, \ldots, \mathbf{w}_m\}$, to the distributed data—one for each local dataset. However, structure between models frequently exists (e.g., people may behave similarly when using their phones), and modeling these relationships via *multi-task learning* is a natural strategy to improve performance and boost the effective sample size for each node [2, 5, 19]. In this section, we suggest a general MTL framework for the federated setting, and propose a novel method, MOCHA, to handle the systems challenges of federated MTL.

6.3.1 Preliminaries

Notation. We use $\mathbf{I}_{d\times d}$ to represent an identity matrix of size $d \times d$. When the context allows, we use the notation $\mathbf{I}$ to denote an identity matrix of an appropriate size. We also use $\otimes$ to denote the Kronecker product between two matrices.

Definition 6 (Matrix norm)**.** *Given a symmetric positive definite matrix* $\mathbf{M}$*, the norm of* $\boldsymbol{u}$ *with respect to* $\mathbf{M}$ *is given by* $\|\boldsymbol{u}\|_{\mathbf{M}} := \sqrt{\boldsymbol{u}^T\mathbf{M}\boldsymbol{u}}$ *.*

Definition 7 (L-smooth)**.** *A convex function* f *is* L*-smooth with respect to* $\mathbf{M}$ *if*

$$f(\boldsymbol{u}) \leq f(\boldsymbol{v}) + \langle \nabla f(\boldsymbol{v}), \boldsymbol{u} - \boldsymbol{v}\rangle + \frac{L}{2}\|\boldsymbol{u} - \boldsymbol{v}\|_{\boldsymbol{M}}^2 \qquad \forall \boldsymbol{u}, \boldsymbol{v}\,. \tag{6.1}$$

If $\boldsymbol{M} = \boldsymbol{I}$ *then, we simply say* f *is* L*-smooth.*

Definition 8 (τ-strongly convex)**.** *A function* f *is* τ*-strongly convex with respect to* $\boldsymbol{M}$ *if*

$$f(\boldsymbol{u}) \geq f(\boldsymbol{v}) + \langle \mathbf{z}, \boldsymbol{u} - \boldsymbol{v}\rangle + \frac{\tau}{2}\|\boldsymbol{u} - \boldsymbol{v}\|_{\boldsymbol{M}}^2 \qquad \forall \boldsymbol{u}, \boldsymbol{v},\ \mathbf{z} \in \partial f(\boldsymbol{v})\,, \tag{6.2}$$

where $\partial f(\boldsymbol{v})$ *is the set of sub-differentials of function* f *at* $\boldsymbol{v}$*. If* $\boldsymbol{M} = \boldsymbol{I}$ *then, we simply say* f *is* τ*-strongly convex.*

Definition 9. *The function* f *is called* L*-Lipchitz if for any* $\boldsymbol{x}$ *and* $\boldsymbol{y}$ *in its domain*

$$|f(\boldsymbol{x}) - f(\boldsymbol{y})| \leq L\|\boldsymbol{x} - \boldsymbol{y}\|_2\,. \tag{6.3}$$

If a function f is L-Lipchitz then its dual will be L-bounded, i.e., for any $\boldsymbol{\alpha}$ such that $\|\boldsymbol{\alpha}\|_2 > L$, then $f^*(\boldsymbol{\alpha}) = +\infty$.

6.3.2 General Multi-Task Learning Setup

Given data $\mathbf{X}_t \in \mathbb{R}^{d \times n_t}$ from m nodes, multi-task learning fits separate weight vectors $\mathbf{w}_t \in \mathbb{R}^d$ to the data for each task (node) through arbitrary convex loss functions ℓ_t (e.g., the hinge loss for SVM models). Many MTL problems can be captured in the following general formulation:

$$\min_{\mathbf{W},\mathbf{\Omega}} \left\{ \sum_{t=1}^{m} \sum_{i=1}^{n_t} \ell_t(\mathbf{w}_t^T \mathbf{x}_t^i, y_t^i) + \mathcal{R}(\mathbf{W}, \mathbf{\Omega}) \right\}, \tag{6.4}$$

where $\mathbf{W} := [\mathbf{w}_1, \ldots, \mathbf{w}_m] \in \mathbb{R}^{d \times m}$ is a matrix whose t-th column is the weight vector for the t-th task. The matrix $\mathbf{\Omega} \in \mathbb{R}^{m \times m}$ models relationships amongst tasks, and is either known a priori or estimated while simultaneously learning task models. MTL problems differ based on their assumptions on $\mathcal{R}$, which takes $\mathbf{\Omega}$ as input and promotes some suitable structure among the tasks.

As an example, several popular MTL approaches assume that tasks form clusters based on whether or not they are related [25, 37, 108, 111]. This can be expressed via the following bi-convex formulation:

$$\mathcal{R}(\mathbf{W}, \mathbf{\Omega}) = \lambda_1 \operatorname{tr}\big(\mathbf{W}\mathbf{\Omega}\mathbf{W}^T\big) + \lambda_2 \|\mathbf{W}\|_F^2, \tag{6.5}$$

with constants $\lambda_1, \lambda_2 > 0$, and where the second term performs L_2 regularization on each local model. We use a jointly convex relaxation of this (6.12) in our experiments in Section 6.5, and provide details on other common classes of MTL models that can be formulated via (6.4) in Section 6.6.

6.3.3 MOCHA: A Framework for Federated Multi-Task Learning

In the federated setting, the aim is to train statistical models directly on the edge, and thus we solve (6.4) while assuming that the data $\{\mathbf{X}_1, \ldots, \mathbf{X}_m\}$ is distributed across m nodes or devices. Before proposing our federated method for solving (6.4), we make the following observations:

- ***Observation 1****: In general,* (6.4) *is not jointly convex in* $\boldsymbol{W}$ *and* $\boldsymbol{\Omega}$*, and even in the cases where* (6.4) *is convex, solving for* $\boldsymbol{W}$ *and* $\boldsymbol{\Omega}$ *simultaneously can be difficult [5].*
- ***Observation 2****: When fixing* $\boldsymbol{\Omega}$*, updating* $\boldsymbol{W}$ *depends on both the data* $\boldsymbol{X}$*, which is distributed across the nodes, and the structure* $\boldsymbol{\Omega}$*, which is known centrally.*
- ***Observation 3****: When fixing* $\boldsymbol{W}$*, optimizing for* $\boldsymbol{\Omega}$ *only depends on* $\boldsymbol{W}$ *and not on the data* $\boldsymbol{X}$*.*

Based on these observations, it is natural to propose an alternating optimization approach to solve problem (6.4), in which at each iteration we fix either $\mathbf{W}$ or $\mathbf{\Omega}$ and optimize over the other, alternating until convergence is reached. Note that solving for $\mathbf{\Omega}$ is not dependent

Algorithm 4 MOCHA: Federated Multi-Task Learning Framework

Input: Data $\mathbf{X}_t$ from $t = 1, \ldots, m$ tasks, stored on m nodes, and initial matrix $\mathbf{\Omega}_0$
Starting point $\boldsymbol{\alpha}^{(0)} := \mathbf{0} \in \mathbb{R}^n$, $\mathbf{v}^{(0)} := \mathbf{0} \in \mathbb{R}^b$
for iterations $i = 0, 1, \ldots$ **do**
    Set subproblem parameter σ' and number of federated iterations, H_i
    for iterations $h = 0, 1, \cdots, H_i$ **do**
        for tasks $t \in \{1, 2, \ldots, m\}$ **in parallel over** m **nodes do**
            compute θ_t^h-approximate solution $\Delta\boldsymbol{\alpha}_t$ of local subproblem (6.7)
            update local variables $\boldsymbol{\alpha}_t \leftarrow \boldsymbol{\alpha}_t + \Delta\boldsymbol{\alpha}_t$
            return updates $\Delta\mathbf{v}_t := \mathbf{X}_t\Delta\boldsymbol{\alpha}_t$
        reduce: $\mathbf{v}_t \leftarrow \mathbf{v}_t + \Delta\mathbf{v}_t$
    Update $\mathbf{\Omega}$ centrally based on $\mathbf{w}(\boldsymbol{\alpha})$ for latest $\boldsymbol{\alpha}$
Central node computes $\mathbf{w} = \mathbf{w}(\boldsymbol{\alpha})$ based on the lastest $\boldsymbol{\alpha}$
return: $\mathbf{W} := [\mathbf{w}_1, \ldots, \mathbf{w}_m]$

on the data and therefore can be computed centrally; as such, we defer to prior work for this step [30, 37, 108, 111]. In Section 6.6, we discuss updates to Ω for several common MTL models.

In this work, we focus on developing an efficient distributed optimization method for the $\mathbf{W}$ step. In traditional data center environments, the task of distributed training is a well-studied problem, and various communication-efficient frameworks have been recently proposed, including the state-of-the-art primal-dual COCOA framework previously discussed in this thesis. Although COCOA can be extended directly to update $\mathbf{W}$ in a distributed fashion across the nodes, it cannot handle the unique systems challenges of the federated environment, such as stragglers and fault tolerance, as discussed in Section 6.3.5. To this end, we extend COCOA and propose a new method, MOCHA, for federated multi-task learning. Our method is given in Algorithm 4 and described in detail in Sections 6.3.4 and 6.3.5.

6.3.4 Federated Update of W

To update $\mathbf{W}$ in the federated setting, we begin by extending works on distributed primal-dual optimization [38, 49, 53] to apply to the generalized multi-task framework (6.4). This involves deriving the appropriate dual formulation, subproblems, and problem parameters, as we detail below.

Dual problem. Considering the dual formulation of (6.4) will allow us to better separate the global problem into distributed subproblems for federated computation across the nodes. Let $n := \sum_{t=1}^m n_t$ and $\mathbf{X} := \text{Diag}(\mathbf{X}_1, \cdots, \mathbf{X}_m) \in \mathbb{R}^{md \times n}$. With $\mathbf{\Omega}$ fixed, the dual of

problem (6.4), defined with respect to dual variables $\boldsymbol{\alpha} \in \mathbb{R}^n$, is given by:

$$\min_{\boldsymbol{\alpha}} \left\{ \mathcal{D}(\boldsymbol{\alpha}) := \sum_{t=1}^{m} \sum_{i=1}^{n_t} \ell_t^*(-\alpha_t^i) + \mathcal{R}^*(\mathbf{X}\boldsymbol{\alpha}) \right\}, \tag{6.6}$$

where ℓ_t^* and $\mathcal{R}^*$ are the conjugate dual functions of ℓ_t and $\mathcal{R}$, respectively, and $\boldsymbol{\alpha}_t^i$ is the dual variable for the data point $(\mathbf{x}_t^i, y_t^i)$. Note that $\mathcal{R}^*$ depends on $\boldsymbol{\Omega}$, but for the sake of simplicity, we have removed this in our notation. To derive distributed subproblems from this global dual, we make an assumption described below on the regularizer $\mathcal{R}$.

Assumption 2. *Given $\boldsymbol{\Omega}$, we assume that there exists a symmetric positive definite matrix $\boldsymbol{M} \in \mathbb{R}^{md \times md}$, depending on $\boldsymbol{\Omega}$, for which the function $\mathcal{R}$ is strongly convex with respect to $\boldsymbol{M}^{-1}$. Note that this corresponds to assuming that $\mathcal{R}^*$ will be smooth with respect to matrix $\boldsymbol{M}$.*

Remark 6. *We can reformulate the MTL regularizer in the form of $\bar{\mathcal{R}}(\boldsymbol{w}, \bar{\boldsymbol{\Omega}}) = \mathcal{R}(\boldsymbol{W}, \boldsymbol{\Omega})$, where $\boldsymbol{w} \in \mathbb{R}^{md}$ is a vector containing the columns of $\boldsymbol{W}$ and $\bar{\boldsymbol{\Omega}} := \boldsymbol{\Omega} \otimes \mathbf{I}_{d\times d} \in \mathbb{R}^{md \times md}$. For example, we can rewrite the regularizer in (6.5) as $\bar{\mathcal{R}}(\boldsymbol{w}, \bar{\boldsymbol{\Omega}}) = tr\big(\boldsymbol{w}^T(\lambda_1 \bar{\boldsymbol{\Omega}} + \lambda_2 \boldsymbol{I})\boldsymbol{w}\big)$. Writing the regularizer in this form, it is clear that it is strongly convex with respect to matrix $\boldsymbol{M}^{-1} = \lambda_1 \bar{\boldsymbol{\Omega}} + \lambda_2 \boldsymbol{I}$.*

Data-local quadratic subproblems. To solve (6.4) across distributed nodes, we define the following data-local subproblems, which are formed via a careful quadratic approximation of the dual problem (6.6) to separate computation across the nodes. These subproblems find updates $\Delta\boldsymbol{\alpha}_t \in \mathbb{R}^{n_t}$ to the dual variables in $\boldsymbol{\alpha}$ corresponding to a single node t, and only require accessing data which is available locally, i.e., $\mathbf{X}_t$ for node t. The t-th subproblem is given by:

$$\min_{\Delta\boldsymbol{\alpha}_t} \mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_t; \mathbf{v}_t, \boldsymbol{\alpha}_t) := \sum_{i=1}^{n_t} \ell_t^*(-\alpha_t^i - \Delta\alpha_t^i) + \langle \mathbf{w}_t(\boldsymbol{\alpha}), \mathbf{X}_t \Delta\boldsymbol{\alpha}_t \rangle + \frac{\sigma'}{2} \left\| \mathbf{X}_t \Delta\boldsymbol{\alpha}_t \right\|_{\mathbf{M}_t}^2 + c(\boldsymbol{\alpha}), \tag{6.7}$$

where $c(\boldsymbol{\alpha}) := \frac{1}{m}\mathcal{R}^*(\mathbf{X}\boldsymbol{\alpha})$, and $\mathbf{M}_t \in \mathbb{R}^{d\times d}$ is the t-th diagonal block of the symmetric positive definite matrix $\mathbf{M}$. Given dual variables $\boldsymbol{\alpha}$, corresponding primal variables can be found via $\mathbf{w}(\boldsymbol{\alpha}) = \nabla\mathcal{R}^*(\mathbf{X}\boldsymbol{\alpha})$, where $\mathbf{w}_t(\boldsymbol{\alpha})$ is the t-th block in the vector $\mathbf{w}(\boldsymbol{\alpha})$. Note that computing $\mathbf{w}(\boldsymbol{\alpha})$ requires the vector $\mathbf{v} = \mathbf{X}\boldsymbol{\alpha}$. The t-th block of $\mathbf{v}$, $\mathbf{v}_t \in \mathbb{R}^d$, is the only information that must be communicated between nodes at each iteration. Finally, $\sigma' > 0$ measures the difficulty of the data partitioning, and helps to relate progress made to the subproblems to the global dual problem. It can be easily selected based on $\mathbf{M}$ for many applications of interest; we provide details in Lemma 20 in Section 6.8.

6.3.5 Practical Considerations

During MOCHA's federated update of $\mathbf{W}$, the central node requires a response from all workers before performing a synchronous update. In the federated setting, a naive execution of this communication protocol could introduce dramatic straggler effects due to node heterogeneity. To avoid stragglers, MOCHA provides the t-th node with the flexibility to *approximately solve* its subproblem $\mathcal{G}_k^{\sigma'}(\cdot)$, where the quality of the approximation is control by a per-node parameter θ_t^h. The following factors determine the quality of the t-th node's subproblem solution:

1. **Statistical challenges**, such as the size of $\mathbf{X}_t$ and the intrinsic difficulty of subproblem $\mathcal{G}_k^{\sigma'}(\cdot)$.

2. **Systems challenges**, such as the node's storage, computational, and communication capacities due to hardware (CPU, memory), network connection (3G, 4G, WiFi), and power (battery level).

3. A **global clock cycle** imposed by the central node specifying a deadline for receiving updates.

We define θ_t^h as a function of these factors, and assume that each node has a controller that may derive θ_t^h from the current clock cycle and statistical/systems setting. θ_t^h ranges from zero to one, where $\theta_t^h = 0$ indicates an exact solution to $\mathcal{G}_k^{\sigma'}(\cdot)$ and $\theta_t^h = 1$ indicates that node t made no progress during iteration h (which we refer to as a *dropped node*). For instance, a node may 'drop' if it runs out of battery, or if its network bandwidth deteriorates during iteration h and it is thus unable to return its update within the current clock cycle. A formal definition of θ_t^h is provided in (6.8) of Section 6.4.

MOCHA mitigates stragglers by enabling the t-th node to define its own θ_t^h. On every iteration h, the local updates that a node performs and sends in a clock cycle will yield a specific value for θ_t^h. As discussed in Section 6.4, MOCHA is additionally robust to a small fraction of nodes periodically dropping and performing no local updates (i.e., $\theta_t^h := 1$) under suitable conditions, as defined in Assumption 3. In contrast, prior work of COCOA may suffer from severe straggler effects in federated settings, as it requires a *fixed* $\theta_t^h = \theta$ *across all nodes and all iterations* while still maintaining synchronous updates, and it does not allow for the case of dropped nodes ($\theta := 1$).

Finally, we note that asynchronous updating schemes are an alternative approach to mitigate stragglers. We do not consider these approaches in this work, in part due to the fact that the bounded-delay assumptions associated with most asynchronous schemes limit fault tolerance. However, it would be interesting to further explore the differences and connections between asynchronous methods and approximation-based, synchronous methods like MOCHA in future work.

6.4 Convergence Analysis

MOCHA is based on a bi-convex alternating approach, which is guaranteed to converge [31, 78] to a stationary solution of problem (6.4). In the case where this problem is jointly convex with respect to $\mathbf{W}$ and $\mathbf{\Omega}$, such a solution is also optimal. In the rest of this section, we therefore focus on the convergence of solving the $\mathbf{W}$ update of MOCHA in a federated setting. Following the discussion in Section 6.3.5, we first introduce the following per-node, per-round approximation parameter.

Definition 10 (Per-Node-Per-Iteration-Approximation Parameter)**.** *At each iteration h, we define the accuracy level of the solution calculated by node t to its subproblem* (6.7) *as:*

$$\theta_t^h := \frac{\mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_t^{(h)}; \boldsymbol{v}^{(h)}, \boldsymbol{\alpha}_t^{(h)}) - \mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_t^{\star}; \boldsymbol{v}^{(h)}, \boldsymbol{\alpha}_t^{(h)})}{\mathcal{G}_k^{\sigma'}(\boldsymbol{0}; \boldsymbol{v}^{(h)}, \boldsymbol{\alpha}_t^{(h)}) - \mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_t^{\star}; \boldsymbol{v}^{(h)}, \boldsymbol{\alpha}_t^{(h)})}\,, \tag{6.8}$$

where $\Delta\boldsymbol{\alpha}_t^{\star}$ is the minimizer of subproblem $\mathcal{G}_k^{\sigma'}(\cdot\ ; \boldsymbol{v}^{(h)}, \boldsymbol{\alpha}_t^{(h)})$. We allow this value to vary between $[0, 1]$, with $\theta_t^h := 1$ meaning that no updates to subproblem $\mathcal{G}_k^{\sigma'}$ are made by node t at iteration h.

While the flexible per-node, per-iteration approximation parameter θ_t^h in (6.8) allows the consideration of stragglers and fault tolerance, these additional degrees of freedom also pose new challenges in providing convergence guarantees for MOCHA. We introduce the following assumption on θ_t^h to provide our convergence guarantees.

Assumption 3. *Let $\mathcal{H}_h := (\boldsymbol{\alpha}^{(h)}, \boldsymbol{\alpha}^{(h-1)}, \cdots, \boldsymbol{\alpha}^{(1)})$ be the dual vector history until the beginning of iteration h, and define $\Theta_t^h := \mathbb{E}[\theta_t^h | \mathcal{H}_h]$. For all tasks t and all iterations h, we assume $p_t^h := \mathbb{P}[\theta_t^h = 1] \leq p_{\max} < 1$ and $\hat{\Theta}_t^h := \mathbb{E}[\theta_t^h | \mathcal{H}_h, \theta_t^h < 1] \leq \Theta_{\max} < 1$.*

This assumption states that at each iteration, the *probability* of a node sending a result is non-zero, and that the quality of the returned result is, on average, better than the previous iterate. Compared to the earlier work in Chapter 5 which assumes $\theta_t^h = \theta < 1$, our assumption is significantly less restrictive and better models the federated setting, where nodes are unreliable and may periodically drop out.

Using Assumption 3, we derive the following theorem, which characterizes the convergence of the federated update of MOCHA in finite horizon when the losses ℓ_t in (6.4) are smooth.

Theorem 12. *Assume that the losses ℓ_t are $(1/\mu)$-smooth. Then, under Assumptions 2 and 3, there exists a constant $s \in (0, 1]$ such that for any given convergence target $\epsilon_{\mathcal{D}}$, choosing H such that*

$$H \geq \frac{1}{(1 - \bar{\Theta})s} \log \frac{n}{\epsilon_{\mathcal{D}}}\,, \tag{6.9}$$

will satisfy $\mathbb{E}[\mathcal{D}(\boldsymbol{\alpha}^{(H)}) - \mathcal{D}(\boldsymbol{\alpha}^{\star})] \leq \epsilon_{\mathcal{D}}$.

Here, $\bar{\Theta} := p_{\max} + (1 - p_{\max})\Theta_{\max} < 1$. While Theorem 12 is concerned with finite horizon convergence, it is possible to get asymptotic convergence results, i.e., $H \to \infty$, with milder assumptions on the stragglers; see Corollary 19 in Section 6.7.1 for details.

When the loss functions are non-smooth, e.g., the hinge loss for SVM models, we provide the following sub-linear convergence for L-Lipschitz losses.

Theorem 13. *If the loss functions ℓ_t are L-Lipschitz, then there exists a constant σ, defined in* (6.24)*, such that for any given $\epsilon_{\mathcal{D}} > 0$, if we choose*

$$H \geq H_0 + \left\lceil \frac{2}{(1-\bar{\Theta})} \max\left(1, \frac{2L^2\sigma\sigma'}{n^2\epsilon_{\mathcal{D}}}\right) \right\rceil, \tag{6.10}$$

with

$$H_0 \geq \left\lceil h_0 + \frac{16L^2\sigma\sigma'}{(1-\bar{\Theta})n^2\epsilon_{\mathcal{D}}} \right\rceil, h_0 = \left[1 + \frac{1}{(1-\bar{\Theta})} \log\left(\frac{2n^2(D(\boldsymbol{\alpha}^\star) - D(\boldsymbol{\alpha}^0))}{4L^2\sigma\sigma'}\right)\right]_+,$$

then $\bar{\boldsymbol{\alpha}} := \frac{1}{H-H0}\sum_{h=H_0+1}^{H} \boldsymbol{\alpha}^{(h)}$ will satisfy $\mathbb{E}[\mathcal{D}(\bar{\boldsymbol{\alpha}}) - \mathcal{D}(\boldsymbol{\alpha}^\star)] \leq \epsilon_{\mathcal{D}}$.

These theorems guarantee that MOCHA will converge in the federated setting, under mild assumptions on stragglers and capabilities of the nodes. While these results consider convergence in terms of the dual, we show that they hold analogously for the duality gap. We provide all proofs in Section 6.7.

Remark 7. *Following from the discussion in Section 6.3.5, our method and theory generalize the results of* COCOA *from Chapter 5. In the limiting case that all θ_t^h are identical, our results extend the results of* COCOA *to the multi-task framework described in* (6.4)*.*

6.5 Simulations

In this section we validate the empirical performance of MOCHA. First, we introduce a benchmarking suite of real-world federated datasets and show that multi-task learning is well-suited to handle the statistical challenges of the federated setting. Next, we demonstrate the ability of MOCHA to handle stragglers, both from statistical and systems heterogeneity. Finally, we explore the performance of MOCHA when devices periodically drop out.

6.5.1 Federated Datasets

In our simulations, we use several real-world datasets that have been generated in federated settings. We provide additional details in the Section 6.9, including information about data size skew, n_t.

- **Google Glass (GLEAM)**[3]: This dataset consists of two hours of high resolution sensor data collected from 38 participants wearing Google Glass for the purpose of activity recognition. Following [74], we featurize the raw accelerometer, gyroscope, and magnetometer data into 180 statistical, spectral, and temporal features. We model each participant as a separate task, and predict between eating and other activities (e.g., walking, talking, drinking).

- **Human Activity Recognition**[4]: Mobile phone accelerometer and gyroscope data collected from 30 individuals, performing one of six activities. We use the provided 561-length feature vectors of time and frequency domain variables generated for each instance [4]. We model each individual as a separate task and predict between sitting and other activities (e.g., walking, lying down).

- **Land Mine**[5]: Radar image data collected from 29 land mine fields. Each instance consists of nine features extracted from the images [99]. We model each field as a task, and predict whether or not landmines are present in each field. Notably, the data is collected from two different terrains—highly foliated and desert regions—and the tasks therefore naturally form two clusters.

- **Vehicle Sensor**[6]: Acoustic, seismic, and infrared sensor data collected from a distributed network of 23 sensors, deployed with the aim of classifying vehicles driving by a segment of road [22]. Each instance is described by 50 acoustic and 50 seismic features. We model each sensor as a separate task and predict between AAV-type and DW-type vehicles.

6.5.2 Multi-Task Learning for the Federated Setting

We demonstrate the benefits of multi-task learning for the federated setting by comparing the error rates of a multi-task model to that of a fully local model (i.e., learning a model for each task separately) and a fully global model (i.e., combining the data from all tasks and learning one single model). Notably, existing work on federated learning focuses on learning fully global models [43, 44, 61].

We use a cluster-regularized multi-task model [37, 111], as described in Section 6.3.2. For each dataset from Section 6.5.1, we randomly split the data into 75% training and 25% testing, and learn multi-task, local, and global support vector machine models, selecting the best regularization parameter, $\lambda \in$ {1e-5, 1e-4, 1e-3, 1e-2, 0.1, 1, 10}, for each model using 5-fold cross-validation. We repeat this process 10 times and report the average prediction error across tasks, averaged across these 10 trials.

[3] `http://www.skleinberg.org/data/GLEAM.tar.gz`

[4] `https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones`

[5] `http://www.ee.duke.edu/~lcarin/LandmineData.zip`

[6] `http://www.ecs.umass.edu/~mduarte/Software.html`

Table 6.1: Average prediction error: Means and standard errors over 10 random shuffles.

Model	Human Activity	Google Glass	Land Mine	Vehicle Sensor
Global	2.23 (0.30)	5.34 (0.26)	27.72 (1.08)	13.4 (0.26)
Local	1.34 (0.21)	4.92 (0.26)	23.43 (0.77)	7.81 (0.13)
MTL	**0.46 (0.11)**	**2.02 (0.15)**	**20.09 (1.04)**	**6.59 (0.21)**

In Table 6.1, we see that for each dataset, multi-task learning significantly outperforms the other models in terms of achieving the lowest average error across tasks. The global model, as proposed in [43, 44, 61] performs the worst, particularly for the Human Activity and Vehicle Sensor datasets. Although the datasets are already somewhat unbalanced, a global modeling approach may additionally benefit tasks that have a very small number of instances, as information can be shared across tasks. For this reason, we additionally explore the performance of global, local, and multi-task modeling for highly skewed data in Table 6.4 of Section 6.9. Although the performance of the global model improves slightly relative to local modeling in this setting, the global model still performs the worst for the majority of the datasets, and MTL still significantly outperforms both global and local approaches.

6.5.3 Straggler Avoidance

Two challenges that are prevalent in federated learning are stragglers and high communication. Stragglers can occur when a subset of the devices take much longer than others to perform local updates, which can be caused either from statistical or systems heterogeneity. Communication can also exacerbate poor performance, as it can be slower than computation by many orders of magnitude in typical cellular or wireless networks [18, 36, 64, 87, 93]. In our experiments below, we simulate the time needed to run each method by tracking the operations and communication complexities, and scaling the communication cost relative to computation by one, two, or three orders of magnitude, respectively. These numbers correspond roughly to the clock rate vs. network bandwidth/latency [cf. 93] for modern cellular and wireless networks. Details are provided in Section 6.9.

Statistical Heterogeneity. We explore the effect of statistical heterogeneity on stragglers, for various methods and communication regimes (3G, LTE, WiFi). For a fixed communication network, we compare MOCHA to COCOA, which has a single θ parameter, and to mini-batch stochastic gradient descent (Mb-SGD) and mini-batch stochastic dual coordinate ascent (Mb-SDCA), which have limited communication flexibility depending on the batch size. We tune all compared methods for best performance, as we detail in Section 6.9. In Figure 6.1, we see that while the performance degrades for mini-batch methods in high

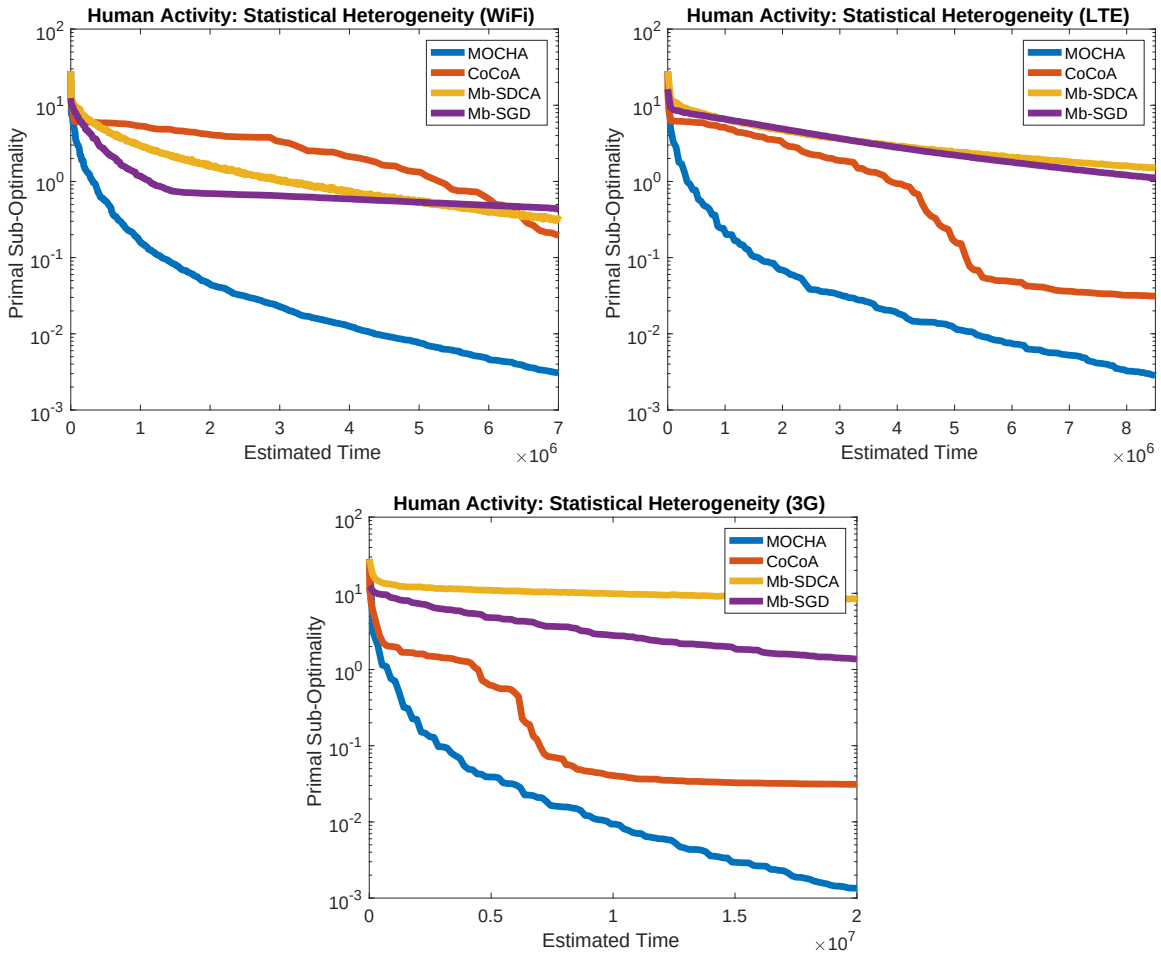

Figure 6.1: The performance of MOCHA compared to other distributed methods for the $\mathbf{W}$ update of (6.4). While increasing communication tends to *decrease* the performance of the mini-batch methods, MOCHA performs well in high communication settings. In all settings, MOCHA with varied approximation values, Θ_t^h, performs better than without (i.e., naively generalizing COCOA), as it avoids stragglers from statistical heterogeneity.

communication regimes, MOCHA and COCOA are robust to high communication. However, COCOA is significantly effected by stragglers—because θ is fixed across nodes and rounds, difficult subproblems adversely impact convergence. In contrast, MOCHA performs well regardless of communication cost, and is robust to statistical heterogeneity.

Systems Heterogeneity. MOCHA is also equipped to handle heterogeneity from changing systems environments, such as battery power, memory, or network connection, as we show in Figure 6.2. In particular, we simulate systems heterogeneity by randomly choosing the num-

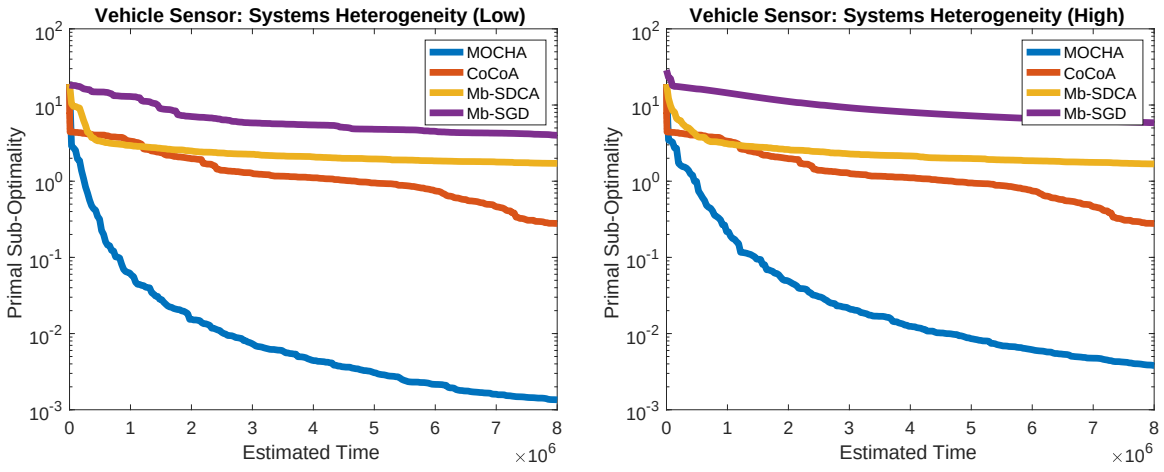

Figure 6.2: The performance of MOCHA relative to other methods is robust to variability from systems heterogeneity (resulting from differences between nodes in terms of, e.g., hardware, network connection, or power). We simulate this heterogeneity by enforcing either high or low variability in the number of local iterations for MOCHA and the mini-batch size for mini-batch methods.

ber of local iterations for MOCHA or the mini-batch size for mini-batch methods, between 10% and 100% of the local data points for high variability environments, to between 90% and 100% for low variability (see Section 6.9 for full details). We do not vary the performance of COCOA, as the impact from statistical heterogeneity alone significantly reduces performance. However, adding systems heterogeneity would reduce performance even further, as the maximum θ value across all nodes would only increase if additional systems challenges were introduced.

6.5.4 Tolerance to Dropped Nodes

Finally, we explore the effect of nodes dropping on the performance of MOCHA. We do not draw comparisons to other methods, as to the best of our knowledge, no other methods for distributed multi-task learning directly address fault tolerance. In MOCHA, we incorporate this setting by allowing $\theta_t^h := 1$, as explored theoretically in Section 6.4. In Figure 6.3, we look at the performance of MOCHA, either for one fixed $\mathbf{W}$ update, or running the entire MOCHA method, as the probability that nodes drop at each iteration (p_t^h in Assumption 3) increases. We see that the performance of MOCHA is robust to relatively high values of p_t^h, both during a single update of $\mathbf{W}$ and in how this effects the performance of the overall method. However, as intuition would suggest, if one of the nodes *never* sends updates (i.e., $p_1^h := 1$ for all h), the method does not converge to the correct solution. This provides validation for our Assumption 3.

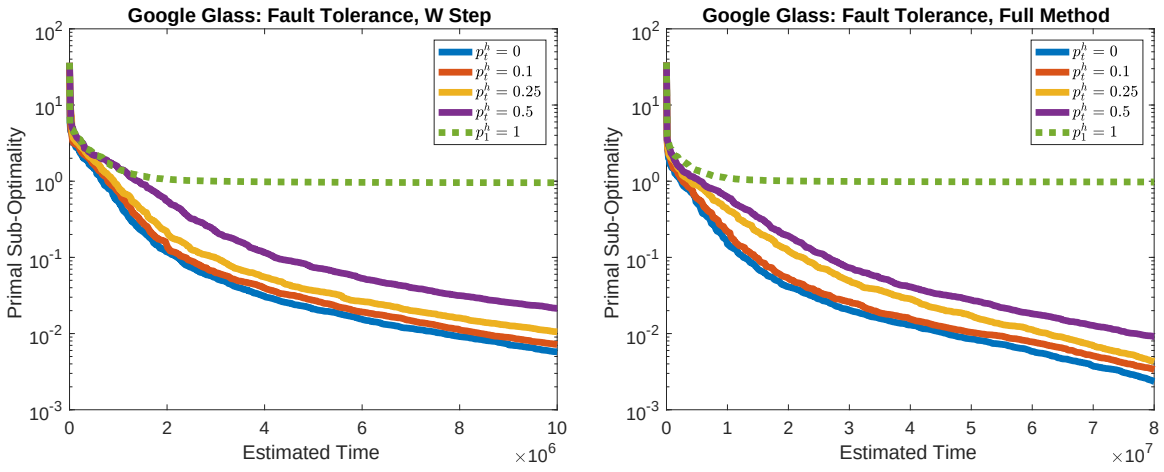

Figure 6.3: The performance of MOCHA is robust to nodes periodically dropping out (fault tolerance). As expected, however, the method will fail to converge to the correct solution if the same node drops out at each round (i.e., $p_1^h := 1$ for all h, as shown in the green-dotted line).

6.6 Multi-Task Learning

In this section, we summarize several popular multi-task learning formulations that can be written in the form of (6.4) and can therefore be addressed by our framework, MOCHA. While the $\mathbf{W}$ update is discussed in Section 6.3, we provide details here on how to solve the Ω update for these formulations.

6.6.1 Multi-Task Learning Formulations

MTL with cluster structure among the tasks. In these MTL models, it is assumed that the weight vectors for each task, $\mathbf{w}_t$, form clusters, i.e., tasks that belong to the same cluster should be 'close' according to some metric. This idea goes back to mean-regularized MTL [25], which assumes that all the tasks form one cluster, and that the weight vectors are close to their mean. Such a regularizer could be formulated in the form of (6.4) by choosing $\mathbf{\Omega} = (\mathbf{I}_{m\times m} - \frac{1}{m}\mathbf{1}\mathbf{1}^T)^2$, where $\mathbf{I}_{m\times m}$ is the identity matrix of size $m \times m$ and $\mathbf{1}_m$ represents a vector of all ones with size m. In this case, we set $\mathcal{R}$ to be

$$\mathcal{R}(\mathbf{W}, \mathbf{\Omega}) = \lambda_1 \operatorname{tr}\big(\mathbf{W}\mathbf{\Omega}\mathbf{W}^T\big) + \lambda_2 \|\mathbf{W}\|_F^2 , \tag{6.11}$$

where $\lambda_1, \lambda_2 > 0$ are parameters. Note that in this formulation, the structural dependence matrix $\mathbf{\Omega}$ is known a-priori. However, it is natural to assume multiple clusters exist, and to learn this clustering structure directly from the data [111]. For such a model, the problem formulation is non-convex if a perfect clustering structure is imposed [37, 111]. However, by

performing a convex relaxation, the following regularizer is obtained [37, 111]

$$\mathcal{R}(\mathbf{W}, \mathbf{\Omega}) = \lambda \ \mathrm{tr}\big(\mathbf{W}(\eta\mathbf{I} + \mathbf{\Omega})^{-1}\mathbf{W}^T\big), \ \mathbf{\Omega} \in \mathcal{Q} = \Big\{\mathbf{Q} \mid \mathbf{Q} \succeq \mathbf{0}, \ \mathrm{tr}(\mathbf{Q}) = k, \ \mathbf{Q} \preceq \mathbf{I}\Big\}, \quad (6.12)$$

where λ and η are regularization parameters, k is the number of clusters, and $\mathbf{\Omega}$ defines the clustering structure.

MTL with probabilistic priors. Another set of MTL models that can be realized by our framework enforce structure by putting probabilistic priors on the dependence among the columns of $\mathbf{W}$. For example, in [108] it is assumed that the weight matrix $\mathbf{W}$ has a prior distribution of the form:

$$\mathbf{W} \ \sim \ \left(\prod_{i=1}^{m} \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})\right) \ \mathcal{MN}(\mathbf{0}, \mathbf{I}_{d\times d} \otimes \mathbf{\Omega}) \,, \quad (6.13)$$

where $\mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$ denotes the normal distribution with mean $\mathbf{0}$ and covariance $\sigma^2\mathbf{I}$, and $\mathcal{MN}(\mathbf{0}, \mathbf{I}_{d\times d} \otimes \mathbf{\Omega})$ denotes the matrix normal distribution with mean $\mathbf{0}$, row covariance $\mathbf{I}_{d\times d}$, and column covariance $\mathbf{\Omega}$. This prior generates a regularizer of the following form [108]:

$$\mathcal{R}(\mathbf{W}, \mathbf{\Omega}) = \lambda \left(\frac{1}{\sigma^2}\|\mathbf{W}\|^2 + \mathrm{tr}\big(\mathbf{W}\mathbf{\Omega}^{-1}\mathbf{W}^T\big) \ + d \ \log|\mathbf{\Omega}|\right), \ \lambda > 0 \,.$$

Unfortunately, such a regularizer is non-convex with respect to $\mathbf{\Omega}$ due to the concavity of $\log|\mathbf{\Omega}|$. To obtain a jointly convex formulation in $\mathbf{\Omega}$ and $\mathbf{W}$, the authors in [108] propose omitting $\log|\mathbf{\Omega}|$ and controling the complexity of $\mathbf{\Omega}$ by adding a constraint on $\mathrm{tr}(\mathbf{\Omega})$:

$$\mathcal{R}(\mathbf{W}, \mathbf{\Omega}) = \lambda \left(\frac{1}{\sigma^2}\|\mathbf{W}\|^2 + \mathrm{tr}\big(\mathbf{W}\mathbf{\Omega}^{-1}\mathbf{W}^T\big)\right), \ \mathbf{\Omega} \in \mathcal{Q} = \Big\{\mathbf{Q} \mid \mathbf{Q} \succeq \mathbf{0}, \ \mathrm{tr}(\mathbf{Q}) = 1\Big\}. \quad (6.14)$$

It is worth noting that unlike the clustered MTL formulations, such as (6.5), the probabilistic formulation in (6.14) can model both positive and negative relationships among the tasks through the covariance matrix.

MTL with graphical models. Another way of modeling task relationships is through the precision matrix. This is popular in graphical models literature [47] because it encodes conditional independence among variables. In other words, if we denote the precision matrix among tasks in matrix variate Gaussian prior with $\mathbf{\Omega}$, then $\mathbf{\Omega}_{i,j} = 0$ if and only if tasks weights $\mathbf{w}_i$ and $\mathbf{w}_j$ are independent given the rest of the task weights [30]. Therefore, assuming sparsity in the structure among the tasks translates to sparsity in matrix $\mathbf{\Omega}$. As a result, we can formulate a sparsity-promoting regularizer by:

$$\mathcal{R}(\mathbf{W}, \mathbf{\Omega}) = \lambda \left(\frac{1}{\sigma^2}\|\mathbf{W}\|^2 + \mathrm{tr}\big(\mathbf{W}\mathbf{\Omega}\mathbf{W}^T\big) - d \ \log|\mathbf{\Omega}|\right) + \lambda_1\|\mathbf{W}\|_1 + \lambda_2\|\mathbf{\Omega}\|_1 \,, \quad (6.15)$$

where $\lambda_1, \lambda_2 \geq 0$ control the sparsity of $\mathbf{W}$ and $\mathbf{\Omega}$ respectively [30]. It is worth noting that although this problem is jointly non-convex in $\mathbf{W}$ and $\mathbf{\Omega}$, it is bi-convex.

6.6.2 Strong Convexity of MTL Regularizers

Recall that in Assumption 2, we presumed that the vectorized formulation of the MTL regularizer is strongly convex with respect to a matrix $\mathbf{M}^{-1}$. In this section we discuss the choice of matrix $\mathbf{M}$ for the widely-used MTL formulations introduced in Section 6.6.1.

Using the notation from Remark 6 for the clustered MTL formulation (6.11), it is easy to see that $\bar{\mathcal{R}}(\mathbf{w}, \bar{\mathbf{\Omega}}) = \lambda_1 \mathbf{w}^T \bar{\mathbf{\Omega}} \mathbf{w} + \lambda_2 \|\mathbf{w}\|_2^2$, where $\bar{\mathbf{\Omega}} := \mathbf{\Omega} \otimes \mathbf{I}_{d\times d}$. As a result, it is clear that $\bar{\mathcal{R}}(\mathbf{w}, \bar{\mathbf{\Omega}})$ is 1-strongly convex with respect to $\mathbf{M}^{-1} = \lambda_1 \bar{\mathbf{\Omega}} + \lambda_2 \mathbf{I}_{md\times md}$.

Using a similar reasoning, it is easy to see that the matrix $\mathbf{M}$ can be chosen as $\lambda^{-1}(\eta\mathbf{I}+\bar{\mathbf{\Omega}})$, $\lambda^{-1}(\frac{1}{\sigma^2}\mathbf{I} + \bar{\mathbf{\Omega}}^{-1})^{-1}$ and $\lambda^{-1}(\frac{1}{\sigma^2}\mathbf{I} + \bar{\mathbf{\Omega}})^{-1}$ for (6.12), (6.14) and (6.15) respectively.

6.6.3 Optimizing $\mathbf{\Omega}$ in MTL Formulations

In this section, we briefly cover approaches to update $\mathbf{\Omega}$ in the MTL formulations introduced in Section 6.6.1. First, it is clear that (6.5) does not require any updates to $\mathbf{\Omega}$, as it is assumed to be fixed. In (6.12), it can be shown [37, 111] that the optimal solution for $\mathbf{\Omega}$ has the same column space as the rows of $\mathbf{W}$. Therefore, the problem boils down to solving a simple convex optimization problem over the eigenvalues of $\mathbf{\Omega}$; see [37, 111] for details. Although outside the scope of this thesis, we note that the bottleneck of this approach to finding $\mathbf{\Omega}$ is computing the SVD of $\mathbf{W}$, which can be a challenging problem when m is large. In the probabilistic model of (6.14), the $\mathbf{\Omega}$ update is given in [108] by $(\mathbf{W}^T\mathbf{W})^{\frac{1}{2}}$, which requires computing the eigenvalue decomposition of $\mathbf{W}^T\mathbf{W}$. For the graphical model formulation, the problem of solving for $\mathbf{\Omega}$ is called sparse precision matrix estimation or graphical lasso [30]. This is a well-studied problem, and many scalable algorithms have been proposed to solve it [30, 35, 94].

Reducing the Size of $\mathbf{\Omega}$ by Sharing Tasks

One interesting aspect of MOCHA is that the method can be easily modified to accommodate the sharing of tasks among the nodes without any change to the local solvers. This property helps the central node to reduce the size of $\mathbf{\Omega}$ and the complexity of its update with minimal changes to the whole system. The following remark highlights this capability.

Remark 8. MOCHA *can be modified to solve problems when there are tasks that are shared among nodes. In this case, each node still solves a data local sub-problem based on its own data for the task, but the central node needs to do an additional aggregation step to add the results for all the nodes that share the data of each task. This reduces the size of matrix* $\mathbf{\Omega}$ *and simplifies its update.*

6.7 Convergence Analysis

Notation. In the rest of this section we use the superscript (h) or h to denote the corresponding variable at iteration (h) of the federated update in MOCHA. When context allows, we drop the superscript to simplify notation.

In order to provide a general convergence analysis, similar to our earlier work, we assume an aggregation parameter $\gamma \in (0, 1]$ in this section. With such an aggregation parameter, the updates in each federated iteration would be $\boldsymbol{\alpha}_t \leftarrow \boldsymbol{\alpha}_t + \gamma \Delta\boldsymbol{\alpha}_t$ and $\mathbf{v}_t \leftarrow \mathbf{v}_t + \gamma \Delta \mathbf{v}_t$. For a more detailed discussion on the role of aggregation parameter, see Section 6.8.1. Note that in Algorithm 4, MOCHA is presented assuming $\gamma = 1$ for simplicity.

Before proving our convergence guarantees, we provide several useful definitions and key lemmas.

Definition 11. *For each task t, define*

$$\sigma_t := \max_{\boldsymbol{\alpha} \in \mathbb{R}^{n_t}} \frac{\|\boldsymbol{X}_t \boldsymbol{\alpha}\|_{M_t}^2}{\|\boldsymbol{\alpha}\|^2} \quad \textit{and} \quad \sigma_{\max} := \max_{t \in [m]} \sigma_t. \tag{6.16}$$

Definition 12. *For any $\boldsymbol{\alpha}$, define the duality gap as*

$$G(\boldsymbol{\alpha}) := \mathcal{D}(\boldsymbol{\alpha}) - (-\mathcal{P}(\boldsymbol{W}(\boldsymbol{\alpha}))), \tag{6.17}$$

where $\mathcal{P}(\boldsymbol{W}) := \sum_{t=1}^{m} \sum_{i=1}^{n_t} \ell_t(\boldsymbol{w}_t^T \boldsymbol{x}_t^i, y_t^i) + \mathcal{R}(\boldsymbol{W}, \boldsymbol{\Omega})$ as in (6.4).

The following lemma uses Assumption 3 to bound the average performance of θ_t^h, which is crucial in providing global convergence guarantees for MOCHA.

Lemma 14. *Under Assumption 3, $\Theta_t^h \leq \bar{\Theta} = p_{\max} + (1 - p_{\max})\Theta_{\max} < 1$.*

Proof. Recalling the definitions $p_t^h := \mathbb{P}[\theta_t^h = 1]$ and $\hat{\Theta}_t^h := \mathbb{E}[\theta_t^h | \theta_t^h < 1, \mathcal{H}_h]$, we have

$$\begin{aligned}
\Theta_t^h &= \mathbb{E}[\theta_t^h | \mathcal{H}_h] \\
&= \mathbb{P}[\theta_t^h = 1] \cdot \mathbb{E}[\theta_t^h | \theta_t^h = 1, \mathcal{H}_h] + (1 - \mathbb{P}[\theta_t^h < 1]) \cdot \mathbb{E}[\theta_t^h | \theta_t^h < 1, \mathcal{H}_h] \\
&= p_t^h \cdot 1 + (1 - p_t^h) \cdot \hat{\Theta}_t^h \leq \bar{\Theta} < 1,
\end{aligned}$$

where the last inequality is due to Assumption 3, and the fact that $\hat{\Theta}_t^h < 1$ by definition. □

The next key lemma bounds the dual objective of an iterate based on the dual objective of the previous iterate and the objectives of local subproblems.

Lemma 15. *For any $\boldsymbol{\alpha}, \Delta\boldsymbol{\alpha} \in \mathbb{R}^n$ and $\gamma \in (0, 1]$ if σ' satisfies (6.28), then*

$$\mathcal{D}\left(\boldsymbol{\alpha} + \gamma \Delta\boldsymbol{\alpha}\right) \leq (1 - \gamma)\mathcal{D}(\boldsymbol{\alpha}) + \gamma \sum_{t=1}^{m} \mathcal{G}_k^{\sigma'}(\Delta\boldsymbol{\alpha}_t; \boldsymbol{v}, \boldsymbol{\alpha}_t)\,. \tag{6.18}$$

Proof. The proof of this lemma is similar to [88, Lemma 1] and follows from the definition of local sub-problems, smoothness of $\mathcal{R}^*$ and the choice of σ' in (6.28). □

Recall that if the functions ℓ_t are $(1/\mu)$-smooth, their conjugates ℓ_t^* will be μ-strongly convex. The lemma below provides a bound on the amount of improvement in dual objective in each iteration.

Lemma 16. *If the functions ℓ_t^* are μ-strongly convex for some $\mu \geq 0$. Then, for any $s \in [0, 1]$.*

$$\mathbb{E}[\mathcal{D}(\boldsymbol{\alpha}^{(h)}) - \mathcal{D}(\boldsymbol{\alpha}^{(h+1)})|\mathcal{H}_h] \geq \gamma \sum_{t=1}^{m} (1 - \bar{\Theta}) \left(sG_t(\boldsymbol{\alpha}^{(h)}) - \frac{\sigma' s^2}{2} J_t \right), \tag{6.19}$$

where

$$G_t(\boldsymbol{\alpha}) := \sum_{i=1}^{n_t} \left[\ell_t^*(-\boldsymbol{\alpha}_t^i) + \ell_t(\boldsymbol{w}_t(\boldsymbol{\alpha})^\top \boldsymbol{x}_t^i, y_t^i) + \boldsymbol{\alpha}_t^i \boldsymbol{w}_t(\boldsymbol{\alpha})^\top \boldsymbol{x}_t^i \right], \tag{6.20}$$

$$J_t := -\tfrac{\mu(1-s)}{\sigma' s} \|(\boldsymbol{u}_t - \boldsymbol{\alpha}_t^{(h)})\|^2 + \|\boldsymbol{X}_t(\boldsymbol{u}_t - \boldsymbol{\alpha}_t^{(h)})\|_{M_t}^2, \tag{6.21}$$

for $\boldsymbol{u}_t \in \mathbb{R}^{n_t}$ with

$$\boldsymbol{u}_t^i \in \partial \ell_t(\boldsymbol{w}_t(\boldsymbol{\alpha})^\top \boldsymbol{x}_t^i, y_t^i). \tag{6.22}$$

Proof. Applying Lemma 15 and recalling $\mathcal{D}(\boldsymbol{\alpha}) = \sum_{t=1}^{m} \mathcal{G}_k^{\sigma'}(\mathbf{0}; \mathbf{v}, \boldsymbol{\alpha}_t)$, we can first bound the improvement for each task separately. Following a similar approach as in the proof of [88, Lemma 7] we can obtain the bound (6.19) which bounds the improvement from $\boldsymbol{\alpha}^{(h)}$ to $\boldsymbol{\alpha}^{(h+1)}$. □

The following lemma relates the improvement of the dual objective in one iteration to the duality gap for the smooth loss functions ℓ_t.

Lemma 17. *If the loss functions ℓ_t are $(1/\mu)$-smooth, then there exists a proper constants $s \in (0, 1]$, such that for any $\gamma \in (0, 1]$ at any iteration h*

$$\mathbb{E}\left[\mathcal{D}(\boldsymbol{\alpha}^{(h)}) - \mathcal{D}(\boldsymbol{\alpha}^{(h+1)})|\mathcal{H}_h\right] \geq s\gamma(1 - \bar{\Theta}) G(\boldsymbol{\alpha}^{(h)}), \tag{6.23}$$

where $G(\boldsymbol{\alpha}^{(h)})$ is the duality gap of $\boldsymbol{\alpha}^{(h)}$ which is defined in (6.17).

Proof. Recall the definition of $\sigma_{\max}$ in (6.16). Now, if we carefully choose $s = \mu/(\mu + \sigma_{\max}\sigma')$, it is easy to show that $J_t \leq 0$ in (6.19); see [88, Theorem 11] for details. The final result follows as a consequence of Lemma 16. □

Note that Lemma 16 holds even if the functions are non-smooth, i.e. $\mu = 0$. However, we cannot infer sufficient decrease of Lemma 17 from Lemma 16 when $\mu = 0$. Therefore, we need additional tools when the losses are L-Liptchitz. The first is the following lemma, which bounds the J term in (6.19).

Lemma 18. *Assume that the loss functions ℓ_t are L-Lipschitz. Denote $J := \sum_{t=1}^{m} J_t$, where J_t is defined in* (6.21)*, then*

$$J \leq 4L^2 \sum_{t=1}^{m} \sigma_t n_t := 4L^2 \sigma, \tag{6.24}$$

where σ_t is defined in (6.16)*.*

Proof. The proof is similar to [53, Lemma 6] and using the definitions of σ and σ_t and the fact that the losses are L-Lipchitz. □

6.7.1 Convergence Analysis for Smooth Losses

Proof of Theorem 12

Let us rewrite (6.23) from Lemma 17 as

$$\begin{aligned}
\mathbb{E}[\mathcal{D}(\boldsymbol{\alpha}^{(h)}) - \mathcal{D}(\boldsymbol{\alpha}^{(h+1)})|\mathcal{H}_h] &= \mathcal{D}(\boldsymbol{\alpha}^{(h)}) - \mathcal{D}(\boldsymbol{\alpha}^{\star}) + \mathbb{E}[\mathcal{D}(\boldsymbol{\alpha}^{\star}) - \mathcal{D}(\boldsymbol{\alpha}^{(h+1)})|\mathcal{H}_h] \\
&\geq s\gamma(1-\bar{\Theta})G(\boldsymbol{\alpha}^{(h)}) \\
&\geq s\gamma(1-\bar{\Theta})\left(\mathcal{D}(\boldsymbol{\alpha}^{(h)}) - \mathcal{D}(\boldsymbol{\alpha}^{\star})\right),
\end{aligned}$$

where the last inequality is due to weak duality, i.e. $G(\boldsymbol{\alpha}^{(h)}) \geq \mathcal{D}(\boldsymbol{\alpha}^{(h)}) - \mathcal{D}(\boldsymbol{\alpha}^{\star})$. Re-arranging the terms in the above inequality, we can easily get

$$\mathbb{E}[\mathcal{D}(\boldsymbol{\alpha}^{(h+1)}) - \mathcal{D}(\boldsymbol{\alpha}^{\star})|\mathcal{H}_h] \leq \left(1 - s\gamma(1-\bar{\Theta})\right)\left(\mathcal{D}(\boldsymbol{\alpha}^{(h)}) - \mathcal{D}(\boldsymbol{\alpha}^{\star})\right) \tag{6.25}$$

Recursively applying this inequality and taking expectations from both sides, we arrive at

$$\mathbb{E}[\mathcal{D}(\boldsymbol{\alpha}^{(h+1)}) - \mathcal{D}(\boldsymbol{\alpha}^{\star})] \leq \left(1 - s\gamma(1-\bar{\Theta})\right)^{h+1}\left(\mathcal{D}(\boldsymbol{\alpha}^{(0)}) - \mathcal{D}(\boldsymbol{\alpha}^{\star})\right). \tag{6.26}$$

Now we can use a simple bound on the initial duality gap [88, Lemma 10], which states that $\mathcal{D}(\boldsymbol{\alpha}^{(0)}) - \mathcal{D}(\boldsymbol{\alpha}^{\star}) \leq n$, to get the final result. It is worth noting that we can translate the bound on the dual distance to optimality to the bound on the duality gap using the following inequalities

$$s\gamma(1-\bar{\Theta})\ \mathbb{E}[G(\boldsymbol{\alpha}^{(H)})] \leq \mathbb{E}[\mathcal{D}(\boldsymbol{\alpha}^{(H)}) - \mathcal{D}(\boldsymbol{\alpha}^{(H+1)})] \leq \mathbb{E}[\mathcal{D}(\boldsymbol{\alpha}^{(H)}) - \mathcal{D}(\boldsymbol{\alpha}^{\star})] \leq \epsilon_{\mathcal{D}}, \tag{6.27}$$

where the first inequality is due to (6.23), the second inequality is due to the optimality of $\boldsymbol{\alpha}^{\star}$, and the last inequality is the bound we just proved for the dual distance to optimality.

Asymptotic Convergence

In the case of smooth loss functions, it is possible to get asymptotic convergence results under milder assumptions. The following corollary is an extension of Theorem 12.

Corollary 19. *If the loss functions ℓ_t are μ-smooth, then under Assumption 2, $\mathbb{E}[\mathcal{D}(\boldsymbol{\alpha}^{(H)}) - \mathcal{D}(\boldsymbol{\alpha}^\star)] \to 0$ as $H \to \infty$ if either of the following conditions hold*

- $\limsup_{h\to\infty} p_t^h < 1$ *and* $\limsup_{h\to\infty} \hat{\Theta}_t^h < 1$.
- *For any task t, $\left(1 - p_t^h\right) \times \left(1 - \hat{\Theta}_t^h\right) = \omega(\frac{1}{h})$. Note that in this case $\lim_{h\to\infty} p_t^h$ can be equal to 1.*

Proof. The proof is similar to the proof of Theorem 12. We can use the same steps to get a sufficient decrease inequality like the one in (6.25), with $\bar{\Theta}$ replaced with $\bar{\Theta}^h := \max_t \ \Theta_t^h$.

$$\mathbb{E}[\mathcal{D}(\boldsymbol{\alpha}^{(h+1)}) - \mathcal{D}(\boldsymbol{\alpha}^\star)|\mathcal{H}_h] \leq \left(1 - s\gamma(1 - \bar{\Theta}^h)\right)\left(\mathcal{D}(\boldsymbol{\alpha}^{(h)}) - \mathcal{D}(\boldsymbol{\alpha}^\star)\right)$$

The rest of the argument follows by applying this inequality recursively and using the assumptions in the corollary. □

6.7.2 Convergence Analysis for Lipschitz Losses: Proof for Theorem 13

Proof. For L-Lipschitz loss functions, the proof follows the same line of reasoning as the proof of Theorem 8 in [53] and therefore we do not cover it in detail. Unlike the case with smooth losses, it is not possible to bound the decrease in dual objective by (6.23). However, we can use Lemma 16 with $\mu = 0$. The next step is to bound $J = \sum_{t=1}^m J_t$ in (6.19), which can be done via Lemma 18. Finally, we apply the inequalities recursively, choose s carefully, and bound the terms in the final inequality. We refer the reader to the proof of Theorem 8 in [53] for more details. It is worth noting that similar to Theorem 12, we can similarly get bounds on the expected duality gap, instead of the dual objective. □

6.8 Choosing σ'

In order to guarantee the convergence of the federated update of MOCHA, the parameter σ', which can be seen as a measure of the difficulty of the data partitioning, must satisfy:

$$\sigma' \sum_{t=1}^m \|\mathbf{X}_t \boldsymbol{\alpha}_t\|_{\mathbf{M}_t}^2 \geq \gamma \|\mathbf{X}\boldsymbol{\alpha}\|_{\mathbf{M}}^2 \quad \forall \boldsymbol{\alpha} \in \mathbb{R}^n \,, \tag{6.28}$$

where $\gamma \in (0, 1]$ is the aggregation parameter for MOCHA Algorithm. Note that in Algorithm 4 we have assumed that $\gamma = 1$. Based on Remark 6, it can be seen that the matrix $\mathbf{M}$ in Assumption 2 can be chosen of the form $\mathbf{M} = \bar{\mathbf{M}} \otimes \mathbf{I}_{d\times d}$, where $\bar{\mathbf{M}}$ is a positive definite matrix of size $m \times m$. For such a matrix, the following lemma shows how to choose σ'.

Lemma 20. *For any positive definite matrix* $\boldsymbol{M} = \bar{\boldsymbol{M}} \otimes \boldsymbol{I}_{d\times d}$*,*

$$\sigma' := \gamma \max_t \sum_{t'=1}^{m} \frac{|\bar{\boldsymbol{M}}_{tt'}|}{\bar{\boldsymbol{M}}_{tt}} \tag{6.29}$$

satisfies the inequality (6.28)*.*

Proof. First of all it is worth noting that for any t, $\mathbf{M}_t = \bar{\mathbf{M}}_t \otimes \mathbf{I}_{d\times d}$. For any $\boldsymbol{\alpha} \in \mathbb{R}^n$

$$\begin{aligned}
\gamma\|\mathbf{X}\boldsymbol{\alpha}\|^2_{\mathbf{M}} &= \gamma \sum_{t,t'} \bar{\mathbf{M}}_{tt'} \langle \mathbf{X}_t\boldsymbol{\alpha}_t, \mathbf{X}_{t'}\boldsymbol{\alpha}_{t'}\rangle \\
&\leq \gamma \sum_{t,t'} \frac{1}{2}|\bar{\mathbf{M}}_{tt'}| \left(\frac{1}{\bar{\mathbf{M}}_{tt}}\|\mathbf{X}_t\boldsymbol{\alpha}_t\|^2_{\mathbf{M}_t} + \frac{1}{\bar{\mathbf{M}}_{t't'}}\|\mathbf{X}_{t'}\boldsymbol{\alpha}_{t'}\|^2_{\mathbf{M}_{t'}} \right) \\
&= \gamma \sum_{t} \left(\sum_{t'} \frac{|\bar{\mathbf{M}}_{tt'}|}{\bar{\mathbf{M}}_{tt}} \right) \|\mathbf{X}_t\boldsymbol{\alpha}_t\|^2_{\mathbf{M}_t} \\
&\leq \sigma' \sum_{t} \|\mathbf{X}_t\boldsymbol{\alpha}_t\|^2_{\mathbf{M}_t},
\end{aligned}$$

where the first inequality is due to Cauchy-Schwartz and the second inequality is due to definition of σ'. □

Remark 9. *Based on the proof of Lemma 20, it is easy to see that we can choose* σ' *differently across the tasks in our algorithm to allow tasks that are more loosely correlated with other tasks to update more aggressively. To be more specific, if we choose* $\sigma'_t = \gamma \sum_{t'} \frac{|\bar{\boldsymbol{M}}_{tt'}|}{\bar{\boldsymbol{M}}_{tt}}$*, then it it is possible to show that* $\gamma\|\boldsymbol{X}\boldsymbol{\alpha}\|^2_{\boldsymbol{M}} \leq \sum_{t=1}^m \sigma'_t \|\boldsymbol{X}_t\boldsymbol{\alpha}_t\|^2_{\boldsymbol{M}_t}$ *for any* $\boldsymbol{\alpha}$*, and the rest of the convergence proofs will follow.*

6.8.1 The Role of Aggregation Parameter γ

The following remark highlights the role of aggregation parameter γ.

Remark 10. *Note that the when* $\gamma < 1$ *the chosen* σ' *in* (6.28) *would be smaller compared to the case where* $\gamma = 1$*. This means that the local subproblems would be solved with less restrictive regularizer. Therefore, the resulting* $\Delta\boldsymbol{\alpha}$ *would be more aggressive. As a result, we need to do a more conservative update* $\boldsymbol{\alpha} + \gamma\Delta\boldsymbol{\alpha}$ *in order to guarantee the convergence.*

Although aggregation parameter γ is proposed to capture this trade off between aggressive subproblems and conservative updates, in most practical scenarios $\gamma = 1$ has the best empirical performance.

6.9 Simulation Details

In this section, we provide additional details and results of our empirical study.

6.9.1 Datasets

In Table 6.2, we provide additional details on the number of tasks (m), feature size (d), and per-task data size (n_t) for each federated dataset described in Section 6.5. The standard deviation n_σ is a measure data skew, and calculates the deviation in the sizes of training data points for each task, n_t. All datasets are publicly available.

Table 6.2: Federated datasets for empirical study.

Dataset	**Tasks** (m)	**Features** (d)	**Min** n_t	**Max** n_t	**Std. Deviation** n_σ
Human Activity	30	561	210	306	26.75
Google Glass	38	180	524	581	11.07
Land Mine	29	9	333	517	65.39
Vehicle Sensor	23	100	872	1,933	267.47

6.9.2 Multi-Task Learning with Highly Skewed Data

Table 6.3: Skewed datasets for empirical study.

Dataset	**Tasks** (m)	**Features** (d)	**Min** n_t	**Max** n_t	**Std. Deviation** σ
HA-Skew	30	561	3	306	84.41
GG-Skew	38	180	6	581	111.79
LM-Skew	29	9	5	517	181.92
VS-Skew	23	100	19	1,933	486.08

To generate highly skewed data, we sample from the original training datasets so that the task dataset sizes differ by at least two orders of magnitude. The sizes of these highly skewed datasets are shown in Table 6.3. When looking at the performance of local, global, and multi-task models for these datasets (Table 6.4), the global model performs slightly better in this setting (particularly for the Human Activity and Land Mine datasets). However, multi-task learning still significantly outperforms all models.

Table 6.4: Average prediction error for skewed data: Means and standard errors over 10 random shuffles.

Model	HA-Skew	GG-Skew	LM-Skew	VS-Skew
Global	2.41 (0.30)	5.38 (0.26)	29.28 (2.47)	13.58 (0.23)
Local	3.87 (0.37)	4.96 (0.20)	27.63 (1.15)	8.15 (0.19)
MTL	**1.93 (0.44)**	**3.28 (0.15)**	**24.12 (1.08)**	**6.91 (0.21)**

6.9.3 Implementation Details

In this section, we provide thorough details on the experimental setup and methods used in our comparison.

Methods.

- **Mb-SGD.** Mini-batch stochastic gradient descent is a standard, widely used method for parallel and distributed optimization. See, e.g., a discussion of this method for the SVM models of interest [81]. We tune both the mini-batch size and step size for best performance using grid search.
- **Mb-SDCA.** Mini-batch SDCA aims to improve mini-batch SGD by employing coordinate ascent in the dual, which has encouraging theoretical and practical backings [85, 90]. For all experiments, we scale the updates for mini-batch stochastic dual coordinate ascent at each round by $\frac{\beta}{b}$ for mini-batch size b and $\beta \in [1, b]$, and tune both parameters with grid search.
- **CoCoA.** We generalize CoCoA [38, 53] to solve (6.4), and tune θ, the fixed approximation parameter, between $(0, 1)$ via grid search. For both CoCoA, and Mocha, we use coordinate ascent as a local solver for the dual subproblems (6.7).
- **Mocha.** The only parameter necessary to tune for Mocha is the level of approximation quality θ_t^h, which can be directly tuned via H_i, the number of local iterations of the iterative method run locally. In Section 6.4, our theory relates this parameter to global convergence, and we discuss the practical effects of this parameter in Section 6.3.5.

Estimated Time. To estimate the time to run methods in the federated setting, we carefully count the floating-point operations (FLOPs) performed in each local iteration for each method, as well as the size and frequency of communication. We convert these counts to estimated time (in milliseconds), using known clock rate and bandwidth/latency numbers for mobile phones in 3G, LTE, and wireless networks [18, 36, 64, 87, 93]. In particular,

we use the following standard model for the cost of one round, h, of local computation / communication on a node t:

$$Time(h,t) := \frac{FLOPs(h,t)}{Clock\ Rate(t)} + Comm(h,t) \tag{6.30}$$

Note that the communication cost $Comm(h,t)$ includes both bandwidth and latency measures. Detailed models of this type have been used to closely match the performance of real-world systems [71].

Statistical Heterogeneity. To account for statistical heterogeneity, MOCHA and the mini-batch methods (Mb-SGD and Mb-SDCA) can adjust the number of local iterations or batch size, respectively, to account for difficult local problems or high data skew. However, because COCOA uses a fixed accuracy parameter θ across both the tasks and rounds, changes in the subproblem difficulty and data skew can make the computation on some nodes much slower than on others. For COCOA, we compute θ via the duality gap, and carefully tune this parameter between $(0, 1)$ for best performance. Despite this, the number of local iterations needed for θ varies significantly across nodes, and as the method runs, the iterations tend to increase as the subproblems become more difficult.

Systems Heterogeneity. Beyond statistical heterogeneity, there can be variability in the systems themselves that cause changes in performance. For example, low battery levels, poor network connections, or low memory may reduce the ability a solver has on a local node to compute updates. As discussed in Section 6.3.5, MOCHA assumes that the central node sets some global clock cycle, and the t-th worker determines the amount of feasible local computation given this clock cycle along with its systems constraints. This specified amount of local computation corresponds to some implicit value of θ_t^h based on the underlying systems and statistical challenges for the t-th node.

To model this setup in our simulations, it suffices to fix a global clock sycle and then randomly assign various amounts of local computation to each local node at each iteration. Specifically, in our simulations we charge all nodes for the cost of one local pass through the data, but some nodes are forced to perform less updates given their current systems constraints. In particular, at each round, we assign the number of updates for node t between $[0.1n_t, n_t]$ for *high variability* environments, and between $[0.9n_t, n_t]$ for *low variability* environments. For the mini-batch methods, we vary the mini-batch size in a similar fashion. However, we do not follow this same process for COCOA, as this would require making the θ parameter worse than what was optimally tuned given statistical heterogeneity. Hence, in these simulations we do not introduce any additional variability for COCOA (and thus present overly optimistic results for COCOA). In spite of this, we see that in both low and high variability settings, MOCHA outperforms all other methods and is robust to systems-related heterogeneity.

Fault Tolerance. Finally, we demonstrate that MOCHA can handle nodes periodically dropping out, which is also supported in our convergence results in Section 6.4. We perform this simulation using the notation defined in Assumption 3, i.e., that each node t temporarily drops on iteration h with probability p_t^h. In our simulations, we modify this probability directly and show that MOCHA is robust to fault tolerance in Figure 6.3. However, note that this robustness is not because of statistical redundancy: If we are to drop out a node entirely (as shown in the green dotted line), MOCHA will not converge to the correct solution. This provides insight into our Assumption 3, which says that the probability that a node drops at each round cannot be exactly equal to one.

Chapter 7

Conclusion

To enable large-scale machine learning, we have developed, analyzed, and evaluated a general-purpose framework for communication-efficient primal-dual optimization in the distributed environment. Our framework, CoCoA, takes a unique approach by using duality to derive subproblems for each machine to solve in parallel. These subproblems closely match the global problem of interest, which allows for state-of-the-art single-machine solvers to be easily re-used in the distributed setting. Further, by allowing the local solvers to find solutions of arbitrary approximation quality to the subproblems on each machine, our framework permits a highly flexible communication scheme. In particular, as the local solvers make updates directly to their local parameters, the need to communicate reduces and can be adapted to the system at hand, which helps to manage the communication bottleneck in the distributed setting.

We have analyzed the impact of the local solver approximation quality, and have derived global primal-dual convergence rates for our framework that are agnostic to the specifics of the local solvers. We have taken particular care in extending our framework to the case of non-strongly convex regularizers, where we introduced a bounded-support modification technique to provide robust convergence guarantees. We demonstrated the efficiency of our framework in an extensive experimental comparison with state-of-the-art distributed solvers. Our framework achieves up to a 50× speedup over other widely-used methods on real-world distributed datasets in the data center setting.

Finally, we have extended our general framework to the burgeoning *federated* setting, which presents a number of new systems and statistical challenges. The proposed modifications in Mocha allow the method to handle practical issues such as non-IID data, stragglers, and fault tolerance. Our methodology is supported by a system-aware analysis that explores the effect of these issues on our convergence guarantees. We demonstrate the effect of the proposed modifications in Mocha with simulations on real-world federated datasets.

In whole, the results in this thesis indicate that by developing methods and theory that carefully expose and consider systems parameters, we can create solutions for modern machine learning are well-aligned with the underlying systems, yielding significant empirical speedups as well as insightful theoretical guarantees.

Bibliography

[1] Amr Ahmed, Abhimanyu Das, and Alexander J Smola. "Scalable hierarchical multi-task learning algorithms for conversion optimization in display advertising". In: *Conference on Web Search and Data Mining*. 2014.

[2] Rie Kubota Ando and Tong Zhang. "A framework for learning predictive structures from multiple tasks and unlabeled data". In: *Journal of Machine Learning Research* 6 (2005), pp. 1817–1853.

[3] Galen Andrew and Jianfeng Gao. "Scalable training of L1-regularized log-linear models". In: *International Conference on Machine Learning*. 2007.

[4] Davide Anguita et al. "A Public Domain Dataset for Human Activity Recognition using Smartphones." In: *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*. 2013.

[5] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. "Convex multi-task feature learning". In: *Machine Learning* 73.3 (2008), pp. 243–272.

[6] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. "Multi-task feature learning". In: *Neural Information Processing Systems*. 2007.

[7] Yossi Arjevani and Ohad Shamir. "Communication complexity of distributed convex learning and optimization". In: *Neural Information Processing Systems*. 2015.

[8] Maria-Florina Balcan et al. "Distributed learning, communication complexity and privacy". In: *Conference on Learning Theory*. 2012.

[9] Heinz H Bauschke and Patrick L Combettes. *Convex Analysis and Monotone Operator Theory in Hilbert Spaces*. New York, NY: Springer Science & Business Media, 2011.

[10] I. M. Baytas et al. "Asynchronous Multi-Task Learning". In: *International Conference on Data Mining*. 2016.

[11] Dimitri P Bersekas and John N Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Englewood Cliffs, NJ: Prentice Hall, 1989.

[12] Yatao Bian et al. "Parallel Coordinate Descent Newton Method for Efficient ℓ_1-Regularized Minimization". In: *arXiv.org* (2013). arXiv: `1306.4080v3 [cs.LG]`.

[13] Flavio Bonomi et al. "Fog computing and its role in the internet of things". In: *SIGCOMM Workshop on Mobile Cloud Computing*. 2012.

[14] J M Borwein and Q Zhu. *Techniques of Variational Analysis*. New York, NY: Springer Science & Buisness Media, 2005.

[15] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge, UK: Cambridge University Press, 2004.

[16] Stephen Boyd et al. "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers". In: *Foundations and Trends in Machine Learning* 3.1 (2010), pp. 1–122.

[17] Joseph K Bradley et al. "Parallel coordinate descent for l1-regularized loss minimization". In: *International Conference on Machine Learning*. 2011.

[18] Aaron Carroll and Gernot Heiser. "An Analysis of Power Consumption in a Smartphone". In: *USENIX Annual Technical Conference*. 2010.

[19] Rich Caruana. "Multitask learning". In: *Machine Learning* 28 (1997), pp. 41–75.

[20] Jianhui Chen, Jiayu Zhou, and Jieping Ye. "Integrating low-rank and group-sparse structures for robust multi-task learning". In: *Conference on Knowledge Discovery and Data Mining*. 2011.

[21] Amol Deshpande et al. "Model-based approximate querying in sensor networks". In: *VLDB Journal* 14.4 (2005), pp. 417–443.

[22] Marco F Duarte and Yu Hen Hu. "Vehicle classification in distributed sensor networks". In: *Journal of Parallel and Distributed Computing* 64.7 (2004), pp. 826–838.

[23] John Duchi, Michael I Jordan, and Brendan McMahan. "Estimation, optimization, and parallelism when data is sparse". In: *Neural Information Processing Systems* (2013).

[24] Celestine Dünner et al. "Primal-dual rates and certificates". In: *International Conference on Machine Learning*. 2016.

[25] Theodoros Evgeniou and Massimiliano Pontil. "Regularized multi-task learning". In: *Conference on Knowledge Discovery and Data Mining*. 2004.

[26] Rong-En Fan et al. "LIBLINEAR: A library for large linear classification". In: *Journal of Machine Learning Research* 9 (2008), pp. 1871–1874.

[27] Olivier Fercoq and Peter Richtárik. "Accelerated, Parallel, and Proximal Coordinate Descent". In: *SIAM Journal on Optimization* 25.4 (2015), pp. 1997–2023.

[28] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. "Regularization paths for generalized linear models via coordinate descent". In: *Journal of Statistical Software* 33.1 (2010), pp. 1–22.

[29] Pedro Garcia Lopez et al. "Edge-centric computing: Vision and Challenges". In: *SIGCOMM Computer Communication Review* 45.5 (2015), pp. 37–42.

[30] André R Gonçalves, Fernando J Von Zuben, and Arindam Banerjee. "Multi-task sparse structure learning with Gaussian copula models". In: *Journal of Machine Learning Research* 17.33 (2016), pp. 1–30.

[31] Jochen Gorski, Frank Pfeuffer, and Kathrin Klamroth. "Biconvex sets and optimization with biconvex functions: a survey and extensions". In: *Mathematical Methods of Operations Research* 66.3 (2007), pp. 373–407.

[32] Christina Heinze, Brian McWilliams, and Nicolai Meinshausen. "DUAL-LOCO: Distributing Statistical Estimation Using Random Projections". In: *International Conference on Artificial Intelligence and Statistics*. 2016.

[33] Jean-Baptiste Hiriart-Urruty and Claude Lemaréchal. *Fundamentals of convex analysis*. Berlin: Springer–Verlag, 2001.

[34] Kirak Hong et al. "Mobile Fog: A Programming Model for Large-scale Applications on the Internet of Things". In: *SIGCOMM Workshop on Mobile Cloud Computing*. 2013.

[35] Cho-Jui Hsieh et al. "Sparse Inverse Covariance Matrix Estimation Using Quadratic Approximation". In: *NIPS 2014 - Advances in Neural Information Processing Systems 27*.

[36] Junxian Huang et al. "An In-depth Study of LTE: Effect of Network Protocol and Application Behavior on Performance". In: *ACM SIGCOMM Conference*. 2013.

[37] Laurent Jacob, Jean-philippe Vert, and Francis R Bach. "Clustered multi-task learning: A convex formulation". In: *Neural Information Processing Systems*. 2009.

[38] Martin Jaggi et al. "Communication-efficient distributed dual coordinate ascent". In: *Neural Information Processing Systems*. 2014.

[39] Xin Jin et al. "Collaborating between local and global learning for distributed online multiple tasks". In: *Conference on Information and Knowledge Management*. 2015.

[40] Tyler Johnson and Carlos Guestrin. "Blitz: A Principled Meta-Algorithm for Scaling Sparse Optimization". In: *International Conference on Machine Learning*. 2015.

[41] Hamed Karimi, Julie Nutini, and Mark Schmidt. "Linear Convergence of Gradient and Proximal-Gradient Methods under the Polyak-Łojasiewicz Condition". In: *European Conference on Machine Learning*. 2016.

[42] Seyoung Kim and Eric P Xing. "Statistical estimation of correlated genome associations to a quantitative trait network". In: *PLoS Genet* 5.8 (2009), e1000587.

[43] Jakub Konečnỳ, H. Brendan McMahan, and Daniel Ramage. "Federated Optimization: Distributed Optimization Beyond the Datacenter". In: *arXiv:1511.03575* (2015).

[44] Jakub Konečnỳ et al. "Federated learning: Strategies for improving communication efficiency". In: *arXiv:1610.05492* (2016).

[45] Tsvi Kuflik, Judy Kay, and Bob Kummerfeld. "Challenges and solutions of ubiquitous user modeling". In: *Ubiquitous display environments*. Springer, 2012, pp. 7–30.

[46] Abhishek Kumar and Hal Daumé. "Learning Task Grouping and Overlap in Multi-task Learning". In: *International Conference on Machine Learning*. 2012.

[47] Steffen L Lauritzen. *Graphical Models*. Vol. 17. Clarendon Press, 1996.

[48] Ching-Pei Lee and Dan Roth. "Distributed box-constrained quadratic optimization for dual linear SVM". In: *International Conference on Machine Learning*. 2015.

[49] Sulin Liu, Sinno Jialin Pan, and Qirong Ho. "Distributed Multi-task Relationship Learning". In: *Conference on Knowledge Discovery and Data Mining* (2017).

[50] Zhaosong Lu and Lin Xiao. "On the Complexity Analysis of Randomized Block-Coordinate Descent Methods". In: *arXiv.org* (2013). arXiv: `1305.4723v1 [math.OC]`.

[51] C. Ma et al. "Distributed Optimization with Arbitrary Local Solvers". In: *arXiv.org* (2015).

[52] Chenxin Ma, Rachael Tappenden, and Martin Takáč. "Linear Convergence of the Randomized Feasible Descent Method Under the Weak Strong Convexity Assumption". In: *arXiv.org* (2015).

[53] Chenxin Ma et al. "Adding vs. Averaging in Distributed Primal-Dual Optimization". In: *International Conference on Machine Learning*. 2015.

[54] Samuel Madden et al. "TAG: A tiny aggregation service for ad-hoc sensor networks". In: *Symposium on Operating Systems Design and Implementation*. 2002.

[55] Samuel Madden et al. "TinyDB: An Acquisitional Query Processing System for Sensor Networks". In: *ACM Transactions on Database Systems* 30.1 (2005), pp. 122–173.

[56] Dhruv Mahajan, S Sathiya Keerthi, and S Sundararajan. "A distributed block coordinate descent method for training l_1 regularized linear classifiers". In: *arXiv.org* (2014). arXiv: `1405.4544v1 [cs.LG]`.

[57] Gideon Mann et al. "Efficient Large-Scale Distributed Training of Conditional Maximum Entropy Models". In: *Neural Information Processing Systems* (2009).

[58] Jakub Marecek, Peter Richtárik, and Martin Takáč. *Distributed Block Coordinate Descent for Minimizing Partially Separable Functions*. Vol. 134. Springer Proceedings in Mathematics & Statistics 261–288. Switzerland: Springer International Publishing, 2015.

[59] David Mateos-Núñez and Jorge Cortés. "Distributed optimization for multi-task learning via nuclear-norm approximation". In: *IFAC Workshop on Distributed Estimation and Control in Networked Systems*. 2015.

[60] H Brendan McMahan and Daniel Ramage. "`http://www.googblogs.com/federated-learning-collaborative-machine-learning-without-centralized-training-data/`". In: *Google* (2017).

[61] H. Brendan McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: *Conference on Artificial Intelligence and Statistics*. 2017.

[62] Brian McWilliams et al. "LOCO: Distributing Ridge Regression with Random Projections". In: *arXiv.org* (2014). arXiv: `1406.3469v2`.

[63] Xiangrui Meng et al. "MLlib: Machine Learning in Apache Spark". In: *Journal of Machine Learning Research* 17.34 (2016), pp. 1–7. arXiv: `1505.06807v1 [cs.LG]`.

[64] Antti P. Miettinen and Jukka K. Nurminen. "Energy Efficiency of Mobile Clients in Cloud Computing". In: *USENIX Conference on Hot Topics in Cloud Computing*. 2010.

[65] Ion Necoara. "Linear convergence of first order methods under weak nondegeneracy assumptions for convex programming". In: *arXiv.org* (2015).

[66] Ion Necoara and Valentin Nedelcu. "Distributed dual gradient methods and error bound conditions". In: *arXiv.org* (2014).

[67] Yurii Nesterov. "Smooth minimization of non-smooth functions". In: *Mathematical Programming* 103.1 (2005), pp. 127–152.

[68] Feng Niu et al. "Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent". In: *Neural Information Processing Systems*. 2011.

[69] Alexandros Pantelopoulos and Nikolaos G Bourbakis. "A survey on wearable sensor-based systems for health monitoring and prognosis". In: *IEEE Transactions on Systems, Man, and Cybernetics* 40.1 (2010), pp. 1–12.

[70] Dmitry Pechyony, Libin Shen, and Rosie Jones. "Solving Large Scale Linear SVM with Distributed Block Minimization". In: *International Conference on Information and Knowledge Management*. 2011.

[71] Hang Qi, Evan R Sparks, and Ameet Talwalkar. "Paleo: A performance model for deep neural networks". In: *International Conference on Learning Representations*. 2017.

[72] Zheng Qu, Peter Richtárik, and Tong Zhang. "Quartz: Randomized dual coordinate ascent with arbitrary sampling". In: *Neural Information Processing Systems*. 2015.

[73] Zheng Qu et al. "SDNA: Stochastic Dual Newton Ascent for Empirical Risk Minimization". In: *International Conference on Machine Learning*. 2016. arXiv: `1502.02268v1 [cs.LG]`.

[74] Shah Atiqur Rahman et al. "Unintrusive eating recognition using Google Glass". In: *Conference on Pervasive Computing Technologies for Healthcare*. 2015.

[75] Parisa Rashidi and Diane J Cook. "Keeping the resident in the loop: Adapting the smart home to the user". In: *IEEE Transactions on systems, man, and cybernetics* 39.5 (2009), pp. 949–959.

[76] Mohammad Rastegari et al. "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks". In: *European Conference on Computer Vision*. 2016.

[77] Sujith Ravi. "`https://research.googleblog.com/2017/02/on-device-machine-intelligence.html`". In: *Google* (2017).

[78] Meisam Razaviyayn, Mingyi Hong, and Zhi-Quan Luo. "A unified convergence analysis of block successive minimization methods for nonsmooth optimization". In: *SIAM Journal on Optimization* 23.2 (2013), pp. 1126–1153.

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

[80] R Tyrrell Rockafellar. *Convex Analysis*. Princeton, NJ: Princeton University Press, 1997.

[81] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. "Pegasos: Primal Estimated sub-GrAdient SOlver for SVM". In: *ICML '07: Proceedings of the 24th international conference on Machine learning* (June 2007).

[82] Shai Shalev-Shwartz and Ambuj Tewari. "Stochastic methods for l_1-regularized loss minimization". In: *Journal of Machine Learning Research* 12 (2011), pp. 1865–1892.

[83] Shai Shalev-Shwartz and Tong Zhang. "Accelerated mini-batch stochastic dual coordinate ascent". In: *Neural Information Processing Systems*. 2013.

[84] Shai Shalev-Shwartz and Tong Zhang. "Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization". In: *Mathematical Programming* Series A (2014), pp. 1–41.

[85] Shai Shalev-Shwartz and Tong Zhang. "Stochastic Dual Coordinate Ascent Methods for Regularized Loss Minimization". In: *Journal of Machine Learning Research* 14 (2013), pp. 567–599.

[86] Ohad Shamir, Nathan Srebro, and Tong Zhang. "Communication-Efficient Distributed Optimization using an Approximate Newton-type Method". In: *International Conference on Machine Learning*. 2014.

[87] Dave Singelée et al. "The communication and computation cost of wireless security". In: *ACM Conference on Wireless Network Security*. 2011.

[88] Virginia Smith et al. "CoCoA: A General Framework for Communication-Efficient Distributed Optimization". In: *arXiv:1611.02189* (2016).

[89] Martin Takáč, Peter Richtárik, and Nathan Srebro. "Distributed Mini-Batch SDCA". In: *arXiv.org* (2015).

[90] Martin Takáč et al. "Mini-Batch Primal and Dual Methods for SVMs". In: *International Conference on Machine Learning*. 2013.

[91] Rachael Tappenden, Martin Takáč, and Peter Richtárik. "On the Complexity of Parallel Coordinate Descent". In: *arXiv.org* (2015). arXiv: `1503.03033v1 [math.OC]`.

[92] Ilya Trofimov and Alexander Genkin. "Distributed Coordinate Descent for L1-regularized Logistic Regression". In: *arXiv.org* (2014). arXiv: `1411.6520v1 [stat.ML]`.

[93] CH Van Berkel. "Multi-core for mobile phones". In: *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association. 2009, pp. 1260–1265.

[94] Huahua Wang et al. "Large scale distributed sparse precision estimation". In: *Neural Information Processing Systems*. 2013.

[95] Jialei Wang, Mladen Kolar, and Nathan Srebro. "Distributed multi-task learning". In: *Conference on Artificial Intelligence and Statistics*. 2016.

[96] Jialei Wang, Mladen Kolar, and Nathan Srebro. "Distributed Multi-Task Learning with Shared Representation". In: *arXiv:1603.02185* (2016).

[97] Po-Wei Wang and Chih-Jen Lin. "Iteration complexity of feasible descent methods for convex optimization." In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1523–1548.

[98] Stephen J Wright. "Coordinate descent algorithms". In: *Mathematical Programming* 151.1 (2015), pp. 3–34.

[99] Ya Xue et al. "Multi-task learning for classification with dirichlet process priors". In: *Journal of Machine Learning Research* 8 (2007), pp. 35–63.

[100] Tianbao Yang. "Trading Computation for Communication: Distributed Stochastic Dual Coordinate Ascent". In: *Neural Information Processing Systems*. 2013.

[101] Tianbao Yang et al. "On Theoretical Analysis of Distributed Stochastic Dual Coordinate Ascent". In: *arXiv.org* (Dec. 2013).

[102] Ian En-Hsu Yen, Shan-Wei Lin, and Shou-De Lin. "A Dual Augmented Block Minimization Framework for Learning with Limited Memory". In: *Neural Information Processing Systems*. 2015.

[103] Hsiang-Fu Yu et al. "Large Linear Classification When Data Cannot Fit in Memory". In: *ACM Transactions on Knowledge Discovery from Data* 5.4 (2012), pp. 1–23.

[104] Jin Yu et al. "A Quasi-Newton Approach to Nonsmooth Convex Optimization Problems in Machine Learning". In: *Journal of Machine Learning Research* 11 (2010), pp. 1145–1200.

[105] Guo-Xun Yuan, Chia-Hua Ho, and Chih-Jen Lin. "An Improved GLMNET for L1-regularized Logistic Regression". In: *Journal of Machine Learning Research* 13 (2012), pp. 1999–2030.

[106] Guo-Xun Yuan et al. "A Comparison of Optimization Methods and Software for Large-scale L1-regularized Linear Classification". In: *Journal of Machine Learning Research* 11 (2010), pp. 3183–3234.

[107] Caoxie Zhang, Honglak Lee, and Kang G. Shin. "Efficient Distributed Linear Classification Algorithms via the Alternating Direction Method of Multipliers". In: *Artificial Intelligence and Statistics Conference*. 2012.

[108] Yu Zhang and Dit-Yan Yeung. "A Convex Formulation for Learning Task Relationships in Multi-task Learning". In: *Conference on Uncertainty in Artificial Intelligence*. 2010.

[109] Yuchen Zhang, John C Duchi, and Martin J Wainwright. "Communication-Efficient Algorithms for Statistical Optimization". In: *Journal of Machine Learning Research* 14 (2013), pp. 3321–3363.

[110] Yuchen Zhang and Xiao Lin. "Stochastic Primal-Dual Coordinate Method for Regularized Empirical Risk Minimization". In: *International Conference on Machine Learning*. 2015.

[111] Jiayu Zhou, Jianhui Chen, and Jieping Ye. "Clustered multi-task learning via alternating structure optimization". In: *Neural Information Processing Systems*. 2011.

[112] Martin A Zinkevich et al. "Parallelized Stochastic Gradient Descent". In: *Neural Information Processing Systems* (2010).