

# **Stochastic, Distributed and Federated Optimization for Machine Learning**

*Jakub Konečný*

Doctor of Philosophy  
University of Edinburgh  
2017



# Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(*Jakub Konečný*)



# Abstract

We study optimization algorithms for the finite sum problems frequently arising in machine learning applications. First, we propose novel variants of stochastic gradient descent with a variance reduction property that enables linear convergence for strongly convex objectives. Second, we study distributed setting, in which the data describing the optimization problem does not fit into a single computing node. In this case, traditional methods are inefficient, as the communication costs inherent in distributed optimization become the bottleneck. We propose a communication-efficient framework which iteratively forms local subproblems that can be solved with arbitrary local optimization algorithms. Finally, we introduce the concept of Federated Optimization/Learning, where we try to solve the machine learning problems without having data stored in any centralized manner. The main motivation comes from industry when handling user-generated data. The current prevalent practice is that companies collect vast amounts of user data and store them in datacenters. An alternative we propose is not to collect the data in first place, and instead occasionally use the computational power of users' devices to solve the very same optimization problems, while alleviating privacy concerns at the same time. In such setting, minimization of communication rounds is the primary goal, and we demonstrate that solving the optimization problems in such circumstances is conceptually tractable.



# Acknowledgements

I would like to express my sincere gratitude to my supervisor, Peter Richtárik, for his guidance through every facet of the research world. I have not only learned how to develop novel research ideas, but also how to clearly write and communicate those ideas, prepare technical presentations and engage audience, interact with other academics and build research collaborations. All of this was crucial for all I have accomplished and lays a solid foundation for my next steps.

I would also like to thank my other supervisors and mentors Jacek Gondzio and Chris Williams for various discussions about research and differences between related fields. I am also grateful to my examination committee, M. Pawan Kumar, Kostas Zygalakis and Andreas Grothey. For stimulating discussions and fun we had together during last four years, I thank other past and current members of our research group: Dominik Csiba, Olivier Fercoq, Robert Gower, Filip Hanzely, Nicolas Loizou, Zheng Qu, Ademir Ribeiro and Rachael Tappenden.

I am indebted to School of Mathematics of the University of Edinburgh for the wonderful, inclusive and productive work environment, to Principal's Career Development Scholarship for initially funding my PhD and to The Centre for Numerical Analysis and Intelligent Software for funding my visit at the Simons Institute for the Theory of Computing. I am extremely grateful to Gill Law, who was always very helpful in overcoming every formal or administrative problems I encountered.

During my study I had the opportunity to work with many brilliant researchers:

- I would like to thank Martin Jaggi and Thomas Hofmann for hosting my visit at ETH Zurich and encouragement in my starting career.
- I would like to thank Martin Takáč for the visit at Lehigh University, and his students Chenxin Ma and Jie Liu. I would also like to thank Katya Scheinberg for her insightful advice regarding my PhD study.
- I would also like to thank Ngai-Man Cheung, Selin Damla Ahipasaoglu and Yiren Zhou who taught me a lot during my visit at Singapore University of Technology and Design.
- I would like to thank Owain Evans and others at Future of Humanity Institute at Oxford University for hosting my extremely inspiring visit which helped me to understand the implications of our work in a much broader context.
- I would like to thank the Simons Institute for the Theory of Computing at University of California in Berkeley, for the opportunity I had at the start of my PhD study.
- Finally, I would like to thank my other collaborators for their ideas, help, jokes and support: Mohamed Osama Ahmed, Dave Bacon, Dmitry Grishchenko, Michal Hagara, Filip Hanzely, Reza Harikandeh, Michael I. Jordan, Nicolas Loizou, H. Brendan McMahan, Barnabás Póczos, Zheng Qu, Daniel Ramage, Sashank J. Reddi, Scott Sallinen, Mark Schmidt, Virginia Smith, Alex Smola, Ananda Theertha Suresh, Alim Virani and Felix X. Yu.

For generous support through Google Doctoral Fellowship I am very thankful to Google, which enabled me to pursue my goals without distractions. I appreciate the advice, patience, discussions and fun I had with many amazing people during my summer internships at Google, including Galen Andrew, Dave Bacon, Keith Bonawitz, Hubert Eichner, Jeffrey Falgout, Emily Fortuna, Gaurav Gite, Seth Hampson, Jeremy Kahn, Peter Kairouz, Eider Moore, Martin Pełikan, John Platt, Daniel Ramage, Marco Tulio Ribeiro, Negar Rostamzadeh, Subarna Tripathi, Felix X. Yu and many others. In particular, I would like to thank for the immerse trust and support I received from Brendan McMahan and Blaise Agüera y Arcas.

For willingness to help and provide recommendation, reference, or connection at various stages of my study, I would like to thank Blaise Agüera y Arcas, Petros Drineas, Martin Jaggi, Michael Mahoney, Mark Schmidt, Nathan Srebro and Lin Xiao. In addition, I am extremely thankful to Isabelle Guyon. I would perhaps not decide to go for PhD without her encouragement and support after we met during my undergraduate study.

I would like to thank the Slovak educational non-profit organizations which helped to shape who I am today, both on academic and personal level — Trojsten, Sezam, P-Mat and Nexteria — and all the people involved in their activities.

Finally, I would like to thank my parents and family, for all the obstacles they quietly removed so I could realize my dreams.

# Contents

<b>Abstract</b>	<b>5</b>
<b>1 Introduction</b>	<b>13</b>
1.1 Empirical Risk Minimization . . . . .	13
1.1.1 Approximation-Estimation-Optimization tradeoff . . . . .	14
1.1.2 Notation . . . . .	15
1.2 Baseline Algorithms . . . . .	15
1.3 Part I: Stochastic Methods with Variance Reduction . . . . .	16
1.3.1 Semi-Stochastic Gradient Descent . . . . .	16
1.3.2 Semi-Stochastic Coordinate Descent . . . . .	16
1.4 Part II: Parallel and Distributed Methods . . . . .	17
1.4.1 Mini-batch Semi-Stochastic Gradient Descent in the Proximal Setting . . .	17
1.4.2 Distributed Optimization with Arbitrary Local Solvers . . . . .	17
1.5 Part III: Federated Optimization . . . . .	19
1.5.1 Distributed Machine Learning for On-device Intelligence . . . . .	19
1.5.2 Distributed Mean Estimation with Communication Constraints . . . . .	20
1.6 Summary . . . . .	21
<b>I Variance Reduced Stochastic Methods</b>	<b>23</b>
<b>2 Semi-Stochastic Gradient Descent</b>	<b>25</b>
2.1 Introduction . . . . .	25
2.1.1 Motivation . . . . .	25
2.1.2 Brief literature review . . . . .	26
2.1.3 Outline . . . . .	26
2.2 Semi-Stochastic Gradient Descent . . . . .	27
2.2.1 S2GD . . . . .	27
2.2.2 S2GD+ . . . . .	28
2.3 Summary of Results . . . . .	28
2.4 Complexity Analysis: Strongly Convex Loss . . . . .	30
2.5 Optimal Choice of Parameters . . . . .	33
2.6 Complexity Analysis: Convex Loss . . . . .	35
2.7 Implementation for sparse data . . . . .	36
2.8 Numerical Experiments . . . . .	38
2.8.1 Comparison with theory . . . . .	38
2.8.2 Comparison with other methods . . . . .	39
2.8.3 Boosted variants of S2GD and SAG . . . . .	41
2.9 Conclusion . . . . .	41
<b>3 Semi-Stochastic Coordinate Descent</b>	<b>43</b>
3.1 Introduction . . . . .	43
3.2 S2CD Algorithm . . . . .	45
3.3 Complexity Result . . . . .	47
3.4 Proof of Lemma 10 . . . . .	49

3.5	Proof of the Main Result . . . . .	51
3.5.1	Coordinate co-coercivity . . . . .	51
3.5.2	Recursion . . . . .	52
3.5.3	Proof of Theorem 11 . . . . .	52
<b>II</b>	<b>Parallel and Distributed Methods with Variance Reduction</b>	<b>55</b>
<b>4</b>	<b>Mini-batch Semi-Stochastic Gradient Descent in the Proximal Setting</b>	<b>57</b>
4.1	Introduction . . . . .	57
4.1.1	Stochastic methods. . . . .	57
4.1.2	Modern stochastic methods . . . . .	58
4.1.3	Linear systems and sketching. . . . .	58
4.2	Contributions . . . . .	59
4.3	The Algorithm . . . . .	59
4.3.1	Deterministic and stochastic proximal gradient methods . . . . .	60
4.3.2	Semi-stochastic methods . . . . .	60
4.3.3	Mini-batch S2GD . . . . .	60
4.4	Analysis . . . . .	61
4.4.1	Assumptions . . . . .	61
4.4.2	Main result . . . . .	61
4.4.3	Special cases: $b = 1$ and $b = n$ . . . . .	62
4.4.4	Mini-batch speedup . . . . .	62
4.4.5	Convergence rate . . . . .	63
4.4.6	Comparison with Acc-Prox-SVRG . . . . .	63
4.5	Efficient implementation for sparse data . . . . .	64
4.6	Experiments . . . . .	66
4.6.1	Speedup of mS2GD . . . . .	67
4.6.2	mS2GD vs other algorithms . . . . .	67
4.6.3	Image deblurring . . . . .	69
4.7	Technical Results . . . . .	69
4.7.1	Proofs . . . . .	70
4.7.2	Proximal lazy updates for $\ell_1$ and $\ell_2$ -regularizers . . . . .	75
4.8	Conclusion . . . . .	76
<b>5</b>	<b>Distributed Optimization with Arbitrary Local Solvers</b>	<b>77</b>
5.1	Motivation . . . . .	77
5.1.1	Contributions . . . . .	77
5.1.2	Outline . . . . .	78
5.2	Background and Problem Formulation . . . . .	78
5.2.1	Problem Formulation . . . . .	80
5.2.2	Technical Assumptions . . . . .	81
5.3	The Framework . . . . .	82
5.3.1	The Local Subproblems . . . . .	83
5.3.2	Practical Communication-Efficient Implementation . . . . .	84
5.3.3	Compatibility of the Subproblems for Aggregating Updates . . . . .	85
5.4	Main Results . . . . .	86
5.4.1	Quality of Local Solutions . . . . .	86
5.4.2	Complexity Bounds . . . . .	86
5.4.3	Discussion and Interpretations of Convergence Results . . . . .	87
5.5	Discussion and Related Work . . . . .	88
5.6	Numerical Experiments . . . . .	91
5.6.1	Exploration of Local Solvers within the Framework . . . . .	91
5.6.2	Averaging vs. Adding the Local Updates . . . . .	94
5.6.3	The Effect of the Subproblem Parameter $\sigma'$ . . . . .	98
5.6.4	Scaling Property . . . . .	99
5.6.5	Performance on a Big Dataset . . . . .	99

5.6.6	Comparison with other distributed methods . . . . .	100
5.7	Conclusion . . . . .	101
5.8	Proofs . . . . .	101
5.8.1	Proof of Lemma 31 . . . . .	101
5.8.2	Proof of Lemma 33 . . . . .	102
5.8.3	Proof of Lemma 34 . . . . .	102
5.8.4	Proofs of Theorems 36 and 37 . . . . .	102
<b>III</b>	<b>Federated Optimization and Learning</b>	<b>111</b>
<b>6</b>	<b>Federated Optimization: Distributed Machine Learning for On-device Intelligence</b>	<b>113</b>
6.1	Introduction . . . . .	113
6.1.1	Problem Formulation . . . . .	114
6.1.2	The Setting of Federated Optimization . . . . .	115
6.2	Related Work . . . . .	116
6.2.1	Baseline Algorithms . . . . .	116
6.2.2	A Novel Breed of Randomized Algorithms . . . . .	117
6.2.3	Distributed Setting . . . . .	119
6.3	Algorithms for Federated Optimization . . . . .	123
6.3.1	Desirable Algorithmic Properties . . . . .	124
6.3.2	SVRG . . . . .	124
6.3.3	Distributed Problem Formulation . . . . .	125
6.3.4	DANE . . . . .	126
6.3.5	SVRG meets DANE . . . . .	127
6.3.6	Federated SVRG . . . . .	128
6.3.7	Further Notes . . . . .	131
6.4	Experiments . . . . .	131
6.4.1	Predicting Comments on Public Google+ Posts . . . . .	131
6.5	Conclusions and Future Challenges . . . . .	134
6.6	Appendix: Distributed Optimization via Quadratic Perturbations . . . . .	134
6.6.1	New Method . . . . .	134
6.6.2	L2-Regularized Linear Predictors . . . . .	136
6.6.3	A Dual Method: Dual Block Proximal Gradient Ascent . . . . .	136
6.6.4	Proof of Theorem 46 . . . . .	137
<b>7</b>	<b>Randomized Distributed Mean Estimation: Accuracy vs Communication</b>	<b>141</b>
7.1	Introduction . . . . .	141
7.1.1	Background and Contributions . . . . .	141
7.1.2	Outline . . . . .	142
7.2	Three Protocols . . . . .	142
7.3	A Family of Randomized Encoding Protocols . . . . .	144
7.3.1	Encoding Protocol with Variable-size Support . . . . .	144
7.3.2	Encoding Protocol with Fixed-size Support . . . . .	145
7.4	Communication Protocols . . . . .	146
7.4.1	Naive . . . . .	146
7.4.2	Varying-length . . . . .	146
7.4.3	Sparse Communication Protocol for Encoder (7.1) . . . . .	147
7.4.4	Sparse Communication Protocol for Encoder (7.4) . . . . .	147
7.4.5	Binary . . . . .	147
7.4.6	Discussion . . . . .	148
7.5	Examples . . . . .	148
7.5.1	Binary Quantization . . . . .	148
7.5.2	Sparse Communication Protocols . . . . .	148
7.6	Optimal Encoders . . . . .	150
7.6.1	Optimal Probabilities for Fixed Node Centers . . . . .	151

7.6.2	Trade-off Curves . . . . .	152
7.7	Further Considerations . . . . .	152
7.7.1	Beyond Binary Encoders . . . . .	152
7.7.2	Preprocessing via Random Rotations . . . . .	153
7.8	Application to Federated Learning . . . . .	154
7.8.1	Structured Update . . . . .	155
7.8.2	Sketched Update . . . . .	156
7.8.3	Experiments . . . . .	156
7.9	Additional Proofs . . . . .	158
<b>IV</b>	<b>Conclusion</b>	<b>161</b>
<b>8</b>	<b>Conclusion and Future Challenges</b>	<b>163</b>

# Chapter 1

## Introduction

In this thesis, we focus on minimization of a finite sum of functions, with particular motivation by machine learning applications:

$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n f_i(w). \quad (1.1)$$

This problem is referred to as Empirical Risk Minimization (ERM) in the machine learning community, and represents optimization problems underpinning a large variety of models — ranging from simple linear regression to deep learning.

This introductory chapter briefly outlines the theoretical framework that gives rise to the ERM problem in Section 1.1, and clarifies what part of the general objective in machine learning we address in this thesis. We follow by a summary of the thesis, highlighting the central contributions without going into the details.

### 1.1 Empirical Risk Minimization

In a prototypical setting of supervised learning, one can access input-output pairs  $x, y \in \mathcal{X} \times \mathcal{Y}$ , which follow an *unknown* probability distribution  $\mathcal{P}(x, y)$ . Typically, inputs and outputs are not known at the same time, and a general goal is to understand the conditional distribution of output given input,  $\mathcal{P}(y|x)$ . For instance, a bank needs to predict whether a transaction is fraudulent, without knowing the true answer immediately, or a recommendation engine predicts which products are users likely to be interested in next. For a more detailed introduction to the following concept, see for instance [161].

The typical learning setting, relies on a definition of a loss function  $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$  and a predictor function  $\phi : \mathcal{X} \rightarrow \mathcal{Y}$ . Loss function  $\ell(\hat{y}, y)$  measures the discrepancy between predicted output  $\hat{y}$  and true output  $y$ , and  $\phi(x)$  maps an input to the predicted output  $\hat{y}$ . With these tools, we are ready to define the *expected risk* of a predictor function  $\phi$  as

$$\mathbf{E}(\phi) \stackrel{\text{def}}{=} \int \ell(\phi(x), y) d\mathcal{P}(x, y).$$

The ideal goal is to find  $\phi^*(x)$  that minimizes the expected risk, defined pointwise as

$$\phi^*(x) \stackrel{\text{def}}{=} \arg \min_{\hat{y} \in \mathcal{Y}} \int \ell(\hat{y}, y) d\mathcal{P}(y|x).$$

Clearly, since we do not assume to know anything about the source distribution  $\mathcal{P}(x, y)$ , finding  $\phi^*$  is an infeasible objective.

Instead, we have access to samples from the distribution. Given a training dataset  $\{(x_i, y_i)\}_{i=1}^n$ , which is assumed to be drawn iid from  $\mathcal{P}(x, y)$ , we can define the *empirical risk*  $\mathbf{E}_n(\phi)$  as a

proxy to the expected risk; also known as monte-carlo integration:

$$\mathbf{E}_n(\phi) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n \ell(\phi(x_i), y_i). \quad (1.2)$$

If we further restrict the predictor  $\phi$  to belong to a specific class of functions, e.g., linear functions, minimizing the empirical risk becomes a tractable objective, motivating the optimization problem (1.1).

### 1.1.1 Approximation-Estimation-Optimization tradeoff

A first learning principle is to restrict the candidate prediction functions to a specific class  $\mathcal{F}$ . This, together with the choice of loss function  $\ell$  effectively corresponds to the choice of a specific machine learning technique. As an example, these can be linear functions of  $x$ , parametrized by a vector  $w$ :  $\phi_w(x) = w^T x$ . A more complex example is a class implicitly defined by the architecture of a neural network.

In practice, an optimization algorithm is applied to obtain an approximate solution  $\hat{\phi}_n$  to the ERM problem. In order to assess the quality of the predictor  $\hat{\phi}_n$ , compared to the ideal but intractable  $\phi^*$ , the standard in learning theory is to define the empirical risk minimizer as

$$\phi_n \stackrel{\text{def}}{=} \arg \min_{\phi \in \mathcal{F}} \mathbf{E}_n(\phi),$$

and the best predictor in terms of expected risk as

$$\phi_{\mathcal{F}}^* \stackrel{\text{def}}{=} \arg \min_{\phi \in \mathcal{F}} \mathbf{E}(\phi).$$

Taking expectation with respect to generation of the sampled data, and possible randomization in an optimization algorithm for solving (1.2), goal of a machine learner is to minimize the *excess error*  $\mathbb{E} [\mathbf{E}(\hat{\phi}_n) - \mathbf{E}(\phi^*)]$  by choosing  $\ell$ ,  $\mathcal{F}$ , and optimization algorithm appropriately, subject to constraints such as computational resources available. The excess error can be decomposed as follows:

$$\mathbb{E} [\mathbf{E}(\hat{\phi}_n) - \mathbf{E}(\phi^*)] = \mathbb{E} [\mathbf{E}(\phi_{\mathcal{F}}^*) - \mathbf{E}(\phi^*)] + \mathbb{E} [\mathbf{E}(\phi_n) - \mathbf{E}(\phi_{\mathcal{F}}^*)] + \mathbb{E} [\mathbf{E}(\hat{\phi}_n) - \mathbf{E}(\phi_n)]. \quad (1.3)$$

The three terms above are referred to as *approximation error*, *estimation error* and *optimization error*, respectively [22]. Approximation error captures how much one loses by restricting the class of candidate predictor functions to  $\mathcal{F}$ . Estimation error captures the loss incurred by minimizing the empirical risk instead of the expected risk we would ideally optimize for. Optimization error is a result of finding an approximate optimum of the empirical risk, using an optimization algorithm.

These terms are subject to various tradeoffs that have been studied for decades. For instance, expanding the functional class  $\mathcal{F}$  will naturally decrease the approximation error, but can increase the estimation error due to overfitting the training dataset. Increasing the size of the dataset available (increasing  $n$ ) makes the empirical risk a better approximation of the expected risk, thus decreasing estimation error. However, it will likely make it computationally more expensive to attain the same optimization error.

Detailed overview of the interplay of these terms is beyond the scope of this work. We focus only on the optimization error, and what computational resources are necessary to obtain particular levels of the error. This can mean using optimization algorithms on a single compute node, with or without parallel processing units, or in a distributed environment. We describe this in the rest of this chapter. A comprehensive literature overview is deferred to Section 6.2, in which we explain why none of the existing methods are suitable for Federated Optimization, a novel conceptual setting for the ERM problem.

### 1.1.2 Notation

With focus on the optimization objective only, we can reformulate (1.2) into notation used throughout the thesis. We are interested in minimizing a function  $P(w)$ , which in full generality takes the form

$$P(w) = \frac{1}{n} \sum_{i=1}^n f_i(w) + R(w). \quad (1.4)$$

The functions  $f_i$  are assumed to be convex, and hide the dependence on the training data  $x_i, y_i$ , which is mostly irrelevant for the subsequent analysis. The (optional) function  $R(w)$  is referred to as a regularizer, and is in practice used primarily to prevent overfitting or enforce structural properties of the solution. Most common choice are L2 ( $R(w) = \lambda/2\|w\|_2^2$ ) or L1 ( $R(w) = \lambda\|w\|_1$ ) regularizers for some choice of  $\lambda > 0$ .

By  $\nabla f_i(w)$  we denote the gradient of  $f_i$  at point  $w$ . We denote  $\langle \cdot, \cdot \rangle$  the standard Euclidean inner product of two vectors, and unless specified otherwise,  $\|\cdot\| = \sqrt{\langle \cdot, \cdot \rangle}$  refers to the standard Euclidean norm. We denote the proximal operator of function  $\psi : \mathbb{R}^d \rightarrow \mathbb{R}$  as  $\text{prox}_\psi(z) = \arg \min_{s \in \mathbb{R}^d} \left\{ \frac{1}{2} \|s - z\|^2 + \psi(s) \right\}$ . The convex (Fenchel) conjugate of a function  $\phi : \mathbb{R}^d \rightarrow \mathbb{R}$  is defined as the function  $\phi^* : \mathbb{R}^d \rightarrow \mathbb{R} \cup \{\infty\}$ , with  $\phi^*(u) = \sup_{s \in \mathbb{R}^d} \{s^T u - \phi(s)\}$ .

## 1.2 Baseline Algorithms

Two of the most basic algorithms that can be used to solve the ERM problem (1.4) are Gradient Descent and Stochastic Gradient Descent, which we introduce now. For simplicity, we assume that  $R(w) = 0$  for all  $w \in \mathbb{R}^d$ .

A trivial benchmark for solving (1.4) is *Gradient Descent* (GD) in the case when functions  $f_i$  are smooth (or Subgradient Descent for non-smooth functions) [127]. The GD algorithm performs the iteration

$$w \leftarrow w - \frac{h}{n} \sum_{i=1}^n \nabla f_i(w) = w - h \nabla P(w),$$

where  $h > 0$  is a stepsize parameter.

Common practice in machine learning is to collect vast amounts of data  $\{x_i, y_i\}_{i=1}^n$ , which in the context of our objective translates to very large  $n$  — the number of functions  $f_i$ . This makes GD impractical, as it needs to process the whole dataset in order to evaluate a single gradient and update the model. This makes GD rather impractical for most state-of-the-art applications. An alternative is to use a randomized algorithm, the computational complexity of which is independent of  $n$ , in a single iteration.

This basic, albeit in practice extremely popular, alternative to GD is *Stochastic Gradient Descent* (SGD), dating back to the seminal work of Robbins and Monro [153]. In the context of (1.4), SGD samples a random function  $i \in \{1, 2, \dots, n\}$  in iteration  $t$ , and performs the update

$$w \leftarrow w - h_t \nabla f_i(w),$$

where  $h_t > 0$  is a stepsize parameter.

Intuitively speaking, this method works because if  $i$  is sampled uniformly at random from indices 1 to  $n$ , the update direction is an unbiased estimate of the gradient:  $\mathbb{E}[\nabla f_i(w)] = \nabla P(w)$ . However, noise introduced by sampling slows down the convergence, and a diminishing sequence of stepsizes  $h_t$  is necessary for the method to converge.

If we consider the case of strongly convex  $P$ , the core differences between GD and SGD can be summarized as follows. Let  $\kappa$  denote the condition number defined as the ratio of smoothness and strong convexity parameters of  $P$ . GD enjoys fast convergence rate, while SGD converges slowly. That is, in order to obtain  $\epsilon$ -accuracy, GD needs  $\mathcal{O}(\kappa \log(1/\epsilon))$  iterations, while SGD needs in general  $\mathcal{O}(\kappa^2/\epsilon)$  iterations. On the other hand, GD requires computation of  $n$  gradients of  $f_i$ , which can be computationally expensive when data is abundant, while SGD needs to evaluate only a single gradient, and thus does not depend on  $n$ .

In most practical applications in machine learning, a high accuracy is not necessary, as the ERM problem is only a proxy to the original problem of interest, and error will eventually be

dominated by approximation and estimation errors described in Section 1.1.1. Indeed, SGD can sometimes yield a decent solution in just a single pass through data — equivalent to a single GD step.

## 1.3 Part I: Stochastic Methods with Variance Reduction

In Chapters 2 and 3, we propose and analyze semi-stochastic methods for minimizing the ERM objective (1.4). These methods interpolate between the baselines (GD and SGD) in the sense that they enjoy benefits of both methods. In particular, we show that using a trick to reduce the variance of stochastic gradients, we are able to maintain the linear convergence of GD, while using stochastic gradients. In order to do so, we still need to evaluate the full gradient  $\nabla P$ , but only a few times during the entire runtime of the method.

### 1.3.1 Semi-Stochastic Gradient Descent

The Semi-Stochastic Gradient Descent (S2GD) method, proposed in Chapter 2 (see Algorithm 1), runs in two nested loops. In the outer loop, it only computes and stores the full gradient of the objective,  $\nabla P(w^t)$ , the expensive operation one tries to avoid in general. In the inner loop, with some choice of stepsize  $h$ , the update step is iteratively computed as

$$w \leftarrow w - h[\nabla f_i(w) - \nabla f_i(w^t) + \nabla P(w^t)] \quad (1.5)$$

for a randomly sampled  $i \in \{1, \dots, n\}$ . The core idea is that the gradients of  $\nabla f_i$  are used to estimate the change of the full gradient  $\nabla P$  between the points  $w^t$  and  $w$ , as opposed to estimating the full gradient directly. It is easy to verify that if  $i$  is sampled uniformly at random, the update direction is an unbiased estimate of the gradient  $\nabla P(w)$ .

We assume that  $P$  is  $\mu$ -strongly convex, and the functions  $f_i$  are  $L$ -smooth. Let  $\kappa = L/\mu$  denote the condition number. In a core result, we are able to show that for the update direction from (1.5), we have that

$$\mathbb{E} [\|\nabla f_i(w) - \nabla f_i(w^t) + \nabla P(w^t)\|^2] \leq 4L[P(w) - P(w^*)] + 4(L - \mu)[P(w^t) - P(w^*)]$$

This shows that as both  $w$  and  $w^t$  progress towards the optimum  $w^*$ , the second moment — and thus also variance — of the estimate of the gradient diminishes. Together with unbiasedness, we use this to build a recursion which yields (see Theorem 4) that for iterates  $w^t$  in the outer loop of the S2GD algorithm, we have

$$\mathbb{E} [P(w^t) - P(w^*)] \leq c^t(P(w^0) - P(w^*)),$$

where  $c$  is a convergence factor depending on the algorithm parameters and properties of the optimization problem.

Each iteration of S2GD requires evaluation of  $\nabla P$  — or  $n$  stochastic gradients  $\nabla f_i$ , followed by a random number of stochastic updates. In Theorem 6 we show that we can obtain an  $\epsilon$ -approximate solution after evaluating  $\mathcal{O}((n + \kappa) \log(1/\epsilon))$  stochastic gradients. This is achieved by running the algorithm for  $\log(1/\epsilon)$  iterations of the outer loop, with  $\mathcal{O}(\kappa)$  stochastic updates (1.5) in the inner loop. Contrast this with the rate of GD, which per iteration requires the evaluation of  $n$  stochastic gradients, and thus needs a total of  $\mathcal{O}(n\kappa \log(1/\epsilon))$  gradient evaluations to attain the same accuracy. Given that  $\kappa$  is commonly of the same order as  $n$ , which is typically very large. This amounts to an improvement by several orders of magnitude!

### 1.3.2 Semi-Stochastic Coordinate Descent

In Chapter 3 we present Semi-Stochastic Coordinate Descent (S2CD) method as Algorithm 4, which builds upon the S2GD algorithm by accessing oracle that returns partial stochastic derivatives  $\nabla_j f_i$ . In general, one can think of S2GD and similar stochastic methods as sampling rows of a data matrix. S2CD is sampling both rows and columns of the data matrix, in order to get computationally even cheaper stochastic iterations. Contrasted with S2GD, the outer

loop stays the same, but stochastic steps in the inner loop update only a single coordinate of the variable  $w$ , and the update (1.5) changes to

$$w \leftarrow w - hp_j^{-1} \left( \frac{1}{nq_{ij}} (\nabla_j f_i(w) - \nabla_j f_i(w^t)) + \nabla_j P(w^t) \right) e_j,$$

where  $p_j, q_{ij}$  are parameters of the algorithm determined by the problem structure, and  $e_j$  is the  $j^{th}$  unit vector in  $\mathbb{R}^d$ . As before, the update direction is an unbiased estimate of the gradient  $\nabla P(w)$ . However, the actual update has only one non-zero element.

We prove that the convergence of S2GD algorithm depends on a different notion of condition number (see Corollary 12), which is always larger or equal to the one driving convergence of S2GD. However, the advantage is the usage of a weaker oracle, which only accesses partial derivatives. Whether S2CD is practically better than S2GD depends on the structure of a given problem, and whether it is possible to implement the oracle efficiently.

## 1.4 Part II: Parallel and Distributed Methods

In Part II, we do not focus on serial algorithms, but explore possibilities of using parallel and distributed computing architectures.

By parallel computation we mean utilization of multiple computing nodes with a shared memory architecture, such as a multi-core processor. The main characteristic is that access to all data is equally fast for every computing node. When we say we solve the ERM problem (1.4) in a distributed setting, we mean that the amount of data describing the problem is too big to fit into a random access memory (RAM) or cannot even be stored on a single computing node. In both cases, the main difference to traditional, or serial, algorithms is that reading any data from a RAM can be several orders of magnitude faster than it is to send it to another node in a network. This single fact presents a considerable challenge to iterative optimization algorithms that are inherently sequential, particularly to stochastic methods with fast iterations such as those described in Part I.

### 1.4.1 Mini-batch Semi-Stochastic Gradient Descent in the Proximal Setting

In Chapter 4 we present parallel version of the S2GD algorithm, which we call mS2GD (see Algorithm 5), which improves upon S2GD algorithm in two major aspects. First, we allow and analyze the effect of mini-batching — sampling multiple  $f_i$  at the same time to obtain a more accurate stochastic gradient. This admits simple use of parallel computing architectures, as the computation of multiple stochastic gradients can be trivially parallelized. Second, the mS2GD algorithm is applicable to problem (1.4) with general  $R(w)$  that admits an efficient proximal operator. This includes non-smooth regularizers such as  $R(w) = \|w\|_1$ . We demonstrate the algorithm is useful also in the area of signal processing and imaging.

In Section 4.4.4 we show that mini-batching alone can decrease the total amount of work necessary for convergence even if we were only to run it as a serial algorithm. More precisely, we show that up to a certain threshold on the mini-batch size (in typical circumstances about 30), the algorithm enjoys superlinear speedup in terms of the number of stochastic iterations needed. Additionally, in Section 4.5, we discuss an efficient implementation of the algorithm for problems with sparse data, which is significantly different and much more efficient than the intuitive straightforward implementation.

### 1.4.2 Distributed Optimization with Arbitrary Local Solvers

In the following, we review a paradigm for comparing efficiency of algorithms for distributed optimization, and describe what conceptual problem of these algorithms we address in Chapter 5.

Let us suppose we have many algorithms  $\mathcal{A}$  readily available to solve problem (1.4). The question is: “How do we decide which algorithm is the best for our purpose?”

First, consider the basic setting on a single machine. Let us define  $\mathcal{I}_{\mathcal{A}}(\epsilon)$  as the number of iterations algorithm  $\mathcal{A}$  needs to converge to some fixed  $\epsilon$  accuracy. Let  $\mathcal{T}_{\mathcal{A}}$  be the time needed for a single iteration. Then, in practice, the best algorithm is one that minimizes the following quantity:<sup>1</sup>

$$\text{TIME} = \mathcal{I}_{\mathcal{A}}(\epsilon) \times \mathcal{T}_{\mathcal{A}}. \quad (1.6)$$

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

The natural extension to distributed setting is the formula (1.7). Let  $c$  be the time needed for communication during a single iteration of the algorithm  $\mathcal{A}$ . For the sake of clarity, we suppose we consider only algorithms that need to communicate a single vector in  $\mathbb{R}^d$  per round of communication. Note that essentially all first-order algorithms fall into this category, so it is not a restrictive assumption. This effectively sets  $c$  to be a constant, given any particular distributed architecture one has at disposal.

$$\text{TIME} = \mathcal{I}_{\mathcal{A}}(\epsilon) \times (c + \mathcal{T}_{\mathcal{A}}). \quad (1.7)$$

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

Most, if not all, of the current state-of-the-art algorithms in setting (1.4) are stochastic and rely on doing very large number (big  $\mathcal{I}_{\mathcal{A}}(\epsilon)$ ) of very fast (small  $\mathcal{T}_{\mathcal{A}}$ ) iterations. Even a relatively small  $c$  can cause the practical performance of their naively distributed variants drop down dramatically, because we still have  $c \gg \mathcal{T}_{\mathcal{A}}$ .

This has been indeed observed in practice, and motivated development of new methods, designed with this fact in mind from scratch, which we review in detail later in Section 6.2.3. Although this is a good development in academia — motivation to explore a novel problem, it is not necessarily good news for the industry.

Many companies have spent significant resources to build excellent algorithms to tackle their problems of form (1.4), fine tuned to the specific patterns arising in their data and side applications required. When the data companies collect grows too large to be processed on a single machine, it is understandable that they would be reluctant to throw away their fine tuned algorithms and start building new ones from scratch.

We address this issue in Chapter 5 and propose the CoCoA<sup>+</sup> framework, which works roughly as follows. The framework formulates a general way to form a specific local subproblem on each node, based on the data available locally, and a single shared vector that needs to be distributed to all nodes. Within an iteration of the framework, each node uses *any* optimization algorithm  $\mathcal{A}$ , to reach a relative  $\Theta$  accuracy on the local subproblem. Updates from all nodes are then aggregated to form an update to the global model.

The efficiency paradigm changes as follows:

$$\text{TIME} = \mathcal{I}(\epsilon, \Theta) \times (c + \mathcal{T}_{\mathcal{A}}(\Theta)). \quad (1.8)$$

Time per iteration  $\mathcal{T}_{\mathcal{A}}(\Theta)$  denotes the time algorithm  $\mathcal{A}$  needs to reach the relative  $\Theta$  accuracy on the local subproblem. The number of iterations  $\mathcal{I}(\epsilon, \Theta)$  is independent of the choice of the algorithm  $\mathcal{A}$  used as a local solver. We provide a theoretical result, which specifies how many iterations of the CoCoA<sup>+</sup> framework are needed to achieve overall  $\epsilon$  accuracy, if we solve the local subproblems to relative  $\Theta$  accuracy. Here,  $\Theta = 0$  would mean we require the local subproblem to be solved to optimality, and  $\Theta = 1$  that we do not need any progress whatsoever. The general upper bound on the number of iterations of the CoCoA<sup>+</sup> framework is  $\mathcal{I}(\epsilon, \Theta) = \frac{\mathcal{O}(\log(1/\epsilon))}{1-\Theta}$  for strongly convex objectives (see Theorem 36). From the inverse dependence on  $1 - \Theta$  we can see that there is a fundamental limit to the number of communication rounds needed. Hence, intuitively speaking, it will probably not be efficient to spend excessive resources

---

<sup>1</sup>Considering only algorithms that can be run on a given machine.

to attain very high local accuracy (small  $\Theta$ ).

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

1. It allows practitioners to continue using their fine-tuned solvers for solving subproblems within the CoCoA<sup>+</sup> framework, that can run only on single machine, instead of having to implement completely new algorithms from scratch.
2. The actual performance in terms of the number of rounds of communication is independent from the choice of the optimization algorithm, making it much easier to optimize the overall performance.
3. Since the constant  $c$  is architecture dependent, running optimal algorithm on one network does not have to be optimal on another. In the setting (1.7), this could mean that when moving from one cluster to another, a completely different algorithm might be necessary for strong performance, which is a major change. In the setting (1.8), this can be improved by simply changing  $\Theta$ , which will be implicitly determined by the number of iterations algorithm  $\mathcal{A}$  runs for.

Extensive experimental evaluation in Section 5.6 demonstrates the versatility of the proposed framework, which has already been implemented and adopted in the popular Apache Spark engine.

## 1.5 Part III: Federated Optimization

Mobile phones and tablets are now the primary computing devices for many people. In many cases, these devices are rarely separated from their owners [34], and the combination of rich user interactions and powerful sensors means they have access to an unprecedented amount of data, much of it private in nature. Machine learning models learned on such data hold the promise of greatly improving usability by powering more intelligent applications. However, the sensitive nature of the data means there are risks and responsibilities related to storing it in a centralized location.

### 1.5.1 Distributed Machine Learning for On-device Intelligence

In Chapter 6 we move beyond distributed optimization and advocate an alternative — *federated learning* — that leaves the training data distributed on the mobile devices, and learns a shared model by aggregating locally computed updates via a central coordinating server. This is a direct application of the principle of focused collection or data minimization proposed by the 2012 White House report on the privacy of consumer data [182]. Since these updates are specific to improving the current model, they can be purely ephemeral — there is no reason to store them on the server once they have been applied. Further, they will never contain more information than the raw training data (by the data processing inequality), and will generally contain much less. A principal advantage of this approach is the decoupling of model training from the need for direct access to the raw training data. Clearly, some trust of the server coordinating the training is still required, and depending on the details of the model and algorithm, the updates may still contain private information. However, for applications where the training objective can be specified on the basis of data available on each client, federated learning can significantly reduce privacy and security risks by limiting the attack surface to only the device, rather than the device and the cloud.

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

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

- **Non-IID:** Data on each node may be drawn from a different distribution; that is, the data points available locally are far from being a representative sample of the overall distribution.
- **Unbalanced:** Different nodes may vary by orders of magnitude in the number of training examples they hold.

In the work presented in Chapter 6, we are particularly concerned with **sparse** data, where some features occur on a small subset of nodes or data points only. Although this is not a necessary characteristic of the setting of federated optimization, we will show that the sparsity structure can be used to develop an effective algorithm for federated optimization. Note that data arising in the largest machine learning problems being solved currently — ad click-through rate predictions — are extremely sparse.

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

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

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

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

The main purpose of the work is initiate research into, and design a first practical implementation of federated optimization. Our results suggest that with suitable optimization algorithms, very little is lost by not having an IID sample of the data available, and that even in the presence of a large number of nodes, we can still achieve convergence in relatively few rounds of communication. Recently, Google announced that they applied this concept in one of their applications used by over 500 million users [117].

### 1.5.2 Distributed Mean Estimation with Communication Constraints

In Chapter 7 we theoretically address the problem of computing the average of vectors stored on different computing devices, while placing a constraint on the amount of bits communicated. This problem could become a bottleneck in practical application of federated optimization, when a server aggregates the updates  $\delta$  from individual users due to in general asymmetric speed of internet connections [1], or cryptographic protocols used to protect individual update [15] that further increase the size of the data needed to be communicated back to server.

We decompose the problem into a choice of encoding and communicating protocol, of which we propose several types. In the setting when we are allowed to communicate a single bit per

element of vectors to be aggregated, we prove the best known bounds on the mean square error of the resulting average.

We apply some of these ideas in the context of federated optimization in Section 7.8, in which we focus on training deep feed-forward models. We propose two major types of techniques to reduce the size of each update — structured and sketching updates. With structured updates, we enforce the local update to be optimized for to be of a specific structure, such as low rank or sparse, which lets us succinctly represent the update using fewer parameters. By sketching updates, we mean the reduction of size of the update by sketching techniques, such as subsampling and quantization used jointly with random structured rotations. In the main contribution, we show we are able to train a deep convolutional model for the CIFAR-10 data, while in total communicating less bits than necessary to represent the original size of the data.

## 1.6 Summary

The content of this thesis is based on the following publications and preprints:

- Chapter 2: *Jakub Konečný and Peter Richtárik: “Semi-stochastic gradient descent methods.” arXiv preprint 1312.1666 (2013).* [89]
- Chapter 3: *Jakub Konečný, Zheng Qu and Peter Richtárik: “Semi-stochastic coordinate descent.” Optimization Methods and Software, 1–13 (2017).* [84]
- Chapter 4: *Jakub Konečný, Jie Liu, Peter Richtárik and Martin Takáč: “Mini-batch semi-stochastic gradient descent in the proximal setting.” IEEE Journal of Selected Topics in Signal Processing 10(2), 242–255 (2016).* [81]
- Chapter 5: *Chenxin Ma, Jakub Konečný, Martin Jaggi, Virginia Smith, Michael I Jordan, Peter Richtárik and Martin Takáč: “Distributed optimization with arbitrary local solvers.” Optimization Methods and Software, 1–36 (2017).* [104]
- Chapter 6: *Jakub Konečný, Brendan McMahan, Daniel Ramage and Peter Richtárik: “Federated optimization: distributed machine learning for on-device intelligence.” arXiv preprint 1610.02527 (2016).* [83] [88]
- Chapter 7: *Jakub Konečný and Peter Richtárik: “Randomized Distributed Mean Estimation: Accuracy vs Communication.” arXiv preprint 1611.07555 (2016).* [86]
- Section 7.8: *Jakub Konečný, Brendan McMahan, Felix Yu, Peter Richtárik, Ananda Theertha Suresh and Dave Bacon: “Federated learning: Strategies for improving communication efficiency.” arXiv preprint 1610.05492 (2016).*

During the course of my study, I also co-authored the following works which were not used in the formation of this thesis:

- *Reza Harikandeh, Mohamed Osama Ahmed, Alim Virani, Mark Schmidt, Jakub Konečný and Scott Sallinen: “Stop wasting my gradients: Practical SVRG.” Advances in Neural Information Processing Systems 28, 2251–2259 (2015).* [74]
- *Sashank J Reddi, Jakub Konečný, Peter Richtárik, Barnabás Póczós and Alex Smola “AIDE: Fast and communication efficient distributed optimization.” arXiv preprint 1608.06879 (2016).* [147]
- *Filip Hanzely, Jakub Konečný, Nicolas Loizou, Peter Richtárik, Dmitry Grishchenko: Privacy Preserving Randomized Gossip Algorithms. arXiv preprint arXiv:1706.07636. (2017)* [73]
- *Jakub Konečný and Peter Richtárik. “Simple complexity analysis of simplified direct search.” arXiv preprint 1410.0390 (2014).* [85]

In [74], we propose several practical improvements to the S2GD algorithm from Chapter 2. In particular, we show that it is not necessary to compute a full gradient in the outer loop; instead, an inexact estimate is sufficient for the same convergence. Additionally, we prove that the algorithm is not only a superior *optimization* algorithm, but is also a better *learning* algorithm, in the sense of the approximation-estimation-optimization tradeoff outlined in Section 1.1.1.

In [147], we propose a framework for distributed optimization in a similar spirit to the one presented in Chapter 5, but one that works only with the primal problem. Accelerated Inexact DANE is the first distributed method for (1.4) that nearly matches communication complexity lower bounds while being implementable using first-order oracle only. This work also makes a link to a distributed algorithm that we propose but do not analyze as Algorithm 12, and indirectly provides its theoretical convergence guarantee.

In [73], we introduce and analyze techniques for preserving privacy of initial values in randomized algorithms for average consensus problem.

Finally, in [85] we simplify and unify complexity proof techniques for direct search — a classical algorithm for derivative-free optimization.

# **Part I**

## **Variance Reduced Stochastic Methods**



# Chapter 2

## Semi-Stochastic Gradient Descent

### 2.1 Introduction

Many problems in data science (e.g., machine learning, optimization and statistics) can be cast as loss minimization problems of the form

$$\min_{w \in \mathbb{R}^d} P(w), \quad (2.1)$$

where

$$P(w) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n f_i(w). \quad (2.2)$$

Here  $d$  typically denotes the number of features / coordinates,  $n$  the number of data points, and  $f_i(w)$  is the loss incurred on data point  $i$ . That is, we are seeking to find a predictor  $w \in \mathbb{R}^d$  minimizing the average loss  $P(w)$ . In big data applications,  $n$  is typically very large; in particular,  $n \gg d$ .

Note that this formulation includes more typical formulation of  $L2$ -regularized objectives —  $P(w) = \frac{1}{n} \sum_{i=1}^n \tilde{f}_i(w) + \frac{\lambda}{2} \|w\|^2$ . We hide the regularizer into the function  $f_i(w)$  for the sake of simplicity of resulting analysis.

#### 2.1.1 Motivation

Let us now briefly review two basic approaches to solving problem (2.1).

1. *Gradient Descent*. Given  $w^k \in \mathbb{R}^d$ , the gradient descent (GD) method sets

$$w^{k+1} = w^k - h \nabla P(w^k),$$

where  $h$  is a stepsize parameter and  $\nabla P(w^k)$  is the gradient of  $P$  at  $w^k$ . We will refer to  $\nabla P(x)$  by the name *full gradient*. In order to compute  $\nabla P(w^k)$ , we need to compute the gradients of  $n$  functions. Since  $n$  is big, it is prohibitive to do this at every iteration.

2. *Stochastic Gradient Descent (SGD)*. Unlike gradient descent, stochastic gradient descent [125, 196] instead picks a random  $i$  (uniformly) and updates

$$w^{k+1} = w^k - h \nabla f_i(w^k).$$

Note that this strategy drastically reduces the amount of work that needs to be done in each iteration (by the factor of  $n$ ). Since

$$\mathbb{E} [\nabla f_i(w^k)] = \nabla P(w^k),$$

we have an unbiased estimator of the full gradient. Hence, the gradients of the component functions  $f_1, \dots, f_n$  will be referred to as *stochastic gradients*. A practical issue with SGD is that consecutive stochastic gradients may vary a lot or even point in opposite directions. This slows down the performance of SGD. On balance, however, SGD is preferable to GD in applications where low accuracy solutions are sufficient. In such cases usually only a small number of passes through the data (i.e., work equivalent to a small number of full gradient evaluations) are needed to find an acceptable  $w$ . For this reason, SGD is extremely popular in fields such as machine learning.

In order to improve upon GD, one needs to reduce the cost of computing a gradient. In order to improve upon SGD, one has to reduce the variance of the stochastic gradients. In this chapter we propose and analyze a *Semi-Stochastic Gradient Descent* (S2GD) method. Our method combines GD and SGD steps and reaps the benefits of both algorithms: it inherits the stability and speed of GD and at the same time retains the work-efficiency of SGD.

### 2.1.2 Brief literature review

Several recent papers, e.g., [148], [156, 158], [163] and [80] proposed methods which achieve similar variance-reduction effect, directly or indirectly. These methods enjoy linear convergence rates when applied to minimizing smooth strongly convex loss functions.

The method in [148] is known as Random Coordinate Descent for Composite functions (RCDC), and can be either applied directly to (2.1), or to a dual version of (2.1). Unless specific conditions on the problem structure are met, application to the primal directly are not as computationally efficient as its dual version. Application of a coordinate descent method to the dual formulation of (2.1) is generally referred to as Stochastic Dual Coordinate Ascent (SDCA) [78]. The algorithm in [163] exhibits this duality, and the method in [172] extends the primal-dual framework to the parallel / mini-batch setting. Parallel and distributed stochastic coordinate descent methods were studied in [151, 58, 57].

Stochastic Average Gradient (SAG) by [156], is one of the first SGD-type methods, other than coordinate descent methods, which were shown to exhibit linear convergence. The method of [80], called Stochastic Variance Reduced Gradient (SVRG), arises as a special case in our setting for a suboptimal choice of a single parameter of our method. The Epoch Mixed Gradient Descent (EMGD) method, [195], is similar in spirit to SVRG, but achieves a quadratic dependence on the condition number instead of a linear dependence, as is the case with SDCA, SAG, SVRG and with our method.

Earlier works of [62], [48] and [10] attempt to interpolate between GD and SGD and decrease variance by varying the sample size. These methods however do not realize the kind of improvements as the recent methods above. For partially related classical work on semi-stochastic approximation methods we refer<sup>1</sup> the reader to the papers of [112, 113], which focus on general stochastic optimization.

### 2.1.3 Outline

We start in Section 2.2 by describing two algorithms: S2GD, which we analyze, and S2GD+, which we do not analyze, but which exhibits superior performance in practice. We then move to summarizing some of the main contributions of this chapter in Section 2.3. Section 2.4 is devoted to establishing expectation and high probability complexity results for S2GD in the case of a strongly convex loss. The results are generic in that the parameters of the method are set arbitrarily. Hence, in Section 2.5 we study the problem of choosing the parameters optimally, with the goal of minimizing the total workload (# of processed examples) sufficient to produce a result of specified accuracy. In Section 2.6 we establish high probability complexity bounds for S2GD applied to a non-strongly convex loss function. Discussion of efficient implementation for sparse data is in Section 2.7. Finally, in Section 2.8 we perform very encouraging numerical experiments on real and artificial problem instances. A brief conclusion can be found in Section 2.9.

---

<sup>1</sup>We thank Zaid Harchaoui who pointed us to these papers a few days before we posted our work to arXiv.

## 2.2 Semi-Stochastic Gradient Descent

In this section we describe two novel algorithms: S2GD and S2GD+. We analyze the former only. The latter, however, has superior convergence properties in our experiments.

We assume throughout the chapter that the functions  $f_i$  are convex and  $L$ -smooth.

**Assumption 1.** *The functions  $f_1, \dots, f_n$  have Lipschitz continuous gradients with constant  $L > 0$  (in other words, they are  $L$ -smooth). That is, for all  $x, z \in \mathbb{R}^d$  and all  $i = 1, 2, \dots, n$ ,*

$$f_i(z) \leq f_i(x) + \langle \nabla f_i(x), z - x \rangle + \frac{L}{2} \|z - x\|^2.$$

(This implies that the gradient of  $P$  is Lipschitz with constant  $L$ , and hence  $P$  satisfies the same inequality.)

In one part of this chapter (Section 2.4) we also make the following additional assumption:

**Assumption 2.** *The average loss  $P$  is  $\mu$ -strongly convex,  $\mu > 0$ . That is, for all  $x, z \in \mathbb{R}^d$ ,*

$$P(z) \geq P(x) + \langle \nabla P(x), z - x \rangle + \frac{\mu}{2} \|z - x\|^2. \quad (2.3)$$

(Note that, necessarily,  $\mu \leq L$ .)

### 2.2.1 S2GD

Algorithm 1 (S2GD) depends on three parameters: stepsize  $h$ , constant  $m$  limiting the number of stochastic gradients computed in a single epoch, and a  $\nu \in [0, \mu]$ , where  $\mu$  is the strong convexity constant of  $P$ . In practice,  $\nu$  would be a known lower bound on  $\mu$ . Note that the algorithm works also without any knowledge of the strong convexity parameter — the case of  $\nu = 0$ .

---

#### Algorithm 1 Semi-Stochastic Gradient Descent (S2GD)

---

```

parameters:  $m = \max$  # of stochastic steps per epoch,  $h$  = stepsize,  $\nu$  = lower bound on  $\mu$ 
for  $k = 0, 1, 2, \dots$  do
     $g^k \leftarrow \frac{1}{n} \sum_{i=1}^n \nabla f_i(w^k)$ 
     $y^{k,0} \leftarrow w^k$ 
    Let  $t^k \leftarrow t$  with probability  $(1 - \nu h)^{m-t}/\beta$  for  $t = 1, 2, \dots, m$ 
    for  $t = 0$  to  $t^k - 1$  do
        Pick  $i \in \{1, 2, \dots, n\}$ , uniformly at random
         $y^{k,t+1} \leftarrow y^{k,t} - h(g^k + \nabla f_i(y^{k,t}) - \nabla f_i(w^k))$ 
    end for
     $w^{k+1} \leftarrow y^{k,t^k}$ 
end for

```

---

The method has an outer loop, indexed by epoch counter  $k$ , and an inner loop, indexed by  $t$ . In each epoch  $k$ , the method first computes  $g^k$ —the *full* gradient of  $P$  at  $w^k$ . Subsequently, the method produces a random number  $t^k \in [1, m]$  of steps, following a geometric law, where

$$\beta \stackrel{\text{def}}{=} \sum_{t=1}^m (1 - \nu h)^{m-t}, \quad (2.4)$$

with only *two stochastic gradients* computed in each step.<sup>2</sup> For each  $t = 0, \dots, t^k - 1$ , the stochastic gradient  $\nabla f_i(w^k)$  is subtracted from  $g^k$ , and  $\nabla f_i(y^{k,t})$  is added to  $g^k$ , which ensures that, one has

$$\mathbb{E}[g^k + \nabla f_i(y^{k,t}) - \nabla f_i(w^k)] = \nabla P(y^{k,t}),$$

---

<sup>2</sup>It is possible to get away with computing only a *single* stochastic gradient per inner iteration, namely  $\nabla f_i(y^{k,t})$ , at the cost of having to store in memory  $\nabla f_i(w^k)$  for  $i = 1, 2, \dots, n$ . This, however, can be impractical for big  $n$ .

where the expectation is with respect to the random variable  $i$ .

Hence, the algorithm is an instance of stochastic gradient descent – albeit executed in a nonstandard way (compared to the traditional implementation described in the introduction).

Note that for all  $k$ , the expected number of iterations of the inner loop,  $\mathbb{E}[t^k]$ , is equal to

$$\xi = \xi(m, h) \stackrel{\text{def}}{=} \sum_{t=1}^m t \frac{(1 - \nu h)^{m-t}}{\beta}. \quad (2.5)$$

Also note that  $\xi \in [\frac{m+1}{2}, m]$ , with the lower bound attained for  $\nu = 0$ , and the upper bound for  $\nu h \rightarrow 1$ .

### 2.2.2 S2GD+

We also implement Algorithm 2, which we call S2GD+. In our experiments, the performance of this method is superior to all methods we tested, including S2GD. However, we do not analyze the complexity of this method and leave this as an open problem.

---

#### Algorithm 2 S2GD+

---

**parameters:**  $\alpha \geq 1$  (e.g.,  $\alpha = 1$ )

1. Run SGD for a single pass over the data (i.e.,  $n$  iterations); output  $w$
  2. Starting from  $w_0 = w$ , run a version of S2GD in which  $t^k = \alpha n$  for all  $k$
- 

In brief, S2GD+ starts by running SGD for 1 epoch (1 pass over the data) and then switches to a variant of S2GD in which the number of the inner iterations,  $t^k$ , is not random, but fixed to be  $n$  or a small multiple of  $n$ .

The motivation for this method is the following. It is common knowledge that SGD is able to progress much more in one pass over the data than GD (where this would correspond to a single gradient step). However, the very first step of S2GD is the computation of the full gradient of  $P$ . Hence, by starting with a single pass over data using SGD and *then* switching to S2GD, we obtain a superior method in practice.<sup>3</sup>

## 2.3 Summary of Results

In this section we summarize some of the main results and contributions of this work.

1. **Complexity for strongly convex  $P$ .** If  $P$  is strongly convex, S2GD needs

$$\mathcal{W} = O((n + \kappa) \log(1/\varepsilon)) \quad (2.6)$$

work (measured as the total number of evaluations of the stochastic gradient, accounting for the full gradient evaluations as well) to output an  $\varepsilon$ -approximate solution (in expectation or in high probability), where  $\kappa = L/\mu$  is the condition number. This is achieved by running S2GD with stepsize  $h = \Theta(1/L)$ ,  $k = \Theta(\log(1/\varepsilon))$  epochs (this is also equal to the number of full gradient evaluations) and  $m = \Theta(\kappa)$  (this is also roughly equal to the number of stochastic gradient evaluations in a single epoch). The complexity results are stated in detail in Sections 2.4 and 2.5 (see Theorems 4, 5 and 6; see also (2.27) and (2.26)).

2. **Comparison with existing results.** This complexity result (2.6) matches the best-known results obtained for strongly convex losses in recent work such as [156], [80] and [195]. Our treatment is most closely related to [80], and contains their method (SVRG) as a special case. However, our complexity results have better constants, which has a discernable effect in practice. In Table 2.1 we summarize our results in the strongly convex case with other existing results for different algorithms.

<sup>3</sup>Using a single pass of SGD as an initialization strategy was already considered in [156]. However, the authors claim that their implementation of vanilla SAG did not benefit from it. S2GD does benefit from such an initialization due to it starting, in theory, with a (heavy) full gradient computation.

Algorithm	Complexity/Work
Nesterov's algorithm	$O(\sqrt{\kappa}n \log(1/\varepsilon))$
EMGD	$O((n + \kappa^2) \log(1/\varepsilon))$
SAG	$O(\max\{n, \kappa\} \log(1/\varepsilon))$
SDCA	$O((n + \kappa) \log(1/\varepsilon))$
SVRG	$O((n + \kappa) \log(1/\varepsilon))$
<b>S2GD</b>	$O((n + \kappa) \log(1/\varepsilon))$

Table 2.1: Comparison of performance of selected methods suitable for solving (2.1). The complexity/work is measured in the number of stochastic gradient evaluations needed to find an  $\varepsilon$ -solution.

We should note that the rate of convergence of Nesterov's algorithm [127] is a deterministic result. EMGD and S2GD results hold with high probability (see Theorem 5 for precise statement). Complexity results for stochastic coordinate descent methods are also typically analyzed in the high probability regime [148]. The remaining results hold in expectation. Notion of  $\kappa$  is slightly different for SDCA, which requires explicit knowledge of the strong convexity parameter  $\mu$  to run the algorithm. In contrast, other methods do not algorithmically depend on this, and thus their convergence rate can adapt to any additional strong convexity locally.

3. **Complexity for convex  $f$ .** If  $P$  is *not* strongly convex, then we propose that S2GD be applied to a perturbed version of the problem, with strong convexity constant  $\mu = O(L/\varepsilon)$ . An  $\varepsilon$ -accurate solution of the original problem is recovered with arbitrarily high probability (see Theorem 8 in Section 2.6). The total work in this case is

$$\mathcal{W} = O((n + L/\varepsilon) \log(1/\varepsilon)),$$

that is,  $\tilde{O}(1/\varepsilon)$ , which is better than the standard rate of SGD.

4. **Optimal parameters.** We derive formulas for optimal parameters of the method which (approximately) minimize the total workload, measured in the number of stochastic gradients computed (counting a single full gradient evaluation as  $n$  evaluations of the stochastic gradient). In particular, we show that the method should be run for  $O(\log(1/\varepsilon))$  epochs, with stepsize  $h = \Theta(1/L)$  and  $m = \Theta(\kappa)$ . No such results were derived for SVRG in [80].
5. **One epoch.** Consider the case when S2GD is run for 1 epoch only, effectively limiting the number of full gradient evaluations to 1, while choosing a target accuracy  $\varepsilon$ . We show that S2GD with  $\nu = \mu$  needs

$$O(n + (\kappa/\varepsilon) \log(1/\varepsilon))$$

work only (see Table 2.2). This compares favorably with the optimal complexity in the  $\nu = 0$  case (which reduces to SVRG), where the work needed is

$$O(n + \kappa/\varepsilon^2).$$

For two epochs one could just say that we need  $\sqrt{\varepsilon}$  decrease in each epoch, thus having complexity of  $O(n + (\kappa/\sqrt{\varepsilon}) \log(1/\sqrt{\varepsilon}))$ . This is already better than general rate of SGD ( $O(1/\varepsilon)$ ).

6. **Special cases.** GD and SVRG arise as special cases of S2GD, for  $m = 1$  and  $\nu = 0$ , respectively.<sup>4</sup>
7. **Low memory requirements.** Note that SDCA and SAG, unlike SVRG and S2GD, need to store all gradients  $\nabla f_i$  (or dual variables) throughout the iterative process. While

<sup>4</sup>While S2GD reduces to GD for  $m = 1$ , our *analysis* does not say anything meaningful in the  $m = 1$  case - it is too coarse to cover this case. This is also the reason behind the empty space in the “Complexity” box column for GD in Table 2.2.

Parameters	Method	Complexity
$\nu = \mu, k = \Theta(\log(\frac{1}{\varepsilon}))$ $\& m = \Theta(\kappa)$	Optimal S2GD	$O((n + \kappa) \log(\frac{1}{\varepsilon}))$
$m = 1$ $\nu = 0$ $\nu = 0, k = 1, m = \Theta(\frac{\kappa}{\varepsilon^2})$ $\nu = \mu, k = 1, m = \Theta(\frac{\kappa}{\varepsilon} \log(\frac{1}{\varepsilon}))$	GD SVRG [80] Optimal SVRG with 1 epoch Optimal S2GD with 1 epoch	$\overline{O}((n + \kappa) \log(\frac{1}{\varepsilon}))$ $O(n + \frac{\kappa}{\varepsilon^2})$ $O(n + \frac{\kappa}{\varepsilon} \log(\frac{1}{\varepsilon}))$

Table 2.2: Summary of complexity results and special cases. Condition number:  $\kappa = L/\mu$  if  $f$  is  $\mu$ -strongly convex and  $\kappa = 2L/\varepsilon$  if  $f$  is *not* strongly convex and  $\epsilon \leq L$ .

this may not be a problem for a modest sized optimization task, this requirement makes such methods less suitable for problems with very large  $n$ .

8. **S2GD+.** We propose a “boosted” version of S2GD, called S2GD+, which we do not analyze. In our experiments, however, it performs vastly superior to all other methods we tested, including GD, SGD, SAG and S2GD. S2GD alone is better than both GD and SGD if a highly accurate solution is required. The performance of S2GD and SAG is roughly comparable, even though in our experiments S2GD turned to have an edge.

## 2.4 Complexity Analysis: Strongly Convex Loss

For the purpose of the analysis, let

$$\mathcal{F}^{k,t} \stackrel{\text{def}}{=} \sigma(w^1, w^2, \dots, w^k; y^{k,1}, y^{k,2}, \dots, y^{k,t}) \quad (2.7)$$

be the  $\sigma$ -algebra generated by the relevant history of S2GD. We first isolate an auxiliary result.

**Lemma 3.** *Consider the S2GD algorithm. For any fixed epoch number  $k$ , the following identity holds:*

$$\mathbb{E}[P(w^{k+1})] = \frac{1}{\beta} \sum_{t=1}^m (1 - \nu h)^{m-t} \mathbb{E}[P(y^{k,t-1})]. \quad (2.8)$$

*Proof.* By the tower law of conditional expectations and the definition of  $w^{k+1}$  in the algorithm, we obtain

$$\begin{aligned} \mathbb{E}[P(w^{k+1})] &= \mathbb{E}[\mathbb{E}[P(w^{k+1}) | \mathcal{F}^{k,m}]] = \mathbb{E}\left[\sum_{t=1}^m \frac{(1 - \nu h)^{m-t}}{\beta} P(y^{k,t-1})\right] \\ &= \frac{1}{\beta} \sum_{t=1}^m (1 - \nu h)^{m-t} \mathbb{E}[P(y^{k,t-1})]. \end{aligned}$$

□

We now state and prove the main result of this section.

**Theorem 4.** *Let Assumptions 1 and 2 be satisfied. Consider the S2GD algorithm applied to solving problem (2.1). Choose  $0 \leq \nu \leq \mu$ ,  $0 < h < \frac{1}{2L}$ , and let  $m$  be sufficiently large so that*

$$c \stackrel{\text{def}}{=} \frac{(1 - \nu h)^m}{\beta \mu h (1 - 2Lh)} + \frac{2(L - \mu)h}{1 - 2Lh} < 1. \quad (2.9)$$

*Then we have the following convergence in expectation:*

$$\mathbb{E}[P(w^k) - P(w^*)] \leq c^k (P(w^0) - P(w^*)). \quad (2.10)$$

Before we proceed to proving the theorem, note that in the special case with  $\nu = 0$ , we recover the result of [80] (with a minor improvement in the second term of  $c$  where  $L$  is replaced by  $L - \mu$ ), namely

$$c = \frac{1}{\mu h(1 - 2Lh)m} + \frac{2(L - \mu)h}{1 - 2Lh}. \quad (2.11)$$

If we set  $\nu = \mu$ , then  $c$  can be written in the form (see (2.4))

$$c = \frac{(1 - \mu h)^m}{(1 - (1 - \mu h)^m)(1 - 2Lh)} + \frac{2(L - \mu)h}{1 - 2Lh}. \quad (2.12)$$

Clearly, the latter  $c$  is a major improvement on the former one. We shall elaborate on this further later.

*Proof.* It is well-known [127, Theorem 2.1.5] that since the functions  $f_i$  are  $L$ -smooth, they necessarily satisfy the following inequality:

$$\|\nabla f_i(w) - \nabla f_i(w^*)\|^2 \leq 2L[f_i(w) - f_i(w^*) - \langle \nabla f_i(w^*), w - w^* \rangle].$$

By summing these inequalities for  $i = 1, \dots, n$ , and using  $\nabla P(w^*) = 0$ , we get

$$\frac{1}{n} \sum_{i=1}^n \|\nabla f_i(w) - \nabla f_i(w^*)\|^2 \leq 2L[P(w) - P(w^*) - \langle \nabla P(w^*), w - w^* \rangle] = 2L(P(w) - P(w^*)). \quad (2.13)$$

Let  $G^{k,t} \stackrel{\text{def}}{=} g^k + \nabla f_i(y^{k,t-1}) - \nabla f_i(w^k)$  be the direction of update at  $k^{th}$  iteration in the outer loop and  $t^{th}$  iteration in the inner loop. Taking expectation with respect to  $i$ , conditioned on the  $\sigma$ -algebra  $\mathcal{F}^{k,t-1}$  (2.7), we obtain<sup>5</sup>

$$\begin{aligned} \mathbb{E} [\|G^{k,t}\|^2] &= \mathbb{E} [\|\nabla f_i(y^{k,t-1}) - \nabla f_i(w^*) - \nabla f_i(w^k) + \nabla f_i(w^*) + g^k\|^2] \\ &\leq 2\mathbb{E} [\|\nabla f_i(y^{k,t-1}) - \nabla f_i(w^*)\|^2] + 2\mathbb{E} [\|\nabla f_i(w^k) - \nabla f_i(w^*)\|^2] \\ &= 2\mathbb{E} [\|\nabla f_i(y^{k,t-1}) - \nabla f_i(w^*)\|^2] + 2\mathbb{E} [\|\nabla f_i(w^k) - \nabla f_i(w^*)\|^2] \\ &\quad - 4\mathbb{E} [\langle \nabla P(w^k), \nabla f_i(w^k) - \nabla f_i(w^*) \rangle] + 2\|\nabla P(w^k)\|^2 \\ &\stackrel{(2.13)}{\leq} 4L[P(y^{k,t-1}) - P(w^*) + P(w^k) - P(w^*)] \\ &\quad - 2\|\nabla P(w^k)\|^2 - 4\langle \nabla P(w^k), \nabla P(w^*) \rangle \\ &\stackrel{(2.3)}{\leq} 4L[P(y^{k,t-1}) - P(w^*)] + 4(L - \mu)[P(w^k) - P(w^*)]. \end{aligned} \quad (2.14)$$

Above we have used the bound  $\|x' + x''\|^2 \leq 2\|x'\|^2 + 2\|x''\|^2$  and the fact that

$$\mathbb{E} [G^{k,t} \mid \mathcal{F}^{k,t-1}] = \nabla P(y^{k,t-1}). \quad (2.15)$$

We now study the expected distance to the optimal solution (a standard approach in the

---

<sup>5</sup>For simplicity, we suppress the  $\mathbb{E} [\cdot \mid \mathcal{F}^{k,t-1}]$  notation here.

analysis of gradient methods):

$$\begin{aligned}
\mathbb{E} [\|y^{k,t} - w^*\|^2 \mid \mathcal{F}^{k,t-1}] &= \|y^{k,t-1} - w^*\|^2 - 2h \langle \mathbb{E} [G^{k,t} \mid \mathcal{F}^{k,t-1}], y^{k,t-1} - w^* \rangle \\
&\quad + h^2 \mathbb{E} [\|G^{k,t}\|^2 \mid \mathcal{F}^{k,t-1}] \\
&\stackrel{(2.14)+(2.15)}{\leq} \|y^{k,t-1} - w^*\|^2 - 2h \langle \nabla P(y^{k,t-1}), y^{k,t-1} - w^* \rangle \\
&\quad + 4Lh^2 [P(y^{k,t-1}) - P(w^*)] \\
&\quad + 4(L-\mu)h^2 [P(w^k) - P(w^*)] \\
&\stackrel{(2.3)}{\leq} \|y^{k,t-1} - w^*\|^2 - 2h [P(y^{k,t-1}) - P(w^*)] \\
&\quad - \nu h \|y^{k,t-1} - w^*\|^2 + 4Lh^2 [P(y^{k,t-1}) - P(w^*)] \\
&\quad + 4(L-\mu)h^2 [P(w^k) - P(w^*)] \\
&= (1 - \nu h) \|y^{k,t-1} - w^*\|^2 \\
&\quad - 2h(1 - 2Lh) [P(y^{k,t-1}) - P(w^*)] \\
&\quad + 4(L-\mu)h^2 [P(w^k) - P(w^*)]. \tag{2.16}
\end{aligned}$$

By rearranging the terms in (2.16) and taking expectation over the  $\sigma$ -algebra  $\mathcal{F}^{k,t-1}$ , we get the following inequality:

$$\begin{aligned}
\mathbb{E} [\|y^{k,t} - w^*\|^2] + 2h(1 - 2Lh) \mathbb{E} [P(y^{k,t-1}) - P(w^*)] \\
\leq (1 - \nu h) \mathbb{E} [\|y^{k,t-1} - w^*\|^2] + 4(L-\mu)h^2 \mathbb{E} [P(w^k) - P(w^*)]. \tag{2.17}
\end{aligned}$$

Finally, we can analyze what happens after one iteration of the outer loop of S2GD, i.e., between two computations of the full gradient. By summing up inequalities (2.17) for  $t = 1, \dots, m$ , with inequality  $t$  multiplied by  $(1 - \nu h)^{m-t}$ , we get the left-hand side

$$\begin{aligned}
LHS &= \mathbb{E} [\|y^{k,m} - w^*\|^2] + 2h(1 - 2Lh) \sum_{t=1}^m (1 - \nu h)^{m-t} \mathbb{E} [P(y^{k,t-1}) - P(w^*)] \\
&\stackrel{(2.8)}{=} \mathbb{E} [\|y^{k,m} - w^*\|^2] + 2\beta h(1 - 2Lh) \mathbb{E} [P(w^{k+1}) - P(w^*)],
\end{aligned}$$

and the right-hand side

$$\begin{aligned}
RHS &= (1 - \nu h)^m \mathbb{E} [\|w^k - w^*\|^2] + 4\beta(L-\mu)h^2 \mathbb{E} [P(w^k) - P(w^*)] \\
&\stackrel{(2.3)}{\leq} \frac{2(1 - \nu h)^m}{\mu} \mathbb{E} [P(w^k) - P(w^*)] + 4\beta(L-\mu)h^2 \mathbb{E} [P(w^k) - P(w^*)] \\
&= 2 \left( \frac{(1 - \nu h)^m}{\mu} + 2\beta(L-\mu)h^2 \right) \mathbb{E} [P(w^k) - P(w^*)].
\end{aligned}$$

Since  $LHS \leq RHS$ , we finally conclude with

$$\mathbb{E} [P(w^{k+1}) - P(w^*)] \leq c \mathbb{E} [P(w^k) - P(w^*)] - \frac{\mathbb{E} [\|y^{k,m} - w^*\|^2]}{2\beta h(1 - 2Lh)} \leq c \mathbb{E} [P(w^k) - P(w^*)].$$

□

Since we have established linear convergence of expected values, a high probability result can be obtained in a straightforward way using Markov inequality.

**Theorem 5.** Consider the setting of Theorem 4. Then, for any  $0 < \rho < 1$ ,  $0 < \varepsilon < 1$  and

$$k \geq \frac{\log \left( \frac{1}{\varepsilon \rho} \right)}{\log \left( \frac{1}{c} \right)}, \tag{2.18}$$

we have

$$\mathbb{P} \left[ \frac{P(w^k) - P(w^*)}{P(w^0) - P(w^*)} \leq \varepsilon \right] \geq 1 - \rho. \quad (2.19)$$

*Proof.* This follows directly from Markov inequality and Theorem 4:

$$\mathbb{P} [P(w^k) - P(w^*) > \varepsilon (P(w^0) - P(w^*))] \stackrel{(2.10)}{\leq} \frac{\mathbb{E} [P(w^k) - P(w^*)]}{\varepsilon (P(w^0) - P(w^*))} \leq \frac{c^k}{\varepsilon} \stackrel{(2.18)}{\leq} \rho$$

□

This result will be also useful when treating the non-strongly convex case.

## 2.5 Optimal Choice of Parameters

The goal of this section is to provide insight into the choice of parameters of S2GD; that is, the number of epochs (equivalently, full gradient evaluations)  $k$ , the maximal number of steps in each epoch  $m$ , and the stepsize  $h$ . The remaining parameters  $(L, \mu, n)$  are inherent in the problem and we will hence treat them in this section as given.

In particular, ideally we wish to find parameters  $k, m$  and  $h$  solving the following optimization problem:

$$\min_{k,m,h} \tilde{\mathcal{W}}(k, m, h) \stackrel{\text{def}}{=} k(n + 2\xi(m, h)), \quad (2.20)$$

subject to

$$\mathbb{E} [P(w^k) - P(w^*)] \leq \varepsilon (P(w^0) - P(w^*)). \quad (2.21)$$

Note that  $\tilde{\mathcal{W}}(k, m, h)$  is the *expected work*, measured by the number number of stochastic gradient evaluations, performed by S2GD when running for  $k$  epochs. Indeed, the evaluation of  $g^k$  is equivalent to  $n$  stochastic gradient evaluations, and each epoch further computes on average  $2\xi(m, h)$  stochastic gradients (see (2.5)). Since  $\frac{m+1}{2} \leq \xi(m, h) < m$ , we can simplify and solve the problem with  $\xi$  set to the conservative upper estimate  $\xi = m$ .

In view of (2.10), accuracy constraint (2.21) is satisfied if  $c$  (which depends on  $h$  and  $m$ ) and  $k$  satisfy

$$c^k \leq \varepsilon. \quad (2.22)$$

We therefore instead consider the parameter fine-tuning problem

$$\min_{k,m,h} \mathcal{W}(k, m, h) \stackrel{\text{def}}{=} k(n + 2m) \quad \text{subject to} \quad c \leq \varepsilon^{1/k}. \quad (2.23)$$

In the following we (approximately) solve this problem in two steps. First, we fix  $k$  and find (nearly) optimal  $h = h(k)$  and  $m = m(k)$ . The problem reduces to minimizing  $m$  subject to  $c \leq \varepsilon^{1/k}$  by fine-tuning  $h$ . While in the  $\nu = 0$  case it is possible to obtain closed form solution, this is not possible for  $\nu > 0$ .

However, it is still possible to obtain a good formula for  $h(k)$  leading to expression for good  $m(k)$  which depends on  $\varepsilon$  in the correct way. We then plug the formula for  $m(k)$  obtained this way back into (2.23), and study the quantity  $\mathcal{W}(k, m(k), h(k)) = k(n + 2m(k))$  as a function of  $k$ , over which we optimize at the end.

**Theorem 6** (Choice of parameters). *Fix the number of epochs  $k \geq 1$ , error tolerance  $0 < \varepsilon < 1$ , and let  $\Delta = \varepsilon^{1/k}$ . If we run S2GD with the stepsize*

$$h = h(k) \stackrel{\text{def}}{=} \frac{1}{\frac{4}{\Delta}(L - \mu) + 2L} \quad (2.24)$$

and

$$m \geq m(k) \stackrel{\text{def}}{=} \begin{cases} \left( \frac{4(\kappa-1)}{\Delta} + 2\kappa \right) \log \left( \frac{2}{\Delta} + \frac{2\kappa-1}{\kappa-1} \right), & \text{if } \nu = \mu, \\ \frac{8(\kappa-1)}{\Delta^2} + \frac{8\kappa}{\Delta} + \frac{2\kappa^2}{\kappa-1}, & \text{if } \nu = 0, \end{cases} \quad (2.25)$$

then  $\mathbb{E} [P(w^k) - P(w^*)] \leq \varepsilon(P(w^0) - P(w^*))$ .

In particular, if we choose  $k^* = \lceil \log(1/\varepsilon) \rceil$ , then  $\frac{1}{\Delta} \leq \exp(1)$ , and hence  $m(k^*) = O(\kappa)$ , leading to the workload

$$\mathcal{W}(k^*, m(k^*), h(k^*)) = \left\lceil \log\left(\frac{1}{\varepsilon}\right) \right\rceil (n + O(\kappa)) = O\left((n + \kappa) \log\left(\frac{1}{\varepsilon}\right)\right). \quad (2.26)$$

*Proof.* We only need to show that  $c \leq \Delta$ , where  $c$  is given by (2.12) for  $\nu = \mu$  and by (2.11) for  $\nu = 0$ . We denote the two summands in expressions for  $c$  as  $c_1$  and  $c_2$ . We choose the  $h$  and  $m$  so that both  $c_1$  and  $c_2$  are smaller than  $\Delta/2$ , resulting in  $c_1 + c_2 = c \leq \Delta$ .

The stepsize  $h$  is chosen so that

$$c_2 \stackrel{\text{def}}{=} \frac{2(L - \mu)h}{1 - 2Lh} = \frac{\Delta}{2},$$

and hence it only remains to verify that  $c_1 = c - c_2 \leq \frac{\Delta}{2}$ . In the  $\nu = 0$  case,  $m(k)$  is chosen so that  $c - c_2 = \frac{\Delta}{2}$ . In the  $\nu = \mu$  case,  $c - c_2 = \frac{\Delta}{2}$  holds for  $m = \log\left(\frac{2}{\Delta} + \frac{2\kappa-1}{\kappa-1}\right) / \log\left(\frac{1}{1-H}\right)$ , where  $H = \left(\frac{4(\kappa-1)}{\Delta} + 2\kappa\right)^{-1}$ . We only need to observe that  $c$  decreases as  $m$  increases, and apply the inequality  $\log\left(\frac{1}{1-H}\right) \geq H$ .  $\square$

We now comment on the above result:

1. **Workload.** Notice that for the choice of parameters  $k^*$ ,  $h = h(k^*)$ ,  $m = m(k^*)$  and any  $\nu \in [0, \mu]$ , the method needs  $\log(1/\varepsilon)$  computations of the full gradient (note this is independent of  $\kappa$ ), and  $O(\kappa \log(1/\varepsilon))$  computations of the stochastic gradient. This result, and special cases thereof, are summarized in Table 2.2.
2. **Simpler formulas for  $m$ .** If  $\kappa \geq 2$ , we can instead of (2.25) use the following (slightly worse but) simpler expressions for  $m(k)$ , obtained from (2.25) by using the bounds  $1 \leq \kappa - 1$ ,  $\kappa - 1 \leq \kappa$  and  $\Delta < 1$  in appropriate places (e.g.,  $\frac{8\kappa}{\Delta} < \frac{8\kappa}{\Delta^2}$ ,  $\frac{\kappa}{\kappa-1} \leq 2 < \frac{2}{\Delta^2}$ ):

$$m \geq \tilde{m}(k) \stackrel{\text{def}}{=} \begin{cases} \frac{6\kappa}{\Delta} \log\left(\frac{5}{\Delta}\right), & \text{if } \nu = \mu, \\ \frac{20\kappa}{\Delta^2}, & \text{if } \nu = 0. \end{cases} \quad (2.27)$$

3. **Optimal stepsize in the  $\nu = 0$  case.** Theorem 6 does not claim to have solved problem (2.23); the problem in general does not have a closed form solution. However, in the  $\nu = 0$  case a closed-form formula can easily be obtained:

$$h(k) = \frac{1}{\frac{4}{\Delta}(L - \mu) + 4L}, \quad m \geq m(k) \stackrel{\text{def}}{=} \frac{8(\kappa - 1)}{\Delta^2} + \frac{8\kappa}{\Delta}. \quad (2.28)$$

Indeed, for fixed  $k$ , (2.23) is equivalent to finding  $h$  that minimizes  $m$  subject to the constraint  $c \leq \Delta$ . In view of (2.11), this is equivalent to searching for  $h > 0$  maximizing the quadratic  $h \rightarrow h(\Delta - 2(\Delta L + L - \mu)h)$ , which leads to (2.28).

Note that both the stepsize  $h(k)$  and the resulting  $m(k)$  are slightly larger in Theorem 6 than in (2.28). This is because in the theorem the stepsize was for simplicity chosen to satisfy  $c_2 = \frac{\Delta}{2}$ , and hence is (slightly) suboptimal. Nevertheless, the dependence of  $m(k)$  on  $\Delta$  is of the correct (optimal) order in both cases. That is,  $m(k) = O\left(\frac{\kappa}{\Delta} \log\left(\frac{1}{\Delta}\right)\right)$  for  $\nu = \mu$  and  $m(k) = O\left(\frac{\kappa}{\Delta^2}\right)$  for  $\nu = 0$ .

4. **Stepsize choice.** In cases when one does not have a good estimate of the strong convexity constant  $\mu$  to determine the stepsize via (2.24), one may choose suboptimal stepsize that does not depend on  $\mu$  and derive similar results to those above. For instance, one may choose  $h = \frac{\Delta}{6L}$ .

In Table 2.3 we provide comparison of work needed for small values of  $k$ , and different values of  $\kappa$  and  $\varepsilon$ . Note, for instance, that for any problem with  $n = 10^9$  and  $\kappa = 10^3$ , S2GD outputs a highly accurate solution ( $\varepsilon = 10^{-6}$ ) in the amount of work equivalent to 2.12 evaluations of the full gradient of  $f$ !

$\varepsilon = 10^{-3}, \kappa = 10^3$			$\varepsilon = 10^{-6}, \kappa = 10^3$			$\varepsilon = 10^{-9}, \kappa = 10^3$		
$k$	$\mathcal{W}_\mu(k)$	$\mathcal{W}_0(k)$	$k$	$\mathcal{W}_\mu(k)$	$\mathcal{W}_0(k)$	$k$	$\mathcal{W}_\mu(k)$	$\mathcal{W}_0(k)$
1	<b>1.06n</b>	17.0n	1	116n	$10^7 n$	2	7.58n	$10^4 n$
2	2.00n	<b>2.03n</b>	2	<b>2.12n</b>	34.0n	3	<b>3.18n</b>	51.0n
3	3.00n	3.00n	3	3.01n	<b>3.48n</b>	4	4.03n	6.03n
4	4.00n	4.00n	4	4.00n	4.06n	5	5.01n	<b>5.32n</b>
5	5.00n	5.00n	5	5.00n	5.02n	6	6.00n	6.09n

$\varepsilon = 10^{-3}, \kappa = 10^6$			$\varepsilon = 10^{-6}, \kappa = 10^6$			$\varepsilon = 10^{-9}, \kappa = 10^6$		
$k$	$\mathcal{W}_\mu(k)$	$\mathcal{W}_0(k)$	$k$	$\mathcal{W}_\mu(k)$	$\mathcal{W}_0(k)$	$k$	$\mathcal{W}_\mu(k)$	$\mathcal{W}_0(k)$
2	4.14n	35.0n	4	8.29n	70.0n	5	17.3n	328n
3	<b>3.77n</b>	8.29n	5	<b>7.30n</b>	26.3n	8	<b>10.9n</b>	32.5n
4	4.50n	<b>6.39n</b>	6	7.55n	16.5n	10	11.9n	21.4n
5	5.41n	6.60n	8	9.01n	<b>12.7n</b>	13	14.3n	<b>19.1n</b>
6	6.37n	7.28n	10	10.8n	13.2n	20	21.0n	23.5n

$\varepsilon = 10^{-3}, \kappa = 10^9$			$\varepsilon = 10^{-6}, \kappa = 10^9$			$\varepsilon = 10^{-9}, \kappa = 10^9$		
$k$	$\mathcal{W}_\mu(k)$	$\mathcal{W}_0(k)$	$k$	$\mathcal{W}_\mu(k)$	$\mathcal{W}_0(k)$	$k$	$\mathcal{W}_\mu(k)$	$\mathcal{W}_0(k)$
6	378n	1293n	13	737n	2409n	15	1251n	4834n
8	<b>358n</b>	1063n	16	<b>717n</b>	2126n	24	<b>1076n</b>	3189n
11	376n	<b>1002n</b>	19	727n	2025n	30	1102n	3018n
15	426n	1058n	22	752n	<b>2005n</b>	32	1119n	<b>3008n</b>
20	501n	1190n	30	852n	2116n	40	1210n	3078n

Table 2.3: Comparison of work sufficient to solve a problem with  $n = 10^9$ , and various values of  $\kappa$  and  $\varepsilon$ . The work was computed using formula (2.23), with  $m(k)$  as in (2.27). The notation  $\mathcal{W}_\nu(k)$  indicates what parameter  $\nu$  was used.

## 2.6 Complexity Analysis: Convex Loss

If  $P$  is convex but not strongly convex, we define  $\hat{f}_i(w) \stackrel{\text{def}}{=} f_i(w) + \frac{\mu}{2} \|w - w^0\|^2$ , for small enough  $\mu > 0$  (we shall see below how the choice of  $\mu$  affects the results), and consider the perturbed problem

$$\min_{w \in \mathbb{R}^d} \hat{P}(w), \quad (2.29)$$

where

$$\hat{P}(w) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n \hat{f}_i(w) = P(w) + \frac{\mu}{2} \|w - w^0\|^2. \quad (2.30)$$

Note that  $\hat{P}$  is  $\mu$ -strongly convex and  $(L + \mu)$ -smooth. In particular, the theory developed in the previous section applies. We propose that S2GD be instead applied to the perturbed problem, and show that an approximate solution of (2.29) is also an approximate solution of (2.1) (we will assume that this problem has a minimizer).

Let  $\hat{w}^*$  be the (necessarily unique) solution of the perturbed problem (2.29). The following result describes an important connection between the original problem and the perturbed problem.

**Lemma 7.** If  $\hat{w} \in \mathbb{R}^d$  satisfies  $\hat{P}(\hat{w}) \leq \hat{P}(\hat{w}^*) + \delta$ , where  $\delta > 0$ , then

$$P(\hat{w}) \leq P(w^*) + \frac{\mu}{2} \|w^0 - w^*\|^2 + \delta.$$

*Proof.* The statement is almost identical to Lemma 9 in [148]; its proof follows the same steps with only minor adjustments.  $\square$

We are now ready to establish a complexity result for non-strongly convex losses.

**Theorem 8.** Let Assumption 1 be satisfied. Choose  $\mu > 0$ ,  $0 \leq \nu \leq \mu$ , stepsize  $0 < h < \frac{1}{2(L+\mu)}$ , and let  $m$  be sufficiently large so that

$$\hat{c} \stackrel{\text{def}}{=} \frac{(1-\nu h)^m}{\beta \mu h (1-2(L+\mu)h)} + \frac{2Lh}{1-2(L+\mu)h} < 1. \quad (2.31)$$

Pick  $w^0 \in \mathbb{R}^d$  and let  $\hat{w}^0 = w^0, \hat{w}^1, \dots, \hat{w}^k$  be the sequence of iterates produced by S2GD as applied to problem (2.29). Then, for any  $0 < \rho < 1$ ,  $0 < \varepsilon < 1$  and

$$k \geq \frac{\log(1/(\varepsilon\rho))}{\log(1/\hat{c})}, \quad (2.32)$$

we have

$$\mathbb{P} \left[ P(\hat{w}^k) - P(w^*) \leq \varepsilon(P(w^0) - P(w^*)) + \frac{\mu}{2} \|w^0 - w^*\|^2 \right] \geq 1 - \rho. \quad (2.33)$$

In particular, if we choose  $\mu = \epsilon < L$  and parameters  $k^*$ ,  $h(k^*)$ ,  $m(k^*)$  as in Theorem 6, the amount of work performed by S2GD to guarantee (2.33) is

$$\mathcal{W}(k^*, h(k^*), m(k^*)) = O \left( \left( n + \frac{L}{\varepsilon} \right) \log \left( \frac{1}{\varepsilon} \right) \right),$$

which consists of  $O(\frac{1}{\varepsilon})$  full gradient evaluations and  $O(\frac{L}{\varepsilon} \log(\frac{1}{\varepsilon}))$  stochastic gradient evaluations.

*Proof.* We first note that

$$\hat{P}(\hat{w}^0) - \hat{P}(\hat{w}^*) \stackrel{(2.30)}{=} P(\hat{w}^0) - \hat{P}(\hat{w}^*) \leq P(\hat{w}^0) - P(\hat{w}^*) \leq P(w^0) - P(w^*), \quad (2.34)$$

where the first inequality follows from  $f \leq \hat{f}$ , and the second one from optimality of  $x_*$ . Hence, by first applying Lemma 7 with  $\hat{w} = \hat{w}^k$  and  $\delta = \varepsilon(P(w^0) - P(w^*))$ , and then Theorem 5, with  $c \leftarrow \hat{c}$ ,  $P \leftarrow \hat{P}$ ,  $w^0 \leftarrow \hat{w}^0$ ,  $w^* \leftarrow \hat{w}^*$ , we obtain

$$\begin{aligned} \mathbb{P} \left[ P(\hat{w}^k) - P(w^*) \leq \delta + \frac{\mu}{2} \|w^0 - w^*\|^2 \right] &\stackrel{\text{(Lemma 7)}}{\geq} \mathbb{P} \left[ \hat{P}(\hat{w}^k) - \hat{P}(\hat{w}^*) \leq \delta \right] \\ &\stackrel{(2.34)}{\geq} \mathbb{P} \left[ \frac{\hat{P}(\hat{w}^k) - \hat{P}(\hat{w}^*)}{\hat{P}(\hat{w}^0) - \hat{P}(\hat{w}^*)} \leq \varepsilon \right] \stackrel{(2.19)}{\geq} 1 - \rho. \end{aligned}$$

The second statement follows directly from the second part of Theorem 6 and the fact that the condition number of the perturbed problem is  $\kappa = \frac{L+\epsilon}{\epsilon} \leq \frac{2L}{\epsilon}$ .  $\square$

## 2.7 Implementation for sparse data

In our sparse implementation of Algorithm 1, described in this section and formally stated as Algorithm 3, we make the following structural assumption:

**Assumption 9.** The loss functions arise as the composition of a univariate smooth loss function  $\ell_i$ , and an inner product with a data point/example  $a_i \in \mathbb{R}^d$ :

$$f_i(w) = \ell_i(a_i^T w), \quad i = 1, 2, \dots, n.$$

In this case,  $\nabla f_i(w) = \nabla \ell_i(a_i^T w) a_i$ .

This is the structure in many cases of interest, including linear or logistic regression.

A natural question one might want to ask is whether S2GD can be implemented efficiently for sparse data.

Let us first take a brief detour and look at SGD, which performs iterations of the type:

$$w^{k+1} \leftarrow w^k - h \nabla \ell_i(a_i^T w) a_i. \quad (2.35)$$

Let  $\omega_i$  be the number of nonzero features in example  $a_i$ , i.e.,  $\omega_i \stackrel{\text{def}}{=} \|a_i\|_0 \leq d$ . Assuming that the computation of the derivative of the univariate function  $\ell_i$  takes  $O(1)$  amount of work, the computation of  $\nabla f_i(w)$  will take  $O(\omega_i)$  work. Hence, the update step (2.35) will cost  $O(\omega_i)$ , too, which means the method can naturally speed up its iterations on sparse data.

The situation is not as simple with S2GD, which for loss functions of the type described in Assumption 9 performs inner iterations as follows:

$$y^{k,t+1} \leftarrow y^{k,t} - h (g^k + \nabla \ell_i(a_i^T y^{k,t}) a_i - \nabla \ell_i(a_i^T w^k) a_i). \quad (2.36)$$

Indeed, note that  $g^k = \nabla P(w^k)$  is in general be fully dense even for sparse data  $\{a_i\}$ . As a consequence, the update in (2.36) might be as costly as  $d$  operations, irrespective of the sparsity level  $\omega_i$  of the active example  $a_i$ . However, we can use the following “lazy/delayed” update trick. We split the update to the  $y$  vector into two parts: immediate, and delayed. Assume index  $i = i_t$  was chosen at inner iteration  $t$ . We immediately perform the update

$$\tilde{y}^{k,t+1} \leftarrow y^{k,t} - h (\nabla \ell_{i_t}(a_{i_t}^T y^{k,t}) - \nabla \ell_{i_t}(a_{i_t}^T w^k)) a_{i_t},$$

which costs  $O(a_{i_t})$ . Note that we have not computed the  $y^{k,t+1}$ . However, we “know” that

$$y^{k,t+1} = \tilde{y}^{k,t+1} - hg^k,$$

without having to actually compute the difference. At the next iteration, we are supposed to perform update (2.36) for  $i = i_{t+1}$ :

$$y^{k,t+2} \leftarrow y^{k,t+1} - hg^k - h (\nabla \ell_{i_{t+1}}(a_{i_{t+1}}^T y^{k,t+1}) - \nabla \ell_{i_{t+1}}(a_{i_{t+1}}^T w^k)) a_{i_{t+1}}. \quad (2.37)$$

---

**Algorithm 3** Semi-Stochastic Gradient Descent (S2GD) for sparse data; “lazy” updates

---

**parameters:**  $m = \max$  # of stochastic steps per epoch,  $h = \text{stepsize}$ ,  $\nu = \text{lower bound on } \mu$

**for**  $k = 0, 1, 2, \dots$  **do**

- $g^k \leftarrow \frac{1}{n} \sum_{i=1}^n \nabla f_i(w^k)$
- $y^{k,0} \leftarrow w^k$
- $\chi_{(s)} \leftarrow 0$  for  $s = 1, 2, \dots, d$  ▷ Store when a coordinate was updated last time
- Let  $t^k \leftarrow t$  with probability  $(1 - \nu h)^{m-t}/\beta$  for  $t = 1, 2, \dots, m$
- for**  $t = 0$  to  $t^k - 1$  **do**

  - Pick  $i \in \{1, 2, \dots, n\}$ , uniformly at random
  - for**  $s \in \text{nnz}(a_i)$  **do**

    - $y_{(s)}^{k,t} \leftarrow y_{(s)}^{k,t} - (t - \chi_{(s)}) hg_{(s)}^k$  ▷ Update what will be needed
    - $\chi_{(s)} = t$

  - end for**
  - $y^{k,t+1} \leftarrow y^{k,t} - h (\nabla \ell_i(a_i^T y^{k,t}) - \nabla \ell_i(a_i^T w^k)) a_i$  ▷ A sparse update

- end for**
- for**  $s = 1$  to  $d$  **do** ▷ Finish all the “lazy” updates

  - $y_{(s)}^{k,t^k} \leftarrow y_{(s)}^{k,t^k} - (t^k - \chi_{(s)}) hg_{(s)}^k$

- end for**
- $w^{k+1} \leftarrow y^{k,t^k}$
- end for**

---

However, notice that we can't compute

$$\nabla \ell_{i_{t+1}}(a_{i_{t+1}}^T y^{k,t+1}) \quad (2.38)$$

as we never computed  $y^{k,t+1}$ . However, here lies the trick: as  $a_{i_{t+1}}$  is sparse, we only need to know those coordinates  $s$  of  $y^{k,t+1}$  for which  $(a_{i_{t+1}})_{(s)}$  is nonzero. So, just before we compute the (sparse part of) the update (2.37), we perform the update

$$y_{(s)}^{k,t+1} \leftarrow \tilde{y}_{(s)}^{k,t+1} - hg_{(s)}^k$$

for coordinates  $s$  for which  $(a_{i_{t+1}})_{(s)}$  is nonzero. This way we know that the inner product appearing in (2.38) is computed correctly (despite the fact that  $y^{k,t+1}$  potentially is not!). In turn, this means that we can compute the sparse part of the update in (2.37).

We now continue as before, again only computing  $\tilde{y}^{k,t+3}$ . However, this time we have to be more careful as it is no longer true that

$$y^{k,t+2} = \tilde{y}^{k,t+2} - hg^k.$$

We need to remember, for each coordinate  $s$ , the last iteration counter  $t$  for which  $(a_{i_t})_{(s)} \neq 0$ . This way we will know how many times did we “forget” to apply the dense update  $-hg_{(s)}^k$ . We do it in a just-in-time fashion, just before it is needed.

Algorithm 3 (sparse S2GD) performs these lazy updates as described above. It produces exactly the same result as Algorithm 1 (S2GD), but is much more efficient for sparse data as iteration picking example  $i$  only costs  $O(\omega_i)$ . This is done with a memory overhead of only  $O(d)$  (as represented by vector  $\chi \in \mathbb{R}^d$ ).

## 2.8 Numerical Experiments

In this section we conduct computational experiments to illustrate some aspects of the performance of our algorithm. In Section 2.8.1 we consider the least squares problem with synthetic data to compare the practical performance and the theoretical bound on convergence in expectations. We demonstrate that for both SVRG and S2GD, the practical rate is substantially better than the theoretical one. In Section 2.8.2 we compare the S2GD algorithm on several real datasets with other algorithms suitable for this task. We also provide efficient implementation of the algorithm, as described in Section 2.7, for the case of logistic regression in the MLOSS repository<sup>6</sup>.

### 2.8.1 Comparison with theory

Figure 2.1 presents a comparison of the theoretical rate and practical performance on a larger problem with artificial data, with a condition number we can control (and choose it to be poor). In particular, we consider the L2-regularized least squares with

$$f_i(w) = \frac{1}{2}(a_i^T w - b_i)^2 + \frac{\lambda}{2}\|w\|^2,$$

for some  $a_i \in \mathbb{R}^d$ ,  $b_i \in \mathbb{R}$  and  $\lambda > 0$  is the regularization parameter.

We consider an instance with  $n = 100,000$ ,  $d = 1,000$  and  $\kappa = 10,000$ . We run the algorithm with both parameters  $\nu = \lambda$  (our best estimate of  $\mu$ ) and  $\nu = 0$ . Recall that the latter choice leads to the SVRG method of [80]. We chose parameters  $m$  and  $h$  as a (numerical) solution of the work-minimization problem (2.20), obtaining  $m = 261,063$  and  $h = 1/11.4L$  for  $\nu = \lambda$  and  $m = 426,660$  and  $h = 1/12.7L$  for  $\nu = 0$ . The practical performance is obtained after a single run of the S2GD algorithm.

The figure demonstrates linear convergence of S2GD in practice, with the convergence rate being significantly better than the already strong theoretical result. Recall that the bound is on the expected function values. We can observe a rather strong convergence to machine precision

---

<sup>6</sup><http://mloss.org/software/view/556/>

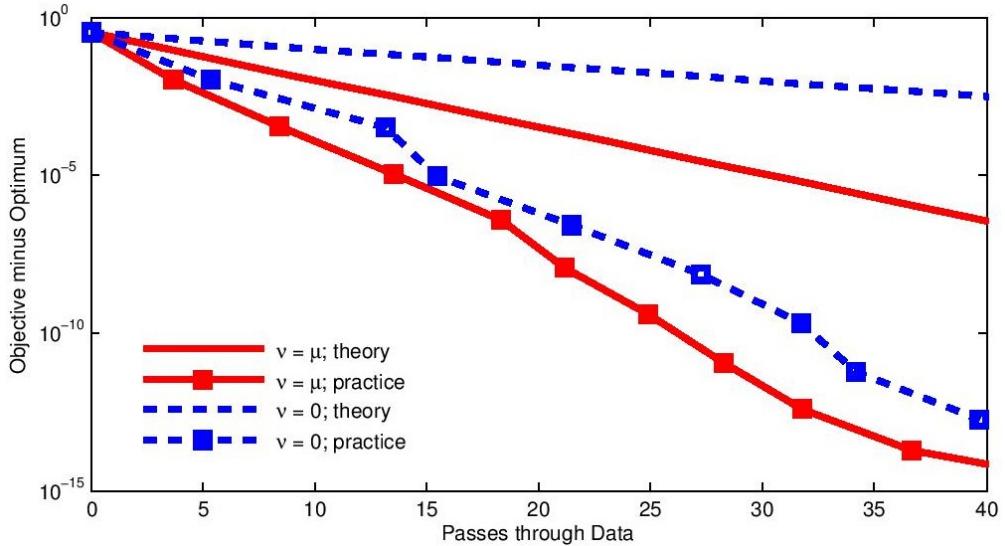


Figure 2.1: Least squares with  $n = 10^5$ ,  $\kappa = 10^4$ . Comparison of theoretical result and practical performance for cases  $\nu = \mu$  (full red line) and  $\nu = 0$  (dashed blue line).

in work equivalent to evaluating the full gradient only 40 times. Needless to say, neither SGD nor GD have such speed. Our method is also an improvement over [80], both in theory and practice.

### 2.8.2 Comparison with other methods

The S2GD algorithm can be applied to several classes of problems. We perform experiments on an important and in many applications used L2-regularized logistic regression for binary classification on several datasets. The functions  $f_i$  in this case are:

$$f_i(w) = \log(1 + \exp(b_i a_i^T w)) + \frac{\lambda}{2} \|w\|^2,$$

where  $b_i \in \{-1, +1\}$  is the label of  $i^{th}$  training example  $a_i$ . In our experiments we set the regularization parameter  $\lambda = \Theta(1/n)$  so that the condition number  $\kappa = \Theta(n)$ , which is about the most ill-conditioned problem used in practice. We added a (regularized) bias term to all datasets.

All the datasets we used, listed in Table 2.4, are freely available<sup>7</sup> benchmark binary classification datasets.

Dataset	Training examples ( $n$ )	Variables ( $d$ )	$L$	$\lambda$	$\kappa$
<i>ijcnn</i>	49 990	23	1.23	$1/n$	61 696
<i>rcv1</i>	20 242	47 237	0.50	$1/n$	10 122
<i>real-sim</i>	72 309	20 959	0.50	$1/n$	36 155
<i>url</i>	2 396 130	3 231 962	128.70	$100/n$	3 084 052

Table 2.4: Datasets used in the experiments.

In the experiment, we compared the following algorithms:

- **SGD:** Stochastic Gradient Descent. After various experiments, we decided to use a variant with constant step-size that gave the best practical performance in hindsight.

<sup>7</sup>Available at <http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>.

- **L-BFGS:** A publicly-available limited-memory quasi-Newton method that is suitable for broader classes of problems. We used a popular implementation by Mark Schmidt.<sup>8</sup>
- **SAG:** Stochastic Average Gradient, [158]. This is the most important method to compare to, as it also achieves linear convergence using only stochastic gradient evaluations. Although the methods has been analyzed for stepsize  $h = 1/16L$ , we experimented with various stepsizes and chose the one that gave the best performance for each problem individually.
- **SDCA:** Stochastic Dual Coordinate Ascent, where we used approximate solution to the one-dimensional dual step, as in Section 6.2 of [163].
- **S2GDcon:** The S2GD algorithm with conservative stepsize choice, i.e., following the theory. We set  $m = \Theta(\kappa)$  and  $h = 1/10L$ , which is approximately the value you would get from Equation (2.24).
- **S2GD:** The S2GD algorithm, with stepsize that gave the best performance in hindsight. The best value of  $m$  was between  $n$  and  $2n$  in all cases, but optimal  $h$  varied from  $1/2L$  to  $1/10L$ .

Note that SAG needs to store  $n$  gradients in memory in order to run. In case of relatively simple functions, one can store only  $n$  scalars, as the gradient of  $f_i$  is always a multiple of  $a_i$ . If we are comparing with SAG, we are implicitly assuming that our memory limitations allow us to do so. Although not included in Algorithm 1, we could also store these gradients we used to compute the full gradient, which would mean we would only have to compute a single stochastic gradient per inner iteration (instead of two).

We plot the results of these methods, as applied to various different, in the Figure 2.2 for first 15-30 passes through the data (i.e., amount of work work equivalent to 15-30 full gradient evaluations).

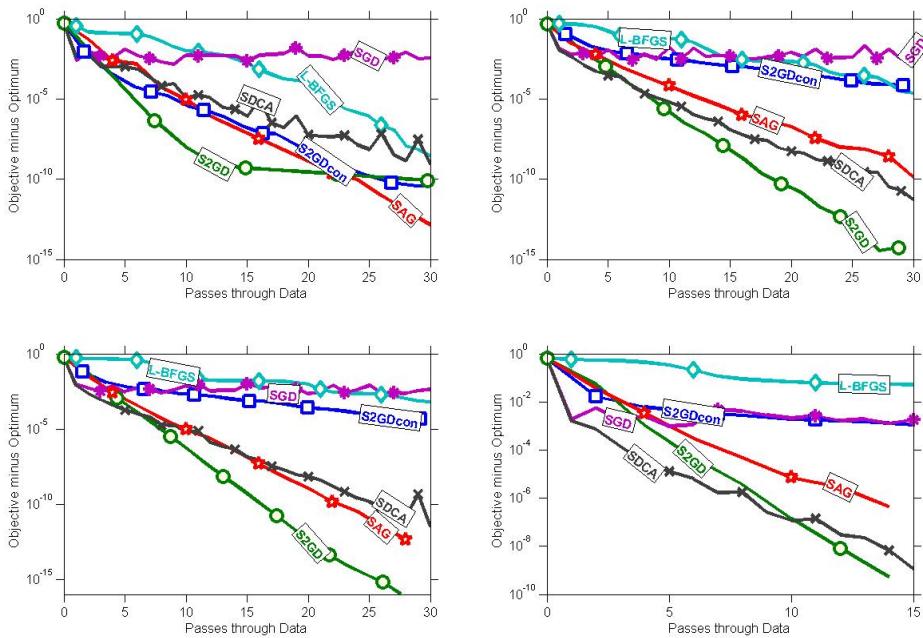


Figure 2.2: Practical performance for logistic regression and the following datasets: *ijcnn*, *rcv* (first row), *realsim*, *url* (second row)

<sup>8</sup><http://www.di.ens.fr/~mschmidt/Software/minFunc.html>

Algorithm	Time in seconds			
	<i>ijcnn</i>	<i>rcv1</i>	<i>real-sim</i>	<i>url</i>
S2GDcon	0.25	0.43	1.01	125.53
S2GD	0.29	0.49	1.02	54.04
SAG	0.41	0.73	1.87	71.74
L-BFGS	0.15	0.67	0.76	309.14
SGD	0.39	0.57	1.54	62.73
SDCA	0.33	0.38	1.10	126.32

Table 2.5: Time required to produce plots in Figure 2.2.

There are several remarks we would like to make. First, our experiments confirm the insight from [158] that for this types of problems, reduced-variance methods consistently exhibit substantially better performance than the popular L-BFGS algorithm.

The performance gap between S2GDcon and S2GD differs from dataset to dataset. A possible explanation for this can be found in an extension of SVRG to proximal setting [187], released after the first version of this work was put onto arXiv (i.e., after December 2013) . Instead Assumption 1, where all loss functions are assumed to be associated with the same constant  $L$ , the authors of [187] instead assume that each loss function  $f_i$  has its own constant  $L_i$ . Subsequently, they sample proportionally to these quantities as opposed to the uniform sampling. In our case,  $L = \max_i L_i$ . This weighted sampling has an impact on the convergence: one gets dependence on the average of the quantities  $L_i$  and not in their maximum.

The number of passes through data seems a reasonable way to compare performance, but some algorithms could need more time to do the same amount of passes through data than others. In this sense, S2GD should be in fact faster than SAG due to the following property. While SAG updates the test point after each evaluation of a stochastic gradient, S2GD does not always make the update — during the evaluation of the full gradient. This claim is supported by computational evidence: SAG needed about 20 – 40% more time than S2GD to do the same amount of passes through data.

Finally, in Table 2.5 we provide the time it took the algorithm to produce these plots on a desktop computer with Intel Core i7 3610QM processor, with  $2 \times 4\text{GB}$  DDR3 1600 MHz memory. The numbers for the *url* dataset is are not representative, as the algorithm needed extra memory, which slightly exceeded the memory limit of our computer.

### 2.8.3 Boosted variants of S2GD and SAG

In this section we study the practical performance of boosted methods, namely S2GD+ (Algorithm 2) and variant of SAG suggested by its authors [158, Section 4.2].

SAG+ is a simple modification of SAG, where one does not divide the sum of the stochastic gradients by  $n$ , but by the number of training examples seen during the run of the algorithm, which has the effect of producing larger steps at the beginning. The authors claim that this method performed better in practice than a hybrid SG/SAG algorithm.

We have observed that, in practice, starting SAG from a point close to the optimum, leads to an initial “away jump”. Eventually, the method exhibits linear convergence. In contrast, S2GD converges linearly from the start, regardless of the starting position.

Figure 2.3 shows that S2GD+ consistently improves over S2GD, while SAG+ does not improve always: sometimes it performs essentially the same as SAG. Although S2GD+ is overall a superior algorithm, one should note that this comes at the cost of having to choose stepsize parameter for SGD initialization. If one chooses these parameters poorly, then S2GD+ could perform worse than S2GD. The other three algorithms can work well without any parameter tuning.

## 2.9 Conclusion

We have developed a new semi-stochastic gradient descent method (S2GD) and analyzed its complexity for smooth convex and strongly convex loss functions. Our methods need

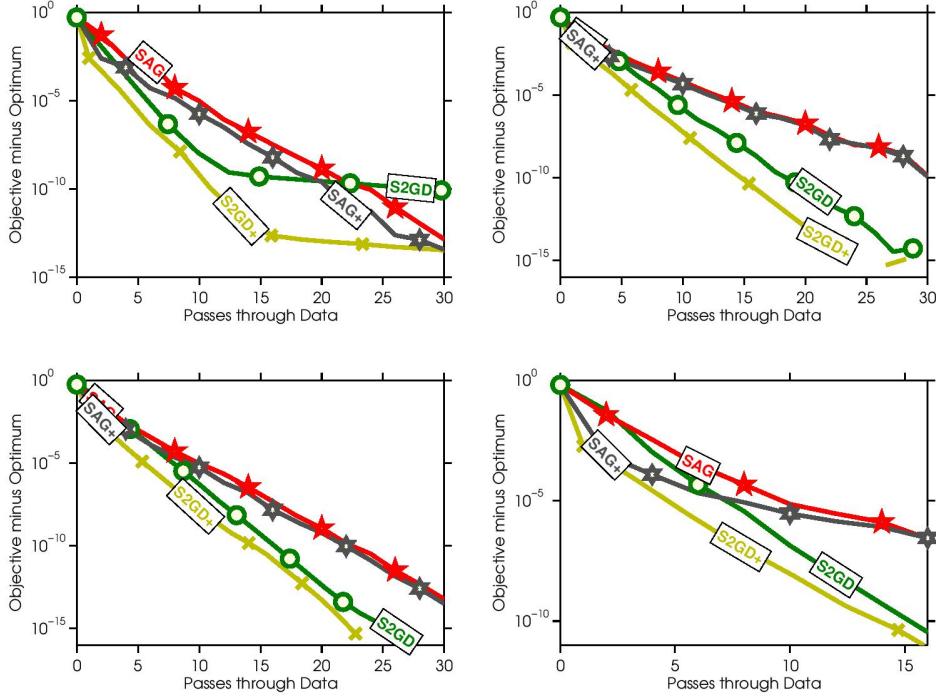


Figure 2.3: Practical performance of boosted methods on datasets *ijcnn*, *rcv* (first row), *realsim*, *url* (second row)

$O((\kappa/n) \log(1/\varepsilon))$  work only, measured in units equivalent to the evaluation of the full gradient of the loss function, where  $\kappa = L/\mu$  if the loss is  $L$ -smooth and  $\mu$ -strongly convex, and  $\kappa \leq 2L/\varepsilon$  if the loss is merely  $L$ -smooth.

Our results in the strongly convex case match or improve on a few very recent results, while at the same time generalizing and simplifying the analysis. Additionally, we proposed S2GD+ —a method which equips S2GD with an SGD pre-processing step—which in our experiments exhibits superior performance to all methods we tested. We left the analysis of this method as an open problem.

# Chapter 3

## Semi-Stochastic Coordinate Descent

### 3.1 Introduction

In this chapter we study the problem of unconstrained minimization of a strongly convex function represented as the average of a large number of smooth convex functions:

$$\min_{w \in \mathbb{R}^d} P(w) \equiv \frac{1}{n} \sum_{i=1}^n f_i(w). \quad (3.1)$$

Many computational problems in various disciplines are of this form. In machine learning,  $f_i(w)$  represents the loss/risk of classifier  $w \in \mathbb{R}^d$  on data sample  $i$ ,  $P$  represents the empirical risk (=average loss), and the goal is to find a predictor minimizing  $P$ . An L2-regularizer of the form  $\frac{\mu}{2}\|w\|^2$ , for  $\mu > 0$ , could be added to the loss, making it strongly convex and hence easier to minimize.

**Assumptions.** We assume that the functions  $f_i : \mathbb{R}^d \rightarrow \mathbb{R}$  are differentiable and convex function, with Lipschitz continuous partial derivatives. Formally, we assume that for each  $i \in [n] \stackrel{\text{def}}{=} \{1, 2, \dots, n\}$  and  $j \in [d] \stackrel{\text{def}}{=} \{1, 2, \dots, d\}$  there exists  $L_{ij} \geq 0$  such that for all  $x \in \mathbb{R}^d$  and  $h \in \mathbb{R}$ ,

$$f_i(x + he_j) \leq f_i(x) + \langle \nabla f_i(x), he_j \rangle + \frac{L_{ij}}{2} h^2, \quad (3.2)$$

where  $e_j$  is the  $j^{th}$  standard basis vector in  $\mathbb{R}^d$ ,  $\nabla f_i(x) \in \mathbb{R}^d$  is the gradient of  $f_i$  at point  $x$  and  $\langle \cdot, \cdot \rangle$  is the standard inner product. This assumption was recently used in the analysis of the accelerated coordinate descent method APPROX [59]. We further assume that  $P$  is  $\mu$ -strongly convex. That is, we assume that there exists  $\mu > 0$  such that for all  $x, y \in \mathbb{R}^d$ ,

$$P(y) \geq P(x) + \langle \nabla P(x), y - x \rangle + \frac{\mu}{2} \|y - x\|^2. \quad (3.3)$$

**Context.** Batch methods such as gradient descent (GD) enjoy a fast (linear) convergence rate: to achieve  $\epsilon$ -accuracy, GD needs  $\mathcal{O}(\kappa \log(1/\epsilon))$  iterations, where  $\kappa$  is a condition number. The drawback of GD is that in each iteration one needs to compute the gradient of  $P$ , which requires a pass through the entire dataset. This is prohibitive to do many times if  $n$  is very large.

Stochastic gradient descent (SGD) in each iteration computes the gradient of a single randomly chosen function  $f_i$  only—this constitutes an unbiased (but noisy) estimate of the gradient of  $P$ —and makes a step in that direction [153, 125, 196, 162]. The rate of convergence of SGD is slower,  $\mathcal{O}(1/\epsilon)$ , but the cost of each iteration is independent of  $n$ . Variants with nonuniform selection probabilities were considered in [201], a mini-batch variant (for SVMs with hinge loss) was analyzed in [172].

Recently, there has been progress in designing algorithms that achieve the fast  $O(\log(1/\epsilon))$  rate without the need to scan the entire dataset in each iteration. The first class of methods to have achieved this are stochastic/randomized coordinate descent methods.

When applied to (3.1), coordinate descent methods (CD) [129, 148] can, like SGD, be seen as an attempt to keep the benefits of GD (fast linear convergence) while reducing the complexity of each iteration. A CD method only computes a single partial derivative  $\nabla_j P(w)$  at each iteration and updates a single coordinate of vector  $w$  only. When chosen uniformly at random, partial derivative is also an unbiased estimate of the gradient. However, unlike the SGD estimate, its variance decrease to zero as one approaches the optimum. While CD methods are able to obtain linear convergence, they typically need  $O((d/\mu) \log(1/\epsilon))$  iterations when applied to (3.1) directly<sup>1</sup>. CD method typically significantly outperform GD, especially on sparse problems with a very large number of variables/coordinates [129, 148].

An alternative to applying CD to (3.1) is to apply it to the dual problem. This is possible under certain additional structural assumptions on the functions  $f_i$ . This is the strategy employed by stochastic dual coordinate ascent (SDCA) [163, 143], whose rate is

$$O((n + \kappa) \log(1/\epsilon)).$$

The condition number  $\kappa$  here is the same as the condition number appearing in the rate of GD. Despite this, this is a vast improvement on the computational complexity achieved by GD which has an iteration cost  $n$  times larger than SDCA. Also, the linear convergence rate is superior to the sublinear rate  $O(1/\epsilon)$  achieved by SGD, and the method indeed typically performs much better in practice. Accelerated [164] and mini-batch [172] variants of SDCA have also been proposed. We refer the reader to QUARTZ [143] for a general analysis involving the update of a random subset of dual coordinates, following an arbitrary distribution.

Recently, there has been progress in designing primal methods which match the fast rate of SDCA. Stochastic average gradient (SAG) [158], and more recently SAGA [45], move in a direction composed of old stochastic gradients. The semi-stochastic gradient descent (S2GD) [89, 87] and stochastic variance reduced gradient (SVRG) [80, 187] methods employ a different strategy: one first computes the gradient of  $P$ , followed by  $O(\kappa)$  steps where only stochastic gradients are computed. These are used to estimate the change of the gradient, and it is this direction which combines the old gradient and the new stochastic gradient information which is used in the update.

**Main result.** In this work we develop a new method—semi-stochastic coordinate descent (S2CD)—for solving (3.1), enjoying a fast rate similar to methods such as SDCA, SAG, S2GD, SVRG, SAGA, mS2GD and QUARTZ. S2CD can be seen as a hybrid between S2GD and CD. In particular, the complexity of our method is the sum of two terms:

$$O(n \log(1/\epsilon))$$

evaluations  $\nabla f_i$  (that is,  $\log(1/\epsilon)$  evaluations of the gradient of  $P$ ) and

$$O(\hat{\kappa} \log(1/\epsilon))$$

evaluations of  $\langle e_j, \nabla f_i \rangle$  for randomly chosen functions  $f_i$  and randomly chosen coordinates  $j$ , where  $\hat{\kappa}$  is a new condition number which is defined in (3.13) and larger than  $\kappa$ . We summarize in Table 3.1 the runtime complexity of the various algorithms. Note that  $\hat{\kappa}$  enters the complexity only in the term involving the evaluation cost of a partial derivative  $\nabla_j f_i$ , which can be substantially smaller than the evaluation cost of  $\nabla f_i$ . Hence, our complexity result can be both better or worse than previous results, depending on whether the increase of the condition number can or can not be compensated by the lower cost of the stochastic steps based on the evaluation of partial derivatives.

---

<sup>1</sup>The complexity can be improved to  $O(\frac{d\alpha}{\tau\mu} \log(1/\epsilon))$  in the case when  $\tau$  coordinates are updated in each iteration, where  $\alpha \in [1, \tau]$  is a problem-dependent constant [151]. This has been further studied for nonsmooth problems via smoothing [58], for arbitrary nonuniform distributions governing the selection of coordinates [150, 143] and in the distributed setting [149, 57, 143]. Also, efficient accelerated variants with  $O(1/\sqrt{\epsilon})$  rate were

Method	Runtime	paper
CD	$\mathcal{O}(n\kappa\mathcal{C}_{grad} \log(1/\epsilon))$	e.g. [127]
SGD	$\mathcal{O}(\mathcal{C}_{grad}/\epsilon)$	[196, 162]
CD	$\mathcal{O}(n\kappa\mathcal{C}_{pd} \log(1/\epsilon))$	[129, 148]
SDCA	$\mathcal{O}((n + \kappa)\mathcal{C}_{grad} \log(1/\epsilon))$	[163, 201, 143]
SVRG/S2GD	$\mathcal{O}((n\mathcal{C}_{grad} + \kappa\mathcal{C}_{grad}) \log(1/\epsilon))$	[80, 187, 89]
S2CD	$\mathcal{O}((n\mathcal{C}_{grad} + \hat{\kappa}\mathcal{C}_{pd}) \log(1/\epsilon))$	this work [84]

Table 3.1: Runtime complexity of various algorithms. We use  $\mathcal{C}_{grad}$  to denote the evaluation cost of the gradient of one single function  $\nabla f_i$  and use  $\mathcal{C}_{pd}$  to denote the evaluation cost of a partial derivative  $\nabla_j f_i$ .

**Outline.** This chapter is organized as follows. In Section 3.2 we describe the S2CD algorithm and in Section 3.3 we state a key lemma and our main complexity result. The proof of the lemma is provided in Section 3.4 and the proof of the main result in Section 3.5.

## 3.2 S2CD Algorithm

In this section we describe the Semi-Stochastic Coordinate Descent method (Algorithm 4).

---

**Algorithm 4** Semi-Stochastic Coordinate Descent (S2CD)

---

```

parameters:  $m$  (max # of stochastic steps per epoch);  $h > 0$  (stepsize parameter);  $w^0 \in \mathbb{R}^d$ 
for  $k = 0, 1, 2, \dots$  do
    Compute and store  $\nabla P(w^k) = \frac{1}{n} \sum_i \nabla f_i(w^k)$ 
    Initialize the inner loop:  $y^{k,0} \leftarrow w^k$ 
    Let  $t^k = T \in \{1, 2, \dots, m\}$  with probability  $(1 - \mu h)^{m-T} / \beta$ 
    for  $t = 0$  to  $t^k - 1$  do
        Pick coordinate  $j \in \{1, 2, \dots, d\}$  with probability  $p_j$ 
        Pick function index  $i$  from the set  $\{i : L_{ij} > 0\}$  with probability  $q_{ij}$ 
         $y^{k,t+1} \leftarrow y^{k,t} - hp_j^{-1} (\nabla_j P(w^k) + \frac{1}{nq_{ij}} (\nabla_j f_i(y^{k,t}) - \nabla_j f_i(w^k)) e_j)$ 
    end for
    Reset the starting point:  $w^{k+1} \leftarrow y^{k,t^k}$ 
end for

```

---

The discussion on the choice of  $m$  and  $h$  in Algorithm 4 is deferred to Section 3.3. As we will see, the parameters  $m$  and  $h$  depends on the target accuracy and the number of iterations. We next provide a more detailed description of the algorithm.

The method has an outer loop (an ‘‘epoch’’), indexed by counter  $k$ , and an inner loop, indexed by  $t$ . At the beginning of epoch  $k$ , we compute and store the gradient of  $f$  at  $w^k$ . Subsequently, S2CD enters the inner loop in which a sequence of vectors  $y^{k,t}$  for  $t = 0, 1, \dots, t^k$  is computed in a stochastic way, starting from  $y^{k,0} = w^k$ . The number  $t^k$  of stochastic steps in the inner loop is random, following a geometric law:

$$\mathbb{P}[(t^k = T)] = \frac{(1 - \mu h)^{m-T}}{\beta}, \quad T \in \{1, \dots, m\},$$

where

$$\beta \stackrel{\text{def}}{=} \sum_{t=1}^m (1 - \mu h)^{m-t}. \quad (3.4)$$

---

developed [59, 57], capable of solving problems with 50 billion variables.

In each step of the inner loop, we seek to compute  $y^{k,t+1}$ , given  $y^{k,t}$ . In order to do so, we sample coordinate  $j$  with probability  $p_j$  and subsequently<sup>2</sup> sample  $i$  with probability  $q_{ij}$ , where the probabilities are given by

$$\omega_i \stackrel{\text{def}}{=} |\{j : L_{ij} \neq 0\}|, \quad v_j \stackrel{\text{def}}{=} \sum_{i=1}^n \omega_i L_{ij}, \quad p_j \stackrel{\text{def}}{=} v_j / \sum_{j=1}^d v_j, \quad q_{ij} \stackrel{\text{def}}{=} \frac{\omega_i L_{ij}}{v_j}, \quad p_{ij} \stackrel{\text{def}}{=} p_j q_{ij}. \quad (3.5)$$

Note that  $L_{ij} = 0$  means that function  $f_i$  does not depend on the  $j^{th}$  coordinate of  $x$ . Hence,  $\omega_i$  is the number of coordinates function  $f_i$  depends on – a measure of sparsity of the data<sup>3</sup>. It can be shown that  $P$  has a 1-Lipschitz gradient with respect to the weighted Euclidean norm with weights  $\{v_j\}$  ([59, Theorem 1]). Hence, we sample coordinate  $j$  proportionally to this weight  $v_j$ . Note that  $p_{ij}$  is the joint probability of choosing the pair  $(i, j)$ .

Having sampled coordinate  $j$  and function index  $i$ , we compute two partial derivatives:  $\nabla_j f_i(w^k)$  and  $\nabla_j f_i(y^{k,t})$  (we compressed the notation here by writing  $\nabla_j f_i(w)$  instead of  $\langle \nabla f_i(w), e_j \rangle$ ), and combine these with the pre-computed value  $\nabla_j P(w^k)$  to form an update of the form

$$y^{k,t+1} \leftarrow y^{k,t} - h p_j^{-1} G_{ij}^{kt} e_j = y^{k,t} - h g_{ij}^{kt}, \quad (3.6)$$

where

$$g_{ij}^{kt} \stackrel{\text{def}}{=} p_j^{-1} G_{ij}^{kt} e_j \quad (3.7)$$

and

$$G_{ij}^{kt} \stackrel{\text{def}}{=} \nabla_j P(w^k) + \frac{1}{n q_{ij}} (\nabla_j f_i(y^{k,t}) - \nabla_j f_i(w^k)). \quad (3.8)$$

Note that only a single coordinate of  $y^{k,t}$  is updated at each iteration.

In the entire text (with the exception of the statement of Theorem 11 and a part of Section 3.5.3, where  $\mathbb{E}[\cdot]$  denotes the total expectation) we will assume that all expectations are conditional on the entire history of the random variables generated up to the point when  $y^{k,t}$  was computed. With this convention, it is possible to think that there are only two random variables:  $j$  and  $i$ . By  $\mathbb{E}[\cdot]$  we then mean the expectation with respect to both of these random variables, and by  $\mathbb{E}_i[\cdot]$  we mean expectation with respect to  $i$  (that is, conditional on  $j$ ). With this convention, we can write

$$\begin{aligned} \mathbb{E}_i[G_{ij}^{kt}] &= \sum_{i=1}^n q_{ij} G_{ij}^{kt} \\ &\stackrel{(3.8)}{=} \nabla_j P(w^k) + \frac{1}{n} \sum_{i=1}^n (\nabla_j f_i(y^{k,t}) - \nabla_j f_i(w^k)) \stackrel{(3.1)}{=} \nabla_j P(y^{k,t}), \end{aligned} \quad (3.9)$$

which means that conditioned on  $j$ ,  $G_{ij}^{kt}$  is an unbiased estimate of the  $j^{th}$  partial derivative of  $P$  at  $y^{k,t}$ . An equally easy calculation reveals that the random vector  $g_{ij}^{kt}$  is an unbiased estimate of the gradient of  $P$  at  $y^{k,t}$ :

$$\begin{aligned} \mathbb{E}[g_{ij}^{kt}] &\stackrel{(3.7)}{=} \mathbb{E}[p_j^{-1} G_{ij}^{kt} e_j] = \mathbb{E}[\mathbb{E}_i[p_j^{-1} G_{ij}^{kt} e_j]] \\ &= \mathbb{E}[p_j^{-1} e_j \mathbb{E}_i[G_{ij}^{kt}]] \stackrel{(3.9)}{=} \mathbb{E}[p_j^{-1} e_j \nabla_j P(y^{k,t})] = \nabla P(y^{k,t}). \end{aligned}$$

Hence, the update step performed by S2CD is a stochastic gradient step of fixed stepsize  $h$ .

Before we describe our main complexity result in the next section, let us briefly comment on a few special cases of S2CD:

- If  $n = 1$  (this can be always achieved simply by grouping all functions in the average into a

---

<sup>2</sup>In S2CD, as presented, coordinates  $j$  is selected first, and then function  $i$  is selected, according to a distribution conditioned on the choice of  $j$ . However, one could equivalently sample  $(i, j)$  with joint probability  $p_{ij}$ . We opted for the sequential sampling for clarity of presentation purposes.

<sup>3</sup>The quantity  $\omega \stackrel{\text{def}}{=} \max_i \omega_i$  (degree of partial separability of  $P$ ) was used in the analysis of a large class of randomized parallel coordinate descent methods in [151]. The more informative quantities  $\{\omega_i\}$  appear in the analysis of parallel/distributed/mini-batch coordinate descent methods [149, 59, 57].

single function), S2CD reduces to a stochastic CD algorithm with importance sampling<sup>4</sup>, as studied in [129, 148, 143], but written with many redundant computations. Indeed, the method in this case does not require the  $w^k$  iterates, nor does it need to compute the gradient of  $P$ , and instead takes on the form:

$$y^{0,t+1} \leftarrow y^{0,t} - hp_j^{-1} \nabla_j P(y^{0,t}) e_j,$$

where  $p_j = L_{1j} / \sum_s L_{1s}$ .

- It is possible to extend the S2CD algorithm and results to the case when coordinates are replaced by (nonoverlapping) blocks of coordinates, as in [148] — we did not do it here for the sake of keeping the notation simple. In such a setting, we would obtain semi-stochastic *block* coordinate descent. In the special case with *all variables forming a single block*, the algorithm reduces to the S2GD method described in [89], but with nonuniform probabilities for the choice of  $i$  — proportional to the Lipschitz constants of the gradient of the functions  $f_i$  (this is also studied in [187]). As in [187], the complexity result then depends on the average of the Lipschitz constants.

Note that the algorithm, as presented, assumes knowledge of the strong convexity parameter  $\mu$ . We have done this for simplicity of exposition: the method works also if  $\mu$  is not explicitly known — in that case, we can simply replace  $\mu$  by 0 and the method will still depend on the true strong convexity parameter. The change to the complexity results will be only minor in constants and all our conclusions hold. Likewise, it is possible to give an  $O(1/\epsilon)$  complexity result in the non-strongly convex case of  $P$ , using standard regularization arguments (e.g., see [89]).

### 3.3 Complexity Result

In this section, we state and describe our complexity result; the proof is provided in Section 3.5.

An important step in our analysis is proving a good upper bound on the variance of the (unbiased) estimator  $g_{ij}^{kt} = p_j^{-1} G_{ij}^{kt} e_j$  of  $\nabla P(y^{k,t})$ , one that we can “believe” would diminish to zero as the algorithm progresses. This is important for several reasons. First, as the method approaches the optimum, we wish  $g_{ij}^{kt}$  to be progressively closer to the true gradient, which in turn will be close to zero. Indeed, if this was the case, then S2CD behaves like gradient descent with fixed stepsize  $h$  close to optimum. In particular, this would indicate that using fixed stepsizes makes sense.

In light of the above discussion, the following lemma plays a key role in our analysis:

**Lemma 10.** *The iterates of the S2CD algorithm satisfy*

$$\mathbb{E} [\|g_{ij}^{kt}\|^2] \leq 4\hat{L} (P(y^{k,t}) - P(w^*)) + 4\hat{L} (P(w^k) - P(w^*)), \quad (3.10)$$

where

$$\hat{L} := \frac{1}{n} \sum_{j=1}^d v_j \stackrel{(3.5)}{=} \frac{1}{n} \sum_{j=1}^d \sum_{i=1}^n \omega_i L_{ij}. \quad (3.11)$$

The proof of this lemma can be found in Section 3.4.

Note that as  $y^{k,t} \rightarrow w^*$  and  $w^k \rightarrow w^*$ , the bound (3.10) decreases to zero. This is the main feature of modern fast stochastic gradient methods: the squared norm of the stochastic gradient estimate progressively diminishes to zero, as the method progresses, in expectation. Therefore it is possible to use constant step-size in this type of algorithms. Note that the standard SGD method does not have this property: indeed, there is no reason for  $\mathbb{E}_i [\|\nabla f_i(w)\|^2]$  to be small even if  $w = w^*$ .

We are now ready to state the main result of this chapter.

---

<sup>4</sup>A parallel CD method in which every subset of coordinates can be assigned a different probability of being chosen/updated was analyzed in [150].

**Theorem 11** (Complexity of S2CD). *If  $0 < h < 1/(2\hat{L})$ , then for all  $k \geq 0$  we have:*<sup>5</sup>

$$\mathbb{E} [P(w^{k+1}) - P(w^*)] \leq \left( \frac{(1 - \mu h)^m}{(1 - (1 - \mu h)^m)(1 - 2\hat{L}h)} + \frac{2\hat{L}h}{1 - 2\hat{L}h} \right) \mathbb{E} [P(w^k) - P(w^*)]. \quad (3.12)$$

By analyzing the above result (one can follow the steps in [89, Theorem 6]), we get the following useful corollary:

**Corollary 12.** *Fix the number of epochs  $k \geq 1$ , error tolerance  $\epsilon \in (0, 1)$  and let  $\Delta \stackrel{\text{def}}{=} \epsilon^{1/k}$  and*

$$\hat{\kappa} \stackrel{\text{def}}{=} \hat{L}/\mu \stackrel{(3.11)}{=} \frac{1}{\mu n} \sum_{j=1}^d \sum_{i=1}^n \omega_i L_{ij}. \quad (3.13)$$

*If we run Algorithm 4 with stepsize  $h$  and  $m$  set as*

$$h = \frac{\Delta}{(4 + 2\Delta)\hat{L}}, \quad m \geq \left( \frac{4}{\Delta} + 2 \right) \log \left( \frac{2}{\Delta} + 2 \right) \hat{\kappa}, \quad (3.14)$$

*then  $\mathbb{E} [P(w^k) - P(w^*)] \leq \epsilon(P(w^0) - P(w^*))$ . In particular, for  $k = \lceil \log(1/\epsilon) \rceil$  we have  $\frac{1}{\Delta} \leq \exp(1)$ , and we can pick*

$$k = \lceil \log(1/\epsilon) \rceil, \quad h = \frac{\Delta}{(4 + 2\Delta)\hat{L}} \approx \frac{1}{(4 \exp(1) + 2)\hat{L}} \approx \frac{1}{12.87\hat{L}}, \quad m \geq 26\hat{\kappa}. \quad (3.15)$$

**Remark 13.** Note that in order to define  $h$  and  $m$  as in (3.14), we need to fix the target accuracy  $\epsilon$  and the number of iterations  $k$  beforehand.

If we run S2CD with the parameters set as in (3.15), then in each epoch the gradient of  $f$  is evaluated once (this is equivalent to  $n$  evaluations of  $\nabla f_i$ ), and the partial derivative of some function  $f_i$  is evaluated  $2m \approx 52\hat{\kappa} = O(\hat{\kappa})$  times. If we let  $C_{grad}$  be the average cost of evaluating the gradient  $\nabla f_i$  and  $C_{pd}$  be the average cost of evaluating the partial derivative  $\nabla_j f_i$ , then the total work of S2CD can be written as

$$(nC_{grad} + mC_{pd})k \stackrel{(3.15)}{=} \mathcal{O} \left( (nC_{grad} + \hat{\kappa}C_{pd}) \log \left( \frac{1}{\epsilon} \right) \right), \quad (3.16)$$

The complexity results of methods such as S2GD/SVRG [89, 80, 187] and SAG/SAGA [158, 45]—in a similar but not identical setup to ours (these papers assume  $f_i$  to be  $L_i$ -smooth)—can be written in a similar form:

$$\mathcal{O} \left( (nC_{grad} + \kappa C_{grad}) \log \left( \frac{1}{\epsilon} \right) \right), \quad (3.17)$$

where  $\kappa = L_{avg}/\mu$  with  $L_{avg} \stackrel{\text{def}}{=} \frac{1}{n} \sum_i L_i$  [187] (or slightly weaker where  $\kappa = L_{max}/\mu$  with  $L_{max} \stackrel{\text{def}}{=} \max_i L_i$  [158, 80, 89, 45]). The difference between our result (3.16) and existing results (3.17) is in the term  $\hat{\kappa}C_{pd}$ —previous results have  $\kappa C_{grad}$  in that place. This difference constitutes a trade-off: while  $\hat{\kappa} \geq \kappa$  (we comment on this below), we clearly have  $C_{pd} \leq C_{grad}$ . The comparison of the quantities  $\kappa C_{grad}$  and  $\hat{\kappa}C_{pd}$  is in general not straightforward and problem dependent.

Let us now compare the condition numbers  $\hat{\kappa}$  and  $\kappa = L_{avg}/\mu$ . It can be shown that (see [148])

$$L_i \leq \sum_{j=1}^d L_{ij}$$

---

<sup>5</sup>It is possible to modify the argument slightly and replace the term  $\hat{L}$  appearing in the numerator by  $\hat{L} - \frac{\mu}{\max_s p_s}$ . However, as this does not bring any significant improvements, we decided to present the result in this simplified form.

and, moreover, this inequality can be tight. Since  $\omega_i \geq 1$  for all  $i$ , we have

$$\hat{\kappa} = \frac{\hat{L}}{\mu} \stackrel{(3.11)}{=} \frac{1}{\mu n} \sum_{j=1}^d \sum_{i=1}^n \omega_i L_{ij} \geq \frac{1}{\mu n} \sum_{i=1}^n \sum_{j=1}^d L_{ij} \geq \frac{1}{\mu n} \sum_{i=1}^n L_i = \frac{L_{avg}}{\mu} = \kappa.$$

Let us denote

$$\underline{\omega} = \min_i \omega_i, \quad \bar{\omega} = \max_i \omega_i.$$

That is,  $\underline{\omega}$  and  $\bar{\omega}$  are respectively the smallest and largest number of coordinates that a subfunction depends on. In the case when

$$L_i = \sum_{j=1}^d L_{ij}, \quad (3.18)$$

it is easy to see that

$$\underline{\omega}\kappa \leq \hat{\kappa} \leq \bar{\omega}\kappa.$$

In addition, when (3.18) holds,  $\hat{\kappa}$  is smaller than  $\kappa_{max} \stackrel{\text{def}}{=} L_{max}/\mu$  if

$$\bar{\omega} \sum_{i=1}^n L_i \leq n \max_i L_i.$$

### 3.4 Proof of Lemma 10

We will prove the following stronger inequality:

$$\mathbb{E} \left[ \|g_{ij}^{kt}\|^2 \right] \leq 4\hat{L} (P(y^{k,t}) - P(w^*)) + 4 \left( \hat{L} - \frac{\mu}{\max_s p_s} \right) (P(w^k) - P(w^*)). \quad (3.19)$$

Lemma 10 follows by dropping the negative term.

**STEP 1.** We first break down the left hand side of (3.19) into  $d$  terms each of which we will bound separately. By first taking expectation conditioned on  $j$  and then taking the full expectation, we can write:

$$\begin{aligned} \mathbb{E} \left[ \|g_{ij}^{kt}\|^2 \right] &\stackrel{(3.7)}{=} \mathbb{E} \left[ \mathbb{E}_i \left[ \|p_j^{-1} G_{ij}^{kt} e_j\|^2 \right] \right] \\ &= \mathbb{E} \left[ p_j^{-2} \mathbb{E}_i \left[ (G_{ij}^{kt})^2 \right] \right] = \sum_{s=1}^d p_s^{-1} \mathbb{E}_i \left[ (G_{is}^{kt})^2 \right]. \end{aligned} \quad (3.20)$$

**STEP 2.** We now further break each of these  $d$  terms into three pieces. That is, for each  $j = 1, \dots, d$  we have:

$$\begin{aligned}
\mathbb{E}_i \left[ (G_{ij}^{kt})^2 \right] &\stackrel{(3.8)}{=} \mathbb{E}_i \left[ \left( \nabla_j P(w^k) + \frac{\nabla_j f_i(y^{k,t}) - \nabla_j f_i(w^k)}{nq_{ij}} + \frac{\nabla_j f_i(w^*) - \nabla_j f_i(w^*)}{nq_{ij}} \right)^2 \right] \\
&= \mathbb{E}_i \left[ \left( \frac{\nabla_j f_i(y^{k,t}) - \nabla_j f_i(w^*)}{nq_{ij}} + \nabla_j P(w^k) - \frac{\nabla_j f_i(w^k) - \nabla_j f_i(w^*)}{nq_{ij}} \right)^2 \right] \\
&\leq 2\mathbb{E}_i \left[ \left( \frac{\nabla_j f_i(y^{k,t}) - \nabla_j f_i(w^*)}{nq_{ij}} \right)^2 \right] + 2\mathbb{E}_i \left[ \left( \nabla_j P(w^k) - \frac{\nabla_j f_i(w^k) - \nabla_j f_i(w^*)}{nq_{ij}} \right)^2 \right] \\
&= 2\mathbb{E}_i \left[ \left( \frac{\nabla_j f_i(y^{k,t}) - \nabla_j f_i(w^*)}{nq_{ij}} \right)^2 \right] \\
&\quad + 2\mathbb{E}_i \left[ \left( \frac{\nabla_j f_i(w^k) - \nabla_j f_i(w^*)}{nq_{ij}} - (\nabla_j P(w^k) - \nabla_j P(w^*)) \right)^2 \right] \\
&= 2\mathbb{E}_i \left[ \left( \frac{\nabla_j f_i(y^{k,t}) - \nabla_j f_i(w^*)}{nq_{ij}} \right)^2 \right] + 2\mathbb{E}_i \left[ \left( \frac{\nabla_j f_i(w^k) - \nabla_j f_i(w^*)}{nq_{ij}} \right)^2 \right] \\
&\quad - 2(\nabla_j P(w^k) - \nabla_j P(w^*))^2,
\end{aligned} \tag{3.21}$$

where the last equality follows from the fact that

$$\mathbb{E}_i \left[ \frac{\nabla_j f_i(w^k) - \nabla_j f_i(w^*)}{nq_{ij}} \right] = \sum_{i=1}^n q_{ij} \frac{\nabla_j f_i(w^k) - \nabla_j f_i(w^*)}{nq_{ij}} = \nabla_j P(w^k) - \nabla_j P(w^*).$$

**STEP 3.** In this step we bound the first two terms in the right hand side of inequality (3.21). It will now be useful to introduce the following notation:

$$Q_j \stackrel{\text{def}}{=} \{i : L_{ij} \neq 0\}, \quad j = 1, \dots, d, \tag{3.22}$$

and

$$1_{ij} \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } L_{ij} \neq 0 \\ 0 & \text{otherwise} \end{cases}, \quad i = 1, \dots, n, \quad j = 1, \dots, d.$$

Let us first examine the first term in the right-hand side of (3.21). Using the coordinate co-coercivity lemma (Lemma 14) with  $y = w^*$ , we obtain the inequality

$$(\nabla_j f_i(w) - \nabla_j f_i(w^*))^2 \leq 2L_{ij} (f_i(w) - f_i(w^*) - \langle \nabla f_i(w^*), w - w^* \rangle), \tag{3.23}$$

using which we get the bound:

$$\begin{aligned}
&2 \sum_{s=1}^d p_s^{-1} \mathbb{E}_i \left[ \left( \frac{1}{nq_{i,s}} (\nabla_s f_i(y^{k,t}) - \nabla_s f_i(w^*)) \right)^2 \right] \\
&= 2 \sum_{s=1}^d p_s^{-1} \sum_{i \in Q_s} \frac{1}{n^2 q_{i,s}} (\nabla_s f_i(y^{k,t}) - \nabla_s f_i(w^*))^2 \\
&\stackrel{(3.23)}{\leq} 4 \sum_{s=1}^d p_s^{-1} \sum_{i \in Q_s} \frac{L_{is}}{n^2 q_{i,s}} (f_i(y^{k,t}) - f_i(w^*) - \langle \nabla f_i(w^*), y^{k,t} - w^* \rangle) \\
&\stackrel{(3.22)}{=} 4 \sum_{i=1}^n \sum_{s=1}^d p_s^{-1} 1_{is} \frac{v_s}{n^2 \omega_i} (f_i(y^{k,t}) - f_i(w^*) - \langle \nabla f_i(w^*), y^{k,t} - w^* \rangle).
\end{aligned} \tag{3.24}$$

Note that by (3.5) and (3.11), we have that for all  $s = 1, 2, \dots, d$ ,

$$p_s^{-1} v_s = n \hat{L}.$$

Continuing from (3.24), we can therefore further write

$$\begin{aligned}
& 2 \sum_{s=1}^d p_s^{-1} \mathbb{E}_i \left[ \left( \frac{1}{nq_{ij}} (\nabla_j f_i(y^{k,t}) - \nabla_j f_i(w^*)) \right)^2 \right] \\
& \leq 4 \sum_{i=1}^n \sum_{s=1}^d 1_{is} \frac{\hat{L}}{n\omega_i} (f_i(y^{k,t}) - f_i(w^*) - \langle \nabla f_i(w^*), y^{k,t} - w^* \rangle) \\
& = \frac{4\hat{L}}{n} \sum_{i=1}^n (f_i(y^{k,t}) - f_i(w^*) - \langle \nabla f_i(w^*), y^{k,t} - w^* \rangle) \\
& = 4\hat{L}(P(y^{k,t}) - P(w^*)).
\end{aligned} \tag{3.25}$$

The same reasoning applies to the second term on the right-hand side of the inequality (3.21) and we have:

$$2 \sum_{s=1}^d p_s^{-1} \mathbb{E}_i \left[ \left( \frac{1}{nq_{ij}} (\nabla_j f_i(w^k) - \nabla_j f_i(w^*)) \right)^2 \right] \leq 4\hat{L}(P(w^k) - P(w^*)). \tag{3.26}$$

**STEP 4.** Next we bound the third term on the right-hand side of the inequality (3.21). First note that since  $P$  is  $\mu$ -strongly convex (see (3.3)), for all  $w \in \mathbb{R}^d$  we have:

$$\langle \nabla P(w), w - w^* \rangle \geq P(w) - P(w^*) + \frac{\mu}{2} \|w - w^*\|^2. \tag{3.27}$$

We can now write:

$$\begin{aligned}
2 \sum_{s=1}^d p_s^{-1} (\nabla_s P(w^k) - \nabla_s P(w^*))^2 & \geq \frac{2}{\max_s p_s} \sum_{j=1}^d (\nabla_j P(w^k) - \nabla_j P(w^*))^2 \\
& \stackrel{(3.27)}{\geq} \frac{4\mu}{\max_s p_s} (P(w^k) - P(w^*)).
\end{aligned} \tag{3.28}$$

**STEP 5.** We conclude by combining (3.20), (3.21), (3.25), (3.26) and (3.28).

## 3.5 Proof of the Main Result

In this section we provide the proof of our main result. In order to present the proof in an organized fashion, we first establish two technical lemmas.

### 3.5.1 Coordinate co-coercivity

It is a well known and widely used fact (see, e.g. [127]) that for a continuously differentiable function  $\phi : \mathbb{R}^d \rightarrow \mathbb{R}$  and constant  $L_\phi > 0$ , the following two conditions are equivalent:

$$\phi(x) \leq \phi(y) + \langle \nabla \phi(y), x - y \rangle + \frac{L_\phi}{2} \|x - y\|^2, \quad \forall x, y \in \mathbb{R}^d$$

and

$$\|\nabla \phi(x) - \nabla \phi(y)\|^2 \leq 2L_\phi(\phi(x) - \phi(y) - \langle \nabla \phi(y), x - y \rangle), \quad \forall x, y \in \mathbb{R}^d.$$

The second condition is often referred to by the name co-coercivity. Note that our assumption (3.2) on  $f_i$  is similar to the first inequality. In our first lemma we establish a coordinate-based co-coercivity result which applies to functions  $f_i$  satisfying (3.2).

**Lemma 14** (Coordinate co-coercivity). *For all  $w, y \in \mathbb{R}^d$  and  $i = 1, \dots, n$ ,  $j = 1, \dots, d$ , we have:*

$$(\nabla_j f_i(w) - \nabla_j f_i(y))^2 \leq 2L_{ij} (f_i(w) - f_i(y) - \langle \nabla f_i(y), w - y \rangle). \tag{3.29}$$

*Proof.* Fix any  $i, j$  and  $y \in \mathbb{R}^d$ . Consider the function  $g_i : \mathbb{R}^d \rightarrow \mathbb{R}$  defined by:

$$g_i(w) \stackrel{\text{def}}{=} f_i(w) - f_i(y) - \langle \nabla f_i(y), w - y \rangle. \quad (3.30)$$

Then since  $f_i$  is convex, we know that  $g_i(w) \geq 0$  for all  $w$ , with  $g_i(y) = 0$ . Hence,  $y$  minimizes  $g_i$ . We also know that for any  $w \in \mathbb{R}^d$ :

$$\nabla_j g_i(w) = \nabla_j f_i(w) - \nabla_j f_i(y). \quad (3.31)$$

Since  $f_i$  satisfies (3.2), so does  $g_i$ , and hence for all  $w \in \mathbb{R}^d$  and  $h \in \mathbb{R}$ , we have

$$g_i(w + he_j) \leq g_i(w) + \langle \nabla g_i(w), he_j \rangle + \frac{L_{ij}}{2} h^2.$$

Minimizing both sides in  $h$ , we obtain

$$g_i(y) \leq \min_h g_i(w + he_j) \leq g_i(w) - \frac{1}{2L_{ij}} (\nabla_j g_i(w))^2,$$

which together with (3.30) yield the result.  $\square$

### 3.5.2 Recursion

We now proceed to the final lemma, establishing a key recursion which ultimately yields the proof of the main theorem, which we present in Section 3.5.3.

**Lemma 15** (Recursion). *The iterates of S2CD satisfy the following recursion:*

$$\begin{aligned} & \frac{1}{2} \mathbb{E} [\|y^{k,t+1} - w^*\|^2] + h(1 - 2h\hat{L}) (P(y^{k,t}) - P(w^*)) \\ & \leq (1 - h\mu) \frac{1}{2} \|y^{k,t} - w^*\|^2 + 2h^2 \hat{L} (P(w^k) - P(w^*)). \end{aligned} \quad (3.32)$$

*Proof.*

$$\begin{aligned} & \frac{1}{2} \mathbb{E} [\|y^{k,t+1} - w^*\|^2] \stackrel{(3.6)}{=} \frac{1}{2} \mathbb{E} [\|y^{k,t} - hp_j^{-1} G_{ij}^{kt} e_j - w^*\|^2] \\ & = \frac{1}{2} \|y^{k,t} - w^*\|^2 - \mathbb{E} [\langle hp_j^{-1} G_{ij}^{kt} e_j, y^{k,t} - w^* \rangle] + \frac{1}{2} \mathbb{E} [\|hp_j^{-1} G_{ij}^{kt} e_j\|^2] \\ & \stackrel{(3.9)}{=} \frac{1}{2} \|y^{k,t} - w^*\|^2 - h \langle \nabla P(y^{k,t}), y^{k,t} - w^* \rangle + \frac{h^2}{2} \mathbb{E} [\|g_{ij}^{kt}\|^2] \\ & \stackrel{(3.27)}{\leq} \frac{1}{2} \|y^{k,t} - w^*\|^2 - h (P(y^{k,t}) - P(w^*) + \frac{\mu}{2} \|y^{k,t} - w^*\|^2) + \frac{h^2}{2} \mathbb{E} [\|g_{ij}^{kt}\|^2] \\ & \stackrel{(3.10)}{\leq} \frac{1}{2} \|y^{k,t} - w^*\|^2 - h (P(y^{k,t}) - P(w^*) + \frac{\mu}{2} \|y^{k,t} - w^*\|^2) \\ & \quad + 2h^2 \hat{L} (P(y^{k,t}) - P(w^*)) + 2h^2 \hat{L} (P(w^k) - P(w^*)) \\ & = (1 - \mu h) \frac{1}{2} \|y^{k,t} - w^*\|^2 - h(1 - 2h\hat{L}) (P(y^{k,t}) - P(w^*)) \\ & \quad + 2h^2 \hat{L} (P(w^k) - P(w^*)). \end{aligned}$$

$\square$

### 3.5.3 Proof of Theorem 11

For simplicity, let us denote:

$$\eta^{k,t} \stackrel{\text{def}}{=} \frac{1}{2} \mathbb{E} [\|y^{k,t} - w^*\|^2], \quad \xi^{k,t} \stackrel{\text{def}}{=} \mathbb{E} [P(y^{k,t}) - P(w^*)],$$

where the expectation now is with respect to the entire history. Notice that

$$y^{k+1,0} = y^{k,t^k},$$

where  $t^k = T \in \{1, \dots, m\}$  with probability  $(1 - \mu h)^{m-T}/\beta$  with  $\beta$  defined in (3.4). Conditioning on  $t^k$  we obtain that

$$\xi^{k+1,0} = \frac{1}{\beta} \sum_{t=0}^{m-1} (1 - \mu h)^t \xi^{k,m-1-t}. \quad (3.33)$$

See also [89, Lemma 3] for a proof. By Lemma 15 we have the following  $m$  inequalities:

$$\begin{aligned} \eta^{k,m} + h(1 - 2h\hat{L})\xi^{k,m-1} &\leq (1 - \mu h)\eta^{k,m-1} + 2h^2\hat{L}\xi^{k,0}, \\ (1 - \mu h)\eta^{k,m-1} + h(1 - 2h\hat{L})(1 - \mu h)\xi^{k,m-2} &\leq (1 - \mu h)^2\eta^{k,m-2} + 2h^2\hat{L}(1 - \mu h)\xi^{k,0}, \\ &\vdots \\ (1 - \mu h)^t\eta^{k,m-t} + h(1 - 2h\hat{L})(1 - \mu h)^t\xi^{k,m-t-1} &\leq (1 - \mu h)^{t+1}\eta^{k,m-t-1} + 2h^2\hat{L}(1 - \mu h)^t\xi^{k,0}, \\ &\vdots \\ (1 - \mu h)^{m-1}\eta^{k,1} + \gamma(1 - 2h\hat{L})(1 - \mu h)^{m-1}\xi^{k,0} &\leq (1 - \mu h)^m\eta^{k,0} + 2h^2\hat{L}(1 - \mu h)^{m-1}\xi^{k,0}. \end{aligned}$$

By summing up the above  $m$  inequalities, we get:

$$\eta^{k,m} + \gamma(1 - 2h\hat{L}) \sum_{t=0}^{m-1} (1 - \mu h)^t \xi^{k,m-1-t} \leq (1 - \mu h)^m \eta^{k,0} + 2h^2\hat{L}\beta\xi^{k,0},$$

It follows from the strong convexity assumption (3.3) that  $P(w^k) - P(w^*) \geq \frac{\mu}{2}\|w^k - w^*\|^2$ , that is,  $\xi^{k,0} \geq \mu\eta^{k,0}$ . Therefore, together with (3.33) we get:

$$h(1 - 2h\hat{L})\xi^{k+1,0} \leq \left( \frac{(1 - \mu h)^m}{\beta\mu} + 2h^2\hat{L} \right) \xi^{k,0}$$

Hence if  $0 < 2h\hat{L} < 1$ , then we obtain:

$$\xi^{k+1,0} \leq \left( \frac{(1 - \mu h)^m}{(1 - (1 - \mu h)^m)(1 - 2h\hat{L})} + \frac{2h\hat{L}}{1 - 2h\hat{L}} \right) \xi^{k,0},$$

which finishes the proof.



## **Part II**

# **Parallel and Distributed Methods with Variance Reduction**



# Chapter 4

## Mini-batch Semi-Stochastic Gradient Descent in the Proximal Setting

### 4.1 Introduction

In this work we are concerned with the problem of minimizing the sum of two convex functions,

$$\min_{w \in \mathbb{R}^d} \{P(w) := f(w) + R(w)\}, \quad (4.1)$$

where the first component,  $f$ , is smooth, and the second component,  $R$ , is possibly nonsmooth (and extended real-valued, which allows for the modeling of constraints).

In the last decade, an intensive amount of research was conducted into algorithms for solving problems of the form (4.1), largely motivated by the realization that the underlying problem has a considerable modeling power. One of the most popular and practical methods for (4.1) is the accelerated proximal gradient method of Nesterov [128], with its most successful variant being FISTA [11].

In many applications in optimization, signal processing and machine learning,  $f$  has an additional structure. In particular, it is often the case that  $f$  is the *average of a number of convex functions*:

$$f(w) = \frac{1}{n} \sum_{i=1}^n f_i(w). \quad (4.2)$$

Indeed, even one of the most basic optimization problems—least squares regression—lends itself to a natural representation of the form (4.2).

#### 4.1.1 Stochastic methods.

For problems of the form (4.1)+(4.2), and especially when  $n$  is large and when a solution of low to medium accuracy is sufficient, deterministic methods do not perform as well as classical *stochastic* methods. The prototype method in this category is stochastic gradient descent (SGD), dating back to the 1951 seminal work of Robbins and Monro [153]. SGD selects an index  $i \in \{1, 2, \dots, n\}$  uniformly at random, and then updates the variable  $w$  using  $\nabla f_i(w)$  — a stochastic estimate of  $\nabla f(w)$ . Note that the computation of  $\nabla f_i$  is  $n$  times cheaper than the computation of the full gradient  $\nabla f$ . For problems where  $n$  is very large, the per-iteration savings can be extremely large, spanning several orders of magnitude.

These savings do not come for free, however (modern methods, such as the one we propose, overcome this – more on that below). Indeed, the stochastic estimate of the gradient embodied by  $\nabla f_i$  has a non-vanishing variance. To see this, notice that even when started from an optimal solution  $w^*$ , there is no reason for  $\nabla f_i(w^*)$  to be zero, which means that SGD drives away from the optimal point. Traditionally, there have been two ways of dealing with this

issue. The first one consists in choosing a decreasing sequence of stepsizes. However, this means that a much larger number of iterations is needed. A second approach is to use a subset (“minibatch”) of indices  $i$ , as opposed to a single index, in order to form a better stochastic estimate of the gradient. However, this results in a method which performs more work per iteration. In summary, while traditional approaches manage to decrease the variance in the stochastic estimate, this comes at a cost.

### 4.1.2 Modern stochastic methods

Very recently, starting with the SAG [156], SDCA [163], SVRG [80] and S2GD [89] algorithms from year 2013, it has transpired that neither decreasing stepsizes nor mini-batching are necessary to resolve the non-vanishing variance issue inherent in the vanilla SGD methods. Instead, these modern stochastic<sup>1</sup> method are able to dramatically improve upon SGD in various different ways, but without having to resort to the usual variance-reduction techniques (such as decreasing stepsizes or mini-batching) which carry with them considerable costs drastically reducing their power. Instead, these modern methods were able to improve upon SGD without any unwelcome side effects. This development led to a revolution in the area of first order methods for solving problem (4.1)+(4.2). Both the theoretical complexity and practical efficiency of these modern methods vastly outperform prior gradient-type methods.

In order to achieve  $\epsilon$ -accuracy, that is,

$$\mathbb{E} [P(w^k) - P(w^*)] \leq \epsilon [P(w^0) - P(w^*)], \quad (4.3)$$

modern stochastic methods such as SAG, SDCA, SVRG and S2GD require only

$$\mathcal{O}((n + \kappa) \log(1/\epsilon)) \quad (4.4)$$

units of work, where  $\kappa$  is a condition number associated with  $f$ , and one unit of work corresponds to the computation of the gradient of  $f_i$  for a random index  $i$ , followed by a call to a prox-mapping involving  $R$ . More specifically,  $\kappa = L/\mu$ , where  $L$  is a uniform bound on the Lipschitz constants of the gradients of functions  $f_i$  and  $\mu$  is the strong convexity constant of  $P$ . These quantities will be defined precisely in Section 4.4.

The complexity bound (4.4) should be contrasted with that of proximal gradient descent (e.g., ISTA), which requires  $\mathcal{O}(n\kappa \log(1/\epsilon))$  units of work, or FISTA, which requires  $\mathcal{O}(n\sqrt{\kappa} \log(1/\epsilon))$  units of work<sup>2</sup>. Note that while all these methods enjoy linear convergence rate, the modern stochastic methods can be many orders of magnitude faster than classical deterministic methods. Indeed, one can have

$$n + \kappa \ll n\sqrt{\kappa} \leq n\kappa.$$

Based on this, we see that these modern methods always beat (proximal) gradient descent ( $n + \kappa \ll n\kappa$ ), and also outperform FISTA as long as  $\kappa \leq \mathcal{O}(n^2)$ . In machine learning, for instance, one usually has  $\kappa \approx n$ , in which case the improvement is by a factor of  $\sqrt{n}$  when compared to FISTA, and by a factor of  $n$  over ISTA. For applications where  $n$  is massive, these improvements are indeed dramatic.

For more information about modern dual and primal methods we refer the reader to the literature on randomized coordinate descent methods [129, 148, 151, 122, 59, 163, 111, 121, 149, 57, 140, 36] and stochastic gradient methods [156, 196, 105, 79, 172, 132, 140, 155], respectively.

### 4.1.3 Linear systems and sketching.

In the case when  $R \equiv 0$ , all stationary points (i.e., points satisfying  $\nabla f(w) = 0$ ) are optimal for (4.1)+(4.2). In the special case when the functions  $f_i$  are convex quadratics of the form

---

<sup>1</sup>These methods are randomized algorithms. However, the term “stochastic” (somewhat incorrectly) appears in their names for historical reasons, and quite possibly due to their aspiration to improve upon *stochastic* gradient descent (SGD).

<sup>2</sup>However, it should be remarked that the condition number  $\kappa$  in these latter methods is slightly different from that appearing in the bound (4.4).

$f_i(w) = \frac{1}{2}(a_i^T w - b_i)$ , the equation  $\nabla f(w) = 0$  reduces to the linear system  $A^T A w = A^T b$ , where  $A = [a_1, \dots, a_n]$ . Recently, there has been considerable interest in designing and analyzing randomized methods for solving linear systems; also known under the name of *sketching* methods. Much of this work was done independently from the developments in (non-quadratic) optimization, despite the above connection between optimization and linear systems. A randomized version of the classical Kaczmarz method was studied in a seminal paper by Strohmer and Vershynin [170]. Subsequently, the method was extended and improved upon in several ways [123, 204, 103, 135]. The randomized Kaczmarz method is equivalent to SGD with a specific stepsize choice [124, 68]. The first randomized coordinate descent method, for linear systems, was analyzed by Lewis and Leventhal [97], and subsequently generalized in various ways by numerous authors (we refer the reader to [140] and the references therein). Gower and Richtárik [68] have recently studied randomized iterative methods for linear systems in a general *sketch and project* framework, which in special cases includes randomized Kaczmarz, randomized coordinate descent, Gaussian descent, randomized Newton, their block variants, variants with importance sampling, and also an infinite array of new specific methods. For approaches of a combinatorial flavor, specific to diagonally dominant systems, we refer to the influential work of Spielman and Teng [169].

## 4.2 Contributions

In this chapter we equip modern stochastic methods—methods which already enjoy the fast rate (4.4)—with the ability to process data in *mini-batches*. None of the *primal*<sup>3</sup> modern methods have been analyzed in the mini-batch setting. This work fills this gap in the literature.

While we have argued above that the modern methods, S2GD included, do not have the “non-vanishing variance” issue that SGD does, and hence do not *need* mini-batching for that purpose, mini-batching is still useful. In particular, we develop and analyze the complexity of mS2GD (Algorithm 5) — a mini-batch proximal variant of *semi-stochastic gradient descent* (S2GD) [89]. While the S2GD method was analyzed in the  $R = 0$  case only, we develop and analyze our method in the proximal<sup>4</sup> setting (4.1). We show that mS2GD enjoys several benefits when compared to previous modern methods. First, it trivially admits a parallel implementation, and hence enjoys a speedup in clocktime in an HPC environment. This is critical for applications with massive datasets and is the main motivation and advantage of our method. Second, our results show that in order to attain a specified accuracy  $\epsilon$ , mS2GD can get by with fewer gradient evaluations than S2GD. This is formalized in Theorem 19, which predicts more than linear speedup up to a certain threshold mini-batch size after which the complexity deteriorates. Third, compared to [187], our method does not need to average the iterates produced in each inner loop; we instead simply continue from the last one. This is the approach employed in S2GD [89].

## 4.3 The Algorithm

In this section we first briefly motivate the mathematical setup of deterministic and stochastic proximal gradient methods in Section 4.3.1, followed by the introduction of semi-stochastic gradient descent in Section 4.3.2. We will be ready to describe the mS2GD method in Section 4.3.3.

---

<sup>3</sup>By a primal method we refer to an algorithm which operates directly to solve (4.1)+(4.2) without explicitly operating on the dual problem. *Dual* methods have very recently been analyzed in the mini-batch setting. For a review of such methods we refer the reader to the paper describing the QUARTZ method [143] and the references therein.

<sup>4</sup>Note that the Prox-SVRG method [187] can also handle the composite problem (4.1).

### 4.3.1 Deterministic and stochastic proximal gradient methods

The classical *deterministic proximal gradient* approach [11, 35, 136] to solving (4.1) is to form a sequence  $\{y^t\}$  via

$$y^{t+1} = \arg \min_{w \in \mathbb{R}^d} U^t(w),$$

where  $U^t(w) \stackrel{\text{def}}{=} f(y^t) + \nabla f(y^t)^T(w - y^t) + \frac{1}{2h}\|w - y^t\|^2 + R(w)$ . Note that in view of Assumption 16, which we shall use in our analysis in Section 4.4,  $U^t$  is an upper bound on  $P$  whenever  $h > 0$  is a stepsize parameter satisfying  $1/h \geq L$ . This procedure can be compactly written using the *proximal operator* as follows:

$$y^{t+1} = \text{prox}_{hR}(y^t - h\nabla f(y^t)),$$

where

$$\text{prox}_{hR}(z) \stackrel{\text{def}}{=} \arg \min_{w \in \mathbb{R}^d} \left\{ \frac{1}{2}\|w - z\|^2 + hR(w) \right\}.$$

In a large-scale setting it is more efficient to instead consider the *stochastic proximal gradient* approach, in which the proximal operator is applied to a stochastic gradient step:

$$y^{t+1} = \text{prox}_{hR}(y^t - hG^t), \quad (4.5)$$

where  $G^t$  is a stochastic estimate of the gradient  $\nabla f(y^t)$ .

### 4.3.2 Semi-stochastic methods

Of particular relevance to our work are the SVRG [80], S2GD [89] and Prox-SVRG [187] methods where the stochastic estimate of  $\nabla f(y^t)$  is of the form

$$G^t = \nabla f(w) + \frac{1}{nq_{i_t}}(\nabla f_{i_t}(y^t) - \nabla f_{i_t}(w)), \quad (4.6)$$

where  $w$  is an “old” reference point for which the gradient  $\nabla f(w)$  was already computed in the past, and  $i_t \in [n] \stackrel{\text{def}}{=} \{1, 2, \dots, n\}$  is a random index equal to  $i$  with probability  $q_i > 0$ . Notice that  $G^t$  is an unbiased estimate of the gradient of  $f$  at  $y^t$ :

$$\mathbb{E}_{i_t} [G^t] \stackrel{(4.6)}{=} \nabla f(w) + \sum_{i=1}^n q_i \frac{1}{nq_i} (\nabla f_i(y^t) - \nabla f_i(w)) \stackrel{(4.2)}{=} \nabla f(y^t).$$

Methods such as S2GD, SVRG, and Prox-SVRG update the points  $y_t$  in an inner loop, and the reference point  $w$  in an outer loop (“epoch”) indexed by  $k$ . With this new outer iteration counter we will have  $w^k$  instead of  $w$ ,  $y^{k,t}$  instead of  $y^t$  and  $G^{k,t}$  instead of  $G_t$ . This is the notation we will use in the description of our algorithm in Section 4.3.3. The outer loop ensures that the squared norm of  $G^{k,t}$  approaches zero as  $k, t \rightarrow \infty$  (it is easy to see that this is equivalent to saying that the stochastic estimate  $G^{k,t}$  has a diminishing variance), which ultimately leads to extremely fast convergence.

### 4.3.3 Mini-batch S2GD

We are now ready to describe the mS2GD method<sup>5</sup> (Algorithm 5).

---

<sup>5</sup> A more detailed algorithm and the associated analysis (in which we benefit from the knowledge of lower-bound on the strong convexity parameters of the functions  $f$  and  $R$ ) can be found in the arXiv preprint [81]. The more general algorithm mainly differs in  $t^k$  being chosen according to a geometric probability law which depends on the estimates of the convexity constants.

---

**Algorithm 5** mS2GD

---

```

1: Input:  $m$  (max # of stochastic steps per epoch);  $h > 0$  (stepsize);  $w^0 \in \mathbb{R}^d$  (starting point); mini-batch size  $b \in [n]$ 
2: for  $k = 0, 1, 2, \dots$  do
3:   Compute and store  $g^k \leftarrow \nabla f(w^k) = \frac{1}{n} \sum_i \nabla f_i(w^k)$ 
4:   Initialize the inner loop:  $y^{k,0} \leftarrow w^k$ 
5:   Choose  $t^k \in \{1, 2, \dots, m\}$  uniformly at random
6:   for  $t = 0$  to  $t^k - 1$  do
7:     Choose mini-batch  $A^{kt} \subseteq [n]$  of size  $b$ ; uniformly at random
8:     Compute a stochastic estimate of  $\nabla f(y^{k,t})$ :
9:      $G^{k,t} \leftarrow g^k + \frac{1}{b} \sum_{i \in A^{kt}} (\nabla f_i(y^{k,t}) - \nabla f_i(w^k))$ 
10:     $y^{k,t+1} \leftarrow \text{prox}_{hR}(y^{k,t} - hG^{k,t})$ 
11:   end for
12:   Set  $w^{k+1} \leftarrow y^{k,t^k}$ 
13: end for

```

---

The algorithm includes an outer loop, indexed by epoch counter  $k$ , and an inner loop, indexed by  $t$ . Each epoch is started by computing  $g^k$ , which is the (full) gradient of  $f$  at  $w^k$ . It then immediately proceeds to the inner loop. The inner loop is run for  $t^k$  iterations, where  $t^k$  is chosen uniformly at random from  $\{1, \dots, m\}$ . Subsequently, we run  $t^k$  iterations in the inner loop (corresponding to Steps 6–10). Each new iterate is given by the proximal update (4.5), however with the stochastic estimate of the gradient  $G^{k,t}$  in (4.6), which is formed by using a *mini-batch* of examples  $A^{kt} \subseteq [n]$  of size  $|A^{kt}| = b$ . Each inner iteration requires  $2b$  units of work<sup>6</sup>.

## 4.4 Analysis

In this section, we lay down the assumptions, state our main complexity result, and comment on how to optimally choose the parameters of the method.

### 4.4.1 Assumptions

Our analysis is performed under the following two assumptions.

**Assumption 16.** *Function  $R : \mathbb{R}^d \rightarrow \mathbb{R} \cup \{+\infty\}$  (regularizer/proximal term) is convex and closed. The functions  $f_i : \mathbb{R}^d \rightarrow \mathbb{R}$  have Lipschitz continuous gradients with constant  $L > 0$ . That is,  $\|\nabla f_i(w) - \nabla f_i(y)\| \leq L\|w - y\|$ , for all  $x, y \in \mathbb{R}^d$ , where  $\|\cdot\|$  is the  $\ell_2$ -norm.*

Hence, the gradient of  $f$  is also Lipschitz continuous with the same constant  $L$ .

**Assumption 17.**  *$P$  is strongly convex with parameter  $\mu > 0$ . That is for all  $x, y \in \text{dom}(P)$  and any  $\xi \in \partial P(x)$ ,*

$$P(y) \geq P(x) + \xi^T(y - x) + \frac{\mu}{2}\|y - x\|^2, \quad (4.7)$$

where  $\partial P(x)$  is the subdifferential of  $P$  at  $x$ .

Lastly, by  $\mu_f \geq 0$  and  $\mu_R \geq 0$  we denote the strong convexity constants of  $f$  and  $R$ , respectively. We allow both of these quantities to be equal to 0, which simply means that the functions are convex (which we already assumed above). Hence, this is not an additional assumption.

### 4.4.2 Main result

We are now ready to formulate our complexity result.

---

<sup>6</sup>It is possible to finish each iteration with only  $b$  evaluations for component gradients, namely  $\{\nabla f_i(y^{k,t})\}_{i \in A^{kt}}$ , at the cost of having to store  $\{\nabla f_i(w^k)\}_{i \in [n]}$ , which is exactly the way that SAG [156] works. This speeds up the algorithm; nevertheless, it is impractical for big  $n$ .

**Theorem 18.** Let Assumptions 16 and 17 be satisfied, let  $w^* \stackrel{\text{def}}{=} \arg \min_w P(w)$  and choose  $b \in \{1, 2, \dots, n\}$ . Assume that  $0 < h \leq 1/L$ ,  $4hL\alpha(b) < 1$  and that  $m, h$  are further chosen so that

$$c \stackrel{\text{def}}{=} \frac{1}{mh\mu(1 - 4hL\alpha(b))} + \frac{4hL\alpha(b)(m+1)}{m(1 - 4hL\alpha(b))} < 1, \quad (4.8)$$

where  $\alpha(b) \stackrel{\text{def}}{=} \frac{n-b}{b(n-1)}$ . Then mS2GD has linear convergence in expectation with rate  $c$ :

$$\mathbb{E} [P(w^k) - P(w^*)] \leq c^k [P(w^0) - P(w^*)].$$

Notice that for any fixed  $b$ , by properly adjusting the parameters  $h$  and  $m$  we can force  $c$  to be arbitrarily small. Indeed, the second term can be made arbitrarily small by choosing  $h$  small enough. Fixing the resulting  $h$ , the first term can then be made arbitrarily small by choosing  $m$  large enough. This may look surprising, since this means that only a single outer loop ( $k = 1$ ) is needed in order to obtain a solution of any prescribed accuracy. While this is indeed the case, such a choice of the parameters of the method  $(m, h, k)$  would not be optimal – the resulting workload would be too high as the complexity of the method would depend sublinearly on  $\epsilon$ . In order to obtain a logarithmic dependence on  $1/\epsilon$ , i.e., in order to obtain linear convergence, one needs to perform  $k = O(\log(1/\epsilon))$  outer loops, and set the parameters  $h$  and  $m$  to appropriate values (generally,  $h = \Theta(1/L)$  and  $m = \Theta(\kappa)$ ).

#### 4.4.3 Special cases: $b = 1$ and $b = n$

In the special case with  $b = 1$  (no mini-batching), we get  $\alpha(b) = 1$ , and the rate given by (4.8) exactly recovers the rate achieved by Prox-SVRG [187] (in the case when the Lipschitz constants of  $\nabla f_i$  are all equal). The rate is also identical to the rate of S2GD [89] (in the case of  $R = 0$ , since S2GD was only analyzed in that case). If we set the number of outer iterations to  $k = \lceil \log(1/\epsilon) \rceil$ , choose the stepsize as  $h = \frac{1}{(2+4e)L}$ , where  $e = \exp(1)$ , and choose  $m = 43\kappa$ , then the total workload of mS2GD for achieving (4.3) is  $(n + 43\kappa) \log(1/\epsilon)$  units of work. Note that this recovers the fast rate (4.4).

In the batch setting, that is when  $b = n$ , we have  $\alpha(b) = 0$  and hence  $c = 1/(mh\mu)$ . By choosing  $k = \lceil \log(1/\epsilon) \rceil$ ,  $h = 1/L$ , and  $m = 2\kappa$ , we obtain the rate  $\mathcal{O}(n\kappa \log(1/\epsilon))$ . This is the standard rate of (proximal) gradient descent.

Hence, by modifying the mini-batch size  $b$  in mS2GD, we interpolate between the fast rate of S2GD and the slow rate of GD.

#### 4.4.4 Mini-batch speedup

In this section we will derive formulas for good choices of the parameter  $m, h$  and  $k$  of our method as a function of  $b$ . Hence, throughout this section we shall consider  $b$  fixed.

Fixing  $0 < c < 1$ , it is easy to see that in order for  $w^k$  to be an  $\epsilon$ -accurate solution (i.e., in order for (4.3) to hold), it suffices to choose  $k \geq (1-c)^{-1} \log(\epsilon^{-1})$ . Notice that the total workload mS2GD will do in order to arrive at  $w^k$  is

$$k(n+2m) \approx (1-c)^{-1} \log(\epsilon^{-1})(n+2m)$$

units of work. If we now consider  $c$  fixed (we may try to optimize for it later), then clearly the total workload is proportional to  $m$ . The free parameters of the method are the stepsize  $h$  and the inner loop size  $m$ . Hence, in order to set the parameters so as to minimize the workload (i.e., optimize the complexity bound), we would like to (approximately) solve the optimization problem

$$\min m \quad \text{subject to} \quad 0 < h \leq \frac{1}{L}, \quad h < \frac{1}{4L\alpha(b)}, \quad c \text{ is fixed.}$$

Let  $(h_b^*, m_b^*)$  denote the optimal pair (we highlight the dependence on  $b$  as it will be useful). Note that if  $m_b^* \leq m_1^*/b$  for some  $b > 1$ , then mini-batching can help us reach the  $\epsilon$ -solution with smaller overall workload. The following theorem presents the formulas for  $h_b^*$  and  $m_b^*$ .

**Theorem 19.** Fix  $b$  and  $0 < c < 1$  and let

$$\tilde{h}_b \stackrel{\text{def}}{=} \sqrt{\left(\frac{1+c}{c\mu}\right)^2 + \frac{1}{4\mu\alpha(b)L}} - \frac{1+c}{c\mu}.$$

If  $\tilde{h}_b \leq \frac{1}{L}$ , then  $h_b^* = \tilde{h}_b$  and

$$m_b^* = \frac{2\kappa}{c} \left\{ \left(1 + \frac{1}{c}\right) 4\alpha(b) + \sqrt{\frac{4\alpha(b)}{\kappa} + \left(1 + \frac{1}{c}\right)^2 [4\alpha(b)]^2} \right\}, \quad (4.9)$$

where  $\kappa \stackrel{\text{def}}{=} \frac{L}{\mu}$  is the condition number. If  $\tilde{h}_b > \frac{1}{L}$ , then  $h_b^* = \frac{1}{L}$  and

$$m_b^* = \frac{\kappa + 4\alpha(b)}{c - 4\alpha(b)(1+c)}. \quad (4.10)$$

Note that if  $b = 1$ , we recover the optimal choice of parameters without mini-batching. Equation (4.9) suggests that as long as the condition  $\tilde{h}_b \leq \frac{1}{L}$  holds,  $m_b^*$  is decreasing at a rate faster than  $1/b$ . Hence, we can find the solution with less overall work when using a minibatch of size  $b$  than when using a minibatch of size 1.

#### 4.4.5 Convergence rate

In this section we study the total workload of mS2GD in the regime of small mini-batch sizes.

**Corollary 20.** Fix  $\epsilon \in (0, 1)$ , choose the number of outer iterations equal to

$$k = \lceil \log(1/\epsilon) \rceil,$$

and fix the target decrease in Theorem 19 to satisfy  $c = \epsilon^{1/k}$ . Further, pick a mini-batch size satisfying  $1 \leq b \leq 29$ , let the stepsize  $h$  be as in (4.33) and let  $m$  be as in (4.32). Then in order for mS2GD to find  $w^k$  satisfying (4.3), mS2GD needs at most

$$(n + 2bm_b)\lceil \log(1/\epsilon) \rceil \quad (4.11)$$

units of work, where  $bm_b = \mathcal{O}(\kappa)$ , which leads to the overall complexity of

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

units of work.

*Proof.* Available in Section 4.7.1. □

This result shows that as long as the mini-batch size is small enough, the total work performed by mS2GD is the same as in the  $b = 1$  case. If the  $b$  updates can be performed in parallel, then this leads to linear speedup.

#### 4.4.6 Comparison with Acc-Prox-SVRG

The Acc-Prox-SVRG [132] method of Nitanda, which was not available online before the first version of this work appeared on arXiv, incorporates both a mini-batch scheme and Nesterov's acceleration [127, 128]. The author claims that when  $b < \lceil b_0 \rceil$ , with the threshold  $b_0$  defined as  $\frac{8\sqrt{\kappa}n}{\sqrt{2p(n-1)+8\sqrt{\kappa}}}$ , the overall complexity of the method is

$$\mathcal{O}\left(\left(n + \frac{n-b}{n-1}\kappa\right)\log(1/\epsilon)\right);$$

and otherwise it is

$$\mathcal{O}((n + b\sqrt{\kappa})\log(1/\epsilon)).$$

This suggests that acceleration will only be realized when the mini-batch size is large, while for small  $b$ , Acc-Prox-SVRG achieves the same overall complexity,  $\mathcal{O}((n + \kappa) \log(1/\epsilon))$ , as mS2GD.

We will now take a closer look at the theoretical results given by Acc-Prox-SVRG and mS2GD, for each  $\epsilon \in (0, 1)$ . In particular, we shall numerically minimize the total work of mS2GD, i.e.,

$$(n + 2b[m_b]) \lceil \log(1/\epsilon) / \log(1/c) \rceil,$$

over  $c \in (0, 1)$  and  $h$  (compare this with (4.11)); and compare these results with similar fine-tuned quantities for Acc-Prox-SVRG.<sup>7</sup>

Fig. 4.1 illustrates these theoretical complexity bounds for both ill-conditioned and well-conditioned data. With small-enough mini-batch size  $b$ , mS2GD is better than Acc-Prox-SVRG. However, for a large mini-batch size  $b$ , the situation reverses because of the acceleration inherent in Acc-Prox-SVRG.<sup>8</sup> Plots with  $b = 64$  illustrate the cases where we cannot observe any differences between the methods.

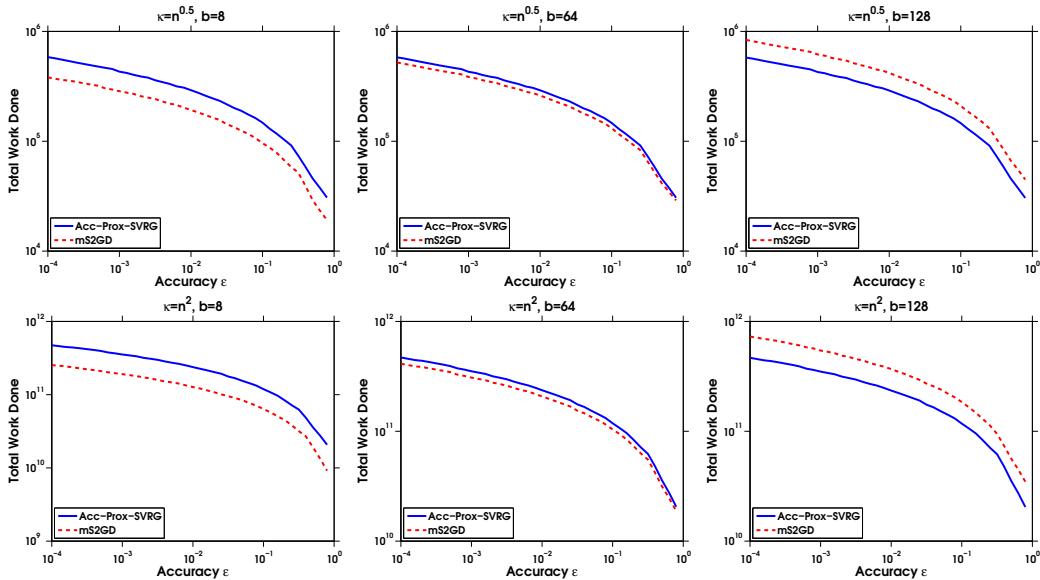


Figure 4.1: Complexity of Acc-Prox-SVRG and mS2GD in terms of total work done for  $n = 10,000$ , and small ( $\kappa = \sqrt{n}$ ; top row) and large ( $\kappa = n^2$ ; bottom row) condition number.

Note however that accelerated methods are very prone to error accumulation. Moreover, it is not clear that an efficient implementation of Acc-Prox-SVRG is possible for sparse data. As shall show in the next section, mS2GD allows for such an implementation.

## 4.5 Efficient implementation for sparse data

Let us make the following assumption about the structure of functions  $f_i$  in (4.2).

**Assumption 21.** *The functions  $f_i$  arise as the composition of a univariate smooth function  $\ell_i$  and an inner product with a datapoint/example  $a_i \in \mathbb{R}^d$ :  $f_i(w) = \ell(a_i^T w)$  for  $i = 1, \dots, n$ .*

Many functions of common practical interest satisfy this assumption including linear and logistic regression. Very often, especially for large scale datasets, the data are extremely sparse, i.e. the vectors  $\{a_i\}$  contains many zeros. Let us denote the number of non-zero coordinates of  $a_i$  by  $\omega_i = \|a_i\|_0 \leq d$  and the set of indexes corresponding to non-zero coordinates by  $\text{support}(a_i) = \{j : (a_i)_j \neq 0\}$ , where  $(a_i)_j$  denotes the  $j^{th}$  coordinate of vector  $a_i$ .

**Assumption 22.** *The regularization function  $R(w)$  is separable in coordinates of  $w$ .*

<sup>7</sup> $m_b$  is the best choice of  $m$  for Acc-Prox-SVRG and mS2GD, respectively. Meanwhile,  $h$  is within the safe upper bounds for both methods.

<sup>8</sup>We have experimented with different values for  $n, b$  and  $\kappa$ , and this result always holds.

This includes the most commonly used regularization functions as  $\frac{\lambda}{2}\|w\|^2$  or  $\lambda\|w\|_1$ .

Let us take a brief detour and look at the classical SGD algorithm with  $R(w) = 0$ . The update would be of the form

$$w^{k+1} \leftarrow w^k - h\nabla\ell_i(a_i^T w^k) a_i = w^k - h\nabla f_i(w^k). \quad (4.12)$$

If evaluation of the univariate function  $\nabla\ell_i$  takes  $O(1)$  amount of work, the computation of  $\nabla f_i$  will account for  $O(\omega_i)$  work. Then the update (4.12) would cost  $O(\omega_i)$  too, which implies that the classical SGD method can naturally benefit from sparsity of data.

Now, let us get back to the Algorithm 5. Even under the sparsity assumption and structural Assumption 21 the Algorithm 5 suggests that each inner iteration will cost  $O(\omega + d) \sim O(d)$  because  $g^k$  is in general fully dense and hence in Step 9 of Algorithm 5 we have to update all  $d$  coordinates.

However, in this Section, we will introduce and describe the implementation trick which is based on “lazy/delayed” updates. The main idea of this trick is not to perform Step 9 of Algorithm 5 for all coordinates, but only for coordinates  $j \in \cup_{i \in A^{kt}} \text{support}(a_i)$ . The efficient algorithm is described in Algorithm 6.

---

**Algorithm 6** ”Lazy” updates for mS2GD (these replace steps 6–10 in Algorithm 5)

---

```

1:  $\chi_j \leftarrow 0$  for  $j = 1, 2, \dots, d$ 
2: for  $t = 0$  to  $t^k - 1$  do
3:   Choose mini-batch  $A^{kt} \subseteq [n]$  of size  $b$ ; uniformly at random
4:   for  $i \in A^{kt}$  do
5:     for  $j \in \text{support}(a_i)$  do
6:        $y_j^{k,t} \leftarrow \text{prox}^{t-\chi_j}[y_j^{k,\chi_j}, g_j^k, R, h]$ 
7:        $\chi_j \leftarrow t$ 
8:     end for
9:   end for
10:   $y^{k,t+1} \leftarrow y^{k,t} - \frac{h}{b} \sum_{i \in A^{kt}} (\nabla\ell_i(a_i^T y^{k,t}) - \nabla\ell_i(a_i^T w^k)) a_i$ 
11: end for
12: for  $j = 1$  to  $d$  do
13:    $y_j^{k,t^k} \leftarrow \text{prox}^{t^k-\chi_j}[y_j^{k,\chi_j}, g_j^k, R, h]$ 
14: end for
```

---

To explain the main idea behind the lazy/delayed updates, consider that it happened that during the first  $\tau$  iterations of the inner loop, the value of the first coordinate in all datapoints which we have used was 0. Then given the values of  $y_1^{k,0}$  and  $g_1^k$  we can compute the true value of  $y_1^{k,t}$  easily. We just need to apply the prox operator  $\tau$  times, i.e.  $y_1^{k,\tau} = \text{prox}_1^\tau[y^{k,0}, g^k, R, h]$ , where the function  $\text{prox}_1^\tau$  is described in Algorithm 7.

---

**Algorithm 7**  $\text{prox}_j^\tau[y, g, R, h]$

---

```

 $\tilde{y}^0 = y$ 
for  $s = 1, 2, \dots, \tau$  do
   $\tilde{y}^s \leftarrow \text{prox}_{hR}(\tilde{y}^{s-1} - hg)$ 
end for
return  $\tilde{y}_j^\tau$ 
```

---

The vector  $\chi$  in Algorithm 6 is enabling us to keep track of the iteration when corresponding coordinate of  $y$  was updated for the last time. E.g. if in iteration  $t$  we will be updating the 1<sup>st</sup> coordinate for the first time,  $\chi_1 = 0$  and after we compute and update the true value of  $y_1$ , its value will be set to  $\chi_1 = t$ . Lines 5–8 in Algorithm 6 make sure that the coordinates of  $y^{k,t}$  which will be read and used afterwards are up-to-date. At the end of the inner loop, we will update all coordinates of  $y$  to the most recent value (lines 12–14). Therefore, those lines make sure that the  $y^{k,t^k}$  of Algorithms 5 and 6 will be the same.

However, one could claim that we are not saving any work, as when needed, we still have to compute the proximal operator many times. Although this can be true for a general function  $R$ , for particular cases  $R(w) = \frac{\lambda}{2}\|w\|^2$  and  $R(w) = \lambda\|w\|_1^2$ , we provide following Lemmas which give a closed form expressions for the  $\text{prox}_j^\tau$  operator.

**Lemma 23** (Proximal Lazy Updates with  $\ell_2$ -Regularizer). *If  $R(w) = \frac{\lambda}{2}\|w\|^2$  with  $\lambda > 0$  then*

$$\text{prox}_j^\tau[y, g, R, h] = \beta^\tau y_j - \frac{h\beta}{1-\beta} (1 - \beta^\tau) g_j,$$

where  $\beta \stackrel{\text{def}}{=} 1/(1 + \lambda h)$ .

**Lemma 24** (Proximal Lazy Updates with  $\ell_1$ -Regularizer). *Assume that  $R(w) = \lambda\|w\|_1$  with  $\lambda > 0$ . Let us define  $M$  and  $m$  as follows,*

$$M = [\lambda + g_j]h, \quad m = -[\lambda - g_j]h,$$

and let  $[\cdot]_+ \stackrel{\text{def}}{=} \max\{\cdot, 0\}$ . Then the value of  $\text{prox}_j^\tau[y, g, R, h]$  can be expressed based on one of the 3 situations described below:

1. If  $g_j \geq \lambda$ , then by letting  $p \stackrel{\text{def}}{=} \lfloor \frac{y_j}{M} \rfloor$ , the operator can be defined as

$$\text{prox}_j^\tau[y, g, R, h] = \begin{cases} y_j - \tau M, & \text{if } p \geq \tau, \\ \min\{y_j - [p]_+ M, m\} - (\tau - [p]_+)m, & \text{if } p < \tau. \end{cases}$$

2. If  $-\lambda < g_j < \lambda$ , then the operator can be defined as

$$\text{prox}_j^\tau[y, g, R, h] = \begin{cases} \max\{y_j - \tau M, 0\}, & \text{if } y_j \geq 0, \\ \min\{y_j - \tau m, 0\}, & \text{if } y_j < 0. \end{cases}$$

3. If  $g_j \leq -\lambda$ , then by letting  $q \stackrel{\text{def}}{=} \lfloor \frac{y_j}{m} \rfloor$ , the operator can be defined as

$$\begin{aligned} \text{prox}_j^\tau[y, g, R, h] &= \\ &= \begin{cases} y_j - \tau m, & \text{if } q \geq \tau, \\ \max\{y_j - [q]_+ m, M\} - (\tau - [q]_+)M, & \text{if } q < \tau. \end{cases} \end{aligned}$$

The proofs of Lemmas 23 and 24 are available in Section 4.7.2.

*Remark:* Upon completion of this work, we learned that similar ideas of lazy updates were proposed in [91] and [28] for online learning and multinomial logistic regression, respectively. However, our method can be seen as a more general result applied to a stochastic gradient method and its variants under Assumptions 21 and 22.

## 4.6 Experiments

In this section we perform numerical experiments to illustrate the properties and performance of our algorithm. In Section 4.6.1 we study the total workload and parallelization speedup of mS2GD as a function of the mini-batch size  $b$ . In Section 4.6.2 we compare mS2GD with several other algorithms. Finally, in Section 4.6.3 we briefly illustrate that our method can be efficiently applied to a deblurring problem.

In Sections 4.6.1 and 4.6.2 we conduct experiments with  $R(w) = \frac{\lambda}{2}\|w\|^2$  and  $f$  of the form (4.2), where  $f_i$  is the logistic loss function:

$$f_i(w) = \log[1 + \exp(-b_i a_i^T w)]. \quad (4.13)$$

These functions are often used in machine learning, with  $(a_i, b_i) \in \mathbb{R}^d \times \{+1, -1\}$ ,  $i = 1, \dots, n$ , being a training dataset of example-label pairs. The resulting optimization problem (4.1)+(4.2)

takes the form

$$P(w) = \frac{1}{n} \sum_{i=1}^n f_i(w) + \frac{\lambda}{2} \|w\|^2, \quad (4.14)$$

and is used in machine learning for binary classification. In these sections we have performed experiments on four publicly available binary classification datasets, namely *rcv1*, *news20*, *covtype*<sup>9</sup> and *astro-ph*<sup>10</sup>.

In the logistic regression problem, the Lipschitz constant of function  $\nabla f_i$  is equal to  $L_i = \|a_i\|^2/4$ . Our analysis assumes (Assumption 16) the same constant  $L$  for all functions. Hence, we have  $L = \max_{i \in [n]} L_i$ . We set the regularization parameter  $\lambda = \frac{1}{n}$  in our experiments, resulting in the problem having the condition number  $\kappa = \frac{L}{\mu} = \mathcal{O}(n)$ . In Table 4.1 we summarize the four datasets, including the sizes  $n$ , dimensions  $d$ , their sparsity levels as a proportion of nonzero elements, and the Lipschitz constants  $L$ .

Dataset	$n$	$d$	Sparsity	$L$
<i>rcv1</i>	20,242	47,236	0.1568%	0.2500
<i>news20</i>	19,996	1,355,191	0.0336%	0.2500
<i>covtype</i>	581,012	54	22.1212%	1.9040
<i>astro-ph</i>	62,369	99,757	0.0767%	0.2500

Table 4.1: Summary of datasets used for experiments.

#### 4.6.1 Speedup of mS2GD

Mini-batches allow mS2GD to be accelerated on a computer with a parallel processor. In Section 4.4.4, we have shown in that up to some threshold mini-batch size, the total workload of mS2GD remains unchanged. Figure 4.2 compares the best performance of mS2GD used with various mini-batch sizes on datasets *rcv1* and *astro-ph*. An effective pass (through the data) corresponds to  $n$  units of work. Hence, the evaluation of a gradient of  $f$  counts as one effective pass. In both cases, by increasing the mini-batch size to  $b = 2, 4, 8$ , the performance of mS2GD is the same or better than that of S2GD ( $b = 1$ ) without any parallelism.

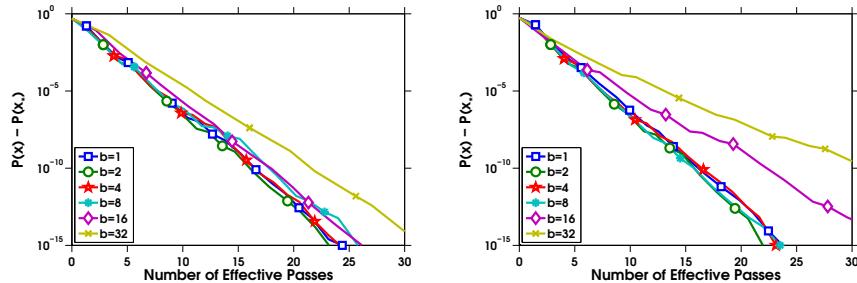


Figure 4.2: Comparison of mS2GD with different mini-batch sizes on *rcv1* (left) and *astro-ph* (right).

Although for larger mini-batch sizes mS2GD would be obviously worse, the results are still promising with parallelism. In Figure 4.3, we show the ideal speedup—one that would be achievable if we could always evaluate the  $b$  gradients in parallel in exactly the same amount of time as it would take to evaluate a single gradient.<sup>11</sup>

#### 4.6.2 mS2GD vs other algorithms

In this part, we implemented the following algorithms to conduct a numerical comparison:

- 1) **SGDcon**: Proximal stochastic gradient descent method with a constant step-size which gave

<sup>9</sup> *rcv1*, *covtype* and *news20* are available at <http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>.

<sup>10</sup> Available at <http://users.cecs.anu.edu.au/~xzhang/data/>.

<sup>11</sup> In practice, it is impossible to ensure that the times of evaluating different component gradients are the same.

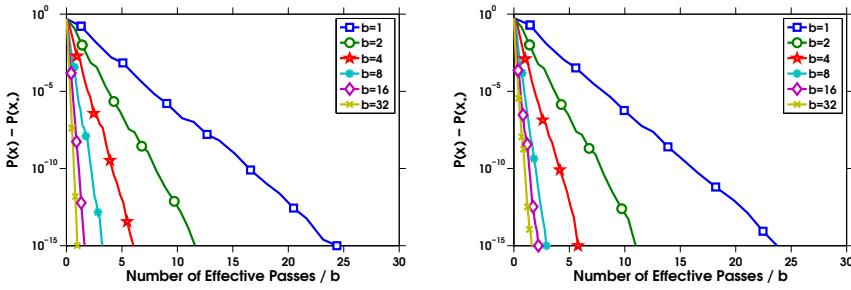


Figure 4.3: Parallelism speedup for *rcv1* (left) and *astro-ph* (right) in theory (unachievable in practice).

the best performance in hindsight.

- 2) **SGD+**: Proximal stochastic gradient descent with variable step-size  $h = h_0/(k+1)$ , where  $k$  is the number of effective passes, and  $h_0$  is some initial constant step-size.
- 3) **FISTA**: Fast iterative shrinkage-thresholding algorithm proposed in [11].
- 4) **SAG**: Proximal version of the stochastic average gradient algorithm [156]. Instead of using  $h = 1/16L$ , which is analyzed in the reference, we used a constant step size.
- 5) **S2GD**: Semi-stochastic gradient descent method proposed in [89]. We applied proximal setting to the algorithm and used a constant stepsize.
- 6) **mS2GD**: mS2GD with mini-batch size  $b = 8$ . Although a safe step-size is given in our theoretical analyses in Theorem 18, we ignored the bound, and used a constant step size.

In all cases, unless otherwise stated, we have used the best constant stepsizes in hindsight.

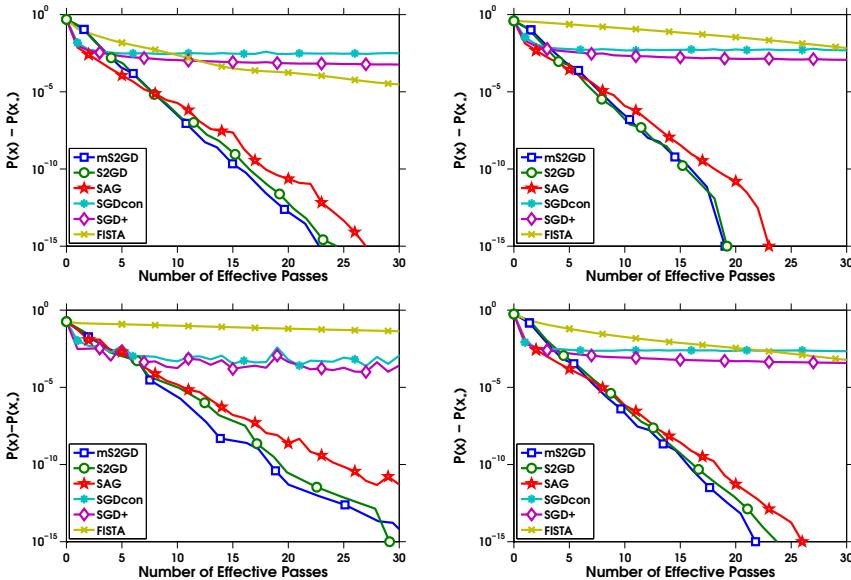


Figure 4.4: Comparison of several algorithms on four datasets: *rcv1* (top left), *news20* (top right), *covtype* (bottom left) and *astro-ph* (bottom right). We have used mS2GD with  $b = 8$ .

Figure 4.4 demonstrates the superiority of mS2GD over other algorithms in the test pool on the four datasets described above. For mS2GD, the best choices of parameters with  $b = 8$  are given in Table 4.2.

Parameter	<i>rcv1</i>	<i>news20</i>	<i>covtype</i>	<i>astro-ph</i>
$m$	$0.11n$	$0.10n$	$0.26n$	$0.08n$
$h$	$5.5/L$	$6/L$	$4.5/L$	$6.5/L$

Table 4.2: Best choices of parameters in mS2GD.

### 4.6.3 Image deblurring

In this section we utilize the Regularization Toolbox [72].<sup>12</sup> We use the *blur* function available therein to obtain the original image and generate a blurred image (we choose following values of parameters for blur function:  $N = 256$ , band=9, sigma=10). The purpose of the blur function is to generate a test problem with an atmospheric turbulence blur. In addition, an additive Gaussian white noise with stand deviation of  $10^{-3}$  is added to the blurred image. This forms our testing image as a vector  $b$ . The image dimension of the test image is  $256 \times 256$ , which means that  $n = d = 65,536$ . We would not expect our method to work particularly well on this problem since mS2GD works best when  $d \ll n$ . However, as we shall see, the method's performance is on a par with the performance of the best methods in our test pool.

Our goal is to reconstruct (deblur) the original image  $x$  by solving a LASSO problem:  $\min_w \|Aw - b\|_2^2 + \lambda\|w\|_1$ . We have chosen  $\lambda = 10^{-4}$ . In our implementation, we normalized the objective function by  $n$ , and hence our objective value being optimized is in fact  $\min_w \frac{1}{n}\|Aw - b\|_2^2 + \lambda\|w\|_1$ , where  $\lambda = \frac{10^{-4}}{n}$ , similarly as was done in [11].

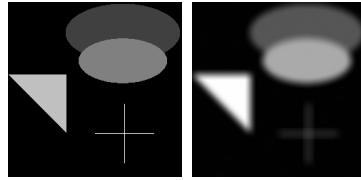


Figure 4.5: Original (left) and blurred & noisy (right) test image.

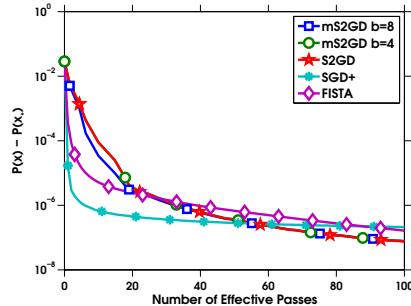


Figure 4.6: Comparison of several algorithms for the deblurring problem.

Figure (4.5) shows the original test image (left) and a blurred image with added Gaussian noise (right). Figure 4.6 compares the mS2GD algorithm with SGD+, S2GD and FISTA. We run all algorithms for 100 epochs and plot the error. The plot suggests that SGD+ decreases the objective function very rapidly at beginning, but slows down after 10-20 epochs.

Finally, Fig. 4.7 shows the reconstructed image after  $T = 20, 60, 100$  epochs.

## 4.7 Technical Results

We first state technical results used in this chapter, followed by proofs deferred from the main body.

**Lemma 25** (Lemma 3.6 in [187]). *Let  $R$  be a closed convex function on  $\mathbb{R}^d$  and  $x, y \in \text{dom}(R)$ , then  $\|\text{prox}_R(x) - \text{prox}_R(y)\| \leq \|x - y\|$ .*

Note that non-expansiveness of the proximal operator is a standard result in optimization literature [118, 154].

---

<sup>12</sup>Regularization Toolbox available for Matlab can be obtained from <http://www.imm.dtu.dk/~pcha/Regutools/>.

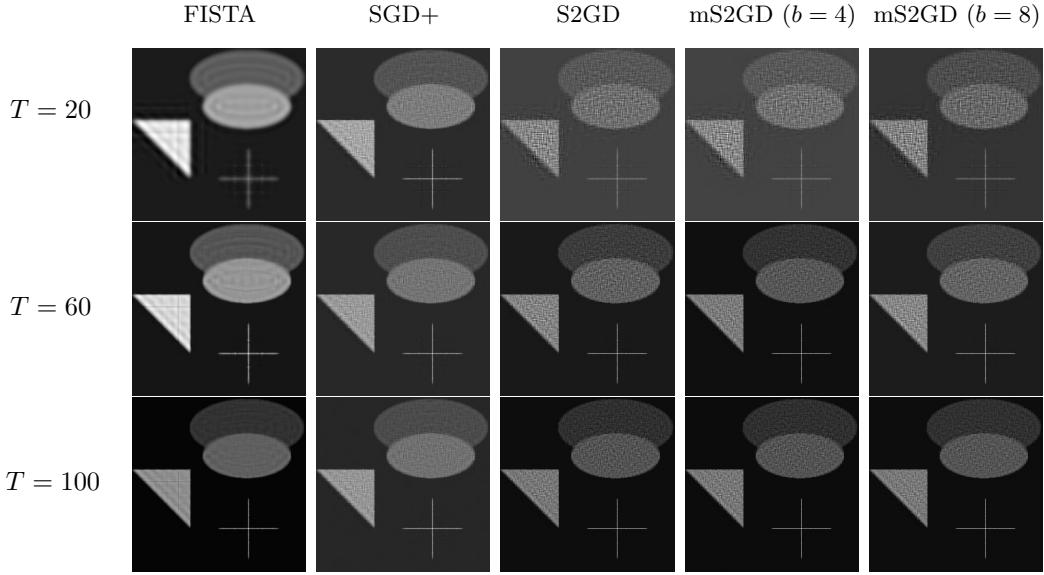


Figure 4.7: Reconstruction of the test image from Figure 4.5 via FISTA, SGD+, S2GD and mS2GD after  $T = 20, 60, 100$  epochs (one epoch corresponds to work equivalent to the computation of one gradient.)

**Lemma 26.** Let  $\{\xi_i\}_{i=1}^n$  be vectors in  $\mathbb{R}^d$  and  $\bar{\xi} \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n \xi_i \in \mathbb{R}^d$ . Let  $\hat{S}$  be a random subset of  $[n]$  of size  $\tau$ , chosen uniformly at random from all subsets of this cardinality. Taking expectation with respect to  $\hat{S}$ , we have

$$\mathbb{E} \left[ \left\| \frac{1}{\tau} \sum_{i \in \hat{S}} \xi_i - \bar{\xi} \right\|^2 \right] \leq \frac{1}{n\tau} \frac{n-\tau}{(n-1)} \sum_{i=1}^n \|\xi_i\|^2. \quad (4.15)$$

Following from the proof of Corollary 3.5 in [187], by applying Lemma 26 with  $\xi_i := \nabla f_i(y^{k,t}) - \nabla f_i(w^k)$ , we have the bound for variance as follows.

**Theorem 27** (Bounding Variance). Let  $\alpha(b) \stackrel{\text{def}}{=} \frac{n-b}{b(n-1)}$ . Considering the definition of  $G^{k,t}$  in Algorithm 5, conditioned on  $y^{k,t}$ , we have  $\mathbb{E}[G^{k,t}] = \nabla f(y^{k,t})$  and the variance satisfies,

$$\begin{aligned} & \mathbb{E} [\|G^{k,t} - \nabla f(y^{k,t})\|^2] \\ & \leq 4L\alpha(b)[P(y^{k,t}) - P(w^*) + P(w^k) - P(w^*)]. \end{aligned} \quad (4.16)$$

### 4.7.1 Proofs

#### Proof of Lemma 26

As in the statement of the lemma, by  $\mathbb{E}[\cdot]$  we denote expectation with respect to the random set  $\hat{S}$ . First, note that

$$\begin{aligned} \eta & \stackrel{\text{def}}{=} \mathbb{E} \left[ \left\| \frac{1}{\tau} \sum_{i \in \hat{S}} \xi_i - \bar{\xi} \right\|^2 \right] = \mathbb{E} \left[ \frac{1}{\tau^2} \left\| \sum_{i \in \hat{S}} \xi_i \right\|^2 \right] - \|\bar{\xi}\|^2 \\ & = \frac{1}{\tau^2} \mathbb{E} \left[ \sum_{i \in \hat{S}} \sum_{j \in \hat{S}} \xi_i^T \xi_j \right] - \|\bar{\xi}\|^2. \end{aligned}$$

If we let  $C \stackrel{\text{def}}{=} \|\bar{\xi}\|^2 = \frac{1}{n^2} \left( \sum_{i,j} \xi_i^T \xi_j \right)$ , we can thus write

$$\begin{aligned} \eta &= \frac{1}{\tau^2} \left( \frac{\tau(\tau-1)}{n(n-1)} \sum_{i \neq j} \xi_i^T \xi_j + \frac{\tau}{n} \sum_{i=1}^n \xi_i^T \xi_i \right) - C \\ &= \frac{1}{\tau^2} \left( \frac{\tau(\tau-1)}{n(n-1)} \sum_{i,j} \xi_i^T \xi_j + \left( \frac{\tau}{n} - \frac{\tau(\tau-1)}{n(n-1)} \right) \sum_{i=1}^n \xi_i^T \xi_i \right) - C \\ &= \frac{1}{n\tau} \left[ - \left( -\frac{(\tau-1)}{(n-1)} + \frac{\tau}{n} \right) \sum_{i,j} \xi_i^T \xi_j + \frac{n-\tau}{n-1} \sum_{i=1}^n \xi_i^T \xi_i \right] \\ &= \frac{1}{n\tau(n-1)} \left[ \sum_{i=1}^n \xi_i^T \xi_i - \frac{1}{n} \sum_{i,j} \xi_i^T \xi_j \right] \leq \frac{1}{n\tau(n-1)} \sum_{i=1}^n \|\xi_i\|^2, \end{aligned}$$

where in the last step we have used the bound  $\frac{1}{n} \sum_{i,j} \xi_i^T \xi_j = n \left\| \sum_{i=1}^n \frac{1}{n} \xi_i \right\|^2 \geq 0$ .

### Proof of Theorem 18

The proof is following the core steps in [187]. For convenience, let us define the stochastic gradient mapping

$$d^{k,t} = \frac{1}{h} (y^{k,t} - y^{k,t+1}) = \frac{1}{h} (y^{k,t} - \text{prox}_{hR}(y^{k,t} - hG^{k,t})),$$

then the iterate update can be written as  $y^{k,t+1} = y^{k,t} - hd^{k,t}$ . Let us estimate the change of  $\|y^{k,t+1} - w^*\|$ . It holds that

$$\begin{aligned} \|y^{k,t+1} - w^*\|^2 &= \|y^{k,t} - hd^{k,t} - w^*\|^2 \\ &= \|y^{k,t} - w^*\|^2 - 2h \langle d^{k,t}, y^{k,t-1} - w^* \rangle + h^2 \|d^{k,t}\|^2. \end{aligned} \quad (4.17)$$

Applying Lemma 3.7 in [187] (this is why we need to assume that  $h \leq 1/L$ ) with  $x = y^{k,t}$ ,  $v = G^{k,t}$ ,  $x^+ = y^{k,t+1}$ ,  $g = d^{k,t}$ ,  $y = x^*$  and  $\Delta = \Delta^{k,t} = G^{k,t} - \nabla f(y^{k,t})$ , we get

$$\begin{aligned} - \langle d^{k,t}, y^{k,t} - w^* \rangle + \frac{h}{2} \|d^{k,t}\|^2 &\leq P(w^*) - P(y^{k,t+1}) - \langle \Delta^{k,t}, y^{k,t+1} - w^* \rangle \\ &\quad - \frac{\mu_F}{2} \|y^{k,t} - w^*\|^2 - \frac{\mu_R}{2} \|y^{k,t+1} - w^*\|^2, \end{aligned} \quad (4.18)$$

and therefore,

$$\begin{aligned} \|y^{k,t+1} - w^*\|^2 &\stackrel{(4.17),(4.18)}{\leq} 2h (P(w^*) - P(y^{k,t+1}) - \langle \Delta^{k,t}, y^{k,t+1} - w^* \rangle) + \|y^{k,t} - w^*\|^2 \\ &= \|y^{k,t} - w^*\|^2 - 2h \langle \Delta^{k,t}, y^{k,t+1} - w^* \rangle - 2h [P(y^{k,t+1}) - P(w^*)]. \end{aligned} \quad (4.19)$$

In order to bound  $-\langle \Delta^{k,t}, y^{k,t+1} - w^* \rangle$ , let us define the proximal full gradient update as<sup>13</sup>  $\bar{y}^{k,t+1} = \text{prox}_{hR}(y^{k,t} - h\nabla f(y^{k,t}))$ . We get

$$\begin{aligned} - \langle \Delta^{k,t}, y^{k,t+1} - w^* \rangle &= - \langle \Delta^{k,t}, y^{k,t+1} - \bar{y}^{k,t+1} \rangle - \langle \Delta^{k,t}, \bar{y}^{k,t+1} - w^* \rangle \\ &= - \langle \Delta^{k,t}, \bar{y}^{k,t+1} - w^* \rangle \\ &\quad - \langle \Delta^{k,t}, \text{prox}_{hR}(y^{k,t} - hG^{k,t}) - \text{prox}_{hR}(y^{k,t-1} - h\nabla f(y^{k,t-1})) \rangle \end{aligned}$$

---

<sup>13</sup>Note that this quantity is never computed during the algorithm. We can use it in the analysis nevertheless.

Using Cauchy-Schwarz and Lemma 25, we conclude that

$$\begin{aligned} -\langle \Delta^{k,t}, y^{k,t+1} - w^* \rangle &\leq \|\Delta^{k,t}\| \| (y^{k,t} - hG^{k,t}) - (y^{k,t} - h\nabla f(y^{k,t})) \| \\ &\quad - \langle \Delta^{k,t}, \bar{y}^{k,t+1} - w^* \rangle \\ &= h\|\Delta^{k,t}\|^2 - \langle \Delta^{k,t}, \bar{y}^{k,t+1} - w^* \rangle. \end{aligned} \quad (4.20)$$

Further, we obtain

$$\begin{aligned} \|y^{k,t+1} - w^*\|^2 &\stackrel{(4.20),(4.19)}{\leq} \|y^{k,t} - w^*\|^2 \\ &\quad + 2h(h\|\Delta^{k,t}\|^2 - \langle \Delta^{k,t}, \bar{y}^{k,t+1} - w^* \rangle - [P(y^{k,t+1}) - P(w^*)]). \end{aligned}$$

By taking expectation, conditioned on  $y^{k,t}$ <sup>14</sup> we obtain

$$\mathbb{E} [\|y^{k,t+1} - w^*\|^2] \leq \|y^{k,t} - w^*\|^2 + 2h(h\mathbb{E} [\|\Delta^{k,t}\|^2] - \mathbb{E} [P(y^{k,t+1}) - P(w^*)]), \quad (4.21)$$

where we have used that  $\mathbb{E} [\Delta^{k,t}] = \mathbb{E} [G^{k,t}] - \nabla f(y^{k,t}) = 0$ , thus  $\mathbb{E} [-\langle \Delta^{k,t}, \bar{y}^{k,t+1} - w^* \rangle] = 0$ <sup>15</sup>. Now, if we substitute (4.16) into (4.21) and decrease index  $t$  by 1, we obtain

$$\begin{aligned} \mathbb{E} [\|y^{k,t} - w^*\|^2] &\leq \|y^{k,t-1} - w^*\|^2 - 2h\mathbb{E} [P(y^{k,t}) - P(w^*)] \\ &\quad + \theta[P(y^{k,t-1}) - P(w^*) + P(w^k) - P(w^*)], \end{aligned} \quad (4.22)$$

where  $\theta \stackrel{\text{def}}{=} 8Lh^2\alpha(b)$  and  $\alpha(b) = \frac{n-b}{b(n-1)}$ . Note that (4.22) is equivalent to

$$\begin{aligned} \mathbb{E} [\|y^{k,t} - w^*\|^2] + 2h(\mathbb{E} [P(y^{k,t}) - P(w^*)]) &\leq \|y^{k,t-1} - w^*\|^2 \\ &\quad + \theta(P(y^{k,t-1}) - P(w^*) + P(w^k) - P(w^*)). \end{aligned} \quad (4.23)$$

Now, by the definition of  $w^k$  in Algorithm 5 we have that

$$\mathbb{E} [P(w^{k+1})] = \frac{1}{m} \sum_{t=1}^m \mathbb{E} [P(y^{k,t})]. \quad (4.24)$$

By summing (4.23) for  $1 \leq t \leq m$ , we get on the left hand side

$$LHS = \sum_{t=1}^m \mathbb{E} [\|y^{k,t} - w^*\|^2] + 2h\mathbb{E} [P(y^{k,t}) - P(w^*)] \quad (4.25)$$

and for the right hand side we have:

$$\begin{aligned} RHS &= \sum_{t=1}^m \{ \mathbb{E} [\|y^{k,t-1} - w^*\|^2] + \theta\mathbb{E} [P(y^{k,t-1}) - P(w^*) + P(w^k) - P(w^*)] \} \\ &\leq \sum_{t=0}^{m-1} \mathbb{E} [\|y^{k,t} - w^*\|^2] + \theta \sum_{t=0}^m \mathbb{E} [P(y^{k,t}) - P(w^*)] + \theta\mathbb{E} [P(w^k) - P(w^*)] m. \end{aligned} \quad (4.26)$$

<sup>14</sup>For simplicity, we omit the  $\mathbb{E} [\cdot | y^{k,t}]$  notation in further analysis

<sup>15</sup> $\bar{y}^{k,t+1}$  is constant, conditioned on  $y^{k,t}$

Combining (4.25) and (4.26) and using the fact that  $LHS \leq RHS$ , we have

$$\begin{aligned} \mathbb{E} [\|y^{k,m} - w^*\|^2] + 2h \sum_{t=1}^m \mathbb{E} [P(y^{k,t}) - P(w^*)] &\leq \mathbb{E} [\|y^{k,0} - w^*\|^2] + \theta \mathbb{E} [P(w^k) - P(w^*)] m \\ &\quad + \theta \sum_{t=0}^m \mathbb{E} [P(y^{k,t}) - P(w^*)]. \end{aligned}$$

Now, using (4.24), we obtain

$$\begin{aligned} \mathbb{E} [\|y^{k,m} - w^*\|^2] + 2hm \mathbb{E} [P(w^{k+1}) - P(w^*)] &\leq \mathbb{E} [\|y^{k,0} - w^*\|^2] + \theta m \mathbb{E} [P(w^k) - P(w^*)] \\ &\quad + \theta m \mathbb{E} [P(w^{k+1}) - P(w^*)] \\ &\quad + \theta \mathbb{E} [P(y^{k,0}) - P(w^*)]. \end{aligned} \quad (4.27)$$

Strong convexity (4.7) and optimality of  $w^*$  imply that  $0 \in \partial P(w^*)$ , and hence for all  $w \in \mathbb{R}^d$  we have

$$\|w - w^*\|^2 \leq \frac{2}{\mu} [P(w) - P(w^*)]. \quad (4.28)$$

Since  $\mathbb{E} [\|y^{k,m} - w^*\|^2] \geq 0$  and  $y^{k,0} = w^k$ , by combining (4.28) and (4.27) we get

$$m(2h - \theta) \mathbb{E} [P(w^{k+1}) - P(w^*)] \leq (P(w^k) - P(w^*)) \left( \frac{2}{\mu} + \theta(m+1) \right).$$

Notice that in view of our assumption on  $h$  and definition of  $\theta$ , we have  $2h > \theta$ , and hence

$$\mathbb{E} [P(w^{k+1}) - P(w^*)] \leq c [P(w^k) - P(w^*)],$$

where  $c = \frac{2}{m\mu(2h-\theta)} + \frac{\theta(m+1)}{m(2h-\theta)}$ . Applying the above linear convergence relation recursively with chained expectations, we finally obtain  $\mathbb{E} [P(w^k) - P(w^*)] \leq c^k [P(w^0) - P(w^*)]$ .

### Proof of Theorem 19

Clearly, if we choose some value of  $h$  then the value of  $m$  will be determined from (4.8) (i.e. we need to choose  $m$  such that we will get desired rate). Therefore,  $m$  as a function of  $h$  obtained from (4.8) is

$$m(h) = \frac{1 + 4\alpha(b)h^2L\mu}{h\mu(c - 4\alpha(b)hL(c+1))}. \quad (4.29)$$

Now, we can observe that the nominator is always positive and the denominator is positive only if  $c > 4\alpha(b)hL(c+1)$ , which implies  $\frac{1}{4\alpha(b)L} \cdot \frac{c}{c+1} > h$  (note that  $\frac{c}{c+1} \in [0, \frac{1}{2}]$ ). Observe that this condition is stronger than the one in the assumption of Theorem 18. It is easy to verify that

$$\lim_{h \searrow 0} m(h) = +\infty, \quad \lim_{h \nearrow \frac{1}{4\alpha(b)L} \cdot \frac{c}{c+1}} m(h) = +\infty.$$

Also note that  $m(h)$  is differentiable (and continuous) at any  $h \in \left(0, \frac{1}{4\alpha(b)L} \cdot \frac{c}{c+1}\right) =: I_h$ . The derivative of  $m$  is given by  $m'(h) = \frac{-c+4\alpha(b)hL(2+(2+h\mu)c)}{h^2\mu(c-4\alpha(b)hL(1+c))^2}$ . Observe that  $m'(h)$  is defined and continuous for any  $h \in I_h$ . Therefore there have to be some stationary points (and in case that there is just one  $I_h$ ) it will be the global minimum on  $I_h$ . The first order condition gives

$$\begin{aligned} \tilde{h}_b &= \frac{-2\alpha(b)L(1+c) + \sqrt{\alpha(b)L(\mu c^2 + 4\alpha(b)L(1+c)^2)}}{2\alpha(b)L\mu c} \\ &= \sqrt{\frac{1}{4\alpha(b)L\mu} + \frac{(1+c)^2}{\mu^2 c^2}} - \frac{1+c}{\mu c}. \end{aligned} \quad (4.30)$$

If this  $\tilde{h}_b \in I_h$  and also  $\tilde{h}_b \leq \frac{1}{L}$  then this is the optimal choice and plugging (4.30) into (4.29) gives us (4.9).

**Claim #1** It always holds that  $\tilde{h}_b \in I_h$ . We just need to verify that

$$\sqrt{\frac{1}{4\alpha(b)L\mu} + \frac{(1+c)^2}{\mu^2 c^2}} - \frac{1+c}{\mu c} < \frac{1}{4\alpha(b)L} \cdot \frac{c}{c+1},$$

which is equivalent to  $\mu c^2 + 4\alpha(b)L(1+c)^2 > 2(1+c)\sqrt{\alpha(b)L(\mu c^2 + 4\alpha(b)L(1+c)^2)}$ . Because both sides are positive, we can square them to obtain the equivalent condition

$$\mu c^2(\mu c^2 + 4\alpha(b)L(1+c)^2) > 0.$$

**Claim #2** If  $\tilde{h}_b > \frac{1}{L}$  then  $h_b^* = \frac{1}{L}$ . The only detail which needs to be verified is that the denominator of (4.10) is positive (or equivalently we want to show that  $c > 4\alpha(b)(1+c)$ ). To see that, we need to realize that in that case we have  $\frac{1}{L} \leq \tilde{h}_b \leq \frac{1}{4\alpha(b)L} \cdot \frac{c}{c+1}$ , which implies that  $4\alpha(b)(1+c) < c$ .

## Proof of Corollary 20

By substituting definition of  $\tilde{h}_b$  in Theorem 19, we get

$$\tilde{h}_b < \frac{1}{L} \iff b < b_0 \stackrel{\text{def}}{=} \frac{8cn\kappa + 8n\kappa + 4cn}{cn\kappa + (7c+8)\kappa + 4c}, \quad (4.31)$$

where  $\kappa = L/\mu$ . Hence, it follows that if  $b < \lceil b_0 \rceil$ , then  $h_b = \tilde{h}_b$  and  $m_b$  is defined in (4.9); otherwise,  $h_b = \frac{1}{L}$  and  $m_b$  is defined in (4.10). Let  $e$  be the base of the natural logarithm. By selecting  $b_0 = \frac{8n\kappa + 8en\kappa + 4n}{n\kappa + (7+8e)\kappa + 4}$ , choosing mini-batch size  $b < \lceil b_0 \rceil$ , and running the inner loop of mS2GD for

$$m_b = \left\lceil 8e\alpha(b)\kappa \left( e + 1 + \sqrt{\frac{1}{4\alpha(b)\kappa} + (1+e)^2} \right) \right\rceil \quad (4.32)$$

iterations with constant stepsize

$$h_b = \sqrt{\left(\frac{1+e}{\mu}\right)^2 + \frac{1}{4\mu\alpha(b)L}} - \frac{1+e}{\mu}, \quad (4.33)$$

we can achieve a convergence rate

$$c \stackrel{(4.8)}{=} \frac{1}{m_b h_b \mu (1 - 4h_b L\alpha(b))} + \frac{4h_b L\alpha(b) (m_b + 1)}{m_b (1 - 4h_b L\alpha(b))} \stackrel{(4.32),(4.33)}{=} \frac{1}{e}. \quad (4.34)$$

Since  $k = \lceil \log(1/\epsilon) \rceil \geq \log(1/\epsilon)$  if and only if  $e^k \geq 1/\epsilon$  if and only if  $e^{-k} \leq \epsilon$ , we can conclude that  $c^k \stackrel{(4.34)}{=} (e^{-1})^k = e^{-k} \leq \epsilon$ . Therefore, running mS2GD for  $k$  outer iterations achieves  $\epsilon$ -accuracy solution defined in (4.3). Moreover, since in general  $\kappa \gg e, n \gg e$ , it can be concluded that

$$b_0 \stackrel{(4.31)}{=} \frac{8(1+e)n\kappa + 4n}{n\kappa + (7+8e)\kappa + 4} \approx 8(e+1) \approx 29.75,$$

then with the definition  $\alpha(b) = \frac{(n-b)}{b(n-1)}$ , we derive

$$\begin{aligned} b m_b &\stackrel{(4.32)}{=} \left\lceil 8e\kappa \frac{(n-b)}{(n-1)} \left( e + 1 + \sqrt{\frac{1}{4\alpha(b)\kappa} + (1+e)^2} \right) \right\rceil \\ &\stackrel{1 \leq b < 30}{\leq} \left\lceil 8e\kappa \left( (e+1) + \sqrt{\frac{b}{4\kappa} + (1+e)^2} \right) \right\rceil = \mathcal{O}(\kappa), \end{aligned}$$

so from (4.11), the total complexity can be translated to  $\mathcal{O}((n + \kappa) \log(1/\epsilon))$ . This result shows that we can reach efficient speedup by mini-batching as long as the mini-batch size is smaller than some threshold  $b_0 \approx 29.75$ , which finishes the proof for Corollary 20.

#### 4.7.2 Proximal lazy updates for $\ell_1$ and $\ell_2$ -regularizers

##### Proof of Lemma 23

For any  $s \in \{1, 2, \dots, \tau\}$  we have  $\tilde{y}^s = \text{prox}_{hR}(\tilde{y}^{s-1} - hg) = \beta(\tilde{y}^{s-1} - hg)$ , where  $\beta \stackrel{\text{def}}{=} 1/(1+\lambda h)$ . Therefore,

$$\tilde{y}^\tau = \beta^\tau \tilde{y}^0 - h \left( \sum_{j=1}^{\tau} \beta^j \right) g = \beta^\tau y - \frac{h\beta}{1-\beta} [1 - \beta^\tau] g.$$

##### Proof of Lemma 24

*Proof.* For any  $s \in \{1, 2, \dots, \tau\}$  and  $j \in \{1, 2, \dots, d\}$ ,

$$\begin{aligned} \tilde{y}_j^s &= \arg \min_{x \in \mathbb{R}} \frac{1}{2} (x - \tilde{y}_j^{s-1} + hg_j)^2 + \lambda h |x| \\ &= \begin{cases} \tilde{y}_j^{s-1} - (\lambda + g_j)h, & \text{if } \tilde{y}_j^{s-1} > (\lambda + g_j)h, \\ \tilde{y}_j^{s-1} + (\lambda - g_j)h, & \text{if } \tilde{y}_j^{s-1} < -(\lambda - g_j)h, \\ 0, & \text{otherwise,} \end{cases} \\ &= \begin{cases} \tilde{y}_j^{s-1} - M, & \text{if } \tilde{y}_j^{s-1} > M, \\ \tilde{y}_j^{s-1} - m, & \text{if } \tilde{y}_j^{s-1} < m, \\ 0, & \text{otherwise.} \end{cases} \end{aligned}$$

where  $M \stackrel{\text{def}}{=} (\lambda + g_j)h$ ,  $m \stackrel{\text{def}}{=} -(\lambda - g_j)h$  and  $M - m = 2\lambda h > 0$ . Now, we will distinguish several cases based on  $g_j$ :

- (1) When  $g_j \geq \lambda$ , then  $M > m = -(\lambda - g_j)h \geq 0$ , thus by letting  $p = \lfloor \frac{y_j}{M} \rfloor$ , we have that: if  $y_j < m$ , then  $\tilde{y}_j^\tau = y_j - \tau m$ ; if  $m \leq y_j < M$ , then  $\tilde{y}_j^\tau = -(\tau - 1)m$ ; and if  $y_j \geq M$ , then

$$\begin{aligned} \tilde{y}_j^\tau &= \begin{cases} y_j - \tau M, & \text{if } \tau \leq p, \\ y_j - pM - (\tau - p)m, & \text{if } \tau > p \& y_j - pM < m, \\ -(\tau - p - 1)m, & \text{if } \tau > p \& y_j - pM \geq m, \end{cases} \\ &= \begin{cases} y_j - \tau M, & \text{if } \tau \leq p, \\ \min\{y_j - pM, m\} - (\tau - p)m, & \text{if } \tau > p. \end{cases} \end{aligned}$$

- (2) When  $-\lambda < g_j < \lambda$ , then  $M = (\lambda + g_j)h > 0$ ,  $m = -(\lambda - g_j)h < 0$ , thus we have that

$$\tilde{y}_j^\tau = \begin{cases} \max\{y_j - \tau M, 0\}, & \text{if } y_j \geq 0, \\ \min\{y_j - \tau m, 0\}, & \text{if } y_j < 0. \end{cases}$$

- (3) When  $g_j \leq -\lambda$ , then  $m < M = (\lambda + g_j)h \leq 0$ , thus by letting  $q = \lfloor \frac{y_j}{m} \rfloor$ , we have that: if  $y_j \leq m$ , then

$$\begin{aligned} \tilde{y}_j^\tau &= \begin{cases} y_j - \tau m, & \text{if } \tau \leq q, \\ y_j - pm - (\tau - q)M, & \text{if } \tau > q \& y_j - qm > M, \\ -(\tau - q - 1)M, & \text{if } \tau > q \& y_j - qm \leq M, \end{cases} \\ &= \begin{cases} y_j - \tau m, & \text{if } \tau \leq q, \\ \max\{y_j - qm, M\} - (\tau - q)M, & \text{if } \tau > q; \end{cases} \end{aligned}$$

if  $m < y_j \leq M$ , then  $\tilde{y}_j^\tau = -(\tau - 1)M$ ; if  $y_j > M$ , then  $\tilde{y}_j^\tau = y_j - \tau M$ .

Now, we will perform a few simplifications: Case (1). When  $y_j < M$ , we can conclude that  $\tilde{y}_j^\tau = \min\{y_j, m\} - \tau m$ . Moreover, since the following equivalences hold if  $g_j \geq \lambda$ :  $y_j \geq M \Leftrightarrow \frac{y_j}{M} \geq 1 \Leftrightarrow p \geq 1$ , and  $y_j < M \Leftrightarrow \frac{y_j}{M} < 1 \Leftrightarrow p \leq 0$ , the situation simplifies to

$$\begin{aligned}\tilde{y}_j^\tau &= \begin{cases} y_j - \tau M, & \text{if } p \geq \tau, \\ \min\{y_j - pM, m\} - (\tau - p)m, & \text{if } 1 \leq p < \tau, \\ \min\{y_j, m\} - \tau m, & \text{if } p \leq 0, \end{cases} \\ &= \begin{cases} y_j - \tau M, & \text{if } p \geq \tau, \\ \min\{y_j - [p]_+ M, m\} - (\tau - [p]_+)m, & \text{if } p < \tau, \end{cases}\end{aligned}$$

where  $[\cdot]_+ \stackrel{\text{def}}{=} \max\{\cdot, 0\}$ . For Case (3), when  $y_j > m$ , we can conclude that  $\tilde{y}_j^\tau = \max\{y_j, M\} - \tau M$ , and in addition, the following equivalences hold when  $g_j \leq -\lambda$ :

$$\begin{aligned}y_j \leq m &\Leftrightarrow \frac{y_j}{m} \geq 1 \Leftrightarrow q \geq 1, \\ y_j > m &\Leftrightarrow \frac{y_j}{m} < 1 \Leftrightarrow q \leq 0,\end{aligned}$$

which summarizes the situation as follows:

$$\begin{aligned}\tilde{y}_j^\tau &= \begin{cases} y_j - \tau m, & \text{if } q \geq \tau, \\ \max\{y_j - qm, M\} - (\tau - q)M, & \text{if } 1 \leq q < \tau, \\ \max\{y_j, M\} - \tau M, & \text{if } q \leq 0, \end{cases} \\ &= \begin{cases} y_j - \tau m, & \text{if } q \geq \tau, \\ \max\{y_j - [q]_+ m, M\} - (\tau - [q]_+)M, & \text{if } q < \tau. \end{cases} \quad \square\end{aligned}$$

## 4.8 Conclusion

We have proposed mS2GD—a mini-batch semi-stochastic gradient method—for minimizing a strongly convex composite function. Such optimization problems arise frequently in inverse problems in signal processing and statistics. Similarly to SAG, SVRG, SDCA and S2GD, our algorithm also outperforms existing deterministic method such as ISTA and FISTA. Moreover, we have shown that the method is by design amenable to a simple parallel implementation. Comparisons to state-of-the-art algorithms suggest that mS2GD, with a small-enough mini-batch size, is competitive in theory and faster in practice than other competing methods even without parallelism. The method can be efficiently implemented for sparse data sets.

# Chapter 5

# Distributed Optimization with Arbitrary Local Solvers

## 5.1 Motivation

Regression and classification techniques, represented in the general class of regularized loss minimization problems [178], are among the most central tools in modern big data analysis, machine learning, and signal processing. For these tasks, much effort from both industry and academia has gone into the development of highly tuned and customized solvers. However, with the massive growth of available datasets, major roadblocks still persist in the distributed setting, where data no longer fits in the memory of a single computer, and computation must be split across multiple machines in a network [19, 111, 106, 8, 149, 53, 173, 57, 32, 108, 166, 167, 140, 199, 82].

On typical real-world systems, communicating data between machines is several orders of magnitude slower than reading data from main memory, e.g., when leveraging commodity hardware. Therefore when trying to translate existing highly tuned single machine solvers to the distributed setting, great care must be taken to avoid this significant communication bottleneck [79, 188].

While several distributed solvers for the problems of interest have been recently developed, they are often unable to fully leverage the competitive performance of their tuned and customized single machine counterparts, which have already received much more research attention. More importantly, it is unfortunate that distributed solvers cannot automatically benefit from improvements made to the single machine solvers, and therefore are forced to lag behind the most recent developments.

In this chapter we make a step towards resolving these issues by proposing a general communication-efficient distributed framework that can employ arbitrary single machine local solvers and thus directly leverage their benefits and problem-specific improvements. Our framework works in rounds, where in each round the local solvers on each machine find a (possibly weak) solution to a specified subproblem of the same structure as the original master problem. On completion of each round, the partial updates between the machines are efficiently combined by leveraging the primal-dual structure of the global problem [188, 79, 105]. The framework therefore completely decouples the local solvers from the distributed communication. Through this decoupling, it is possible to balance communication and computation in the distributed setting, by controlling the desired accuracy and thus computational effort spent to determine the solution to each local subproblem. Our framework holds with this abstraction even if the user wishes to use a different local solver on each machine.

### 5.1.1 Contributions

*Reusability of Existing Local Solvers.* The proposed framework allows for distributed optimization with the use of arbitrary local solvers on each machine. This abstraction makes the resulting framework highly flexible, and means that it can easily leverage the benefits of

well-studied, problem-specific single machine solvers. In addition to increased flexibility and ease-of-use, this can result in large performance gains, as single machine solvers for the problems of interest have typically been thoroughly tuned for optimal performance. Moreover, any performance improvements that are made to these local solvers can be automatically translated by the framework into the distributed setting.

*Adaptivity to Communication Cost.* On real-world compute systems, the cost of communication versus computation typical varies by many orders of magnitude, from high-performance computing environments to very slow disk-based distributed workflow systems such as MapReduce/Hadoop. For optimization algorithms, it is thus essential to accommodate varying amounts of work performed locally per round, while still providing convergence guarantees. Our framework provides exactly such control.

*Strong Theoretical Guarantees.* In this chapter we extend and improve upon the CoCoA [79] method. Our theoretical convergence rates apply to both smooth and non-smooth losses, and for both CoCoA as well as CoCoA<sup>+</sup>, the more general framework presented here. Our new rates exhibit favorable *strong scaling* properties for the class of problems considered, as the number of machines  $K$  increases and the data size is kept fixed. More precisely, while the convergence rate of CoCoA degrades as  $K$  is increased, the stronger theoretical convergence rate here is—in the worst case complexity—*independent* of  $K$ . As only one vector is communicated per round and worker, this favorable scaling might be surprising. Indeed, for existing methods, splitting data among more machines generally increases communication requirements [166, 7], which can severely affect overall runtime.

*Primal-Dual Convergence.* We additionally strengthen the rates by showing stronger primal-dual convergence for both algorithmic frameworks, which are almost tight to their dual-only (or primal-only) counterparts. Primal-dual rates for CoCoA had not previously been analyzed in the general convex case. Our primal-dual rates allow efficient and practical certificates for the optimization quality, e.g., for stopping criteria.

*Experimental Results.* Finally, we provide an extensive experimental comparison that highlights the impact of using various arbitrary solvers locally on each machine, with experiments on several real-world, distributed datasets. We compare the performance of CoCoA and CoCoA<sup>+</sup> across these datasets and choices of solvers, in particular illustrating the performance on a 280 GB dataset. Our code is available in an open source C++ library, at: <https://github.com/optml/CoCoA>.

### 5.1.2 Outline

The rest of the chapter is organized as follows. Section 5.2 provides context and states the problem of interest, including necessary assumptions and their consequences. In Section 5.3 we formulate the algorithm in detail and explain how to implement it efficiently in practice. The main theoretical results are presented in Section 5.4, followed by a discussion of relevant related work in Section 5.5. Practical experiments demonstrating the strength of the proposed framework are given in Section 5.6. Finally, we prove the main results in Section 5.8.4.

## 5.2 Background and Problem Formulation

To provide context for our framework, we first state traditional complexity measures and convergence rates for single machine algorithms, and then demonstrate that these must be adapted to more accurately represent the performance of an algorithm in the distributed setting.

When running an iterative optimization algorithm  $\mathcal{A}$  on a single machine, its performance is typically measured by the total runtime:

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

Here,  $\mathcal{T}_{\mathcal{A}}$  stands for the time it takes to perform a single iteration of algorithm  $\mathcal{A}$ , and  $\mathcal{I}_{\mathcal{A}}(\epsilon)$  is the number of iterations  $\mathcal{A}$  needs to attain an  $\epsilon$ -accurate objective.<sup>1</sup>

On a single machine, most of the state-of-the-art first-order optimization methods can achieve quick convergence in practice in terms of (T-A) by performing a large amount of relatively fast iterations. In the distributed setting, however, time to communicate between two machines can be several orders of magnitude slower than even a single iteration of such an algorithm. As a result, the overall time needed to perform this single iteration can increase significantly.

Distributed timing can therefore be more accurately illustrated using the following practical distributed efficiency model (see also [111]), where

$$\text{TIME}(\mathcal{A}) = \mathcal{I}_{\mathcal{A}}(\epsilon) \times (c + \mathcal{T}_{\mathcal{A}}). \quad (\text{T-B})$$

The extra term  $c$  is the time required to perform one round of communication.<sup>2</sup> As a result, an algorithm that performs well in the setting of (T-A) does not necessarily perform well in the distributed setting (T-B), especially when implemented in a straightforward or naïve way. In particular, if  $c \gg \mathcal{T}_{\mathcal{A}}$ , we could intuitively expect less potential for improvement from fast computation, as most of the time in the method will be spent on communication, not on actual computational effort to solve the problem. In this setting, novel optimization procedures are needed that carefully consider the amount of communication and the distribution of data across multiple machines.

One approach to this challenge is to design novel optimization algorithms from scratch, designed to be efficient in the distributed setting. This approach has one obvious practical drawback: There have been numerous highly efficient solvers developed and fine-tuned to particular problems of interest, as long as the problem fits onto a single machine. These solvers are ideal if run on a single machine, but with the growth of data and necessity of data distribution, they must be re-designed to work in modern data regimes.

Recent work [79, 105, 188, 189, 191, 168] has attempted to address this issue by designing algorithms that reduce the communication bottleneck by allowing infrequent communication, while utilizing already existing algorithms as local sub-procedures. The presented work here builds on the promising approach of [188, 79] in this direction. See Section 5.5 for a detailed discussion of the related literature.

The core idea in this line of work is that one can formulate a local subproblem for each individual machine, and run an arbitrary local solver dependent only on local data for a number of iterations—obtaining a partial local update. After each worker returns its partial update, a global update is formed by their aggregation.

The big advantage of this is that companies and practitioners do not have to implement new algorithms that would be suitable for the distributed setting. We provide a way for them to utilize their existing algorithms that work on a single machine, and provide a novel communication protocol on top of this.

In the original work on CoCoA [79], authors provide convergence analysis only for the case when the overall update is formed as an average of the partial updates, and note that in practice it is possible to improve performance by making a longer step in the same direction. The main contribution of this work is a more general convergence analysis of various settings, which enables us to do better than averaging. In one case, we can even sum the partial updates to obtain the overall update, which yields the best result, both in theory and practice. We will see that this can result in significant performance gains, see also [105, 168].

In the analysis, we will allow local solvers of arbitrarily weak accuracy, each working on its own subproblem which is defined in a completely data-local way for each machine. The relative accuracy obtained by each local solver will be denoted by  $\Theta \in [0, 1]$ , where  $\Theta = 0$  describes an exact solution of the subproblem, and  $\Theta = 1$  means that the local subproblem objective has not

---

<sup>1</sup>While for many algorithms the cost of a single iteration will vary throughout the iterative process, this simple model will suffice for our purpose to highlight the key issues associated with extending algorithms to a distributed framework.

<sup>2</sup>For simplicity, we assume here a fixed network architecture, and compare only to classes of algorithms that communicate a single vector in each iteration, rendering  $c$  to be a constant, assuming we have a fixed number of machines. Most first-order algorithms would fall into this class.

Loss function	$\ell_i(a)$	$\ell_i^*(b)$	Property of $\ell$
Quadratic	$\frac{1}{2}(a - y_i)^2$	$\frac{1}{2}b^2 + y_i b$	Smooth
Hinge	$\max\{0, y_i - a\}$	$y_i b, \quad b \in [-1, 0]$	Continuous
Squared hinge	$(\max\{0, y_i - a\})^2$	$\frac{b^2}{4}, \quad b \in [-\infty, 0]$	Smooth
Logistic	$\log(1 + \exp(-y_i a))$	$-\frac{b}{y_i} \log\left(-\frac{b}{y_i}\right) + \left(1 + \frac{b}{y_i}\right) \log\left(1 + \frac{b}{y_i}\right)$	Smooth

Table 5.1: Examples of commonly used loss functions.

improved at all, for this run of the local solver. This paradigm results in a substantial change in how we analyze efficiency in the distributed setting. The formula practitioners are interested in minimizing thus changes to become:

$$\text{TIME}(\mathcal{A}, \Theta) = \mathcal{I}(\epsilon, \Theta) \times (c + \mathcal{T}_{\mathcal{A}}(\Theta)). \quad (\text{T-C})$$

Here, the function  $\mathcal{T}_{\mathcal{A}}(\Theta)$  represents the time the local algorithm  $\mathcal{A}$  needs to obtain an accuracy of  $\Theta$  on the local subproblem. Note that the number of outer iterations  $\mathcal{I}(\epsilon, \Theta)$  is independent of choice of the inner algorithm  $\mathcal{A}$ , which will also be reflected by our convergence analysis presented in Section 5.4. Our convergence rates will hold for any local solver  $\mathcal{A}$  achieving local accuracy of  $\Theta$ . For strongly convex problems, the general form will be  $\mathcal{I}(\epsilon, \Theta) = \frac{\mathcal{O}(\log(1/\epsilon))}{1-\Theta}$ . The inverse dependence on  $1 - \Theta$  suggests that there is a limit to how much we can gain by solving local subproblems to high accuracy, i.e., for  $\Theta$  close to 0. There will always be on the order of  $\log(1/\epsilon)$  outer iterations needed. Hence, excessive local accuracy should not be necessary. On the other hand, if  $\Theta \rightarrow 1$ , meaning that the cost and quality of the local solver diminishes, then the number of outer iterations  $\mathcal{I}(\epsilon, \Theta)$  will increase dramatically, which is to be expected.

To illustrate the strength of the paradigm (T-C) compared to (T-B), suppose that we run just a single iteration of gradient descent as the local solver  $\mathcal{A}$ . Within our framework, choosing this local solver would lead to a method which is equivalent to naively distributed gradient descent<sup>3</sup>. Indeed, running gradient descent for a single iteration would attain a particular value of  $\Theta$ . Note that we typically do not set this explicitly:  $\Theta$  is implicitly chosen by the number of iterations or stopping criterion specified by the user for the local solver. There is no reason to think that the value attained by single iteration of gradient descent would be optimal. For instance, it may be the case that running gradient descent for, say, 200 iterations, instead of just one, would give substantially better results in practice, due to better communication efficiency. Considerations of this form are discussed in detail in Section 5.6.

In general, one would intuitively expect that the optimal choice would be to have  $\Theta$  such that  $\mathcal{T}_{\mathcal{A}}(\Theta) = \mathcal{O}(1) \times c$ . In practice, however, the best strategy for any given local solver is to estimate the optimal choice by trying several values for the number of local iterations. We discuss the importance of  $\Theta$ , both theoretically and empirically, in Sections 5.4 and 5.6.

### 5.2.1 Problem Formulation

Let the training data  $\{x_i \in \mathbb{R}^d, y_i \in \mathbb{R}\}_{i=1}^n$  be the set of input-output pairs, where  $y_i$  can be real valued or from a discrete set in the case of classification problems. We will assume without loss of generality that  $\forall i : \|x_i\| \leq 1$ . Many common tasks in machine learning and signal processing can be cast as the following optimization problem:

$$\min_{w \in \mathbb{R}^d} \left\{ P(w) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n \ell_i(x_i^T w) + \lambda g(w) \right\}, \quad (5.1)$$

where  $\ell_i$  is some convex loss function and  $g$  is a regularizer. Note that  $y_i$  is typically hidden in the formulation of functions  $\ell_i$ . Table 5.1 lists several common loss functions together with their convex conjugates  $\ell_i^*$  [164].

<sup>3</sup>Note that this is not obvious at this point. They are identical, subject to choice of local subproblems as specified in Section 5.3.1.

The dual optimization problem for formulation (5.1)—as a special case of Fenchel duality—can be written as follows [192, 164]:

$$\max_{\alpha \in \mathbb{R}^n} \left\{ D(\alpha) \stackrel{\text{def}}{=} \frac{1}{n} \left( \sum_{i=1}^n -\ell_i^*(-\alpha_i) \right) - \lambda g^* \left( \frac{1}{\lambda n} X \alpha \right) \right\}, \quad (5.2)$$

where  $X = [x_1, x_2, \dots, x_n] \in \mathbb{R}^{d \times n}$ , and  $\ell_i^*$  and  $g^*$  are the convex conjugate functions of  $\ell_i$  and  $g$ , respectively. The convex (Fenchel) conjugate of a function  $\phi : \mathbb{R}^k \rightarrow \mathbb{R}$  is defined as the function  $\phi^* : \mathbb{R}^k \rightarrow \mathbb{R}$ , with  $\phi^*(u) := \sup_{s \in \mathbb{R}^k} \{s^T u - \phi(s)\}$ .

For simplicity throughout the chapter, let us denote

$$f(\alpha) \stackrel{\text{def}}{=} \lambda g^* \left( \frac{1}{\lambda n} X \alpha \right) \quad \text{and} \quad R(\alpha) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n \ell_i^*(-\alpha_i), \quad (5.3)$$

such that  $D(\alpha) \stackrel{(5.2)+(5.3)}{=} -f(\alpha) - R(\alpha)$ .

It is well known [143, 172, 164, 55] that the first-order optimality conditions give rise to a natural mapping that relates pairs of primal and dual variables. This mapping employs the linear map given by the data  $X$ , and maps any dual variable  $\alpha \in \mathbb{R}^n$  to a primal candidate vector  $w \in \mathbb{R}^d$  as follows:

$$w(\alpha) := \nabla g^*(v(\alpha)) = \nabla g^* \left( \frac{1}{\lambda n} X \alpha \right),$$

where we denote  $v(\alpha) := \frac{1}{\lambda n} X \alpha$ .

For this mapping, under the assumptions that we make in Section 5.2.2 below, it holds that if  $\alpha^*$  is an optimal solution of (5.2), then  $w(\alpha^*)$  is an optimal solution of (5.1). In particular, *strong duality* holds between the primal and dual problems. If we define the duality gap function as

$$\text{Gap}(\alpha) := P(w(\alpha)) - D(\alpha), \quad (5.4)$$

then  $\text{Gap}(\alpha^*) = 0$ , which ensures that by solving the dual problem (5.2) we also solve the original primal problem of interest (5.1). As we will later see, there are many benefits to leveraging this primal-dual relationship, including the ability to use the duality gap as a certificate of solution quality, and, in the distributed setting, the ability to effectively distribute computation.

*Notation.* We assume that to solve problem (5.2), we have a network of  $K$  machines at our disposal. The data  $\{x_i, y_i\}_{i=1}^n$  is residing on the  $K$  machines in a distributed fashion, with every machine holding a subset of the whole dataset. We distribute the dual variables in the same manner, with each dual variable  $\alpha_i$  corresponding to an individual data point  $x_i$ . The given data distribution is described using a partition  $\mathcal{P}_1, \dots, \mathcal{P}_K$  that corresponds to the indices of the data and dual variables residing on machine  $k$ . Formally,  $\mathcal{P}_k \subseteq \{1, 2, \dots, n\}$  for each  $k$ ;  $\mathcal{P}_k \cap \mathcal{P}_l = \emptyset$  whenever  $k \neq l$ ; and  $\bigcup_{k=1}^K \mathcal{P}_k = \{1, 2, \dots, n\}$ .

Finally, we introduce the following notation dependent on this partitioning. For any  $h \in \mathbb{R}^n$ , let  $h_{[k]}$  be the vector in  $\mathbb{R}^n$  defined such that  $(h_{[k]})_i = h_i$  if  $i \in \mathcal{P}_k$  and 0 otherwise. Note that, in particular,  $h = \sum_{k=1}^K h_{[k]}$ . Analogously, we write  $X_{[k]}$  for the matrix consisting only of the columns  $i \in \mathcal{P}_k$ , padded with zeros in all other columns.

### 5.2.2 Technical Assumptions

Here we first state the properties and assumptions used throughout the chapter. We assume that for all  $i \in \{1, \dots, n\}$ , the function  $\ell_i$  in (5.1) is convex, i.e.,  $\forall \lambda \in [0, 1]$  and  $\forall x, y \in \mathbb{R}$  we have  $\ell_i(\lambda x + (1 - \lambda)y) \leq \lambda \ell_i(x) + (1 - \lambda) \ell_i(y)$ .

We also assume that the function  $g$  is 1-strongly convex, i.e., for all  $w, u \in \mathbb{R}^d$  it holds that  $g(w + u) \geq g(w) + \langle \nabla g(w), u \rangle + \frac{1}{2} \|u\|^2$ , where  $\nabla g(w)$  is any subgradient<sup>4</sup> of the function  $g$ .

---

<sup>4</sup>A subgradient of a convex function  $\phi$  in a point  $x' \in \mathbb{R}^d$  is defined as any  $\xi \in \mathbb{R}^d$  satisfying for all  $x \in \mathbb{R}^d$ ,  $\phi(x) \geq \phi(x') + \langle \xi, x - x' \rangle$ .

Here,  $\|\cdot\|$  denotes the standard Euclidean norm.

Note that we use subgradients in the definition of strong convexity. This is due to the fact that while we will need the function  $g$  to be strongly convex in our analysis, we do not require smoothness. An example used in practice is  $g(w) = \|w\|^2 + \lambda'\|w\|_1$  for some  $\lambda' \in \mathbb{R}$ . Also note that in the problem formulation (5.1) we have a regularization parameter  $\lambda$ , which controls the strong convexity parameter of the entire second term. Hence, fixing the strong convexity parameter of  $g$  to 1 is not restrictive in this regard. For instance, this setting has been used previously in [164, 143, 37].

The following assumptions state properties of the functions  $\ell_i$ , which we use only in certain results in the chapter. We always explicitly state when we require each assumption.

**Assumption 28** (( $1/\gamma$ )-Smoothness). *Functions  $\ell_i : \mathbb{R} \rightarrow \mathbb{R}$  are  $1/\gamma$ -smooth, if  $\forall i \in \{1, \dots, n\}$  and  $\forall x, h \in \mathbb{R}$  it holds that*

$$\ell_i(x + h) \leq \ell_i(x) + h\nabla\ell_i(x) + \frac{1}{2\gamma}h^2, \quad (5.5)$$

where  $\nabla\ell_i(x)$  denotes the gradient of the function  $\ell_i$ .

**Assumption 29** ( $L$ -Lipschitz Continuity). *Functions  $\ell_i : \mathbb{R} \rightarrow \mathbb{R}$  are  $L$ -Lipschitz continuous, if  $\forall i \in \{1, \dots, n\}$  and  $\forall x, h \in \mathbb{R}$  it holds that*

$$|\ell_i(x + h) - \ell_i(x)| \leq L|h|. \quad (5.6)$$

**Remark 30.** As a consequence of having  $1/\gamma$ -smoothness of  $\ell_i$  and 1-strong convexity of  $g$ , we have that the functions  $\ell_i^*(\cdot)$  are  $\gamma$ -strongly convex and  $g^*(\cdot)$  is 1-smooth [154]. These are the properties we will ultimately use as we will be solving the dual problem (5.2). Note that 1-smoothness of  $g^* : \mathbb{R}^d \rightarrow \mathbb{R}$  means that for all  $x, h \in \mathbb{R}^d$ ,

$$g^*(x + h) \leq g^*(x) + \langle \nabla g^*(x), h \rangle + \frac{1}{2}\|h\|^2. \quad (5.7)$$

The following lemma, which is a consequence of 1-smoothness of  $g^*$  and the definition of  $f$ , will be crucial in deriving a meaningful local subproblem for the proposed distributed framework.

**Lemma 31.** *Let  $f$  be defined in (5.3). Then for all  $\alpha, h \in \mathbb{R}^n$  we have*

$$f(\alpha + h) \leq f(\alpha) + \langle \nabla f(\alpha), h \rangle + \frac{1}{2\lambda n^2}h^T X^T X h. \quad (5.8)$$

**Remark 32.** Note that although the above inequality appears as a consequence of the problem structure (5.2) and of the strong convexity of  $g$ , there are other ways to satisfy it. Hence, our dual analysis holds for all optimization problems of the form  $\max_{\alpha} D(\alpha)$ , where  $D(\alpha) = -f(\alpha) - R(\alpha)$ , and where  $f$  satisfies inequality (5.8). However, for the duality gap analysis we naturally do require that the dual problem arises from the primal problem, with  $g$  being strongly convex.

### 5.3 The Framework

In this section we start by giving a general view of the proposed framework, explaining the most important concepts needed to make the framework efficient. In Section 5.3.1 we discuss the formulation of the local subproblems, and in Section 5.3.2 we provide specific details and best practices for implementation.

The data distribution plays a crucial role in Algorithm 8, where in each outer iteration indexed by  $t$ , machine  $k$  runs an arbitrary local solver on a problem described only by the data that particular machine owns and other fixed constants or linear functions.

The crucial property is that the optimization algorithm on machine  $k$  changes only coordinates of the dual optimization variable  $\alpha^t$  corresponding to the partition  $\mathcal{P}_k$  to obtain an approximate solution to the local subproblem. We will formally specify this in Assumption 35.

After each such step, updates from all machines are aggregated to form a new iterate  $\alpha^{t+1}$ . The aggregation parameter  $\nu$  will typically be between  $\nu = 1/K$ , corresponding to averaging, and  $\nu = 1$ , adding.

---

**Algorithm 8** Improved CoCoA+ Framework

---

```

1: Input: starting point  $\alpha^0 \in \mathbb{R}^n$ , aggregation parameter  $\nu \in (0, 1]$ , data partition  $\{\mathcal{P}_k\}_{k=1}^K$ 
2: for  $t = 0, 1, 2, \dots$  do
3:   for  $k \in \{1, 2, \dots, K\}$  in parallel over machines do
4:     Let  $h_{[k]}^t$  be an approximate solution of the local problem (LO), i.e.

$$\max_{h_{[k]} \in \mathbb{R}^n} \mathcal{G}_k(h_{[k]}; \alpha^t)$$

5:   end for
6:   Set  $\alpha^{t+1} := \alpha^t + \nu \sum_{k=1}^K h_{[k]}^t$ 
7: end for

```

---

Here we list the core conceptual properties of Algorithm 8, which are important qualities that allow it to run efficiently.

**Locality.** The local subproblem  $\mathcal{G}_k$  (LO) is defined purely based on the data points residing on machine  $k$ , as well as a single shared vector in  $\mathbb{R}^d$  (representing the state of the  $\alpha^t$  variables of the other machines). Each local solver can then run independently and in parallel, i.e., there is no need for communication while solving the local subproblems.

**Local changes.** The optimization algorithm used to solve the local subproblem  $\mathcal{G}_k$  outputs a vector  $h_{[k]}^t$  with nonzero elements only in coordinates corresponding to variables  $\alpha_{[k]}$  stored locally (i.e.,  $i \in \mathcal{P}_k$ ).

**Efficient maintenance.** Given the description of the local problem  $\mathcal{G}_k(\cdot; \alpha^t)$  at time  $t$ , the new local problem  $\mathcal{G}_k(\cdot; \alpha^{t+1})$  at time  $t+1$  can be formed on each machine, requiring only communication of a single vector in  $\mathbb{R}^d$  from each machine  $k$  to the master node, and vice versa, back to each machine  $k$ .

Let us now comment on these properties in more detail. Locality is important for making the method versatile, and is the way we escape the restricted setting described by (T-B) that allows us much greater flexibility in designing the overall optimization scheme. Local changes result from the fact that we distribute coordinates of the dual variables  $\alpha$  in the same manner as the data, and thus only make updates to the coordinates stored locally. As we will see, efficient maintenance of the subproblems can be obtained. For this, a communication-efficient encoding of the current shared state  $\alpha$  is necessary. To this goal, we will in Section 5.3.2 show that communication of a single  $d$ -dimensional vector is enough to formulate the subproblems (LO) in each round, by carefully exploiting their partly separable structure.

Note that Algorithm 8 is the “analysis friendly” formulation of our algorithm framework, and it is not yet fully illustrative for implementation purposes. In Section 5.3.2 we will precisely formulate the actual communication scheme, and illustrate how the above properties can be achieved.

Before that, we formulate the precise subproblem  $\mathcal{G}_k$  in the following section.

### 5.3.1 The Local Subproblems

We can define a data-local subproblem of the original dual optimization problem (5.2), which can be solved on machine  $k$  and only requires accessing data which is already available locally, i.e., datapoints with  $i \in \mathcal{P}_k$ . More formally, each machine  $k$  is assigned the following local subproblem, depending only on the previous shared primal vector  $w \in \mathbb{R}^d$ , and the change in the local dual variables  $\alpha_i$  with  $i \in \mathcal{P}_k$ :

$$\max_{h_{[k]} \in \mathbb{R}^n} \mathcal{G}_k^{\sigma'}(h_{[k]}; \alpha). \quad (5.9)$$

We are now ready to define the local objective  $\mathcal{G}_k^{\sigma'}(\cdot; \alpha)$  as follows:

$$\mathcal{G}_k^{\sigma'}(h_{[k]}; \alpha) := -\frac{1}{K}f(\alpha) - \langle \nabla f(\alpha), h_{[k]} \rangle - \frac{\lambda\sigma'}{2} \left\| \frac{1}{\lambda n} X_{[k]} h_{[k]} \right\|^2 - R_k(\alpha_{[k]} + h_{[k]}), \quad (\text{LO})$$

where  $R_k(\alpha_{[k]}) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i \in \mathcal{P}_k} \ell_i^*(-\alpha_i)$ . The role of the parameter  $\sigma' \geq 1$  is to measure the “difficulty” of the data partition, in a sense which we will discuss in detail in Section 5.3.3.

The interpretation of the subproblems defined above is that they will form a quadratic approximation of the smooth part of the true objective  $D$ , which becomes separable over the machines. The approximation keeps the non-smooth  $R$  part intact. The variable  $h_{[k]}$  expresses the update proposed by machine  $k$ . In this spirit, note also that the approximation coincides with  $D$  at the reference point  $\alpha$ , i.e.  $\sum_{k=1}^K \mathcal{G}_k^{\sigma'}(\mathbf{0}; \alpha) = D(\alpha)$ . We will discuss the interpretation and properties of these subproblems in more detail below in Section 5.3.3.

### 5.3.2 Practical Communication-Efficient Implementation

We now discuss how Algorithm 8 can efficiently be implemented in a distributed environment. Most importantly, we clarify how the “local” subproblems can be formulated and solved while using only local information from the corresponding machines, and we make precise what information needs to be communicated in each round.

Recall that the local subproblem objective  $\mathcal{G}_k^{\sigma'}(\cdot; \alpha)$  was defined in (LO). We will now equivalently rewrite this optimization problem, illustrating how it can be expressed using only *local* information. To do so, we use our simplifying notation  $v = v(\alpha) := \frac{1}{\lambda n} X \alpha$  for a given  $\alpha$ . As we see in the reformulation, it is precisely this vector  $v \in \mathbb{R}^d$  which contains all the necessary shared information between the machines. Given the vector  $v$ , the subproblem (LO) can be equivalently written as

$$\begin{aligned} \mathcal{G}_k^{\sigma'}(h_{[k]}; v, \alpha_{[k]}) &:= -\frac{\lambda}{K} g^*(v) - \left\langle \frac{1}{n} X_{[k]}^T \nabla g^*(v), h_{[k]} \right\rangle - \frac{\lambda\sigma'}{2} \left\| \frac{1}{\lambda n} X_{[k]} h_{[k]} \right\|^2 \\ &\quad - R_k(\alpha_{[k]} + h_{[k]}). \end{aligned} \quad (\text{LO}')$$

Here for the reformulation of the gradient term, we have simply used the chain rule on the objective  $f$  (recall the definition  $f(\alpha) \stackrel{\text{def}}{=} \lambda g^*(v)$ ), giving

$$\nabla f(\alpha)_{[k]} = \frac{1}{n} X_{[k]}^T \nabla g^*(v).$$

*Practical Distributed Framework.* In summary, we have seen that each machine can formulate the local subproblem given purely local information (the local data  $X_{[k]}$  as well as the local dual variables  $\alpha_{[k]}$ ). No information about the data or variables  $\alpha$  stored on the other machines is necessary.

The only requirement for the method to work is that between the rounds, the changes in the  $\alpha_{[k]}$  variables on each machine and the resulting global change in  $v$  are kept consistent, in the sense that  $v^t = v(\alpha^t) := \frac{1}{\lambda n} X \alpha^t$  must always hold. Note that for the evaluation of  $\nabla g^*(v)$ , the vector  $v$  is all that is needed. In practice,  $g$  as well as its conjugate  $g^*$  are simple vector-valued regularization functions, the most prominent example being  $g(v) = g^*(v) = \frac{1}{2} \|v\|^2$ .

In the following more detailed formulation of the CoCoA+ framework shown in Algorithm 9 (an equivalent reformulation of Algorithm 8), the crucial communication pattern of the framework finally becomes more clear: Per round, *only a single vector* (the update on  $v \in \mathbb{R}^d$ ) needs to be sent over the communication network. The reduce-all operation in line 10 means that each machine sends their vector  $\Delta v_k^t \in \mathbb{R}^d$  to the network, which performs the addition operation of the  $K$  vectors to the old  $v^t$ . The resulting vector  $v^{t+1}$  is then communicated back to all machines, so that all have the same copy of  $v^{t+1}$  before the beginning of the next round.

The framework as shown below in Algorithm 9 clearly maintains the consistency of  $\alpha^t$  and  $v^t = v^t(\alpha^t)$  after each round, no matter which local solver is used to approximately solve (LO'). A diagram illustrating the communication and computation involved in the first two

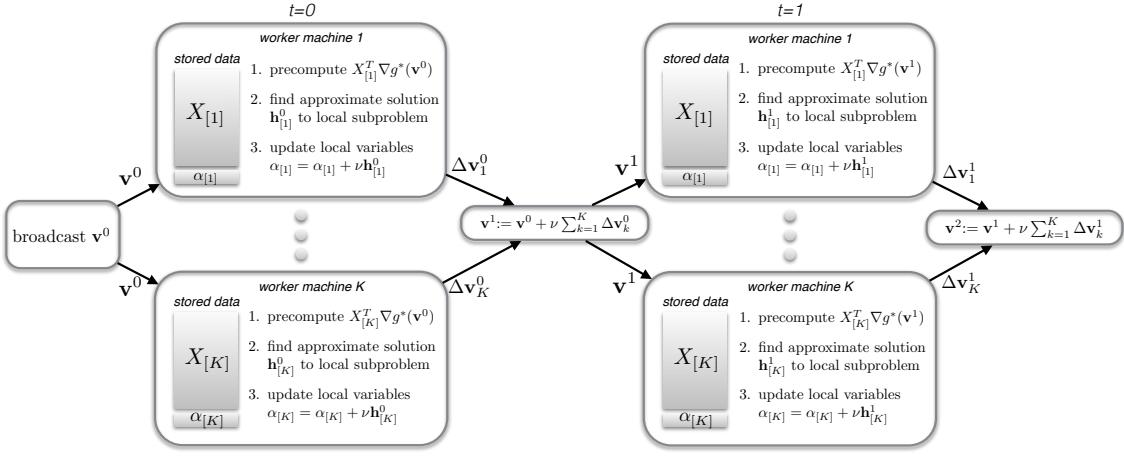


Figure 5.1: The first two iterations of the improved framework (practical implementation).

full iterations of Algorithm 9 is given in Figure 5.1.

---

**Algorithm 9** Improved CoCoA+ Framework, Practical Implementation

---

```

1: Input: starting point  $\alpha^0 \in \mathbb{R}^n$ , aggregation parameter  $\nu \in (0, 1]$ , data partition  $\{\mathcal{P}_k\}_{k=1}^K$ 
2:  $v^0 := \frac{1}{\lambda n} X \alpha^0 \in \mathbb{R}^d$ 
3: for  $t = 0, 1, 2, \dots$  do
4:   for  $k \in \{1, 2, \dots, K\}$  in parallel over machines do
5:     Precompute  $X_{[k]}^T \nabla g^*(v^t)$ 
6:     Let  $h_{[k]}^t$  be an approximate solution of the local problem (LO'), i.e.
      
$$\max_{h_{[k]} \in \mathbb{R}^n} \mathcal{G}_k^{\sigma'}(h_{[k]}; v^t, \alpha_{[k]}^t) \quad \triangleright \text{computation}$$

7:     Update local variables  $\alpha_{[k]}^{t+1} := \alpha_{[k]}^t + \nu h_{[k]}^t$ 
8:     Let  $\Delta v_k^t := \frac{1}{\lambda n} X_{[k]} h_{[k]}^t$ 
9:   end for
10:  reduce all to compute  $v^{t+1} := v^t + \nu \sum_{k=1}^K \Delta v_k^t \quad \triangleright \text{communication}$ 
11: end for

```

---

### 5.3.3 Compatibility of the Subproblems for Aggregating Updates

In this subsection, we shed more light on the local subproblems on each machine, as defined in (LO) above, and their interpretation. More formally, we show how the aggregation parameter  $\nu$  (controlling the level of adding versus averaging the resulting updates from each machine) and  $\sigma'$  (the subproblem parameter) interplay together, so that in each round they achieve a valid approximation to the global objective function  $D$ .

The role of the subproblem parameter  $\sigma'$  is to measure the difficulty of the given data partition. For the convergence results discussed below to hold,  $\sigma'$  must be chosen not smaller than

$$\sigma' \geq \sigma'_{min} \stackrel{\text{def}}{=} \nu \cdot \max_{h \in \mathbb{R}^n} \{h^T X^T X h \mid h^T G h \leq 1\}. \quad (5.10)$$

Here,  $G$  is the block diagonal submatrix of the data covariance matrix  $X^T X$ , corresponding to the partition  $\{\mathcal{P}_k\}_{k=1}^K$ , i.e.,

$$G_{ij} \stackrel{\text{def}}{=} \begin{cases} x_i^T x_j = (X^T X)_{ij}, & \text{if } \exists k \text{ such that } i, j \in \mathcal{P}_k, \\ 0, & \text{otherwise.} \end{cases} \quad (5.11)$$

In this notation, it is easy to see that the crucial quantity defining  $\sigma'_{min}$  above is written as  $h^T G h = \sum_{k=1}^K \|X_{[k]} h_{[k]}\|^2$ .

The following lemma shows that if the aggregation and subproblem parameters  $\nu$  and  $\sigma'$  satisfy (5.10), then the sum of the subproblems  $\sum_k \mathcal{G}_k^{\sigma'}$  will closely approximate the global objective function  $D$ . More precisely, this sum is a block-separable lower bound on  $D$ .

**Lemma 33.** Let  $\sigma' \geq 1$  and  $\nu \in [0, 1]$  satisfy (5.10) (that is  $\sigma' \geq \sigma'_{\min}$ ). Then  $\forall \alpha, h \in \mathbb{R}^n$ , it holds that

$$D\left(\alpha + \nu \sum_{k=1}^K h_{[k]}\right) \geq (1 - \nu)D(\alpha) + \nu \sum_{k=1}^K \mathcal{G}_k^{\sigma'}(h_{[k]}; \alpha), \quad (5.12)$$

The following lemma gives a simple choice for the subproblem parameter  $\sigma'$ , which is trivial to calculate for all values of the aggregation parameter  $\nu \in \mathbb{R}$ , and safe in the sense of the desired condition (5.10) above. Later we will show experimentally (Section 5.6) that the choice of this safe upper bound for  $\sigma'$  only has a minimal effect on the overall performance of the algorithm.

**Lemma 34.** For any aggregation parameter  $\nu \in [0, 1]$ , the choice of the subproblem parameter  $\sigma' := \nu K$  is valid for (5.10), i.e.,  $\nu K \geq \sigma'_{\min}$ .

## 5.4 Main Results

In this section we state the main theoretical results of this chapter. Before doing so, we elaborate on one of the most important aspects of the algorithmic framework: the *quality of approximate local solutions*.

### 5.4.1 Quality of Local Solutions

The notion of approximation quality provided by the local solvers is measured according to the following:

**Assumption 35** (Quality of local solution). Let  $\Theta \in [0, 1]$  and  $\alpha \in \mathbb{R}^n$  be fixed, and let  $h_{[k]}^*$  be the optimal solution of a local subproblem  $\mathcal{G}_k(\cdot; \alpha)$ . We assume the local optimization procedure run on every node  $k \in [K]$  in each iteration  $t$  produces a (possibly random) output  $h_{[k]}$  satisfying

$$\mathbb{E} \left[ \mathcal{G}_k(h_{[k]}^*; \alpha) - \mathcal{G}_k(h_{[k]}; \alpha) \right] \leq \Theta \left[ \mathcal{G}_k(h_{[k]}^*; \alpha) - \mathcal{G}_k(\mathbf{0}; \alpha) \right]. \quad (5.13)$$

The assumption specifies the (relative) accuracy  $\Theta$  obtained on solving the local subproblem  $\mathcal{G}_k$ . Considering the two extreme examples, setting  $\Theta = 0$  would require to find the exact maximum, while  $\Theta = 1$  states that no improvement was achieved at all by the local solver. Intuitively, we would prefer  $\Theta$  to be small, but spending many computational resources to drive  $\Theta$  to 0 can be excessive in practice, since  $\mathcal{G}_k$  is actually not the problem we are interested in solving (5.2), but is the problem to be solved per communication round. The best choice in practice will therefore be to choose  $\Theta$  such that the local solver runs for a time comparable to the time it takes for a single communication round. This freedom of choice of  $\Theta \in [0, 1]$  is a crucial property of our proposed framework, allowing it to adapt to the full range of communication speeds on real world systems, ranging from supercomputers on one extreme to very slow communication rounds like MapReduce systems on the other extreme.

In Section 5.6 we study the impact of different values of this parameter on the overall performance of solving (5.2).

### 5.4.2 Complexity Bounds

Now we are ready to state the main results. Theorem 36 covers the case when  $\forall i$ , the loss function  $\ell_i$  is  $1/\gamma$  smooth, and Theorem 37 covers the case when  $\ell_i$  is  $L$ -Lipschitz continuous. For simplicity in the rates, we define the following two quantities:

$$\forall k : \sigma_k \stackrel{\text{def}}{=} \max_{\alpha_{[k]} \in \mathbb{R}^n} \frac{\|X_{[k]}\alpha_{[k]}\|^2}{\|\alpha_{[k]}\|^2} \quad \text{and} \quad \sigma \stackrel{\text{def}}{=} \sum_{k=1}^K \sigma_k |\mathcal{P}_k|.$$

**Theorem 36** (Smooth loss functions). *Assume the loss functions  $\ell_i$  are  $(1/\gamma)$ -smooth  $\forall i \in [n]$ . We define  $\sigma_{\max} = \max_{k \in [K]} \sigma_k$ . Then after  $T$  iterations of Algorithm 9, with*

$$T \geq \frac{1}{\nu(1-\Theta)} \frac{\lambda\gamma n + \sigma_{\max}\sigma'}{\lambda\gamma n} \log \frac{1}{\epsilon_D},$$

it holds that

$$\mathbb{E} [D(\alpha^*) - D(\alpha^T)] \leq \epsilon_D.$$

Furthermore, after  $T$  iterations with

$$T \geq \frac{1}{\nu(1-\Theta)} \frac{\lambda\gamma n + \sigma_{\max}\sigma'}{\lambda\gamma n} \log \left( \frac{1}{\nu(1-\Theta)} \frac{\lambda\gamma n + \sigma_{\max}\sigma'}{\lambda\gamma n} \frac{1}{\epsilon_{\text{Gap}}} \right), \quad (5.14)$$

we have the expected duality gap

$$\mathbb{E} [P(w(\alpha^T)) - D(\alpha^T)] \leq \epsilon_{\text{Gap}}.$$

**Theorem 37** (Lipschitz continuous loss functions). *Consider Algorithm 9 with Assumption 35. Let  $\ell_i(\cdot)$  be  $L$ -Lipschitz continuous, and  $\epsilon_{\text{Gap}} > 0$  be the desired duality gap (and hence an upper-bound on primal sub-optimality). Then after  $T$  iterations, where*

$$\begin{aligned} T &\geq T_0 + \max \left\{ \left\lceil \frac{1}{\nu(1-\Theta)} \right\rceil, \frac{4L^2\sigma\sigma'}{\lambda n^2\epsilon_{\text{Gap}}\nu(1-\Theta)} \right\}, \\ T_0 &\geq t_0 + \max \left\{ 0, \frac{2}{\nu(1-\Theta)} \left( \frac{8L^2\sigma\sigma'}{\lambda n^2\epsilon_{\text{Gap}}} - 1 \right) \right\}, \\ t_0 &\geq \max \left\{ 0, \left\lceil \frac{1}{\nu(1-\Theta)} \log \left( \frac{2\lambda n^2(D(\alpha^*) - D(\alpha^0))}{4L^2\sigma\sigma'} \right) \right\rceil \right\}, \end{aligned} \quad (5.15)$$

we have that the expected duality gap satisfies

$$\mathbb{E} [P(w(\bar{\alpha})) - D(\bar{\alpha})] \leq \epsilon_{\text{Gap}},$$

at the averaged iterate

$$\bar{\alpha} := \frac{1}{T - T_0} \sum_{t=T_0+1}^{T-1} \alpha^t. \quad (5.16)$$

The most important observation regarding the above result is that we do not impose any assumption on the choice of the local solver, apart from the sufficient decrease condition on the local objective in Assumption 35.

Let us now comment on the leading terms of the complexity results. The inverse dependence on  $1 - \Theta$  suggests that it is worth pushing the rate of local accuracy  $\Theta$  down to zero. However, when thinking about overall complexity, we have to bear in mind that achieving high accuracy on the local subproblems might be too expensive. The optimal choice would depend on the time we estimate a round of communication would take. In general, if communication is slow, it would be worth spending more time on solving local subproblems, but not so much if communication is relatively fast. We discussed this tradeoff in Section 5.2.

We achieve a significant speedup by replacing the slow averaging aggregation (as in [79]) by more aggressive adding instead, that is  $\nu = 1$  instead of  $\nu = 1/K$ . Note that the safe subproblem parameter for the averaging case ( $\nu = 1/K$ ) is  $\sigma' := 1$ , while for adding ( $\nu = 1$ ) it is given by  $\sigma' := K$ , both proven in Lemma 34. The speedup that results from more aggressive adding is reflected in the convergence rate as shown above, when plugging in the actual parameter values  $\nu$  and  $\sigma'$  for the two cases, as we will illustrate more clearly in the next subsection.

### 5.4.3 Discussion and Interpretations of Convergence Results

As the above theorems suggest, it is not possible to meaningfully change the aggregation parameter  $\nu$  in isolation. It comes naturally coupled with a particular subproblem.

In this section, we explain a simple way to be able to set the aggregation parameter as  $\nu = 1$ ,

that is to aggressively add up the updates from each machine. The motivation for this comes from a common practical setting. When solving the SVM dual (Hinge loss:  $\ell_i(a) = \max\{0, y_i - a\}$ ), the optimization problem comes with ‘‘box constraints’’, i.e., for all  $i \in \{1, \dots, n\}$ , we have  $\alpha_i \in [0, 1]$  (see Table 5.1). The particular values of  $\alpha_i$  being 0 or 1 have a particular interpretation in the context of original problem (5.1). If we used  $\nu < 1$ , we would never be able reach the upper boundary of any variable  $\alpha_i$ , when starting the algorithm with all-zeros  $\alpha$ . This example illustrates some of the downsides of averaging vs. adding updates, coming from the fact that the step-size from using averaging (by being  $1/K$  times shorter) can result in  $1/K$  times slower convergence.

For the case of aggressive adding, the convergence from Theorem 36 becomes:

**Corollary 38** (Smooth loss functions - adding). *Let the assumptions of Theorem 36 be satisfied. If we run Algorithm 8 with  $\nu = 1, \sigma' = K$  for*

$$T \stackrel{(5.14)}{=} \frac{1}{1 - \Theta} \frac{\lambda \gamma n + \sigma_{\max} K}{\lambda \gamma n} \log \left( \frac{1}{1 - \Theta} \frac{\lambda \gamma n + \sigma_{\max} K}{\lambda \gamma n} \frac{1}{\epsilon_{\text{Gap}}} \right) \quad (5.17)$$

*iterations, we have  $\mathbb{E}[P(w(\alpha^T)) - D(\alpha^T)] \leq \epsilon_{\text{Gap}}$ .*

On the other hand, if we would just average results (as proposed in [79]), we would obtain following corollary:

**Corollary 39** (Smooth loss functions - averaging). *Let the assumptions of Theorem 36 be satisfied. If we run Algorithm 8 with  $\nu = 1/K, \sigma' = 1$  for*

$$T \stackrel{(5.14)}{\geq} \frac{1}{1 - \Theta} \frac{K \lambda \gamma n + \sigma_{\max} K}{\lambda \gamma n} \log \left( \frac{1}{1 - \Theta} \frac{K \lambda \gamma n + \sigma_{\max} K}{\lambda \gamma n} \frac{1}{\epsilon_{\text{Gap}}} \right) \quad (5.18)$$

*iterations, we have  $\mathbb{E}[P(w(\alpha^T)) - D(\alpha^T)] \leq \epsilon_{\text{Gap}}$ .*

Comparing the leading terms in Equations (5.17) and (5.18), we see that the leading term for the  $\nu = 1$  choice is  $\mathcal{O}(\lambda \gamma n + \sigma_{\max} K)$ , which is always better than for the  $\nu = 1/K$  case, when the leading term is  $\mathcal{O}(K \lambda \gamma n + \sigma_{\max} K)$ . This strongly suggests that adding in Framework 9 is preferable, especially when  $\lambda \gamma n \gg \sigma_{\max}$ .

An analogous improvement (by a factor on the order of  $K$ ) follows for the case of the sub-linear convergence rate for general Lipschitz loss functions, as shown in Theorem 37.

Note that the differences in the convergence rate are bigger for relatively big values of the regularizer  $\lambda$ . When the regularizer is  $\mathcal{O}(1/n)$ , the difference is negligible. This behavior is also present in practice, as we will illustrate in Section 5.6.

## 5.5 Discussion and Related Work

In this section, we review a number of methods designed to solve optimization problems of the form of interest here, which are typically referred to as regularized empirical risk minimization (ERM) problems in the machine learning literature. This problem class (5.1), which is formally described in Section 5.2.1, underlies many prominent methods in supervised machine learning.

*Single-Machine Solvers.* Stochastic Gradient Descent (SGD) is the simplest stochastic method one can use to solve (5.1), and dates back to the work of Robbins and Monro [153]. We refer the reader to [120, 124, 125, 20] for a recent theoretical and practical assessment of SGD. Generally speaking, the method is extremely easy to implement, and converges to modest accuracies very quickly, which is often satisfactory in applications in machine learning. On the other hand, the method can sometimes be rather cumbersome because it can be difficult to tune its hyperparameters, and it can be impractical if higher solution accuracy is needed.

The current state of the art for empirical loss minimization with strongly convex regularizers is randomized coordinate ascent on the dual objective—Stochastic Dual Coordinate Ascent (SDCA) [163]. In contrast to primal SGD methods, the SDCA algorithm family is often preferred as it is free of learning-rate parameters, and has faster (geometric) convergence

guarantees. This algorithm and its variants are increasingly used in practice [186, 164]. On the other hand, primal-only methods apply to a larger problem class, not only of form (5.1) that enables formation of dual problem (5.2) as considered here.

Another class of algorithms gaining attention in recent very few years are ‘variance reduced’ modifications of the original SGD algorithm. They are applied directly to the primal problem (5.1), but unlike SGD, have the property that the variance of estimates of the gradients tend to zero as they approach the optimal solution. Algorithms such as SAG [158], SAGA [45] and others [159, 46] come at the cost of extra memory requirements—they have to store a gradient for each training example. This can be addressed efficiently in the case of generalized linear models, but prohibits its use in more complicated models such as in deep learning. On the other hand, Stochastic Variance Reduced Gradient (SVRG) and its variants [80, 89, 187, 81, 132] are often interpreted as ‘memory-free’ methods with variance reduction. However, these methods need to compute the full gradient occasionally to drive the variance reduction, which requires a full pass through the data and is an operation one generally tries to avoid. This and several other practical issues have been recently addressed in [74]. Finally, another class of extensions to SGD are stochastic quasi-Newton methods [17, 26]. Despite their clear potential, a lack of theoretical understanding and complicated implementation issues compared to those above may still limit their adoption in the wider community. A stochastic dual Newton ascent (SDNA) method was proposed and analyzed in [142]. However, the method needs to modified substantially before it can be implemented in a distributed environment.

*SGD-based Algorithms.* For the empirical loss minimization problems of interest, stochastic subgradient descent (SGD) based methods are well-established. Several distributed variants of SGD have been proposed, many of which build on the idea of a parameter server [133, 149, 53]. Despite their simplicity and accessibility in terms of implementation, the downside of this approach 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 1 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.

*One-Shot Communication Schemes.* At the other extreme, there are distributed methods using only a single round of communication, such as [198, 203, 114, 76, 75]. 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 [167]. Additional relevant lower bounds on the minimum number of communication rounds necessary for a given approximation quality are presented in [8, 7].

*Mini-Batch Methods.* Mini-batch methods (which instead of just one data-example use updates from several examples per iteration) are more flexible and lie within these two communication vs. computation extremes. However, mini-batch versions of both SGD and coordinate descent (CD) [151, 149, 164, 111, 188, 176, 150, 140, 141, 143, 38, 40] 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  $w$ , in contrast to methods that allow immediate local updates like CoCoA.

Another disadvantage of mini-batch methods is that the aggregation parameter is harder to tune, as it can lie anywhere in the order of mini-batch size. The optimal choice is often either unknown, or difficult to compute. In the CoCoA setting, the parameter lies in the typically much smaller range given by  $K$ . In this work the aggregation parameter is further simplified and can be simply set to 1, i.e., adding updates, which is achieved by formulating a more conservative local problem as described in Section 5.3.1.

*Distributed Batch Solvers.* With traditional batch gradient solvers not being competitive for the problem class (5.1), improved batch methods have also received much research attention

recently, in the single machine case as well as in the distributed setting. In distributed environments, popular methods include the alternating direction method of multipliers (ADMM) [23] as well as quasi-Newton methods such as L-BFGS, which can be attractive because of their relatively low communication requirements. Namely, communication is in the order of a constant number of vectors (the batch gradient information) per full pass through the data.

ADMM also comes with an additional penalty parameter balancing between the equality constraint on the primal variable vector  $w$  and the original optimization objective [23], which is typically hard to tune in many applications. Nevertheless, the method has been used for distributed SVM training in, e.g., [60]. The known convergence rates for ADMM are weaker than the more problem-tailored methods mentioned we study here, and the choice of the penalty parameter is often unclear in practice.

Standard ADMM and quasi-Newton methods do not allow a gradual trade-off between communication and computation available here. An exception is the approach of Zhang, Lee and Shin [194], which is similar to our approach in spirit, albeit based on ADMM, in that they allow for the subproblems to be solved inexactly. However, this work focuses on L2-regularized problems and a few selected loss functions, and offers no complexity results.

Interestingly, our proposed CoCoA<sup>+</sup> framework—despite being aimed at cheap stochastic local solvers—does have similarities to block-wise variants of batch proximal methods. In particular, the purpose of our subproblems as defined in (LO) is to form a data-dependent block-separable quadratic approximation to the smooth part of the original (dual) objective (5.2), while leaving the non-smooth part  $R$  intact (recall that  $R(\alpha)$  was defined to collect the  $\ell_i^*$  functions, and is separable over the coordinate blocks). Now if hypothetically each of our regularized quadratic subproblems (LO) were to be minimized exactly, the resulting steps could be interpreted as block-wise proximal Newton-type steps on each coordinate block  $k$  of the dual (5.2), where the Newton-subproblem is modified to also contain the proximal part  $R$ . This connection only holds for the special case of adding ( $\nu = 1$ ), and would correspond to a carefully adapted step-size in the block-wise Newton case.

One of the main crucial differences of our proposed CoCoA<sup>+</sup> framework compared to all known batch proximal methods (no matter if block-wise or not) is that the latter do require high accuracy subproblem solutions, and do not allow arbitrary solvers of weak accuracy  $\Theta$  such as we do here, see also the next paragraph. Distributed Newton methods have been analyzed theoretically only when the subproblems are solved to high precision, see e.g. [167]. This makes the local solvers very expensive and the convergence rates less general than in our framework (which allows weak local solvers). Furthermore, the analysis of [167] requires additional strong assumptions on the data partitioning, such that the local Hessian approximations are consistent between the machines.

*Distributed Methods Allowing Local Optimization.* Developing distributed optimization methods that allow for arbitrary weak local optimizers requires carefully devising data-local subproblems to be solved after each communication round.

By making use of the primal-dual structure in the line of work of [191, 137, 188, 189, 94], the CoCoA and CoCoA<sup>+</sup> frameworks proposed here are the first to allow the use of any local solver—of weak local approximation quality—in each round. Furthermore, the approach here also allows more control over the aggregation of updates between machines. The practical variant of the DisDCA algorithm of [188], called DisDCA-p, also allows additive updates but is restricted to coordinate decent (CD) being the local solver, and was initially proposed without convergence guarantees. The work of [189] has provided the first theoretical convergence analysis for an ideal case, when the distributed data parts are all orthogonal to each other, which is an unrealistic setting in practice. DisDCA-p can be recovered as a special case of the CoCoA<sup>+</sup> framework when using CD as a local solver, if  $|\mathcal{P}_k| = n/K$ , and when using the conservative bound  $\sigma' := K$ ; see also [94, 105]. The convergence theory presented here therefore also covers that method, and extends it to arbitrary local solvers.

Since the first version of this work, Accelerated Inexact Dane (AIDE) [147]—a method based on related set of ideas but applied to the primal problem—was developed. Like CoCoA<sup>+</sup>, AIDE promotes an efficient balance between communication and computation costs in the sense of (T-C).

*Inexact Block Coordinate Descent.* Our framework is related, but not identical, to running an *inexact* version of block coordinate ascent, applied to all blocks in parallel, and to the dual problem. From this perspective, the level of inexactness is controlled by the parameter  $\Theta$  through the use of a (possibly randomized) iterative ‘‘local’’ solver applied to the local subproblems. For previous work on *randomized* block coordinate descent we refer to the reader to [175] and [174].

## 5.6 Numerical Experiments

In this section we explore numerous aspects of our distributed framework and demonstrate its competitive performance in practice. Section 5.6.1 first explores the impact of the local solver on overall performance, by comparing examples of various local solvers that can be used in the framework (the improved CoCoA<sup>+</sup> framework as shown in Algorithms 8 and 9) as well as testing the effect of approximate solution quality. The results indicate that the choice of local solver can have a significant impact on overall performance. In Sections 5.6.2 and 5.6.3 we further explore framework parameters, looking at the impact of the aggregation parameter  $\nu$  and the subproblem parameter  $\sigma'$ , respectively. Finally, Section 5.6.5 demonstrates the competitive practical performance of the overall framework on a large 280GB distributed dataset.

We conduct experiments on three datasets of moderate and large size, namely *rcv1-test*, *epsilon* and *splice-site.t*<sup>5</sup>. The details of these datasets are listed in Table 5.2.

Dataset	$n$	$d$	size (GB)
rcv1-test	677,399	47,236	1.2
epsilon	400,000	2,000	3.1
splice-site.t	4,627,840	11,725,480	273.4

Table 5.2: Datasets used for numerical experiments.

For solving subproblems, we compare numerous local solver methods, as listed in Table 5.3. We use the Euclidean norm as the regularizer  $g(x) = \|x\|^2$  for all the experiments. All the algorithms are implemented in C++ with MPI, and experiments are run on a cluster of 4 Amazon EC2 m3.xlarge instances. Our open-source code is available online at: <https://github.com/optml/CoCoA>.

CD	Coordinate Descent [148]
APPROX	Accelerated, Parallel and Proximal Coordinate Descent [59]
GD	Gradient Descent with Backtracking Line Search [134]
CG	Conjugate Gradient Method [77]
L-BFGS	Quasi-Newton with Limited-Memory BFGS Updating [27]
BB	Barzilai-Borwein Gradient Method [9]
FISTA	Fast Iterative Shrinkage-Thresholding Algorithm [11]

Table 5.3: Local solvers used in numerical experiments.

### 5.6.1 Exploration of Local Solvers within the Framework

In this section we compare the performance of our framework for various local solvers and various choices of inner iterations performed by a given local solver, resulting in different local accuracy measures  $\Theta$ . For simplicity, we choose the subproblem parameter  $\sigma' := \nu K$  (see Lemma 34) as a simple obtainable and theoretically safe value.

<sup>5</sup>The datasets are available at <http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>.

## Comparison of Different Local Solvers

Here we compare the performance of the seven local solvers listed in Table 5.3. We show results for the quadratic loss function  $\ell_i(a) = \frac{1}{2}(a - y_i)^2$  with three different values of the regularization parameter,  $\lambda = 10^{-3}$ ,  $10^{-4}$ , and  $10^{-5}$ , and  $g(\cdot)$  being the default Euclidean squared norm regularizer:  $g(\cdot) = \frac{1}{2}\|\cdot\|^2$ . The dataset is rcv1\\_test and we ran the CoCoA<sup>+</sup> framework for a maximum of  $T := 100$  communication rounds. We set  $\nu = 1$  (adding) and choose  $H$  which gave the best performance in CPU time (see Table 5.4) for each solver.

Local Solver	CD	APPROX	GD	CG	L-BFGS	BB	FISTA
$H$	40,000	40,000	20	5	10	15	20

Table 5.4: Optimal  $H$  for different local solvers and the rcv1\\_test dataset.

From Figure 5.2, we find that if a high-enough accuracy solution is needed, the coordinate descent (CD) local solver always outperforms the other solvers. However, when a low accuracy solution is sufficient, as is often the case in machine learning applications, and if the regularization parameter is not too small, then L-BFGS performs best. The local subproblems arising with the rcv1\\_test dataset are reasonably well conditioned. If more ill-conditioning was present, however, we would expect the APPROX local solver to do better than CD. This is because this method is an *accelerated* variant of CD. In summary, randomized methods, such as CD and APPROX, and quasi-Newton methods (L-BFGS), perform best on this dataset.

Based on the above observations, it seems reasonable to expect that a method combining the power of both of these successful approaches—randomization and second-order information—would perform even better. One might therefore want to look at local solvers based on ideas appearing in [142] or [67].

Note that it is not the goal of this work to decide on what the best local solver is. Our goals are quite the opposite, we provide a framework which allows the incorporation of *any* local solver. This choice might depend on which solvers are readily available to the practitioner/company. It will also depend on the conditioning of the local subproblems, their size, and other similar considerations. Future research will undoubtedly lead to the development of new and better local solvers which can be incorporated within CoCoA<sup>+</sup>.

Finally, note that some of the solvers cannot guarantee strict decrease of the duality gap, and sometimes this fluctuation can be very dramatic.

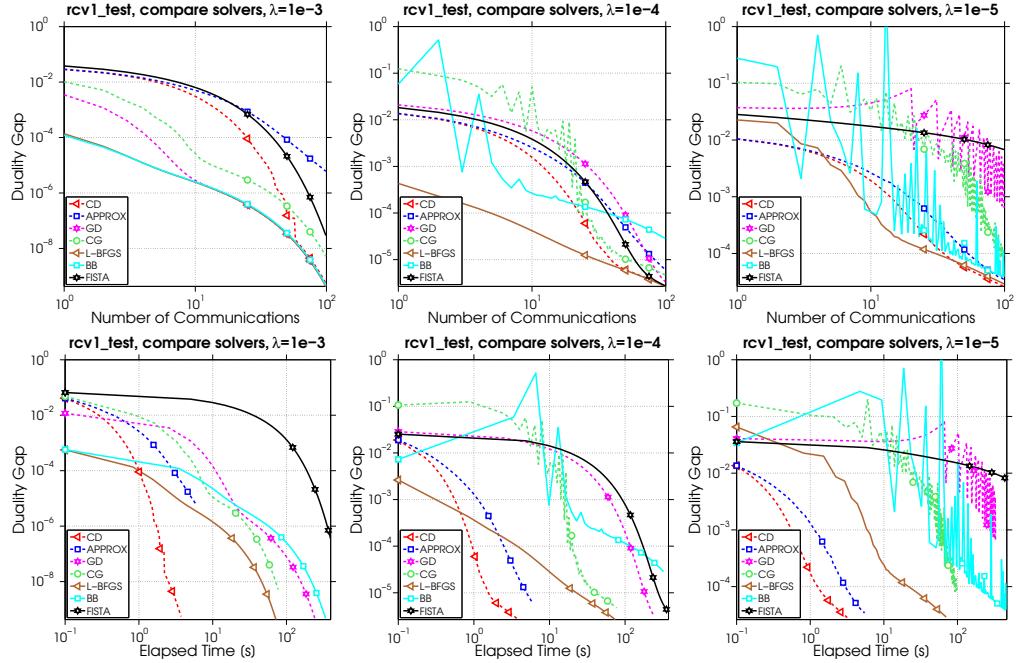


Figure 5.2: Performance of 7 local solvers on rcv1\_test dataset for three values of the regularization parameter.

### Effect of the Quality of Local Solver Solutions on Overall Performance

Here we discuss how the quality of subproblem solutions affects the overall performance of Algorithm 9. In order to do so, we denote  $H$  as the number of iterations the local solver is run for, within each communication round of the framework. We choose various values for  $H$  for the two local solvers that had the best performance in general, CD [148, 163] and L-BFGS [27]. For CD,  $H$  represents the number of local iterations performed on the subproblem. For L-BFGS,  $H$  not only means the number of iterations, but also stands for the size of past information used to approximate the Hessian (i.e., the size of limited memory).

Looking at Figures 5.3 and 5.4, we see that for both of these local solvers and all values of  $\lambda$ , increasing  $H$  will lead to less iterations of Algorithm 9. Of course, increasing  $H$  comes at the cost of the time spent on local solvers increasing. Hence, a larger value of  $H$  is not always the optimal choice with respect to total elapsed time. For example, for the rcv1\_test dataset, when choosing CD to solve the subproblems, choosing  $H$  to be 40,000 uses less time and provides faster convergence. When using L-BFGS,  $H = 10$  seems to be the best choice.

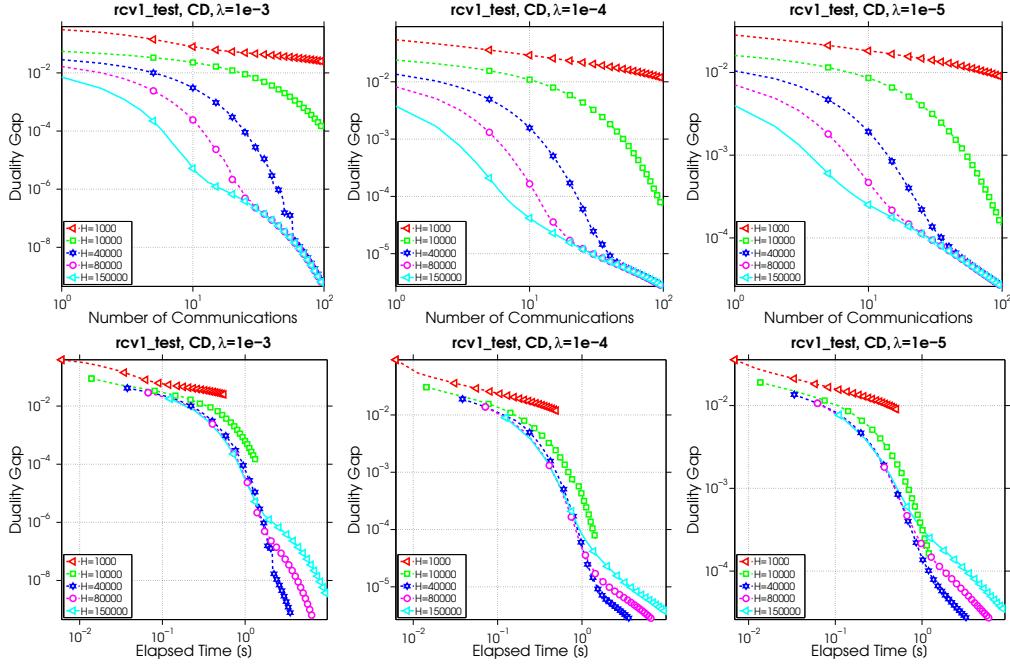


Figure 5.3: Varying the number of iterations of CD as a local solver.

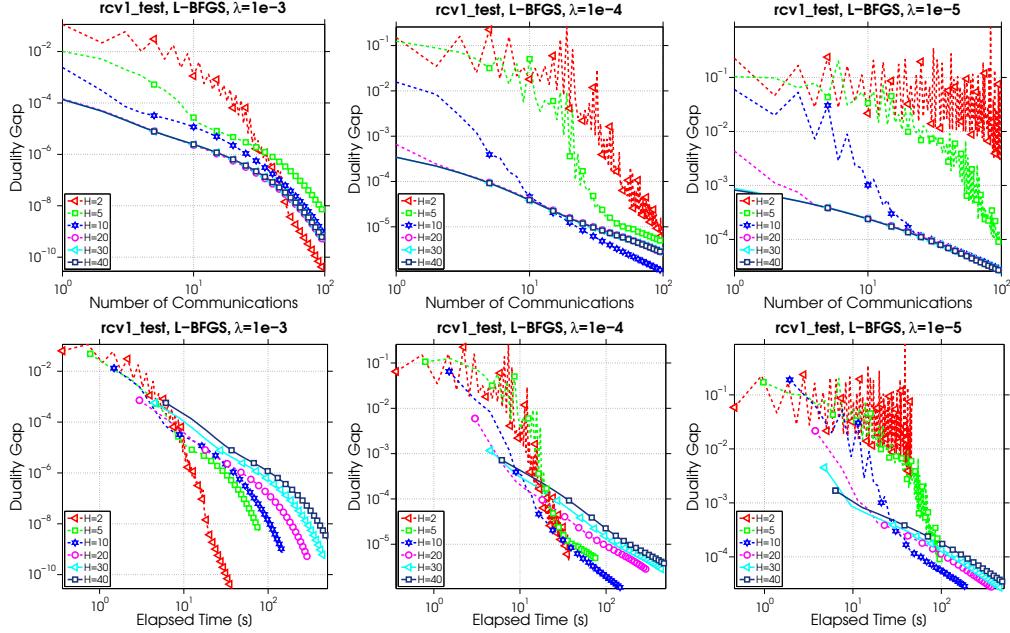


Figure 5.4: Varying the number of iterations of L-BFGS as a local solver.

### 5.6.2 Averaging vs. Adding the Local Updates

In this section, we compare the performance of our algorithm using two different schemes for aggregating partial updates: adding vs. averaging. This corresponds to comparing two extremes for the parameter  $\nu$ , either  $\nu := \frac{1}{K}$  (averaging partial solutions) or  $\nu := 1$  (adding partial solutions). As discussed in Section 5.4, adding the local updates ( $\nu = 1$ ) will lead to less iterations than averaging, due to choosing different  $\sigma'$  in the subproblems. We verify this experimentally by considering several of the local solvers listed in Table 5.3.

We show results for the rcv1\_test dataset, and we apply the quadratic loss function with

three different choices for the regularization parameter,  $\lambda=1e-03$ ,  $1e-04$ , and  $1e-05$ . The experiments in Figures 5.6–5.11 indicate that the “adding” strategy will always lead to faster convergence than averaging, even though the difference is minimal when we apply a large number of iterations in the local solver. All the blue solid plots (adding) outperform the red dashed plots (averaging), which indicates the advantage of choosing  $\nu = 1$ . Another note here is that for smaller  $\lambda$ , we will have to spend more iterations to get the same accuracy, because the original objective function (5.1) is less strongly convex.

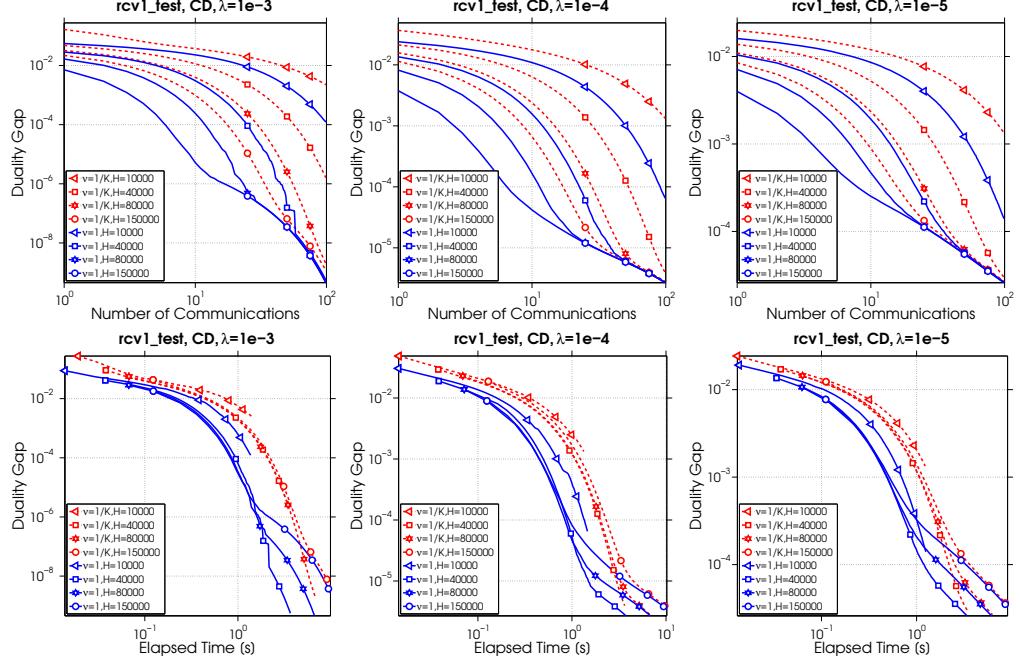


Figure 5.5: Adding (blue solid line) vs Averaging (red dashed line) for CD as the local solver.

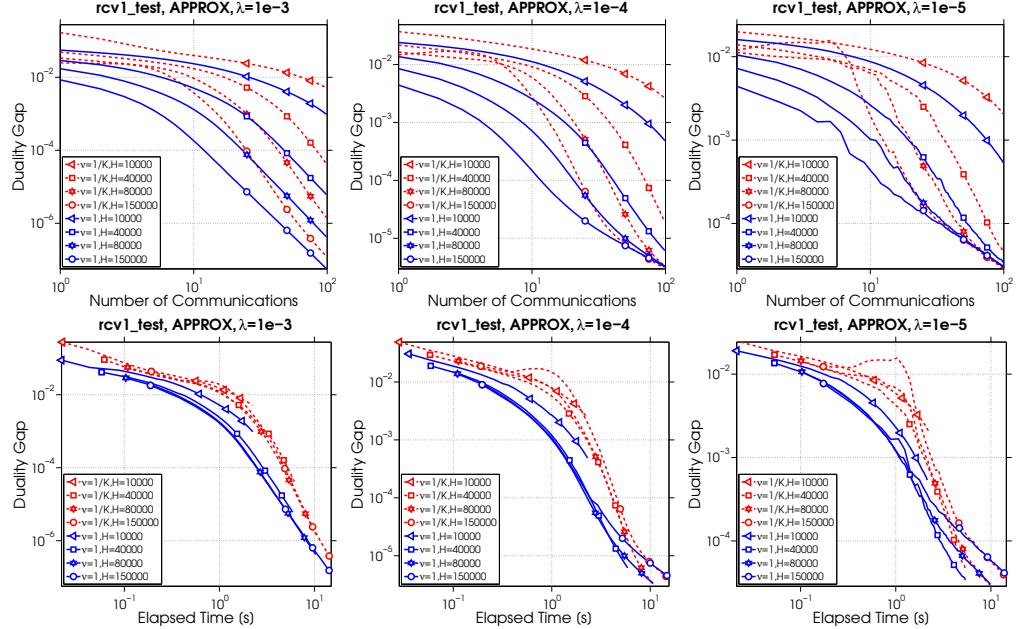


Figure 5.6: Adding (blue solid line) vs Averaging (red dashed line) for APPROX as the local solver.

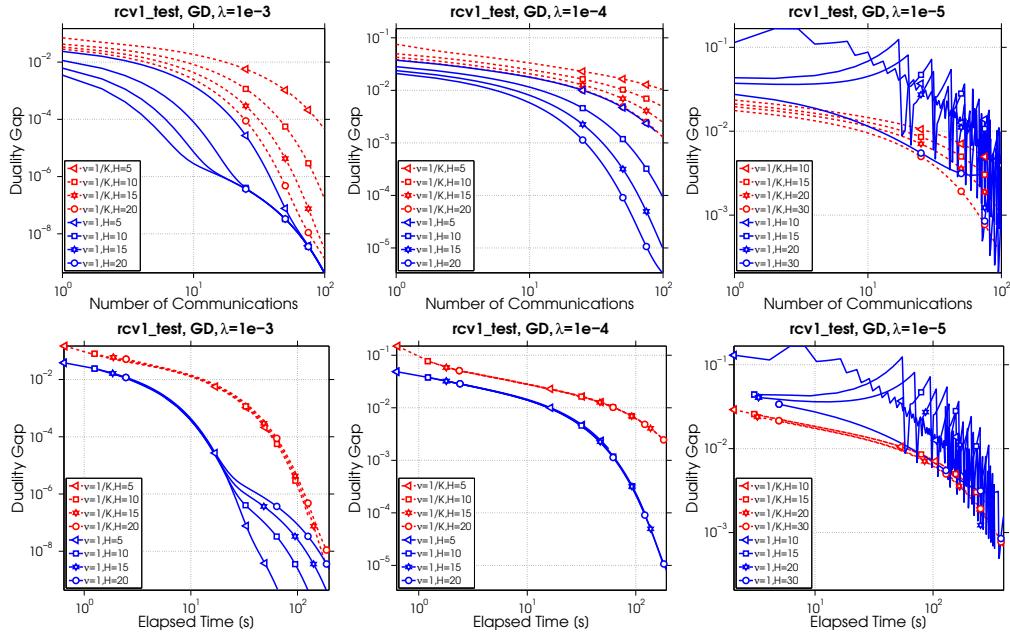


Figure 5.7: Adding (blue solid line) vs Averaging (red dashed line) for Gradient Descent as the local solver.

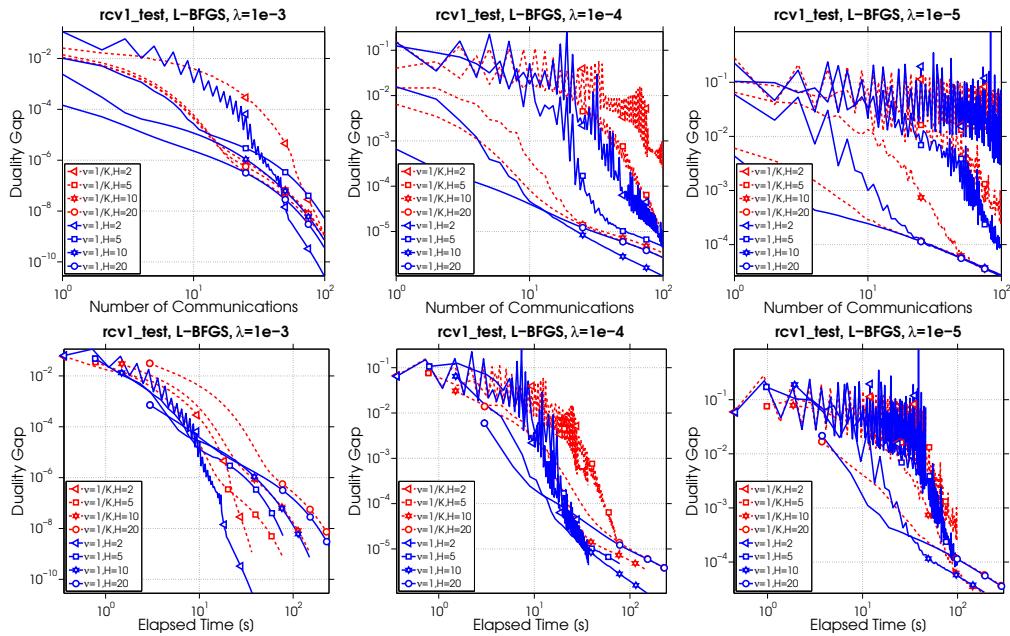


Figure 5.8: Adding (blue solid line) vs Averaging (red dashed line) for L-BFGS as the local solver.

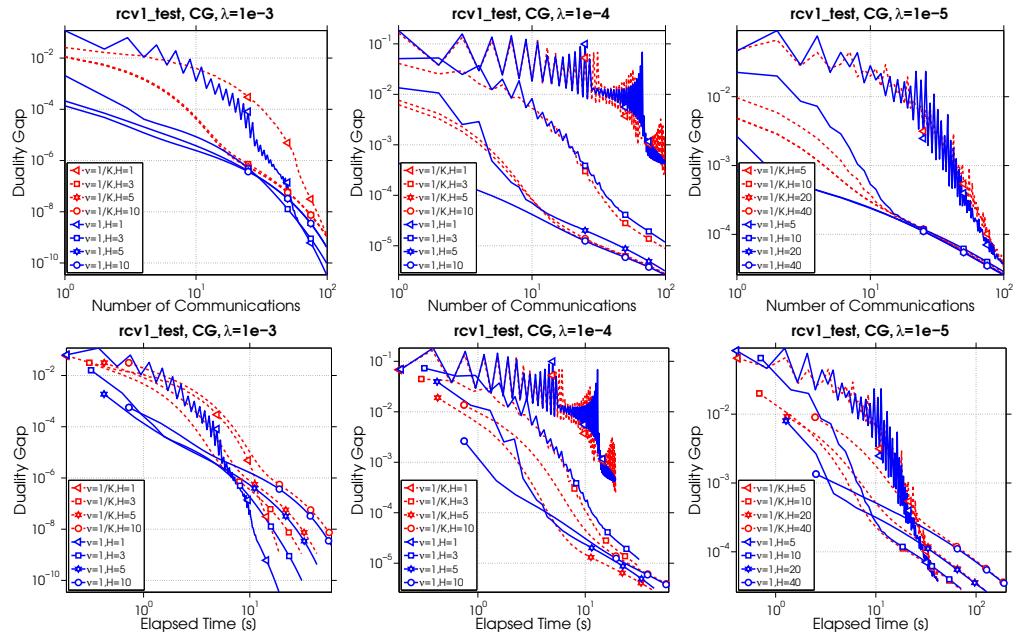


Figure 5.9: Adding (blue solid line) vs Averaging (red dashed line) for Conjugate Gradient Method as the local solver.

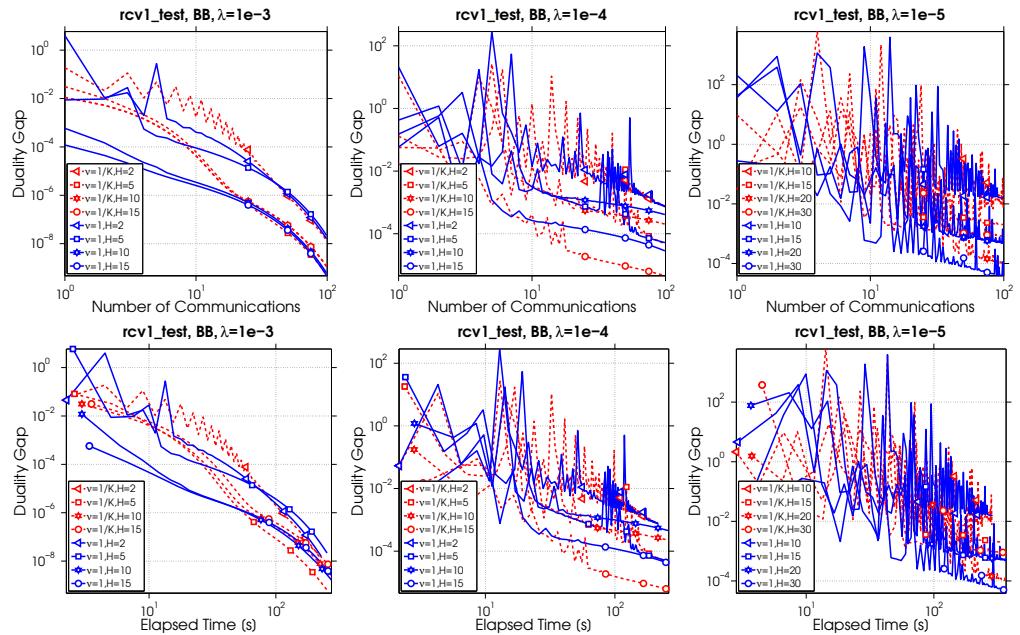


Figure 5.10: Adding (blue solid line) vs Averaging (red dashed line) for BB as the local solver.

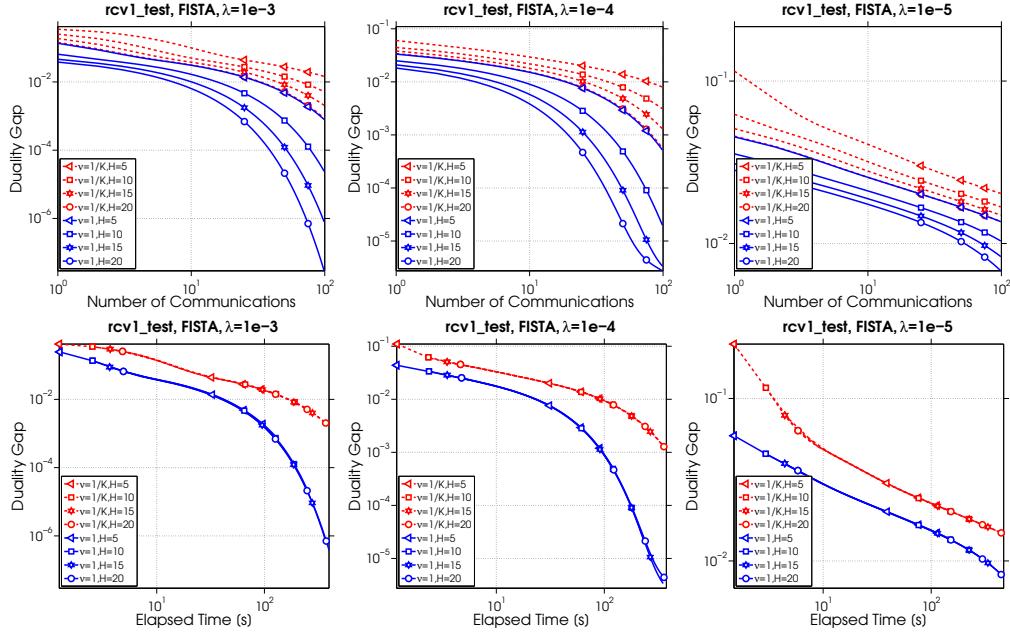


Figure 5.11: Adding (blue solid line) vs Averaging (red dashed line) for FISTA as the local solver.

### 5.6.3 The Effect of the Subproblem Parameter $\sigma'$

In this section we consider the effect of the choice of the subproblem parameter on convergence (Figure 5.12). We plot the duality gap over the number of communications for the rcv1\_test and epsilon datasets with quadratic loss, and set  $K = 8$ ,  $\lambda = 10^{-5}$ . For  $\nu = 1$  (adding the local updates), we consider several different values of  $\sigma'$ , ranging from 1 to 8. The value  $\sigma' = 8$  represents the safe upper bound of  $\nu K$ , as given in Lemma 34.

Decreasing  $\sigma'$  improves performance in terms of communication until a certain point, after which the algorithm diverges. For the rcv1\_test dataset, the optimal convergence occurs around  $\sigma' = 5$ , and diverges fast for  $\sigma' \leq 3$ . For the epsilon dataset,  $\sigma'$  around 6 is the best choice and the algorithm will not converge to the optimal solution if  $\sigma' \leq 5$ . However, more importantly, the “safe” upper bound of  $\sigma' := \nu K = 8$  has only slightly worse performance than the practically best (but “un-safe”) value of  $\sigma'$ .

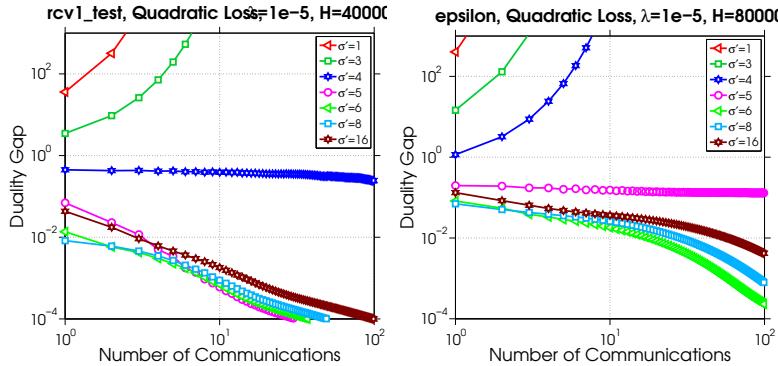


Figure 5.12: The effect of  $\sigma'$  on convergence for the rcv1\_test and epsilon datasets distributed across 8 machines.

### 5.6.4 Scaling Property

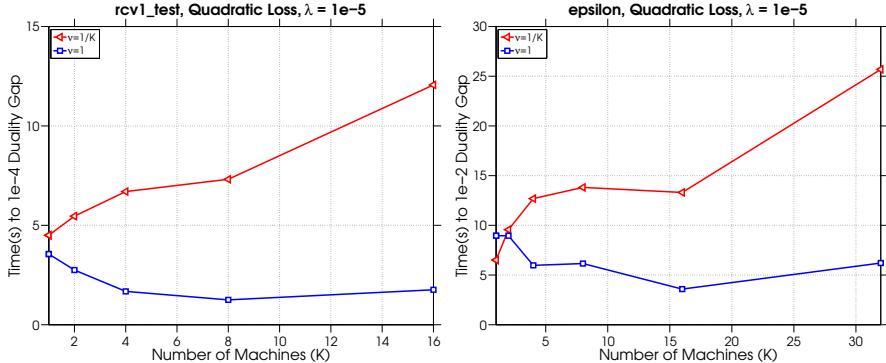


Figure 5.13: The effect of increasing the number of machines  $K$  on the time (s) to reach a solution with expected duality gap.

Here we demonstrate the ability of our framework to scale with  $K$  (number of machines). We compare the runtime to reach a specific tolerance on duality gap ( $10^{-4}$  and  $10^{-2}$ ) for two choices of  $\nu$ . Looking at Figure 5.13, we see that when choosing  $\nu = 1$ , the performance improves as the number of machines increases. However, when  $\nu = \frac{1}{K}$ , the algorithm slows down as  $K$  increases. These observations support our analysis in Section 4.

### 5.6.5 Performance on a Big Dataset

As shown in Figure 5.14, we test the algorithm on the *splice-site.t* dataset, whose size is about 280 GB. We show experiments for three different loss functions  $\ell$ , namely logistic loss, hinge loss and least squares loss (see Table 5.1). We set  $\lambda = 10^{-6}$  for the squared norm regularizer. The dataset is distributed across  $K = 4$  machines and we use CD as the local solver with  $H = 50,000$ . In all the cases, an optimal solution can be reached in about 20 minutes and again, we observe that setting the aggregation parameter  $\nu := 1$  leads to faster convergence than  $\nu := \frac{1}{K}$  (averaging).

Also, the number of communication rounds for the three different loss functions are almost the same if we set all the other parameters to be same. However, the duality gap decreases in a different manner for the three loss functions.

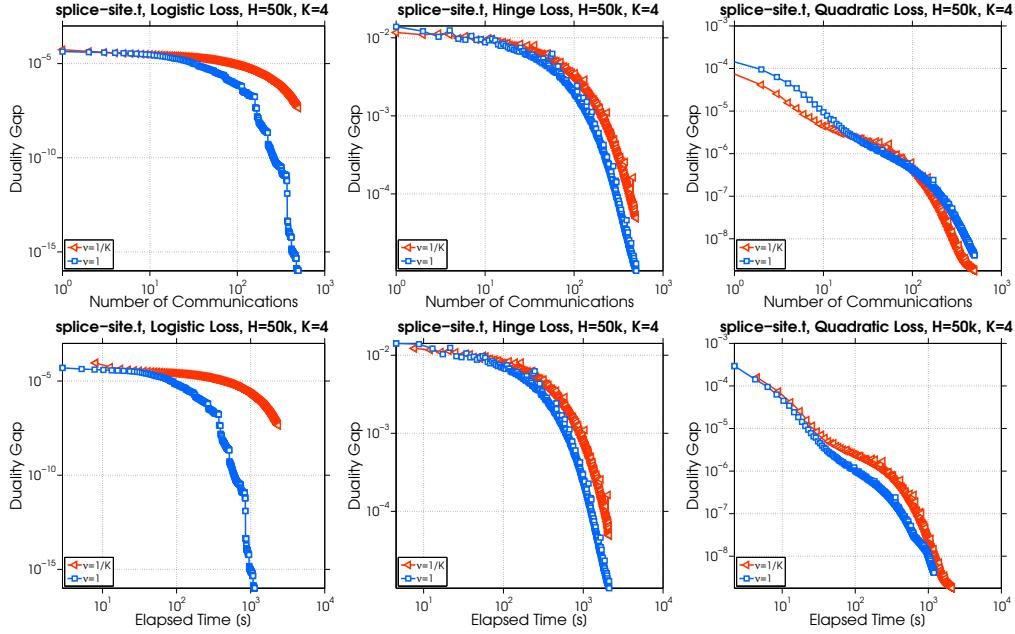


Figure 5.14: Performance of Algorithm 9 on *splice-site.t* dataset, with three different loss functions.

### 5.6.6 Comparison with other distributed methods

Finally, we compare the CoCoA<sup>+</sup> framework with several competing distributed optimization algorithms. The DiSCO algorithm [199] is a Newton-type method, where in each iteration the updates on iterates are computed inexactly using a Preconditioned Conjugate Gradients (PCG) method. As suggested in [199], in our implementation of DISCO we apply the Stochastic Average Gradient (SAG) method [158] as the solver to get the initial solutions for each local machine and solve the linear system during PCG. DiSCO-F [107], improves on the computational efficiency of original DiSCO, by partitioning the data across features rather than examples. The DANE algorithm [167] is another distributed Newton-type method, where in each iteration there are two rounds of communication. Also, a subproblem is to be solved in each iteration to obtain updates. For each of these algorithms, we tune the hyperparameters manually to optimize performance.

The experiments are conducted on a compute cluster with  $K = 4$  machines. We run all algorithms using two datasets. Since not all methods are primal-based in nature, it is difficult to continue using duality gap as a measure of optimality. Instead, the norm of the gradient of the primal objective function (5.1) is used to compare the relative quality of the solutions obtained.

As shown in Figure 5.15, in terms of the number of communications, CoCoA<sup>+</sup> usually converges more rapidly than competing methods during the early iterations, but tends to get slower later on in the iterative process. This illustrates that the Newton-type methods can accelerate in the vicinity of the optimal solution, as expected. However, CoCoA<sup>+</sup> can still beat other methods in running time. The main reason for this is the fact that the subproblems in our framework can be solved more efficiently, compared with DiSCO and DANE.

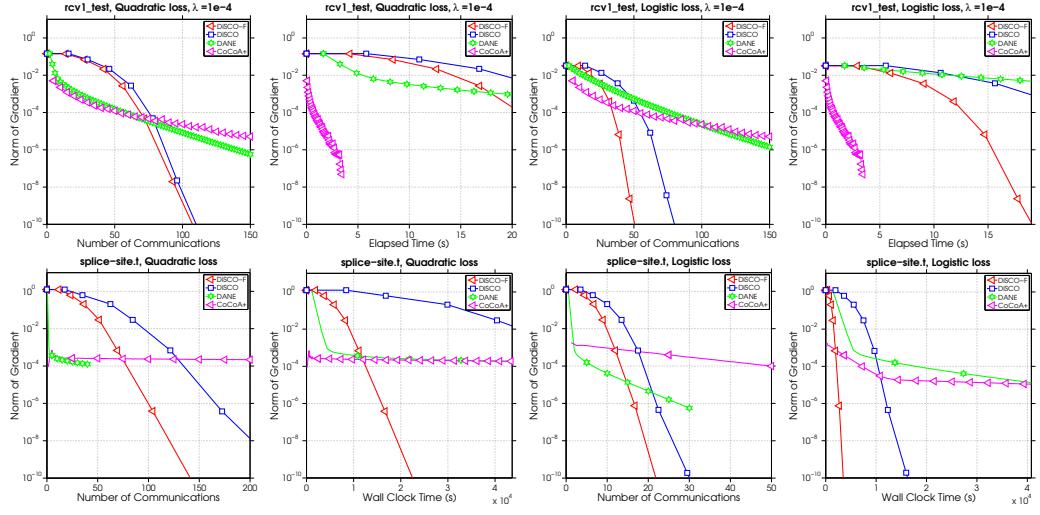


Figure 5.15: Performance of several distributed frameworks on solving (5.1) with different losses on two datasets.

## 5.7 Conclusion

We present CoCoA<sup>+</sup>, a novel framework that enables fast and communication-efficient *additive aggregation* in distributed primal-dual optimization. We analyze the theoretical complexity of CoCoA<sup>+</sup>, giving strong primal-dual convergence rates with outer iterations scaling independently of the number of machines. We extended the basic theory to allow for non-smooth loss functions, arbitrary strongly convex regularizers, and primal-dual convergence results. Our experimental results show significant speedups in terms of runtime over previous methods, including the original CoCoA framework as well as other state-of-the-art methods.

## 5.8 Proofs

### 5.8.1 Proof of Lemma 31

Since  $g$  is 1-strongly convex,  $g^*$  is 1-smooth, and thus we can use (5.7) as follows

$$\begin{aligned}
 f(\alpha + h) &= \lambda g^* \left( \frac{1}{\lambda n} X \alpha + \frac{1}{\lambda n} X h \right) \\
 &\stackrel{(5.7)}{\leq} \lambda \left( g^* \left( \frac{1}{\lambda n} X \alpha \right) + \left\langle \nabla g^* \left( \frac{1}{\lambda n} X \alpha \right), \frac{1}{\lambda n} X h \right\rangle + \frac{1}{2} \left\| \frac{1}{\lambda n} X h \right\|^2 \right) \\
 &= f(\alpha) + \langle \nabla f(\alpha), h \rangle + \frac{1}{2\lambda n^2} h^T X^T X h.
 \end{aligned}$$

### 5.8.2 Proof of Lemma 33

Indeed,

$$\begin{aligned}
D(\alpha + \nu \sum_{k=1}^K h_{[k]}) &= D(\alpha + \nu h) \\
&\stackrel{(5.2)}{=} \frac{1}{n} \sum_{i=1}^n -\ell_i^*(-\alpha_i - \nu h_i) - \lambda g^* \left( \frac{1}{\lambda n} X(\alpha + \nu h) \right) \\
&\stackrel{(5.3)}{=} \frac{1}{n} \sum_{i=1}^n -\ell_i^*(-\alpha_i - \nu h_i) - f(\alpha + \nu h) \\
&\stackrel{(5.8)}{\geq} \frac{1-\nu}{n} \sum_{i=1}^n -\ell_i^*(-\alpha_i) + \nu \frac{1}{n} \sum_{i=1}^n -\ell_i^*(-\alpha_i - h_i) \\
&\quad - f(\alpha) - \nu \langle \nabla f(\alpha), h \rangle - \nu^2 \frac{1}{2\lambda n^2} h^T X^T X h \\
&\stackrel{(5.2),(5.10)}{\geq} (1-\nu)D(\alpha) - \nu \sum_{k=1}^K R_k (\alpha_{[k]} + h_{[k]}) \\
&\quad - \nu \frac{1}{K} \sum_{k=1}^K f(\alpha) - \nu \sum_{k=1}^K \langle \nabla f(\alpha), h_{[k]} \rangle - \nu \sigma' \frac{1}{2\lambda n^2} h^T G h \\
&\stackrel{(LO)}{=} (1-\nu)D(\alpha) + \nu \frac{1}{K} \mathcal{G}_k^{\sigma'}(h_{[k]}; \alpha),
\end{aligned}$$

where the first inequality follows from Jensen's inequality and the last equality follows from the block diagonal definition of  $G$  given in (5.11), i.e.

$$h^T G h = \sum_{k=1}^K h_{[k]} X_{[k]}^T X_{[k]} h_{[k]}. \quad (5.19)$$

### 5.8.3 Proof of Lemma 34

Considering  $h \in \mathbb{R}^n$  with zeros in all coordinates except those that belong to the  $k$ -th block  $\mathcal{P}_k$ , we have  $h^T X^T X h = h^T G h$ , and thus  $\sigma' \geq \nu$ . Let  $h_{[k,l]}$  denote  $h_{[k]} - h_{[l]}$ . Since  $X^T X$  is a positive semi-definite matrix, for  $k, l \in \{1, \dots, K\}$ ,  $k \neq l$  we have

$$0 \leq h_{[k,l]}^T X^T X h_{[k,l]} = h_{[k]}^T X^T X h_{[k]} + h_{[l]}^T X^T X h_{[l]} - 2h_{[k]}^T X^T X h_{[l]}. \quad (5.20)$$

By taking any  $h \in \mathbb{R}^n$  for which  $h^T G h \leq 1$ , in view of (5.10), we get

$$\begin{aligned}
h^T X^T X h &= \sum_{k=1}^K \sum_{l=1}^K h_{[k]}^T X^T X h_{[l]}^T = \sum_{k=1}^K h_{[k]}^T X^T X h_{[k]}^T + \sum_{k \neq l} h_{[k]}^T X^T X h_{[l]}^T \\
&\stackrel{(5.20)}{\leq} \sum_{k=1}^K h_{[k]}^T X^T X h_{[k]}^T + \sum_{k \neq l} \frac{1}{2} [h_{[k]}^T X^T X h_{[k]} + h_{[l]}^T X^T X h_{[l]}] \\
&= K \sum_{k=1}^K h_{[k]}^T X^T X h_{[k]} = K h^T G h \leq K.
\end{aligned}$$

Therefore we can conclude that  $\nu h^T X^T X h \leq \nu K$  for all  $h$  included in the definition (5.10) of  $\sigma'_{\min}$ , proving the claim.

### 5.8.4 Proofs of Theorems 36 and 37

Before we state the proofs of the main theorems, we will write and prove a few crucial lemmas.

**Lemma 40.** Let  $\ell_i^*$  be strongly<sup>6</sup> convex with convexity parameter  $\gamma \geq 0$  with respect to the norm  $\|\cdot\|$ ,  $\forall i \in [n]$ . Then for all iterations  $t$  of Algorithm 8 under Assumption 35, and any  $s \in [0, 1]$ , it holds that

$$\mathbb{E} [D(\alpha^{t+1}) - D(\alpha^t)] \geq \nu(1 - \Theta) \left( s \text{Gap}(\alpha^t) - \frac{\sigma' s^2}{2\lambda n^2} R^t \right), \quad (5.21)$$

where

$$R^t := -\frac{\lambda \gamma n(1-s)}{\sigma' s} \|u^t - \alpha^t\|^2 + \sum_{k=1}^K \|X(u^t - \alpha^t)_{[k]}\|^2, \quad (5.22)$$

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

$$-u_i^t \in \partial \ell_i(w(\alpha^t)^T x_i). \quad (5.23)$$

*Proof.* For sake of notation, we will write  $\alpha$  instead of  $\alpha^t$ ,  $w$  instead of  $w(\alpha^t)$  and  $u$  instead of  $u^t$ .

Now, let us estimate the expected change of the dual objective. Using the definition of the dual update  $\alpha^{t+1} := \alpha^t + \nu \sum_k h_{[k]}$  resulting in Algorithm 9, we have

$$\begin{aligned} \mathbb{E} [D(\alpha^t) - D(\alpha^{t+1})] &= \mathbb{E} \left[ D(\alpha) - D(\alpha + \nu \sum_{k=1}^K h_{[k]}) \right] \\ &\stackrel{(5.12)}{\leq} \mathbb{E} \left[ D(\alpha) - (1 - \nu) D(\alpha) - \nu \sum_{k=1}^K \mathcal{G}_k^{\sigma'}(h_{[k]}^t; \alpha) \right] \\ &= \nu \mathbb{E} \left[ D(\alpha) - \sum_{k=1}^K \mathcal{G}_k^{\sigma'}(h_{[k]}^t; \alpha) \right] \\ &= \nu \mathbb{E} \left[ D(\alpha) - \sum_{k=1}^K \mathcal{G}_k^{\sigma'}(h_{[k]}^*; \alpha) + \sum_{k=1}^K \mathcal{G}_k^{\sigma'}(h_{[k]}^*; \alpha) - \sum_{k=1}^K \mathcal{G}_k^{\sigma'}(h_{[k]}^t; \alpha) \right] \\ &\stackrel{(5.13)}{\leq} \nu \left( D(\alpha) - \sum_{k=1}^K \mathcal{G}_k^{\sigma'}(h_{[k]}^*; \alpha) + \Theta \left( \underbrace{\sum_{k=1}^K \mathcal{G}_k^{\sigma'}(h_{[k]}^*; \alpha) - \sum_{k=1}^K \mathcal{G}_k^{\sigma'}(\mathbf{0}; \alpha)}_{D(\alpha)} \right) \right) \\ &= \nu(1 - \Theta) \left( \underbrace{D(\alpha) - \sum_{k=1}^K \mathcal{G}_k^{\sigma'}(h_{[k]}^*; \alpha)}_C \right). \end{aligned} \quad (5.24)$$

---

<sup>6</sup>Note that the case of weakly convex  $\ell_i^*(.)$  is explicitly allowed here as well, as the Lemma holds for the case  $\gamma = 0$ .

Now, let us upper bound the  $C$  term (we will denote by  $h^* = \sum_{k=1}^K h_{[k]}^*$ ):

$$\begin{aligned}
C &\stackrel{(5.2), (\text{LO})}{=} \frac{1}{n} \sum_{i=1}^n (\ell_i^*(-\alpha_i - h_i^*) - \ell_i^*(-\alpha_i)) + \langle \nabla f(\alpha), h \rangle + \sum_{k=1}^K \frac{\lambda}{2} \sigma' \left\| \frac{1}{\lambda n} X h_{[k]}^* \right\|^2 \\
&\leq \frac{1}{n} \sum_{i=1}^n (\ell_i^*(-\alpha_i - s(u_i - \alpha_i)) - \ell_i^*(-\alpha_i)) + \langle \nabla f(\alpha), s(u - \alpha) \rangle \\
&\quad + \sum_{k=1}^K \frac{\lambda}{2} \sigma' \left\| \frac{1}{\lambda n} X s(u - \alpha)_{[k]} \right\|^2 \\
&\stackrel{\text{Strong conv.}}{\leq} \frac{1}{n} \sum_{i=1}^n \left( s\ell_i^*(-u_i) + (1-s)\ell_i^*(-\alpha_i) - \frac{\gamma}{2}(1-s)s(u_i - \alpha_i)^2 - \ell_i^*(-\alpha_i) \right) \\
&\quad + \langle \nabla f(\alpha), s(u - \alpha) \rangle + \sum_{k=1}^K \frac{\lambda}{2} \sigma' \left\| \frac{1}{\lambda n} X s(u - \alpha)_{[k]} \right\|^2 \\
&= \frac{1}{n} \sum_{i=1}^n \left( s\ell_i^*(-u_i) - s\ell_i^*(-\alpha_i) - \frac{\gamma}{2}(1-s)s(u_i - \alpha_i)^2 \right) \\
&\quad + \langle \nabla f(\alpha), s(u - \alpha) \rangle + \sum_{k=1}^K \frac{\lambda}{2} \sigma' \left\| \frac{1}{\lambda n} X s(u - \alpha)_{[k]} \right\|^2.
\end{aligned}$$

The convex conjugate maximal property implies that

$$\ell_i^*(-u_i) \stackrel{(5.23)}{=} -u_i w(\alpha)^T x_i - \ell_i(w(\alpha)^T x_i). \quad (5.25)$$

Moreover, from the definition of the primal and dual optimization problems (5.1), (5.2), we can write the duality gap as

$$\begin{aligned}
\text{Gap}(\alpha) &:= P(w(\alpha)) - D(\alpha) \\
&\stackrel{(5.1), (5.2)}{=} \frac{1}{n} \sum_{i=1}^n (\ell_i(x_i^T w(\alpha)) + \ell_i^*(-\alpha_i)) + \lambda g(w(\alpha)) + \lambda g^*(v(\alpha)) \\
&= \frac{1}{n} \sum_{i=1}^n (\ell_i(x_i^T w(\alpha)) + \ell_i^*(-\alpha_i)) + \lambda g(\nabla g^*(v(\alpha))) + \lambda g^*(v(\alpha)) \\
&= \frac{1}{n} \sum_{i=1}^n (\ell_i(x_i^T w(\alpha)) + \ell_i^*(-\alpha_i)) + \lambda v(\alpha)^T w(\alpha) \\
&= \frac{1}{n} \sum_{i=1}^n (\ell_i(x_i^T w(\alpha)) + \ell_i^*(-\alpha_i) + w(\alpha)^T x_i \alpha_i).
\end{aligned} \quad (5.26)$$

Hence,

$$\begin{aligned}
C &\stackrel{(5.25)}{\leq} \frac{1}{n} \sum_{i=1}^n \left( -su_i w(\alpha)^T x_i - s\ell_i(w(\alpha)^T x_i) - s\ell_i^*(-\alpha_i) \underbrace{-sw(\alpha)^T x_i \alpha_i + sw(\alpha)^T x_i \alpha_i}_0 \right) \\
&\quad + \frac{1}{n} \sum_{i=1}^n \left( -\frac{\gamma}{2}(1-s)s(u_i - \alpha_i)^2 \right) + \langle \nabla f(\alpha), s(u - \alpha) \rangle + \sum_{k=1}^K \frac{\lambda}{2} \sigma' \left\| \frac{1}{\lambda n} X s(u - \alpha)_{[k]} \right\|^2 \\
&= \frac{1}{n} \sum_{i=1}^n (-s\ell_i(w(\alpha)^T x_i) - s\ell_i^*(-\alpha_i) - sw(\alpha)^T x_i \alpha_i) \\
&\quad + \frac{1}{n} \sum_{i=1}^n \left( sw(\alpha)^T x_i (\alpha_i - u_i) - \frac{\gamma}{2}(1-s)s(u_i - \alpha_i)^2 \right) \\
&\quad + \frac{1}{n} w(\alpha)^T X s(u - \alpha) + \sum_{k=1}^K \frac{\lambda}{2} \sigma' \left\| \frac{1}{\lambda n} X s(u - \alpha)_{[k]} \right\|^2 \\
&\stackrel{(5.26)}{=} -sGap(\alpha) - \frac{\gamma}{2}(1-s)s \frac{1}{n} \sum_{i=1}^n \|u - \alpha\|^2 + \frac{\sigma' s^2}{2\lambda n^2} \sum_{k=1}^K \|X(u - \alpha)_{[k]}\|^2. \tag{5.27}
\end{aligned}$$

Now, the claimed improvement bound (5.21) follows by plugging (5.27) into (5.24).  $\square$

**Lemma 41.** *If  $\ell_i$  are  $L$ -Lipschitz continuous for all  $i \in [n]$ , then*

$$\forall t : R^t \leq 4L^2 \underbrace{\sum_{k=1}^K \sigma_k |\mathcal{P}_k|}_{=: \sigma}, \tag{5.28}$$

where

$$\sigma_k \stackrel{\text{def}}{=} \max_{\alpha_{[k]} \in \mathbb{R}^n} \frac{\|X_{[k]} \alpha_{[k]}\|^2}{\|\alpha_{[k]}\|^2}. \tag{5.29}$$

*Proof.* For general convex functions, the strong convexity parameter is  $\gamma = 0$ , and hence the definition of  $R^t$  becomes

$$R^t \stackrel{(5.22)}{=} \sum_{k=1}^K \|X(u^t - \alpha^t)_{[k]}\|^2 \stackrel{(5.29)}{\leq} \sum_{k=1}^K \sigma_k \|(u^t - \alpha^t)_{[k]}\|^2 \stackrel{[163, \text{ Lemma 3}]}{\leq} \sum_{k=1}^K \sigma_k |\mathcal{P}_k| 4L^2.$$

$\square$

### Proof of Theorem 37

At first let us estimate expected change of dual feasibility. By using the main Lemma 40, we have

$$\begin{aligned}
\mathbb{E} [D(\alpha^*) - D(\alpha^{t+1})] &= \mathbb{E} [D(\alpha^*) - D(\alpha^{t+1}) + D(\alpha^t) - D(\alpha^t)] \\
&\stackrel{(5.21)}{=} D(\alpha^*) - D(\alpha^t) - \nu(1-\Theta)sGap(\alpha^t) + \nu(1-\Theta) \frac{\sigma'}{2\lambda} \left( \frac{s}{n} \right)^2 R^t \\
&\stackrel{(5.4)}{=} D(\alpha^*) - D(\alpha^t) - \nu(1-\Theta)s(P(w(\alpha^t)) - D(\alpha^t)) + \nu(1-\Theta) \frac{\sigma'}{2\lambda} \left( \frac{s}{n} \right)^2 R^t \\
&\leq D(\alpha^*) - D(\alpha^t) - \nu(1-\Theta)s(D(\alpha^*) - D(\alpha^t)) + \nu(1-\Theta) \frac{\sigma'}{2\lambda} \left( \frac{s}{n} \right)^2 R^t \\
&\stackrel{(5.28)}{\leq} (1 - \nu(1-\Theta)s)(D(\alpha^*) - D(\alpha^t)) + \nu(1-\Theta) \frac{\sigma'}{2\lambda} \left( \frac{s}{n} \right)^2 4L^2 \sigma. \tag{5.30}
\end{aligned}$$

Using (5.30) recursively we have

$$\begin{aligned}
\mathbb{E} [D(\alpha^*) - D(\alpha^t)] &= (1 - \nu(1 - \Theta)s)^t (D(\alpha^*) - D(\alpha^0)) \\
&\quad + \nu(1 - \Theta) \frac{\sigma'}{2\lambda} \left(\frac{s}{n}\right)^2 4L^2 \sigma \sum_{j=0}^{t-1} (1 - \nu(1 - \Theta)s)^j \\
&= (1 - \nu(1 - \Theta)s)^t (D(\alpha^*) - D(\alpha^0)) + \nu(1 - \Theta) \frac{\sigma'}{2\lambda} \left(\frac{s}{n}\right)^2 4L^2 \sigma \frac{1 - (1 - \nu(1 - \Theta)s)^t}{\nu(1 - \Theta)s} \\
&\leq (1 - \nu(1 - \Theta)s)^t (D(\alpha^*) - D(\alpha^0)) + s \frac{4L^2 \sigma \sigma'}{2\lambda n^2}.
\end{aligned} \tag{5.31}$$

The choice of  $s := 1$  and  $t = t_0 := \max\{0, \lceil \frac{1}{\nu(1-\Theta)} \log(2\lambda n^2(D(\alpha^*) - D(\alpha^0))/(4L^2 \sigma \sigma')) \rceil\}$  will lead to

$$\begin{aligned}
\mathbb{E} [D(\alpha^*) - D(\alpha^t)] &\leq (1 - \nu(1 - \Theta))^{t_0} (D(\alpha^*) - D(\alpha^0)) + \frac{4L^2 \sigma \sigma'}{2\lambda n^2} \\
&\leq \frac{4L^2 \sigma \sigma'}{2\lambda n^2} + \frac{4L^2 \sigma \sigma'}{2\lambda n^2} = \frac{4L^2 \sigma \sigma'}{\lambda n^2}.
\end{aligned} \tag{5.32}$$

Now, we are going to show that

$$\forall t \geq t_0 : \mathbb{E} [D(\alpha^*) - D(\alpha^t)] \leq \frac{4L^2 \sigma \sigma'}{\lambda n^2 (1 + \frac{1}{2} \nu(1 - \Theta)(t - t_0))}. \tag{5.33}$$

Clearly, (5.32) implies that (5.33) holds for  $t = t_0$ . Now imagine that it holds for any  $t \geq t_0$  then we show that it also has to hold for  $t + 1$ . Indeed, using

$$s = \frac{1}{1 + \frac{1}{2} \nu(1 - \Theta)(t - t_0)} \in [0, 1] \tag{5.34}$$

we obtain

$$\begin{aligned}
\mathbb{E} [D(\alpha^*) - D(\alpha^{t+1})] &\stackrel{(5.30)}{\leq} (1 - \nu(1 - \Theta)s) (D(\alpha^*) - D(\alpha^t)) + \nu(1 - \Theta) \frac{\sigma'}{2\lambda} \left(\frac{s}{n}\right)^2 4L^2 \sigma \\
&\stackrel{(5.33)}{\leq} (1 - \nu(1 - \Theta)s) \frac{4L^2 \sigma \sigma'}{\lambda n^2 (1 + \frac{1}{2} \nu(1 - \Theta)(t - t_0))} + \nu(1 - \Theta) \frac{\sigma'}{2\lambda} \left(\frac{s}{n}\right)^2 4L^2 \sigma \\
&\stackrel{(5.34)}{=} \frac{4L^2 \sigma \sigma'}{\lambda n^2} \left( \frac{1 + \frac{1}{2} \nu(1 - \Theta)(t - t_0) - \nu(1 - \Theta) + \nu(1 - \Theta) \frac{1}{2}}{(1 + \frac{1}{2} \nu(1 - \Theta)(t - t_0))^2} \right) \\
&= \frac{4L^2 \sigma \sigma'}{\lambda n^2} \underbrace{\left( \frac{1 + \frac{1}{2} \nu(1 - \Theta)(t - t_0) - \frac{1}{2} \nu(1 - \Theta)}{(1 + \frac{1}{2} \nu(1 - \Theta)(t - t_0))^2} \right)}_E.
\end{aligned}$$

Now, we will upperbound  $E$  as follows

$$\begin{aligned}
E &= \frac{1}{1 + \frac{1}{2} \nu(1 - \Theta)(t + 1 - t_0)} \underbrace{\frac{(1 + \frac{1}{2} \nu(1 - \Theta)(t + 1 - t_0))(1 + \frac{1}{2} \nu(1 - \Theta)(t - 1 - t_0))}{(1 + \frac{1}{2} \nu(1 - \Theta)(t - t_0))^2}}_{\leq 1} \\
&\leq \frac{1}{1 + \frac{1}{2} \nu(1 - \Theta)(t + 1 - t_0)},
\end{aligned}$$

where in the last inequality we have used the fact that geometric mean is less or equal to arithmetic mean.

If  $\bar{\alpha}$  is defined as in (5.16) then we obtain that

$$\begin{aligned}
\mathbb{E}[Gap(\bar{\alpha})] &= \mathbb{E}\left[Gap\left(\sum_{t=T_0}^{T-1} \frac{1}{T-T_0} \alpha^t\right)\right] \leq \frac{1}{T-T_0} \mathbb{E}\left[\sum_{t=T_0}^{T-1} Gap(\alpha^t)\right] \\
&\stackrel{(5.21),(5.28)}{\leq} \frac{1}{T-T_0} \mathbb{E}\left[\sum_{t=T_0}^{T-1} \left(\frac{1}{\nu(1-\Theta)s}(D(\alpha^{t+1}) - D(\alpha^t)) + \frac{4L^2\sigma\sigma's}{2\lambda n^2}\right)\right] \\
&= \frac{1}{\nu(1-\Theta)s} \frac{1}{T-T_0} \mathbb{E}[D(\alpha^T) - D(\alpha^{T_0})] + \frac{4L^2\sigma\sigma's}{2\lambda n^2} \\
&\leq \frac{1}{\nu(1-\Theta)s} \frac{1}{T-T_0} \mathbb{E}[D(\alpha^*) - D(\alpha^{T_0})] + \frac{4L^2\sigma\sigma's}{2\lambda n^2}.
\end{aligned} \tag{5.35}$$

Now, if  $T \geq \left\lceil \frac{1}{\nu(1-\Theta)} \right\rceil + T_0$  such that  $T_0 \geq t_0$  we obtain

$$\begin{aligned}
\mathbb{E}[Gap(\bar{\alpha})] &\stackrel{(5.35),(5.33)}{\leq} \frac{1}{\nu(1-\Theta)s} \frac{1}{T-T_0} \left( \frac{4L^2\sigma\sigma'}{\lambda n^2(1 + \frac{1}{2}\nu(1-\Theta)(T_0-t_0))} \right) + \frac{4L^2\sigma\sigma's}{2\lambda n^2} \\
&= \frac{4L^2\sigma\sigma'}{\lambda n^2} \left( \frac{1}{\nu(1-\Theta)s} \frac{1}{T-T_0} \frac{1}{1 + \frac{1}{2}\nu(1-\Theta)(T_0-t_0)} + \frac{s}{2} \right).
\end{aligned} \tag{5.36}$$

Choosing

$$s = \frac{1}{(T-T_0)\nu(1-\Theta)} \in [0, 1] \tag{5.37}$$

gives us

$$\mathbb{E}[Gap(\bar{\alpha})] \stackrel{(5.36),(5.37)}{\leq} \frac{4L^2\sigma\sigma'}{\lambda n^2} \left( \frac{1}{1 + \frac{1}{2}\nu(1-\Theta)(T_0-t_0)} + \frac{1}{(T-T_0)\nu(1-\Theta)} \frac{1}{2} \right). \tag{5.38}$$

To have right hand side of (5.38) smaller then  $\epsilon_{Gap}$  it is sufficient to choose  $T_0$  and  $T$  such that

$$\frac{4L^2\sigma\sigma'}{\lambda n^2} \left( \frac{1}{1 + \frac{1}{2}\nu(1-\Theta)(T_0-t_0)} \right) \leq \frac{1}{2} \epsilon_{Gap}, \tag{5.39}$$

$$\frac{4L^2\sigma\sigma'}{\lambda n^2} \left( \frac{1}{(T-T_0)\nu(1-\Theta)} \frac{1}{2} \right) \leq \frac{1}{2} \epsilon_{Gap}. \tag{5.40}$$

Hence, if

$$\begin{aligned}
t_0 + \frac{2}{\nu(1-\Theta)} \left( \frac{8L^2\sigma\sigma'}{\lambda n^2 \epsilon_{Gap}} - 1 \right) &\leq T_0, \\
T_0 + \frac{4L^2\sigma\sigma'}{\lambda n^2 \epsilon_{Gap} \nu(1-\Theta)} &\leq T,
\end{aligned}$$

then (5.39) and (5.40) are satisfied.

### Proof of Theorem 36

If the function  $\ell_i(\cdot)$  is  $(1/\gamma)$ -smooth then  $\ell_i^*(\cdot)$  is  $\gamma$ -strongly convex with respect to the  $\|\cdot\|$  norm. From (5.22) we have

$$\begin{aligned} R^t &\stackrel{(5.22)}{=} -\frac{\lambda\gamma n(1-s)}{\sigma's}\|u^t - \alpha^t\|^2 + \sum_{k=1}^K \|X(u^t - \alpha^t)_{[k]}\|^2 \\ &\stackrel{(5.29)}{\leq} -\frac{\lambda\gamma n(1-s)}{\sigma's}\|u^t - \alpha^t\|^2 + \sum_{k=1}^K \sigma_k \|(u^t - \alpha^t)_{[k]}\|^2 \\ &\leq -\frac{\lambda\gamma n(1-s)}{\sigma's}\|u^t - \alpha^t\|^2 + \sigma_{\max} \sum_{k=1}^K \|(u^t - \alpha^t)_{[k]}\|^2 \\ &= \left(-\frac{\lambda\gamma n(1-s)}{\sigma's} + \sigma_{\max}\right) \|u^t - \alpha^t\|^2. \end{aligned} \quad (5.41)$$

If we plug

$$s = \frac{\lambda\gamma n}{\lambda\gamma n + \sigma_{\max}\sigma'} \in [0, 1] \quad (5.42)$$

into (5.41) we obtain that  $\forall t : R^t \leq 0$ . Putting the same  $s$  into (5.21) will give us

$$\begin{aligned} \mathbb{E}[D(\alpha^{t+1}) - D(\alpha^t)] &\stackrel{(5.21), (5.42)}{\geq} \nu(1-\Theta) \frac{\lambda\gamma n}{\lambda\gamma n + \sigma_{\max}\sigma'} \text{Gap}(\alpha^t) \\ &\geq \nu(1-\Theta) \frac{\lambda\gamma n}{\lambda\gamma n + \sigma_{\max}\sigma'} D(\alpha^*) - D(\alpha^t). \end{aligned} \quad (5.43)$$

Using the fact that  $\mathbb{E}[D(\alpha^{t+1}) - D(\alpha^t)] = \mathbb{E}[D(\alpha^{t+1}) - D(\alpha^*)] + D(\alpha^*) - D(\alpha^t)$  we have

$$\mathbb{E}[D(\alpha^{t+1}) - D(\alpha^*)] + D(\alpha^*) - D(\alpha^t) \stackrel{(5.43)}{\geq} \nu(1-\Theta) \frac{\lambda\gamma n}{\lambda\gamma n + \sigma_{\max}\sigma'} D(\alpha^*) - D(\alpha^t)$$

which is equivalent with

$$\mathbb{E}[D(\alpha^*) - D(\alpha^{t+1})] \leq \left(1 - \nu(1-\Theta) \frac{\lambda\gamma n}{\lambda\gamma n + \sigma_{\max}\sigma'}\right) D(\alpha^*) - D(\alpha^t). \quad (5.44)$$

Therefore if we denote by  $\epsilon_D^t = D(\alpha^*) - D(\alpha^t)$  we have that

$$\begin{aligned} \mathbb{E}[\epsilon_D^t] &\stackrel{(5.44)}{\leq} \left(1 - \nu(1-\Theta) \frac{\lambda\gamma n}{\lambda\gamma n + \sigma_{\max}\sigma'}\right)^t \epsilon_D^0 \leq \left(1 - \nu(1-\Theta) \frac{\lambda\gamma n}{\lambda\gamma n + \sigma_{\max}\sigma'}\right)^t \\ &\leq \exp\left(-t\nu(1-\Theta) \frac{\lambda\gamma n}{\lambda\gamma n + \sigma_{\max}\sigma'}\right). \end{aligned}$$

The right hand side will be smaller than some  $\epsilon_D$  if

$$t \geq \frac{1}{\nu(1-\Theta)} \frac{\lambda\gamma n + \sigma_{\max}\sigma'}{\lambda\gamma n} \log \frac{1}{\epsilon_D}.$$

Moreover, to bound the duality gap, we have

$$\nu(1-\Theta) \frac{\lambda\gamma n}{\lambda\gamma n + \sigma_{\max}\sigma'} \text{Gap}(\alpha^t) \stackrel{(5.43)}{\leq} \mathbb{E}[D(\alpha^{t+1}) - D(\alpha^t)] \leq \mathbb{E}[D(\alpha^*) - D(\alpha^t)].$$

Therefore  $\text{Gap}(\alpha^t) \leq \frac{1}{\nu(1-\Theta)} \frac{\lambda\gamma n + \sigma_{\max}\sigma'}{\lambda\gamma n} \epsilon_D^t$ . Hence if  $\epsilon_D \leq \nu(1-\Theta) \frac{\lambda\gamma n}{\lambda\gamma n + \sigma_{\max}\sigma'} \epsilon_{\text{Gap}}$  then  $\text{Gap}(\alpha^t) \leq \epsilon_{\text{Gap}}$ . Therefore after

$$t \geq \frac{1}{\nu(1-\Theta)} \frac{\lambda\gamma n + \sigma_{\max}\sigma'}{\lambda\gamma n} \log \left( \frac{1}{\nu(1-\Theta)} \frac{\lambda\gamma n + \sigma_{\max}\sigma'}{\lambda\gamma n} \frac{1}{\epsilon_{\text{Gap}}} \right)$$

iterations we have obtained a duality gap less than  $\epsilon_{\text{Gap}}$ .



# **Part III**

# **Federated Optimization and Learning**



# Chapter 6

## Federated Optimization: Distributed Machine Learning for On-device Intelligence

### 6.1 Introduction

Mobile phones and tablets are now the primary computing devices for many people. In many cases, these devices are rarely separated from their owners [34], and the combination of rich user interactions and powerful sensors means they have access to an unprecedented amount of data, much of it private in nature. Models learned on such data hold the promise of greatly improving usability by powering more intelligent applications, but the sensitive nature of the data means there are risks and responsibilities to storing it in a centralized location.

We advocate an alternative — *federated learning* — that leaves the training data distributed on the mobile devices, and learns a shared model by aggregating locally computed updates via a central coordinating server. This is a direct application of the principle of focused collection or data minimization proposed by the 2012 White House report on the privacy of consumer data [182]. Since these updates are specific to improving the current model, they can be purely ephemeral — there is no reason to store them on the server once they have been applied. Further, they will never contain more information than the raw training data (by the data processing inequality), and will generally contain much less. A principal advantage of this approach is the decoupling of model training from the need for direct access to the raw training data. Clearly, some trust of the server coordinating the training is still required, and depending on the details of the model and algorithm, the updates may still contain private information. However, for applications where the training objective can be specified on the basis of data available on each client, federated learning can significantly reduce privacy and security risks by limiting the attack surface to only the device, rather than the device and the cloud.

If additional privacy is needed, randomization techniques from differential privacy can be used. The centralized algorithm could be modified to produce a differentially private model [29, 56, 3], which allows the model to be released while protecting the privacy of the individuals contributing updates to the training process. If protection from even a malicious (or compromised) coordinating server is needed, techniques from local differential privacy can be applied to privatize the individual updates [54]. Details of this are beyond the scope of the current work, but it is a promising direction for future research.

A more complete discussion of applications of federated learning as well as privacy ramifications can be found in [115]. Our focus in this work will be on federated optimization, the optimization problem that must be solved in order to make federated learning a practical alternative to current approaches.

### 6.1.1 Problem Formulation

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

$$\min_{w \in \mathbb{R}^d} P(w) \quad \text{where} \quad P(w) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n f_i(w). \quad (6.1)$$

The main source of motivation are problems arising in machine learning. The problem structure (6.1) covers linear or logistic regressions, support vector machines, but also more complicated models such as conditional random fields or neural networks.

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

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

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

The amount of data that businesses, governments and academic projects collect is rapidly increasing. Consequently, solving problem (6.1) arising in practice is often impossible on a single node, as merely storing the whole dataset on a single node becomes infeasible. This necessitates the use of a distributed computational framework, in which the training data describing the problem is stored in a distributed fashion across a number of interconnected nodes and the optimization problem is solved collectively by the cluster of nodes.

Loosely speaking, one can use any network of nodes to simulate a single powerful node, on which one can run any algorithm. The practical issue is that the time it takes to communicate between a processor and memory on the same node is normally many orders of magnitude smaller than the time needed for two nodes to communicate; similar conclusions hold for the energy required [165]. Further, in order to take advantage of parallel computing power on each node, it is necessary to subdivide the problem into subproblems suitable for independent/parallel computation.

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

These considerations have lead to the development of novel algorithms specialized for distributed optimization (we defer thorough review until Section 6.2). For now, we note that most of the results in literature work in the setting where the data is evenly distributed, and further suppose that  $K \ll n/K$  where  $K$  is the number of nodes. This is indeed often close to reality when data is stored in a large data center. Additionally, an important subfield of the field of distributed learning relies on the assumption that each machine has a representative sample of the data available locally. That is, it is assumed that each machine has an IID sample from the underlying distribution. However, this assumption is often too strong; in fact, even in the data center paradigm this is often not the case since the data on a single node can be close to each other on a temporal scale, or clustered by its geographical origin. Since the patterns in the data can change over time, a feature might be present frequently on one node, while not appear on another at all.

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

### 6.1.2 The Setting of Federated Optimization

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

- **Massively Distributed:** Data points are stored across a large number of nodes  $K$ . In particular, the number of nodes can be much bigger than the average number of training examples stored on a given node ( $n/K$ ).
- **Non-IID:** Data on each node may be drawn from a different distribution; that is, the data points available locally are far from being a representative sample of the overall distribution.
- **Unbalanced:** Different nodes may vary by orders of magnitude in the number of training examples they hold.

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

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

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

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

Further, informally, we choose  $\delta$  to be the minimum piece of information necessary to improve the global model; its utility for other uses is significantly reduced compared to the original data. Thus, it is natural to design a system that does not store these  $\delta$ 's longer than necessary to update the model, again increasing privacy and reducing liability on the part of the centralized model trainer. This setting, in which a single vector  $\delta \in \mathbb{R}^d$  is communicated in each round, covers most existing first-order methods, including dual methods such as CoCoA+ [104].

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

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

## 6.2 Related Work

In this section we provide a detailed overview of the relevant literature. We particularly focus on algorithms that can be used to solve problem (6.1) in various contexts. First, in Sections 6.2.1 and 6.2.2 we look at algorithms designed to be run on a single computer. In Section 6.2.3 we follow with a discussion of the distributed setting, where no single node has direct access to all data describing  $f$ . We describe a paradigm for measuring the efficiency of distributed methods, followed by overview of existing methods and commentary on whether they were designed with communication efficiency in mind or not.

### 6.2.1 Baseline Algorithms

In this section we shall describe several fundamental baseline algorithms which can be used to solve problems of the form (6.1).

**Gradient Descent.** A trivial benchmark for solving problems of structure (6.1) is *Gradient Descent* (GD) in the case when functions  $f_i$  are smooth (or Subgradient Descent for non-smooth functions) [127]. The GD algorithm performs the iteration

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

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

Gradient descent can be substantially accelerated, in theory and practice, via the addition of a momentum term. Acceleration ideas for gradient methods in convex optimization can be traced back to the work of Polyak [139] and Nesterov [126, 127]. While accelerated GD methods have a substantially better convergence rate, in each iteration they still need to do at least one pass over all data. As a result, they are not practical for problems where  $n$  very large.

**Stochastic Gradient Descent.** At present a basic, albeit in practice extremely popular, alternative to GD is *Stochastic Gradient Descent* (SGD), dating back to the seminal work of Robbins and Monro [153]. In the context of (6.1), SGD samples a random function (i.e., a random data-label pair)  $i_t \in \{1, 2, \dots, n\}$  in iteration  $t$ , and performs the update

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

where  $h_t > 0$  is a stepsize parameter. Intuitively speaking, this method works because if  $i_t$  is sampled uniformly at random from indices 1 to  $n$ , the update direction is an unbiased estimate of the gradient —  $\mathbb{E}[\nabla f_{i_t}(w)] = \nabla P(w)$ . However, noise introduced by sampling slows down the convergence, and a diminishing sequence of stepsizes  $h_k$  is necessary for convergence. For a theoretical analysis for convex functions we refer the reader to [125, 120, 124] and [162, 172] for SVM problems. In a recent review [21], the authors outline further research directions. For a more practically-focused discussion, see [20]. In the context of neural networks, computation of stochastic gradients is referred to as *backpropagation* [93]. Instead of specifying the functions  $f_i$  and its gradients explicitly, backpropagation is a general way of computing the gradient. Performance of several competitive algorithms for training deep neural networks has been compared in [130].

One common trick that has been practically observed to provide superior performance, is to replace random sampling in each iteration by going through all the functions in a random

order. This ordering is replaced by another random order after each such cycle [18]. Theoretical understanding of this phenomenon had been a long standing open problem, understood recently in [71].

The core differences between GD and SGD can be summarized as follows. GD has a fast convergence rate, but each iteration in the context of (6.1) is potentially very slow, as it needs to process the entire dataset in each iteration. On the other hand, SGD has slower convergence rate, but each iteration is fast, as the work needed is independent of number of data points  $n$ . For the problem structure of (6.1), SGD is usually better, as for practical purposes relatively low accuracy is required, which SGD can in extreme cases achieve after single pass through data, while GD would make just a single update. However, if a high accuracy was needed, GD or its faster variants would prevail.

### 6.2.2 A Novel Breed of Randomized Algorithms

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

**Randomized Coordinate Descent.** Although the idea of coordinate descent has been around for several decades in various contexts (and for quadratic functions dates back even much further, to works on the Gauss-Seidel methods), it came to prominence in machine learning and optimization with the work of Nesterov [129] which equipped the method with a randomization strategy. Nesterov’s work on *Randomized Coordinate Descent* (RCD) popularized the method and demonstrated that randomization can be very useful for problems of structure (6.1).

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

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

where  $h_{j_t} > 0$  is a stepsize parameter,  $\nabla_j P(w)$  denotes the  $j^{th}$  partial derivative of function  $f$ , and  $e_j$  is the  $j^{th}$  unit standard basis vector in  $\mathbb{R}^d$ . For the case of generalized linear models, when the data exhibits certain sparsity structure, it is possible to evaluate the partial derivative  $\nabla_j P(w)$  efficiently, i.e., without need to process the entire dataset, leading to a practically efficient algorithm, see for instance [148, Section 6].

Numerous follow-up works extended the concept to proximal setting [148], single processor parallelism [24, 151] and develop efficiently implementable acceleration [96]. All of these three properties were connected in a single algorithm in [59], to which we refer the reader for a review of the early developments in the area of RCD, particularly to overview in Table 1 therein.

**Stochastic Dual Coordinate Ascent.** When an explicit strongly convex, but not necessarily smooth, regularizer is added to the average loss (6.1), it is possible to write down its (Fenchel) dual and the dual variables live in  $n$ -dimensional space. Applying RCD leads to an algorithm for solving (6.1) known under the name *Stochastic Dual Coordinate Ascent* [163]. This method has gained broad popularity with practitioners, likely due to the fact that for a number of loss functions, the method comes without the need to tune any hyper-parameters. The work [163] was first to show that by applying RCD [148] to the dual problem, one also solves the primal problem (6.1). For a theoretical and computational comparison of applying RCD to the primal versus the dual problems, see [39].

A directly primal-dual randomized coordinate descent method called Quartz, was developed in [143]. It has been recently shown in SDNA [142] that incorporating curvature information contained in random low dimensional subspaces spanned by a few coordinates can sometimes lead to dramatic speedups. Recent works [160, 38] interpret the SDCA method in primal-only setting, shedding light onto why this method works as a SGD method with a version of variance reduction property.

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

**Stochastic Average Gradient.** The first notable algorithm from this class is the *Stochastic Average Gradient* (SAG) [156, 158]. The SAG algorithm stores an average of  $n$  gradients of functions  $f_i$  evaluated at different points in the history of the algorithm. In each iteration, the algorithm, updates randomly chosen gradient out of this average, and makes a step in the direction of the average. This way, complexity of each iteration is independent of  $n$ , and the algorithm enjoys a fast convergence. The drawback of this algorithm is that it needs to store  $n$  gradients in memory because of the update operation. In the case of generalized linear models, this memory requirement can be reduced to the need of  $n$  scalars, as the gradient is a scalar multiple of the data point. This methods has been recently extended for use in Conditional Random Fields [157]. Nevertheless, the memory requirement makes the algorithm infeasible for application even in relatively small neural networks.

A followup algorithm SAGA [45] and its simplification [44], modifies the SAG algorithm to achieve unbiased estimate of the gradients. The memory requirement is still present, but the method significantly simplifies theoretical analysis, and yields a slightly stronger convergence guarantee.

**Stochastic Variance Reduced Gradient.** Another algorithm from the SGD class of methods is *Stochastic Variance Reduced Gradient*<sup>1</sup> (SVRG) [80] and [89, 187, 81]. The SVRG algorithm runs in two nested loops. In the outer loop, it computes full gradient of the whole function,  $\nabla P(w^t)$ , the expensive operation one tries to avoid in general. In the inner loop, the update step is iteratively computed as

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

The core idea is that the stochastic gradients are used to estimate the change of the gradient between point  $w^t$  and  $w$ , as opposed to estimating the gradient directly. We return to more detailed description of this algorithm in Section 6.3.2.

The SVRG has the advantage that it does not have the additional memory requirements of SAG/SAGA, but it needs to process the whole dataset every now and then. Indeed, comparing to SGD, which typically makes significant progress in the first pass through data, SVRG does not make any update whatsoever, as it needs to compute the full gradient. This and several other practical issues have been recently addressed in [74], making the algorithm competitive with SGD early on, and superior in later iterations. Although there is nothing that prevents one from applying SVRG and its variants in deep learning, we are not aware of any systematic assessment of its performance in this setting. Vanilla experiments in [80, 146] suggest that SVRG matches basic SGD, and even outperforms in the sense that variance of the iterates seems to be significantly smaller for SVRG. However, in order to draw any meaningful conclusions, one would need to perform extensive experiments and compare with state-of-the-art methods usually equipped with numerous heuristics.

There already exist attempts at combining SVRG type algorithms with randomized coordinate descent [84, 180]. Although these works highlight some interesting theoretical properties, the algorithms do not seem to be practical at the moment; more work is needed in this area. The first attempt to unify algorithms such as SVRG and SAG/SAGA already appeared in the SAGA paper [45], where the authors interpret SAGA as a midpoint between SAG and SVRG. Recent work [145] presents a general algorithm, which recovers SVRG, SAGA, SAG and GD as special cases, and obtains an asynchronous variant of these algorithms as a byproduct of the formulation. SVRG can be equipped with momentum (and negative momentum), leading to a new accelerated SVRG method known as Katyusha [5]. SVRG can be further accelerated via a raw clustering mechanism [6].

---

<sup>1</sup>The same algorithm was simultaneously introduced as Semi-Stochastic Gradient Descent (S2GD) [89]. Since the former work gained more attention, we will for clarity use the name SVRG throughout this chapter.

**Stochastic Quasi-Newton Methods.** A third class of new algorithms are the *Stochastic quasi-Newton* methods [26, 17]. These algorithms in general try to mimic the limited memory BFGS method (L-BFGS) [100], but model the local curvature information using inexact gradients — coming from the SGD procedure. A recent attempt at combining these methods with SVRG can be found in [119]. In [67], the authors utilize recent progress in the area of stochastic matrix inversion [69] revealing new connections with quasi-Newton methods, and devise a new stochastic limited memory BFGS method working in tandem with SVRG. The fact that the theoretical understanding of this branch of research is the least understood and having several details making the implementation more difficult compared to the methods above may limit its wider use. However, this approach could be most promising for deep learning once understood better.

One important aspect of machine learning is that the Empirical Risk Minimization problem (6.1) we are solving is just a proxy for the Expected Risk we are ultimately interested in. When one can find exact minimum of the empirical risk, everything reduces to balancing approximation–estimation tradeoff that is the object of abundant literature — see for instance [179]. An assessment of asymptotic performance of some optimization algorithms as *learning* algorithms in large-scale learning problems<sup>2</sup> has been introduced in [22]. Recent extension in [74] has shown that the variance reduced algorithms (SAG, SVRG, ...) can in certain setting be better *learning* algorithms than SGD, not just better optimization algorithms.

**Further Remarks.** A general method, referred to as Universal Catalyst [99, 63], effectively enables conversion of a number of the algorithms mentioned in the previous sections to their ‘accelerated’ variants. The resulting convergence guarantees nearly match lower bounds in a number of cases. However, the need to tune additional parameter makes the method rather impractical.

Recently, lower and upper bounds for complexity of stochastic methods on problems of the form (6.1) were recently obtained in [185].

### 6.2.3 Distributed Setting

In this section we review the literature concerning algorithms for solving (6.1) in the distributed setting. When we speak about distributed setting, we refer to the case when the data describing the functions  $f_i$  are not stored on any single storage device. This can include setting where one’s data just don’t fit into a single RAM/computer/node, but two is enough. This also covers the case where data are distributed across several datacenters around the world, and across many nodes in those datacenters. The point is that in the system, there is no single processing unit that would have direct access to all the data. Thus, the distributed setting does not include single processor parallelism<sup>3</sup>. Compared with local computation on any single node, the cost of communication between nodes is much higher both in terms of speed and energy consumption [12, 165], introducing new computational challenges, not only for optimization procedures.

We first review a theoretical decision rule for determining the practically best algorithm for a given problem in Section 6.2.3, followed by overview of distributed algorithms in Section 6.2.3, and communication efficient algorithms in Section 6.2.3. The following paradigm highlights why the class of communication efficient algorithms are not only preferable choice in the trivial sense. The communication efficient algorithms provide us with much more flexible tools for designing overall optimization procedure, which can make the algorithms inherently adaptive to differences in computing resources and architectures.

#### A Paradigm for Measuring Distributed Optimization Efficiency

This section reviews a paradigm for comparing efficiency of distributed algorithms. Let us suppose we have many algorithms  $\mathcal{A}$  readily available to solve the problem (6.1). The question is: “How do we decide which algorithm is the best for our purpose?” Initial version of this reasoning already appeared in [104], and applies also to [147].

---

<sup>2</sup>See [22, Section 2.3] for their definition of large scale learning problem.

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

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

$$\text{TIME} = \mathcal{I}_{\mathcal{A}}(\epsilon) \times \mathcal{T}_{\mathcal{A}}. \quad (6.2)$$

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

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

$$\text{TIME} = \mathcal{I}_{\mathcal{A}}(\epsilon) \times (c + \mathcal{T}_{\mathcal{A}}) \quad (6.3)$$

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

Most, if not all, of the current state-of-the-art algorithms that are the best in setting of (6.2), are stochastic and rely on doing very large number (big  $\mathcal{I}_{\mathcal{A}}(\epsilon)$ ) of very fast (small  $\mathcal{T}_{\mathcal{A}}$ ) iterations. As a result, even relatively small  $c$  can cause the practical performance of those algorithms drop down dramatically, because  $c \gg \mathcal{T}_{\mathcal{A}}$ .

This has been indeed observed in practice, and motivated development of new methods, designed with this fact in mind from scratch, which we review in Section 6.2.3. Although this is a good development for academia — motivation to explore new setting, it is not necessarily a good news for the industry.

Many companies have spent significant resources to build excellent algorithms to tackle their problems of form (6.1), fine tuned to the specific patterns arising in their data and side applications required. When the data companies collect grows too large to be processed on a single machine, it is understandable that they would be reluctant to throw away their fine tuned algorithms. This issue was first time explicitly addressed in CoCoA [104], which is rather framework than a algorithm, which works as follows (more detailed description follows in Section 6.2.3).

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

The efficiency paradigm changes as follows:

$$\text{TIME} = \mathcal{I}(\epsilon, \Theta) \times (c + \mathcal{T}_{\mathcal{A}}(\Theta)) \quad (6.4)$$

The number of iterations  $\mathcal{I}(\epsilon, \Theta)$  is independent of choice of the algorithm  $\mathcal{A}$  used as a local solver, because there is theory predicting how many iterations of the CoCoA framework are needed to achieve  $\epsilon$  accuracy, if we solve the local subproblems to relative  $\Theta$  accuracy. Here,  $\Theta = 0$  would mean we require the subproblem to be solved to optimality, and  $\Theta = 1$  that we don't need any progress whatsoever. The general upper bound on number of iterations of the CoCoA framework is  $\mathcal{I}(\epsilon, \Theta) = \frac{\mathcal{O}(\log(1/\epsilon))}{1-\Theta}$  [79, 105, 104] for strongly convex objectives. From the inverse dependence on  $1 - \Theta$ , we can see that there is a fundamental limit to the number of communication rounds needed. Hence, it will probably not be efficient to spend excessive resources to attain very high local accuracy (small  $\Theta$ ). Time per iteration  $\mathcal{T}_{\mathcal{A}}(\Theta)$  denotes the time algorithm  $\mathcal{A}$  needs to reach the relative  $\Theta$  accuracy on the local subproblem.

---

<sup>4</sup>Considering only algorithms that can be run on a given machine.

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

1. It allows practitioners to continue using their fine-tuned solvers, that can run only on single machine, instead of having to implement completely new algorithms from scratch.
2. The actual performance in terms of number of rounds of communication is independent from the choice of optimization algorithm, making it much easier to optimize the overall performance.
3. Since the constant  $c$  is architecture dependent, running optimal algorithm on one node network does not have to be optimal on another. In the setting (6.3), this could mean moving from one cluster to another, a completely different algorithm is optimal, which is a major change. In the setting (6.4), this can be improved by simply changing  $\Theta$ , which is typically implicitly determined by number of iterations algorithm  $\mathcal{A}$  runs for.

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

### Distributed Algorithms

As discussed below in Section 6.2.3, this setting creates unique challenges. Distributed optimization algorithms typically require a small number (1–4) of communication rounds per iteration. By communication round we typically understand a single MapReduce operation [43], implemented efficiently for iterative procedures [61], such as optimization algorithms. Spark [193] has been established as a popular open source framework for implementing distributed iterative algorithms, and includes several of the algorithms mentioned in this section.

Optimization in distributed setting has been studied for decades, tracing back to at least works of Bertsekas and Tsitsiklis [14, 13, 177]. Recent decade has seen an explosion of interest in this area, greatly motivated by rapid increase of data availability in machine learning applications.

Much of the recent effort was focused on creating new optimization algorithms, by building variants of popular algorithms suitable for running on a single processor (See Section 6.2.1). A relatively common feature of many of these efforts is a) The computation overhead in the case of synchronous algorithms, and b) The difficulty of analysing asynchronous algorithms without restrictive assumptions. By computation overhead we mean that if optimization program runs in a compute-communicate-update cycle, the update part cannot start until all nodes finish their computation. This causes some of the nodes to be idle, while remaining nodes finish their part of computation, clearly an inefficient use of computational resources. This pattern often diminishes or completely reverts potential speed-ups from distributed computation. In the asynchronous setting in general, an update can be applied to a parameter vector, followed by computation done based on a now-outdated version of that parameter vector. Formally grasping this pattern, while keeping the setting realistic is often quite challenging. Consequently, this is very open area, and optimal choice of algorithm in any particular case is often heavily dependent on the problem size, details in its structure, computing architecture available, and above all, expertise of the practitioner.

This general issue is best exhibited with numerous attempts at parallelizing the Stochastic Gradient Descent and its variants. As an example, [47, 51] provide theoretically linear speedup with number of nodes, but are difficult to implement efficiently, as the nodes need to synchronize frequently in order to compute reasonable gradient averages. As an alternative, no synchronization between workers is assumed in [133, 4, 53]. Consequently, each worker reads  $w^t$  from memory, parameter vector  $w$  at time point  $t$ , computes a stochastic gradient  $\nabla f_i(w^t)$  and applies it to already changed state of the parameter vector  $w^{t+\tau}$ . The above mentioned methods assume that the delay  $\tau$  is bounded by a constant, which is not necessarily realistic assumption<sup>5</sup>. Some of the works also introduce assumptions on the sparsity structures or conditioning of the Hessian of  $f$ . Asymptotically optimal convergent rates were proven in [52] with

---

<sup>5</sup>A bound on the delay  $\tau$  can be deterministic or probabilistic. However, in practice, the delays are mostly about the number of nodes in the network, and there are very long delays, when a variety of operating systems

considerably milder assumptions. Improved analysis of asynchronous SGD was also presented in [41], simultaneously with a version that uses lower-precision arithmetic was introduced without sacrificing performance, which is a trend that might find use in other parts of machine learning in the following years.

The negative effect of asynchronous distributed implementations of SGD seem to be negligible, when applied to the task of training very large deep networks — which is the ultimate industrial application of today. The practical usefulness has been demonstrated for instance by Google’s Downpour SGD [42] and Microsoft’s Project Adam [33].

The first distributed versions of Coordinate Descent algorithms were the Hydra and its accelerated variant, Hydra<sup>2</sup>, [149, 57], which has been demonstrated to be very efficient on large sparse problems implemented on a computing cluster. An extended version with description of implementation details is presented in [111]. Effect of asynchrony has been explored and partially theoretically understood in the works of [102, 101]. Another asynchronous, rather framework than an algorithm, for coordinate updates, applicable to wider class of objectives is presented in [138].

The data are assumed to be partitioned to nodes by features/coordinates in the above algorithms. This setting can be restrictive if one is not able to distribute the data beforehand, but instead the data are distributed “as is” — in which case the data are most commonly distributed by data points. This does not need to be an issue, if a dual version of coordinate descent is used — in which the distribution is done by data points [173] followed by works on Communication Efficient Dual Coordinate Ascent, described in next section. The use of duality however requires usage of additional explicit strongly convex regularization term, hence can be used to solve smaller class of problems. Despite the apparent practical disadvantages, variants of distributed coordinate descent algorithms are among the most widely used methods in practice.

Moving to variance reduced methods, distributed versions of SAG/SAGA algorithms have not been proposed yet. However, several distributed versions of the SVRG algorithm already exist. A scheme for replicating data to simulate iid sampling in distributed environment was proposed in [95]. Although the performance is well analyzed, the setting requires significantly stronger control of data distribution which is rarely practically feasible. A relatively similar method to Algorithm 12 presented here has been proposed in [147], which was analyzed, and in [109], a largely experimental work that can be also cast as communication efficient — described in detail in Section 6.2.3.

Another class of algorithms relevant for this work is Alternating Direction Method of Multipliers (ADMM) [23, 49]. These algorithms are in general applicable to much broader class of problems, and hasn’t been observed to perform better than other algorithms presented in this section, in the machine learning setting of (6.1).

## Communication-Efficient Algorithms

In this Section, we describe algorithms that can be cast as “communication efficient”. The common theme of the algorithms presented here, is that in order to perform better in the sense of (6.3), one should design algorithms with high  $\mathcal{T}_A$ , in order to make the cost of communication  $c$  negligible.

Before moving onto specific methods, it is worth noting some of the core limits concerning the problem we are trying to solve in distributed setting. Fundamental limitations of stochastic versions of the problem (6.1) in terms of runtime, communication costs and number of samples used are studied in [166]. Efficient algorithms and lower bounds for distributed statistical estimation are established in [198, 197].

However, these works do not fit into our framework, because they assume that each node has access to data generated IID from a single distribution. In the case of [198, 197] also  $K \ll n/K$ , that the number of nodes  $K$  is much smaller than the number of data point on each node is also assumed. As we stress in the Introduction, these assumptions are far from being

---

related events can temporarily postpone computation of a single node. To the best of our knowledge, no formal assumptions reflect this setting well. In fact, two recent works [110, 92] highlight subtle but important issue with labeling of iterates in the presence of asynchrony, rendering most of the existing analyses of asynchronous optimization algorithms incorrect.

satisfied in our setting. Intuitively, relaxing these assumptions should make the problem harder. However, it is not as straightforward to conclude this, as there are certainly particular non-iid data distributions that simplify the problem — for instance if data are distributed according to separability structure of the objective. Lower bounds on communication complexity of distributed convex optimization of (6.1) are presented in [7], concluding that for IID data distributions, existing algorithms already achieve optimal complexity in specific settings.

Probably first, rather extreme, work [203] proposed to parallelize SGD in a single round of communication. Each node simply runs SGD on the data available locally, and their outputs are averaged to form a final result. This approach is however not very robust to differences in data distributions available locally, and it has been shown [167, Appendix A] that in general it cannot perform better than using output of a single machine, ignoring all the other data.

Shamir et al. proposed the DANE algorithm, Distributed Approximate Newton [167], to exactly solve a general subproblem available locally, before averaging their solutions. The method relies on similarity of Hessians of local objectives, representing their iterations as an average of inexact Newton steps. We describe the algorithm in greater detail in Section 6.3.4 as our proposed work builds on it. A quite similar approach was proposed in [109], with richer class of subproblems that can be formulated locally, and solved approximately. An analysis of inexact version of DANE and its accelerated variant, AIDE, appeared recently in [147]. Inexact DANE is closely related to the algorithms presented in this chapter. We, however, continue in different direction shaped by the setting of federated optimization.

The DiSCO algorithm [199] of Zhang and Xiao is based on inexact damped Newton method. The core idea is that the inexact Newton steps are computed by distributed preconditioned conjugate gradient, which can be very fast, if the data are distributed in an IID fashion, enabling a good preconditioner to be computed locally. The theoretical upper bound on number of rounds of communication improves upon DANE and other methods, and in certain settings matches the lower bound presented in [7]. The DiSCO algorithm is related to [98, 202], a distributed truncated Newton method. Although it was reported to perform well in practice, the total number of conjugate gradient iterations may still be high to be considered a communication efficient algorithm.

Common to the above algorithms is the assumption that each node has access to data points sampled IID from the same distribution. This assumption is not required only in theory, but can cause the algorithms to converge significantly slower or even diverge (as reported for instance in [167, Table 3]). Thus, these algorithms, at least in their default form, are not suitable for the setting of Federated Optimization presented here.

An algorithm that bypasses the need for IID data assumption is CoCoA, which provably converges under any distribution of the data, while the convergence rate does depend on properties of the data distribution. The first version of the algorithm was proposed as DisDCA in [188], without convergence guarantees. First analysis was introduced in [79], with further improvements in [105], and a more general version in [104]. Recently, its variant for L1-regularized objectives was introduced in [168].

The CoCoA framework formulates general local subproblems based on the dual form of (6.1) (See for instance [104, Eq. (2)]). Data points are distributed to nodes, along with corresponding dual variables. Arbitrary optimization algorithm is used to attain a relative  $\Theta$  accuracy on the local subproblem — by changing only local dual variables. These updates have their corresponding updates to primal variable  $w$ , which are synchronously aggregated (could be averaging, adding up, or anything in between; depending on the local subproblem formulation).

From the description in this section it appears that the CoCoA framework is the only usable tool for the setting of Federated Optimization. However, the theoretical bound on number of rounds of communications for ill-conditioned problems scales with the number of nodes  $K$ . Indeed, as we will show in Section 6.4 on real data, CoCoA framework does converge very slowly.

### 6.3 Algorithms for Federated Optimization

In this section we introduce the first algorithm that was designed with the unique challenges of federated optimization in mind. Before proceeding with the explanation, we first revisit

two important and at first sight unrelated algorithms. The connection between these algorithms helped to motivate our research. Namely, the algorithms are the Stochastic Variance Reduced Gradient (SVRG) [80, 89], a stochastic method with explicit variance reduction, and the Distributed Approximate Newton (DANE) [167] for distributed optimization.

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

Although this algorithm seems to work well in practice in simple circumstances, its performance is still unsatisfactory in the general setting we specify in Section 6.3.3. We proceed by making the FSVRG algorithm adaptive to different local data sizes, general sparsity patterns and significant differences in patterns in data available locally, and those present in the entire data set.

### 6.3.1 Desirable Algorithmic Properties

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

- (A) If the algorithm is initialized to the optimal solution, it stays there.
- (B) If all the data is on a single node, the algorithm should converge in  $\mathcal{O}(1)$  rounds of communication.
- (C) If each feature occurs on a single node, so the problems are fully decomposable (each machine is essentially learning a disjoint block of parameters), then the algorithm should converge in  $\mathcal{O}(1)$  rounds of communication<sup>6</sup>.
- (D) If each node contains an identical dataset, then the algorithm should converge in  $\mathcal{O}(1)$  rounds of communication.

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

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

### 6.3.2 SVRG

The SVRG algorithm [80, 89] is a stochastic method designed to solve problem (6.1) on a single node. We present it as Algorithm 10 in a slightly simplified form.

---

#### Algorithm 10 SVRG

---

```

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

```

---

<sup>6</sup>This is valid only for generalized linear models.

The algorithm runs in two nested loops. In the outer loop, it computes gradient of the entire function  $P$  (Line 3). This constitutes for a full pass through data — in general expensive operation one tries to avoid unless necessary. This is followed by an inner loop, where  $m$  fast stochastic updates are performed. In practice,  $m$  is typically set to be a small multiple (1–5) of  $n$ . Although the theoretically optimal choice for  $m$  is a small multiple of a condition number [89, Theorem 6], this is often of the same order as  $n$  in practice.

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

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

The performance is well understood in theory. For  $\lambda$ -strongly convex  $P$  and  $L$ -smooth functions  $f_i$ , convergence results are in the form

$$\mathbb{E} [P(w^t) - P(w^*)] \leq c^t [P(w^0) - P(w^*)], \quad (6.5)$$

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

It is possible to show [89, Theorem 6] that for appropriate choice of parameters  $m$  and  $h$ , the convergence rate (6.5) translates to the need of

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

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

### 6.3.3 Distributed Problem Formulation

In this section, we introduce notation and specify the structure of the distributed version of the problem we consider (6.1), focusing on the case where the  $f_i$  are convex. We assume the data  $\{x_i, y_i\}_{i=1}^n$ , describing functions  $f_i$  are stored across a large number of nodes.

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

$$F_k(w) \stackrel{\text{def}}{=} \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w), \quad (6.6)$$

which is the local objective based on the data stored on machine  $k$ . We can then rephrase the objective (6.1) as

$$P(w) = \sum_{k=1}^K \frac{n_k}{n} F_k(w) = \sum_{k=1}^K \frac{n_k}{n} \cdot \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w). \quad (6.7)$$

The way to interpret this structure is to see the empirical loss  $P(w) = \frac{1}{n} \sum_{i=1}^n f_i(w)$  as a convex combination of the local empirical losses  $F_k(w)$ , available locally to node  $k$ . Problem (6.1) then takes the simplified form

$$\min_{w \in \mathbb{R}^d} P(w) \equiv \sum_{k=1}^K \frac{n_k}{n} F_k(w). \quad (6.8)$$

---

<sup>7</sup>See [89, Theorem 4] and [80, Theorem 1] for details.

### 6.3.4 DANE

In this section, we introduce a general reasoning providing stronger intuitive support for the DANE algorithm [167], which we describe in detail below. We will follow up on this reasoning in Appendix 6.6 and draw a connection between two existing methods that was not known in the literature.

If we wanted to design a distributed algorithm for solving the above problem (6.8), where node  $k$  contains the data describing function  $F_k$ . The first, and as we shall see, a rather naive idea is to ask each node to minimize their local functions, and average the results (a variant of this idea appeared in [203]):

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

Clearly, it does not make sense to run this algorithm for more than one iteration as the output  $w$  will always be the same. This is simply because  $w_k^{t+1}$  does not depend on  $t$ . In other words, this method effectively performs just a single round of communication. While the simplicity is appealing, the drawback of this method is that it can't work. Indeed, there is no reason to expect that in general the solution of (6.8) will be a weighted average of the local solutions, unless the local functions are all the same — in which case we do not need a distributed algorithm in the first place and can instead solve the much simpler problem  $\min_{w \in \mathbb{R}^d} F_1(w)$ . This intuitive reasoning can be also formally supported, see for instance [167, Appendix A].

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

$$w_k^{t+1} = \arg \min_{w \in \mathbb{R}^d} F_k(w) - (a_k^t)^T w + \frac{\mu}{2} \|w - w^t\|^2, \quad w^{t+1} = \frac{1}{K} \sum_{k=1}^K w_k^{t+1}. \quad (6.9)$$

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

As described, the method is not yet well defined, since we have not described how the vectors  $a_k^t$  would change from iteration to iteration, and how one should choose  $\mu$ . In order to get some insight into how such a method might work, let us examine the optimality conditions. Asymptotically as  $t \rightarrow \infty$ , we would like  $a_k^t$  to be such that the minimum of each subproblem is equal to  $w^*$ ; the minimizer of (6.8). Hence, we would wish for  $w^*$  to be the solution of

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

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

**DANE.** We present the Distributed Approximate Newton algorithm (DANE) [167], as Algorithm 11. The algorithm was originally analyzed for solving the problem of structure (6.7), with  $n_k$  being identical for each  $k$  — i.e., each computer has the same number of data points. Nothing prevents us from running it in our more general setting though.

---

**Algorithm 11** Distributed Approximate Newton (DANE)

---

```

1: Input: regularizer  $\mu \geq 0$ , parameter  $\eta$  (default:  $\mu = 0, \eta = 1$ )
2: for  $s = 0, 1, 2, \dots$  do
3:   Compute  $\nabla P(w^t) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(w^t)$  and distribute to all machines
4:   For each node  $k \in \{1, \dots, K\}$ , solve

$$w_k = \arg \min_{w \in \mathbb{R}^d} \left\{ F_k(w) - (\nabla F_k(w^t) - \eta \nabla P(w^t))^T w + \frac{\mu}{2} \|w - w^t\|^2 \right\} \quad (6.10)$$

5:   Compute  $w^{t+1} = \frac{1}{K} \sum_{k=1}^K w_k$ 
6: end for

```

---

As alluded to earlier, the main idea of DANE is to form a local subproblem, dependent only on local data, and gradient of the entire function — which can be computed in a single round of communication (Line 3). The subproblem is then solved exactly (Line 4), and updates from individual nodes are averaged to form a new iterate (Line 5). This approach allows any algorithm to be used to solve the local subproblem (6.10). As a result, it often achieves communication efficiency in the sense of requiring expensive local computation between rounds of communication, hopefully rendering the time needed for communication insignificant (see Section 6.2.3). Further, note that DANE belongs to the family of distributed method that operate via the quadratic perturbation trick (6.9) with

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

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

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

The convergence analysis for DANE assumes that the functions are twice differentiable, and relies on the assumption that each node has access to IID samples from the same underlying distribution. This implies that the Hessians of  $\nabla^2 F_k(w)$  are similar to each other [167, Lemma 1]. In case of linear regression, with  $\lambda = \mathcal{O}(1/\sqrt{n})$ -strongly convex functions, the number of DANE iterations needed to achieve  $\epsilon$ -accuracy is  $\mathcal{O}(K \log(1/\epsilon))$ . However, for general  $L$ -smooth loss, the theory is significantly worse, and does not match its practical performance.

The practical performance also depends on the additional local regularization parameter  $\mu$ . For small number of nodes  $K$ , the algorithm converges quickly with  $\mu = 0$ . However, as reported [167, Figure 3], it can diverge quickly with growing  $K$ . Bigger  $\mu$  makes the algorithm more stable at the cost of slower convergence. Practical choice of  $\mu$  remains an open question.

### 6.3.5 SVRG meets DANE

As we mentioned above, the DANE algorithm can perform poorly in certain settings, even without the challenging aspects of federated optimization. Another point that is seen as drawback of DANE is the need to find the *exact* minimum of (6.10) — this can be feasible for quadratics with relatively small dimension, but infeasible or extremely expensive to achieve for other problems. We adapt the idea from the CoCoA algorithm [104], in which an arbitrary optimization algorithm is used to obtain relative  $\Theta$  accuracy on a locally defined subproblem. We replace the exact optimization with an approximate solution obtained by using any optimization algorithm.

Considering all the algorithms one could use to solve (6.10), the SVRG algorithm seems to be a particularly good candidate. Starting the local optimization of (6.10) from point  $w^t$ , the algorithm automatically has access to the derivative at  $w^t$ , which is identical for each node —  $\nabla P(w^t)$ . Hence, the SVRG algorithm can skip the initial expensive operation, evaluation of

the entire gradient (Line 3, Algorithm 10), and proceed only with the stochastic updates in the inner loop.

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

**Proposition 42.** *Consider the following two algorithms.*

1. *Run the DANE algorithm (Algorithm 11) with  $\eta = 1$  and  $\mu = 0$ , and use SVRG (Algorithm 10) as a local solver for (6.10), running it for a single iteration, initialized at point  $w^t$ .*
2. *Run a distributed variant of the SVRG algorithm, described in Algorithm 12.*

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

*Proof.* We construct the proof by showing that single step of the SVRG algorithm applied to the problem (6.10) on computer  $k$  is identical to the update on Line 8 in Algorithm 12.

The way to obtain a stochastic gradient of (6.10) is to sample one of the functions composing  $F_k(w) = \frac{1}{n_k} \sum_{i \in \mathcal{P}_k} f_i(w)$ , and add the linear term  $\nabla F_k(w^t) - \eta P(w^t)$ , which is known and does not need to be estimated. Upon sampling an index  $i \in \mathcal{P}_k$ , the update direction follows as

$$\begin{aligned} [\nabla f_i(w) - \nabla F_k(w^t) - \nabla P(w^t)] - [\nabla f_i(w^t) - \nabla F_k(w^t) - \nabla P(w^t)] + \nabla P(w^t) = \\ \nabla f_i(w) - \nabla f_i(w^t) + \nabla P(w^t) \end{aligned}$$

which is identical to the direction in Line 8 in Algorithm 12. The claim follows by chaining the identical updates to form identical iterate  $w^{t+1}$ .  $\square$

---

#### Algorithm 12 naive Federated SVRG (FSVRG)

---

```

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

```

---

**Remark 43.** *The algorithms considered in Proposition 42 are inherently stochastic. The statement of the proposition is valid under the assumption that in both cases, identical sequence of samples  $i \in \mathcal{P}_k$  would be generated by all nodes  $k \in \{1, 2, \dots, K\}$ .*

**Remark 44.** *In the Proposition 42 we consider the DANE algorithm with particular values of  $\eta$  and  $\mu$ . The Algorithm 12 and the Proposition can be easily generalized, but we present only the default version for the sake of clarity.*

Since the first version of this work, this connection has been mentioned in [147], which analyzes an inexact version of the DANE algorithm. We proceed by adapting the above algorithm to other challenges arising in the context of federated optimization.

### 6.3.6 Federated SVRG

Empirically, the Algorithm 12 fits in the model of distributed optimization efficiency described in Section 6.2.3, since we can balance how many stochastic iterations should be performed locally

against communication costs. However, several modifications are necessary to achieve good performance in the full federated optimization setting (Section 6.3.3). Very important aspect that needs to be addressed is that the number of data points available to a given node can differ greatly from the average number of data points available to any single node. Furthermore, this setting always comes with the data available locally being clustered around a specific pattern, and thus not being a representative sample of the overall distribution we are trying to learn. In the Experiments section we focus on the case of L2 regularized logistic regression, but the ideas carry over to other generalized linear prediction problems.

## Notation

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

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

With these quantities defined, we can state our proposed algorithm as Algorithm 13. Our experiments show that this algorithm works very well in practice, but the motivation for the particular scaling of the updates may not be immediately clear. In the following section we provide the intuition that lead to the development of this algorithm.

---

**Algorithm 13** Federated SVRG (FSVRG)

- ```

1: parameters:  $h$  = stepsize, data partition  $\{\mathcal{P}_k\}_{k=1}^K$ ,  

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

```

## Intuition Behind FSVRG Updates

The difference between the Algorithm 13 and Algorithm 12 is in the introduction of the following properties.

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

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

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

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

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

We have

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

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

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

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

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

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

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

Let us now highlight some properties of the modification in Point 4. Without any extra information, or in the case of fully dense data, averaging the local updates is the only way that actually makes sense — because each node outputs approximate solution of a proxy to the overall objective, and there is no induced separability structure in the outputs such as in CoCoA [104]. However, we could do much more in the other extreme. If the sparsity structure is such that each data point only depends on one of disjoint groups of variables, and the data were distributed according to this structure, we would efficiently have several disjoint problems. Solving each of them locally, and adding up the results would solve the problem in single iteration — desired algorithm property (C).

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

### 6.3.7 Further Notes

Looking at the Proposition 42, we identify equivalence of two algorithms, take the second one and try modify it to make it suitable for the setting of federated optimization. A question naturally arise: Is it possible to achieve the same by modifying the first algorithm suitable for federated optimization — by only altering the local optimization objective?

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

## 6.4 Experiments

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

### 6.4.1 Predicting Comments on Public Google+ Posts

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

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

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

It is natural to expect that different users can exhibit very different patterns in the data generated. This is indeed the case, and hence the distribution to nodes cannot be considered an IID sample from the overall distribution. Since we have a bag-of-words model, our data are very sparse — most posts contain only small fraction of all the words in the dictionary.

---

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

This, together with the fact that the data are naturally clustered on a per-user basis, creates additional challenge that is not present in the traditional distributed setting.

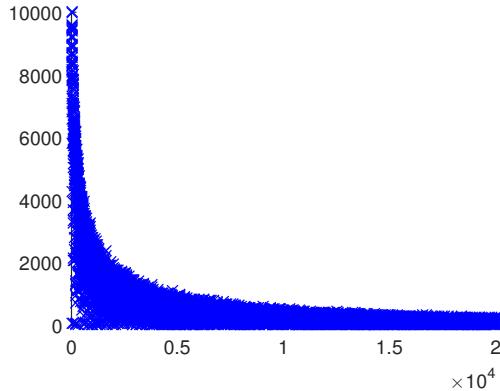


Figure 6.1: Features vs. appearance on nodes. The  $x$ -axis is a feature index, and the  $y$ -axis represents the number of nodes where a given feature is present.

Figure 6.1 shows the frequency of different features across nodes. Some features are present everywhere, such as the bias term, while most features are relatively rare. In particular, over 88% of features are present on fewer than 1,000 nodes. However, this distribution does not necessarily resemble the overall appearance of the features in data examples. For instance, while an unknown word is present in data of almost every user, it is far from being contained in every data point.

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

- If one chooses to predict  $-1$  (no comment), classification error is **33.16%**.
- The optimal solution of the global logistic regression problem yields **26.27%** test set error.
- Predicting the per-author majority from the training data yields **17.14%** test error. That is, predict  $+1$  or  $-1$  for all the posts of an author, based on which label was more common in that author’s training data. This indicates that knowing the author is actually more useful than knowing what they said, which is perhaps not surprising.

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

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

The performance of the Algorithm 13 is presented below. The only parameter that remains to be chosen by user is the stepsize  $h$ . We tried a set of stepsizes, and retrospectively choose one that works best — a typical practice in machine learning.

In Figure 6.2, we compare the following optimization algorithms<sup>9</sup>:

---

<sup>9</sup>We thank Mark Schmidt for his `prettyPlot` function, available on his website.

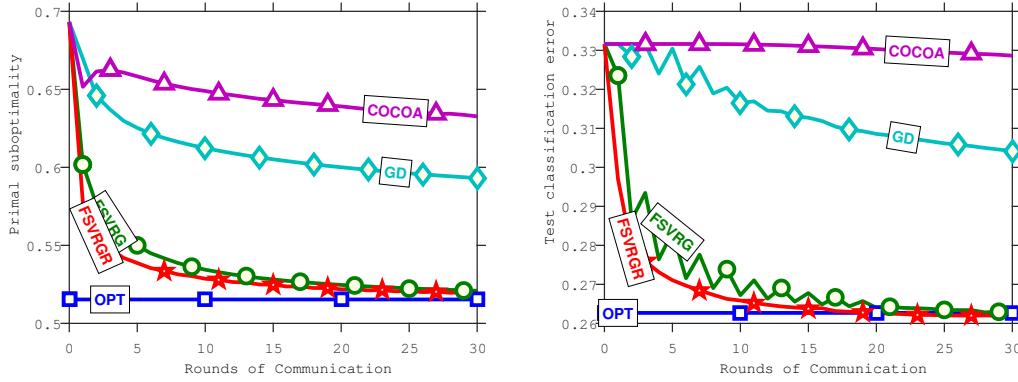


Figure 6.2: Rounds of communication vs. objective function (left) and test prediction error (right).

- The blue squares (OPT) represent the best possible offline value (the optimal value of the optimization task in the first plot, and the test error corresponding to the optimum in the second plot).
- The teal diamonds (GD) correspond to a simple distributed gradient descent.
- The purple triangles (COCOA) are for the CoCoA+ algorithm [104].
- The green circles (FSVRG) give values for our proposed algorithm.
- The red stars (FSVRGR) correspond to the same algorithm applied to the same problem with randomly reshuffled data. That is, we keep the unbalanced number of examples per node, but populate each node with randomly selected examples.

The first thing to notice is that CoCoA+ seems to be worse than trivial benchmark — distributed gradient descent. This behavior can be predicted from theory, as the overall convergence rate directly depends on the best choice of aggregation parameter  $\sigma'$ . For sparse problems, it is upperbounded by the maximum of the values reported in Figure 6.1, which is  $K$ , and it is close to it also in practice. Although it is expected that the algorithm could be modified to depend on average of these quantities (which could be orders of magnitude smaller), akin to coordinate descent algorithms [148], it has not been done yet. Note that other communication efficient algorithms fail to converge altogether.

The algorithm we propose, FSVRG, converges to optimal test classification accuracy in just 30 iterations. Recall that in the setting of federated optimization we introduced in Section 6.1.2, minimization of rounds of communication is the principal goal. However, concluding that the approach is stunningly superior to existing methods would not be completely fair nor correct. The conclusion is that the *FSVRG is the first algorithm to tackle federated optimization*, a problem that existing methods fail to generalize to. It is important to stress that none of the existing methods were designed with these particular challenges in mind, and we formulate the first benchmark.

Since the core reason other methods fail to converge is the non-IID data distribution, we test our method on the same problem, with data randomly reshuffled among the same number of nodes (FSVRGR; red stars). Since the difference in convergence is subtle, we can conclude that the techniques described in Section 6.3.6 serve its purpose and make the algorithm robust to challenges present in federated optimization.

This experiment demonstrates that learning from massively decentralized data, clustered on a per-user basis is indeed problem we can tackle in practice. Since the earlier version of this work [82], additional experimental results were presented in [115]. We refer the reader to this paper for experiments in more challenging setting of deep learning, and a further discussion on how such system would be implemented in practice.

## 6.5 Conclusions and Future Challenges

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

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

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

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

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

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

## 6.6 Appendix: Distributed Optimization via Quadratic Perturbations

This appendix follows from the discussion motivating DANE algorithm by a general algorithmic perturbation template (6.9) for  $\lambda$ -strongly convex objectives. We use this to propose a similar but new method, which unlike DANE converges under arbitrary data partitioning  $\{\mathcal{P}_k\}_{k=1}^K$ , and we highlight its relation to the dual CoCoA algorithm for distributed optimization.

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

### 6.6.1 New Method

We now present a new method (Algorithm 14), which also belongs to the family of quadratic perturbation methods (6.9). However, the perturbation vectors  $a_k^t$  are different from those of

---

**Algorithm 14** Primal Method

---

```

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

```

---

DANE. In particular, we set

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

where  $\eta > 0$  is a parameter, and the vectors  $g_k^t$  are maintained by the method. As we show in Lemma 45, Algorithm 14 satisfies

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

for all iterations  $t$ . This implies that  $\frac{1}{K} \sum_{k=1}^K a_k^t = (1 - \eta) \nabla P(w^t)$ . That is, both DANE and the new method use a linear perturbation which, when averaged over the nodes, involves the gradient of the objective function  $f$  at the latest iterate  $w^t$ . Therefore, the methods have one more property in common beyond both being of the form (6.9). However, as we shall see in the rest of this section, Algorithm 14 allows an insightful dual interpretation. Moreover, while DANE may not converge for arbitrary problems (even when restricted to ridge regression)—and is only known to converge under the assumption that the data stored on each node are in some precise way similar, Algorithm 14 converges for any ridge regression problem and any data partitioning.

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

**Lemma 45.** *For all  $t \geq 0$  we have  $\sum_{k=1}^K g_k^t = 0$ .*

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

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

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

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

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

### 6.6.2 L2-Regularized Linear Predictors

In the rest of this section we consider the case of L2-regularized *linear predictors*. That is, we focus on problem (6.1) with  $f_i$  of the form

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

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

$$\min_{w \in \mathbb{R}^d} \left\{ P(w) \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n \phi_i(x_i^T w) + \frac{\lambda}{2} \|w\|^2 \right\}. \quad (6.11)$$

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

Let  $X = [x_1, \dots, x_n] \in \mathbb{R}^{d \times n}$ . As described in Section 6.3.3, we assume that the data  $(x_i, y_i)_{i=1}^n$  is distributed among  $K$  nodes of a computer cluster as follows: node  $k = 1, 2, \dots, K$  contains pairs  $(x_i, y_i)$  for  $i \in \mathcal{P}_k$ , where  $\mathcal{P}_1, \dots, \mathcal{P}_K$  forms a partition of the set  $[n] = \{1, 2, \dots, n\}$ . Letting  $X = [X_1, \dots, X_K]$ , where  $X_k \in \mathbb{R}^{d \times |\mathcal{P}_k|}$  is a submatrix of  $A$  corresponding to columns  $i \in \mathcal{P}_k$ , and  $y_k \in \mathbb{R}^{|\mathcal{P}_k|}$  is the subvector of  $y$  corresponding to entries  $i \in \mathcal{P}_k$ . Hence, node  $k$  contains the pair  $(X_k, y_k)$ . With this notation, we can write the problem in the form (6.8), where

$$F_k(w) = \frac{K}{n} \sum_{i \in \mathcal{P}_k} \phi_i(x_i^T w) + \frac{\lambda}{2} \|w\|^2. \quad (6.12)$$

### 6.6.3 A Dual Method: Dual Block Proximal Gradient Ascent

The dual of (6.11) is the problem

$$\max_{\alpha \in \mathbb{R}^n} \left\{ D(\alpha) \stackrel{\text{def}}{=} -\frac{1}{2\lambda n^2} \|X\alpha\|^2 - \frac{1}{n} \sum_{i=1}^n \phi_i^*(-\alpha_i) \right\}, \quad (6.13)$$

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

**From dual solution to a primal solution.** It is well known that if  $\alpha^*$  is the optimal solution of the dual problem (6.11), then  $w^* \stackrel{\text{def}}{=} \frac{1}{\lambda n} X\alpha^*$  is the optimal solution of the primal problem. Therefore, for any dual algorithm producing a sequence of iterates  $\alpha^t$ , we can define a corresponding primal algorithm via the linear mapping

$$w^t \stackrel{\text{def}}{=} \frac{1}{\lambda n} X\alpha^t. \quad (6.14)$$

Clearly, if  $\alpha^t \rightarrow \alpha^*$ , then  $w^t \rightarrow w^*$ . We shall now design a method for maximizing the dual function  $D$  and then in Theorem 46 we claim that for quadratic loss functions, Algorithm 14 arises as an image, defined via (6.14), of dual iterations of this dual ascent method.

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

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

---

**Algorithm 15** Dual Method

---

```

1: Input:  $\sigma \in [1, K]$ 
2: Choose:  $\alpha_k^0 \in \mathbb{R}^{|\mathcal{P}_k|}$  for  $k = 1, 2, \dots, K$ 
3: for  $t = 0, 1, 2, \dots$  do
4:   for  $k = 1$  to  $K$  do
5:      $h_k^{t+1} = \arg \min_{u \in \mathbb{R}^{|\mathcal{P}_k|}} D_k^t(u)$  ▷ See (6.15)
6:   end for
7:    $\alpha^{t+1} = \alpha^t + h^t$ 
8: end for
9: return  $w^t$ 

```

---

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

$$\|h\|_B^2 \stackrel{\text{def}}{=} \sum_{k=1}^K \|X_k h_k\|^2.$$

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

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

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

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

$$h_k^t := \arg \min_{u \in \mathbb{R}^{|\mathcal{P}_k|}} \left\{ D_k^t(u) \stackrel{\text{def}}{=} \frac{1}{\lambda n^2} \langle \nabla_k \xi(\alpha^t), u \rangle + \frac{\sigma}{2\lambda n^2} \|X_k u\|^2 + \frac{1}{n} \sum_{i \in \mathcal{P}_k} \phi_i^*(-\alpha_i^t - u_i) \right\}. \quad (6.15)$$

Having computed  $h_k^t$  for all  $k$ , we can set  $\alpha_k^{t+1} = \alpha_k^t + h_k^t$  for all  $k$ , or equivalently,  $\alpha^{t+1} = \alpha^t + h^t$ . This is formalized as Algorithm 15. Algorithm 15 is a proximal gradient ascent method applied to the dual problem, with smoothness being measured using the block norm  $\|h\|_B$ . It is known that gradient ascent converges at a linear rate for smooth and strongly convex (for minimization problems) objectives.

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

**Theorem 46** (Equivalence of Algorithms 14 and 15 for Quadratic Loss). *Consider the ridge regression problem. That is, set  $\phi_i(t) = \frac{1}{2}(t - y_i)^2$  for all  $i$ . Assume  $\alpha_1^0, \dots, \alpha_K^0$  is chosen in the same way in Algorithms 14 and 15. Then the dual iterates  $\alpha^t$  and the primal iterates  $w^t$  produced by the two algorithms are related via (6.14) for all  $t \geq 0$ .*

Since the dual method converges linearly, in view of the above theorem, so does the primal method. Here we only remark that the popular algorithm CoCoA+ [104] arises if Step 5 in Algorithm 15 is done inexactly. Hence, we show that duality provides a deep relationship between the CoCoA+ and DANE algorithms, which were previously considered completely different.

#### 6.6.4 Proof of Theorem 46

In this part we prove the theorem.

**Primal and Dual Problems.** Since  $\phi_i(t) = \frac{1}{2}(t - y_i)^2$ , the primal problem (6.11) is a ridge regression problem of the form

$$\min_{w \in \mathbb{R}^d} P(w) = \frac{1}{2n} \|X^T w - y\|^2 + \frac{\lambda}{2} \|w\|^2, \quad (6.16)$$

where  $X \in \mathbb{R}^{d \times n}$  and  $y \in \mathbb{R}^n$ . In view of (6.13), the dual of (6.16) is

$$\min_{\alpha \in \mathbb{R}^n} D(\alpha) = \frac{1}{2\lambda n^2} \|X\alpha\|^2 + \frac{1}{2n} \|\alpha\|^2 - \frac{1}{n} y^T \alpha. \quad (6.17)$$

**Primal Problem: Distributed Setup.** The primal objective function is of the form (6.8), where in view of (6.12), we have  $F_k(w) = \frac{K}{2n} \|X_k^T w - y_k\|^2 + \frac{\lambda}{2} \|w\|^2$ . Therefore,

$$\nabla F_k(w) = \frac{K}{n} X_k (X_k^T w - y_k) + \lambda w \quad (6.18)$$

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

**Dual Method.** Since  $D$  is a quadratic, we have

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

with

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

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

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

Next, we shall define

$$h_k^t \stackrel{\text{def}}{=} \arg \min_{h_k \in \mathbb{R}^{|P_k|}} \frac{\sigma}{2\lambda n} \|X_k h_k\|^2 + \frac{1}{2} \|h_k\|^2 - (y_k - X_k^T w^t - \alpha_k^t)^T h_k \quad (6.19)$$

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

$$\alpha^{t+1} = \alpha^t + h^t. \quad (6.20)$$

**Primal Version of the Dual Method.** Note that (6.19) has the same form as (6.17), with  $X$  replaced by  $X_k$ ,  $\lambda$  replaced by  $\lambda/\sigma$  and  $y$  replaced by  $c_k := y_k - X_k^T w^t - \alpha_k^t$ . Hence, we know that

$$s_k^t \stackrel{\text{def}}{=} \frac{1}{(\lambda/\sigma)n} X_k h_k^t \quad (6.21)$$

is the optimal solution of the primal problem of (6.22):

$$s_k^t = \arg \min_{s \in \mathbb{R}^d} \frac{1}{2n} \|X_k^T s - c_k\|^2 + \frac{\lambda/\sigma}{2} \|s\|^2. \quad (6.22)$$

Hence, the primal version of method (6.20) is given by

$$\begin{aligned} w^{t+1} &\stackrel{(6.14)}{=} \frac{1}{\lambda n} X \alpha^{t+1} \stackrel{(6.20)}{=} \frac{1}{\lambda n} X(\alpha^t + h^t) \stackrel{(6.14)}{=} w^t + \frac{1}{\lambda n} \sum_{k=1}^K X_k h_k^t \\ &= \frac{1}{K} \sum_{k=1}^K \left( w^t + \frac{K}{\sigma} \frac{\sigma}{\lambda n} X_k h_k^t \right) \stackrel{(6.21)}{=} \frac{1}{K} \sum_{k=1}^K \left( w^t + \frac{K}{\sigma} s_k^t \right). \end{aligned}$$

With the change of variables  $w := w^t + \frac{K}{\sigma} s$  (i.e.,  $s = \frac{\sigma}{K}(w - w^t)$ ), from (6.22) we know that  $w_k^{t+1} := w^t + \frac{K}{\sigma} s_k^t$  solves

$$w_k^{t+1} = \arg \min_{w \in \mathbb{R}^d} \left\{ L_k(w) \stackrel{\text{def}}{=} \frac{1}{2n} \left\| X_k^T \frac{\sigma}{K} (w - w^t) - c_k \right\|^2 + \frac{\lambda/\sigma}{2} \left\| \frac{\sigma}{K} (w - w^t) \right\|^2 \right\} \quad (6.23)$$

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

Let us now rewrite the function in (6.23) so as to connect it to Algorithm 14:

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

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

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

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

$$\begin{aligned}
\eta z_k^t &= \eta \frac{K}{n} X_k (X_k^T w^t - y_k + \alpha_k^t) \\
&= \eta \frac{K}{n} X_k (X_k^T w^t - y_k) + \eta \frac{K}{n} X_k \alpha_k^t \\
&= \eta \left( \frac{K}{n} X_k (X_k^T w^t - y_k) + \lambda w^t \right) + \eta \left( \frac{K}{n} X_k \alpha_k^t - \lambda w^t \right) \\
&\stackrel{(6.18)}{=} \eta \nabla F_k(w^t) + \eta \left( \frac{K}{n} X_k \alpha_k^t - \lambda w^t \right).
\end{aligned}$$

Due to the definition of  $g_k^0$  in Step 5 of Algorithm 14 as  $g_k^0 = \eta(\frac{K}{n} X_k \alpha_k^0 - \lambda w^0)$ , we observe that the claim holds for  $t = 0$ . If we show that

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

for all  $t \geq 0$ , then we are done. This can be shown by induction. This finishes the proof of Theorem 46.

# Chapter 7

## Randomized Distributed Mean Estimation: Accuracy vs Communication

### 7.1 Introduction

In this chapter, we address the problem of approximately computing the arithmetic mean of  $n$  vectors,  $X_1, \dots, X_n \in \mathbb{R}^d$ , stored in a distributed fashion across  $n$  compute nodes, subject to a constraint on the communication cost.

In particular, we consider a star network topology with a single server at the center and  $n$  nodes connected to it. All nodes send an encoded (possibly via a lossy randomized transformation) version of their vector to the server, after which the server performs a decoding operation to estimate the true mean

$$X \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n X_i.$$

The purpose of the encoding operation is to compress the vector so as to save on communication cost, which is typically the bottleneck in practical applications.

To better illustrate the setup, consider the naive approach in which all nodes send the vectors without performing any encoding operation, followed by the application of a simple averaging decoder by the server. This results in zero estimation error at the expense of maximum communication cost of  $n dr$  bits, where  $r$  is the number of bits needed to communicate a single floating point entry/coordinate of  $X_i$ .

#### 7.1.1 Background and Contributions

The distributed mean estimation problem was recently studied in a statistical framework where it is assumed that the vectors  $X_i$  are independent and identically distributed samples from some specific underlying distribution. In such a setup, the goal is to estimate the true mean of the underlying distribution [200, 197, 65, 25]. These works formulate lower and upper bounds on the communication cost needed to achieve the minimax optimal estimation error.

In contrast, we do not make any statistical assumptions on the source of the vectors, and study the trade-off between expected communication costs and mean square error of the estimate. Arguably, this setup is a more robust and accurate model of the distributed mean estimation problems arising as subproblems in applications such as reduce-all operations within algorithms for distributed and federated optimization [149, 105, 104, 147, 83]. In these applications, the averaging operations need to be done repeatedly throughout the iterations of a master learning/optimization algorithm, and the vectors  $\{X_i\}$  correspond to updates to a global model/variable. In these applications, the vectors evolve throughout the iterative process in a complicated pattern, typically approaching zero as the master algorithm converges to optimality. Hence, their statistical properties change, which renders fixed statistical assumptions not

satisfied in practice.

For instance, when training a deep neural network model in a distributed environment, the vector  $X_i$  corresponds to a stochastic gradient based on a minibatch of data stored on node  $i$ . In this setup we do not have any useful prior statistical knowledge about the high-dimensional vectors to be aggregated. It has recently been observed that when communication cost is high, which is typically the case for commodity clusters, and even more so in a federated optimization framework, it is can be very useful to sacrifice on estimation accuracy in favor of reduced communication [115, 88].

In this chapter we propose a *parametric family of randomized methods for estimating the mean  $X$* , with parameters being a set of *probabilities*  $p_{ij}$  for  $i = 1, \dots, n$  and  $j = 1, 2, \dots, d$  and *node centers*  $\mu_i \in \mathbb{R}^d$  for  $i = 1, 2, \dots, n$ . The exact meaning of these parameters is explained in Section 7.3. By varying the probabilities, at one extreme, we recover the exact method described, enjoying zero estimation error at the expense of full communication cost. At the opposite extreme are methods with arbitrarily small expected communication cost, which is achieved at the expense of suffering an exploding estimation error. Practical methods appear somewhere on the continuum between these two extremes, depending on the specific requirements of the application at hand. Suresh et al. [171] propose a method combining a pre-processing step via a random structured rotation, followed by randomized binary quantization. Their quantization protocol arises as a suboptimal special case of our parametric family of methods.

To illustrate our results, consider the special case in which we choose to communicate a single bit per element of  $X_i$  only. We then obtain an  $\mathcal{O}\left(\frac{r}{n}R\right)$  bound on the mean square error, where  $r$  is number of bits used to represent a floating point value, and  $R = \frac{1}{n} \sum_{i=1}^n \|X_i - \mu_i 1\|^2$  with  $\mu_i \in \mathbb{R}$  being the average of elements of  $X_i$ , and  $1$  the all-ones vector in  $\mathbb{R}^d$  (see Example 7 in Section 7.5). Note that this bound improves upon the performance of the method of [171] in two aspects. First, the bound is independent of  $d$ , improving from logarithmic dependence. Further, due to a preprocessing rotation step, their method requires  $\mathcal{O}(d \log d)$  time to be implemented on each node, while our method is linear in  $d$ . This and other special cases are summarized in Table 7.1 in Section 7.5.

While the above already improves upon the state of the art, the improved results are in fact obtained for a suboptimal choice of the parameters of our method (constant probabilities  $p_{ij}$ , and node centers fixed to the mean  $\mu_i$ ). One can decrease the MSE further by optimizing over the probabilities and/or node centers (see Section 7.6). However, apart from a very low communication cost regime in which we have a closed form expression for the optimal probabilities, the problem needs to be solved numerically, and hence we do not have expressions for how much improvement is possible. We illustrate the effect of fixed and optimal probabilities on the trade-off between communication cost and MSE experimentally on a few selected datasets in Section 7.6 (see Figure 7.1).

### 7.1.2 Outline

In Section 7.2 we formalize the concepts of encoding and decoding protocols. In Section 7.3 we describe a parametric family of randomized (and unbiased) encoding protocols and give a simple formula for the mean squared error. Subsequently, in Section 7.4 we formalize the notion of communication cost, and describe several communication protocols, which are optimal under different circumstances. We give simple instantiations of our protocol in Section 7.5, illustrating the trade-off between communication costs and accuracy. In Section 7.6 we address the question of the optimal choice of parameters of our protocol. Brief remarks on possible extensions that are summarized in Section 7.7. Finally, we present experimental application in Federated Learning in Section 7.8.

## 7.2 Three Protocols

In this work we consider (randomized) *encoding protocols*  $\alpha$ , *communication protocols*  $\beta$  and *decoding protocols*  $\gamma$  using which the averaging is performed inexactly as follows. Node  $i$  computes a (possibly stochastic) estimate of  $X_i$  using the encoding protocol, which we denote  $Y_i = \alpha(X_i) \in \mathbb{R}^d$ , and sends it to the server using communication protocol  $\beta$ . By  $\beta(Y_i)$  we

denote the number of bits that need to be transferred under  $\beta$ . The server then estimates  $X$  using the decoding protocol  $\gamma$  of the estimates:

$$Y \stackrel{\text{def}}{=} \gamma(Y_1, \dots, Y_n).$$

The objective of this work is to study the trade-off between the (expected) number of bits that need to be communicated, and the accuracy of  $Y$  as an estimate of  $X$ .

In this work we focus on encoders which are unbiased, in the following sense.

**Definition 47** (Unbiased and Independent Encoder). *We say that encoder  $\alpha$  is unbiased if  $\mathbb{E}_\alpha [\alpha(X_i)] = X_i$  for all  $i = 1, 2, \dots, n$ . We say that it is independent, if  $\alpha(X_i)$  is independent from  $\alpha(X_j)$  for all  $i \neq j$ .*

**Example 48** (Identity Encoder). *A trivial example of an encoding protocol is the identity function:  $\alpha(X_i) = X_i$ . It is both unbiased and independent. This encoder does not lead to any savings in communication that would be otherwise infeasible though.*

We now formalize the notion of accuracy of estimating  $X$  via  $Y$ . Since  $Y$  can be random, the notion of accuracy will naturally be probabilistic.

**Definition 49** (Estimation Error / Mean Squared Error). *The mean squared error of protocol  $(\alpha, \gamma)$  is the quantity*

$$\begin{aligned} MSE_{\alpha, \gamma}(X_1, \dots, X_n) &= \mathbb{E}_{\alpha, \gamma} [\|Y - X\|^2] \\ &= \mathbb{E}_{\alpha, \gamma} [\|\gamma(\alpha(X_1), \dots, \alpha(X_n)) - X\|^2]. \end{aligned}$$

To illustrate the above concept, we now give a few examples:

**Example 50** (Averaging Decoder). *If  $\gamma$  is the averaging function, i.e.,  $\gamma(Y_1, \dots, Y_n) = \frac{1}{n} \sum_{i=1}^n Y_i$ , then*

$$MSE_{\alpha, \gamma}(X_1, \dots, X_n) = \frac{1}{n^2} \mathbb{E}_\alpha \left[ \left\| \sum_{i=1}^n \alpha(X_i) - X_i \right\|^2 \right].$$

The next example generalizes the identity encoder and averaging decoder.

**Example 51** (Linear Encoder and Inverse Linear Decoder). *Let  $A : \mathbb{R}^d \rightarrow \mathbb{R}^d$  be linear and invertible. Then we can set  $Y_i = \alpha(X_i) \stackrel{\text{def}}{=} AX_i$  and  $\gamma(Y_1, \dots, Y_n) \stackrel{\text{def}}{=} A^{-1} \left( \frac{1}{n} \sum_{i=1}^n Y_i \right)$ . If  $A$  is random, then  $\alpha$  and  $\gamma$  are random (e.g., a structured random rotation, see [190]). Note that*

$$\gamma(Y_1, \dots, Y_n) = \frac{1}{n} \sum_{i=1}^n A^{-1} Y_i = \frac{1}{n} \sum_{i=1}^n X_i = X,$$

and hence the MSE of  $(\alpha, \gamma)$  is zero.

We shall now prove a simple result for unbiased and independent encoders used in subsequent sections.

**Lemma 52** (Unbiased and Independent Encoder + Averaging Decoder). *If the encoder  $\alpha$  is unbiased and independent, and  $\gamma$  is the averaging decoder, then*

$$MSE_{\alpha, \gamma}(X_1, \dots, X_n) = \frac{1}{n^2} \sum_{i=1}^n \mathbb{E}_\alpha [\|Y_i - X_i\|^2] = \frac{1}{n^2} \sum_{i=1}^n \mathbf{Var}_\alpha [\alpha(X_i)].$$

*Proof.* Note that  $\mathbb{E}_\alpha [Y_i] = X_i$  for all  $i$ . We have

$$\begin{aligned} MSE_\alpha(X_1, \dots, X_n) &= \mathbb{E}_\alpha [\|Y - X\|^2] \\ &\stackrel{(*)}{=} \frac{1}{n^2} \mathbb{E}_\alpha \left[ \left\| \sum_{i=1}^n Y_i - X_i \right\|^2 \right] \\ &\stackrel{(**)}{=} \frac{1}{n^2} \sum_{i=1}^n \mathbb{E}_\alpha [\|Y_i - \mathbb{E}_\alpha [Y_i]\|^2] \\ &= \frac{1}{n^2} \sum_{i=1}^n \text{Var}_\alpha [\alpha(X_i)], \end{aligned}$$

where  $(*)$  follows from unbiasedness and  $(**)$  from independence.  $\square$

One may wish to define the encoder as a combination of two or more separate encoders:  $\alpha(X_i) = \alpha_2(\alpha_1(X_i))$ . See [171] for an example where  $\alpha_1$  is a random rotation and  $\alpha_2$  is binary quantization.

### 7.3 A Family of Randomized Encoding Protocols

Let  $X_1, \dots, X_n \in \mathbb{R}^d$  be given. We shall write  $X_i = (X_i(1), \dots, X_i(d))$  to denote the entries of vector  $X_i$ . In addition, with each  $i$  we also associate a parameter  $\mu_i \in \mathbb{R}$ . We refer to  $\mu_i$  as the center of data at node  $i$ , or simply as *node center*. For now, we assume these parameters are fixed and we shall later comment on how to choose them optimally.

We shall define *support* of  $\alpha$  on node  $i$  to be the set  $S_i \stackrel{\text{def}}{=} \{j : Y_i(j) \neq \mu_i\}$ . We now define two parametric families of randomized encoding protocols. The first results in  $S_i$  of random size, the second has  $S_i$  of a fixed size.

#### 7.3.1 Encoding Protocol with Variable-size Support

With each pair  $(i, j)$  we associate a parameter  $0 < p_{ij} \leq 1$ , representing a probability. The collection of parameters  $\{p_{ij}, \mu_i\}$  defines an encoding protocol  $\alpha$  as follows:

$$Y_i(j) = \begin{cases} \frac{X_i(j)}{p_{ij}} - \frac{1-p_{ij}}{p_{ij}}\mu_i & \text{with probability } p_{ij}, \\ \mu_i & \text{with probability } 1 - p_{ij}. \end{cases} \quad (7.1)$$

**Remark 53.** Enforcing the probabilities to be positive, as opposed to nonnegative, leads to vastly simplified notation in what follows. However, it is more natural to allow  $p_{ij}$  to be zero, in which case we have  $Y_i(j) = \mu_i$  with probability 1. This raises issues such as potential lack of unbiasedness, which can be resolved, but only at the expense of a larger-than-reasonable notational overload.

In the rest of this section, let  $\gamma$  be the averaging decoder (Example 50). Since  $\gamma$  is fixed and deterministic, we shall for simplicity write  $\mathbb{E}_\alpha [\cdot]$  instead of  $\mathbb{E}_{\alpha, \gamma} [\cdot]$ . Similarly, we shall write  $MSE_\alpha(\cdot)$  instead of  $MSE_{\alpha, \gamma}(\cdot)$ .

We now prove two lemmas describing properties of the encoding protocol  $\alpha$ . Lemma 54 states that the protocol yields an unbiased estimate of the average  $X$  and Lemma 55 provides the expected mean square error of the estimate.

**Lemma 54** (Unbiasedness). *The encoder  $\alpha$  defined in (7.1) is unbiased. That is,  $\mathbb{E}_\alpha [\alpha(X_i)] = X_i$  for all  $i$ . As a result,  $Y$  is an unbiased estimate of the true average:  $\mathbb{E}_\alpha [Y] = X$ .*

*Proof.* Due to linearity of expectation, it is enough to show that  $\mathbb{E}_\alpha [Y(j)] = X(j)$  for all  $j$ . Since  $Y(j) = \frac{1}{n} \sum_{i=1}^n Y_i(j)$  and  $X(j) = \frac{1}{n} \sum_{i=1}^n X_i(j)$ , it suffices to show that  $\mathbb{E}_\alpha [Y_i(j)] = X_i(j)$ :

$$\mathbb{E}_\alpha [Y_i(j)] = p_{ij} \left( \frac{X_i(j)}{p_{ij}} - \frac{1-p_{ij}}{p_{ij}}\mu_i(j) \right) + (1-p_{ij})\mu_i(j) = X_i(j),$$

and the claim is proved.  $\square$

**Lemma 55** (Mean Squared Error). *Let  $\alpha = \alpha(p_{ij}, \mu_i)$  be the encoder defined in (7.1). Then*

$$MSE_\alpha(X_1, \dots, X_n) = \frac{1}{n^2} \sum_{i,j} \left( \frac{1}{p_{ij}} - 1 \right) (X_i(j) - \mu_i)^2. \quad (7.2)$$

*Proof.* Using Lemma 52, we have

$$\begin{aligned} MSE_\alpha(X_1, \dots, X_n) &= \frac{1}{n^2} \sum_{i=1}^n \mathbb{E}_\alpha \left[ \|Y_i - X_i\|^2 \right] \\ &= \frac{1}{n^2} \sum_{i=1}^n \mathbb{E}_\alpha \left[ \sum_{j=1}^d (Y_i(j) - X_i(j))^2 \right] \\ &= \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^d \mathbb{E}_\alpha [(Y_i(j) - X_i(j))^2]. \end{aligned} \quad (7.3)$$

For any  $i, j$  we further have

$$\begin{aligned} \mathbb{E}_\alpha [(Y_i(j) - X_i(j))^2] &= p_{ij} \left( \frac{X_i(j)}{p_{ij}} - \frac{1-p_{ij}}{p_{ij}} \mu_i - X_i(j) \right)^2 + (1-p_{ij}) (\mu_i - X_i(j))^2 \\ &= \frac{(1-p_{ij})^2}{p_{ij}} (X_i(j) - \mu_i)^2 + (1-p_{ij}) (\mu_i - X_i(j))^2 \\ &= \left( \frac{1-p_{ij}}{p_{ij}} \right) (X_i(j) - \mu_i)^2. \end{aligned}$$

It suffices to substitute the above into (7.3).  $\square$

### 7.3.2 Encoding Protocol with Fixed-size Support

Here we propose an alternative encoding protocol, one with deterministic support size. As we shall see later, this results in deterministic communication cost.

Let  $\sigma_k(d)$  denote the set of all subsets of  $\{1, 2, \dots, d\}$  containing  $k$  elements. The protocol  $\alpha$  with a single integer parameter  $k$  is then working as follows: First, each node  $i$  samples  $\mathcal{D}_i \in \sigma_k(d)$  uniformly at random, and then sets

$$Y_i(j) = \begin{cases} \frac{dX_i(j)}{k} - \frac{d-k}{k} \mu_i & \text{if } j \in \mathcal{D}_i, \\ \mu_i & \text{otherwise.} \end{cases} \quad (7.4)$$

Note that due to the design, the size of the support of  $Y_i$  is always  $k$ , i.e.,  $|S_i| = k$ . Naturally, we can expect this protocol to perform practically the same as the protocol (7.1) with  $p_{ij} = k/d$ , for all  $i, j$ . Lemma 57 indeed suggests this is the case. While this protocol admits a more efficient communication protocol (as we shall see in Section ), protocol (7.1) enjoys a larger parameters space, ultimately leading to better MSE. We comment on this tradeoff in subsequent sections.

As for the data-dependent protocol, we prove basic properties. The proofs are similar to those of Lemmas 54 and 55 and we defer them to Appendix 7.9.

**Lemma 56** (Unbiasedness). *The encoder  $\alpha$  defined in (7.1) is unbiased. That is,  $\mathbb{E}_\alpha [\alpha(X_i)] = X_i$  for all  $i$ . As a result,  $Y$  is an unbiased estimate of the true average:  $\mathbb{E}_\alpha [Y] = X$ .*

**Lemma 57** (Mean Squared Error). *Let  $\alpha = \alpha(k)$  be encoder defined as in (7.4). Then*

$$MSE_\alpha(X_1, \dots, X_n) = \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^d \left( \frac{d-k}{k} \right) (X_i(j) - \mu_i)^2. \quad (7.5)$$

## 7.4 Communication Protocols

Having defined the encoding protocols  $\alpha$ , we need to specify the way the encoded vectors  $Y_i = \alpha(X_i)$ , for  $i = 1, 2, \dots, n$ , are communicated to the server. Given a specific *communication protocol*  $\beta$ , we write  $\beta(Y_i)$  to denote the (expected) number of bits that are communicated by node  $i$  to the server. Since  $Y_i = \alpha(X_i)$  is in general not deterministic,  $\beta(Y_i)$  can be a random variable.

**Definition 58** (Communication Cost). *The communication cost of communication protocol  $\beta$  under randomized encoding  $\alpha$  is the total expected number of bits transmitted to the server:*

$$C_{\alpha,\beta}(X_1, \dots, X_n) = \mathbb{E}_\alpha \left[ \sum_{i=1}^n \beta(\alpha(X_i)) \right]. \quad (7.6)$$

Given  $Y_i$ , a good communication protocol is able to encode  $Y_i = \alpha(X_i)$  using a few bits only. Let  $r$  denote the number of bits used to represent a floating point number. Let  $\bar{r}$  be the the number of bits representing  $\mu_i$ .

In the rest of this section we describe several communication protocols  $\beta$  and calculate their communication cost.

### 7.4.1 Naive

Represent  $Y_i = \alpha(X_i)$  as  $d$  floating point numbers. Then for all encoding protocols  $\alpha$  and all  $i$  we have  $\beta(\alpha(X_i)) = dr$ , whence

$$C_{\alpha,\beta} = \mathbb{E}_\alpha \left[ \sum_{i=1}^n \beta(\alpha(X_i)) \right] = ndr.$$

### 7.4.2 Varying-length

We will use a single variable for every element of the vector  $Y_i$ , which does not have constant size. The first bit decides whether the value represents  $\mu_i$  or not. If yes, end of variable, if not, next  $r$  bits represent the value of  $Y_i(j)$ . In addition, we need to communicate  $\mu_i$ , which takes  $\bar{r}$  bits<sup>1</sup>. We thus have

$$\beta(\alpha(X_i)) = \bar{r} + \sum_{j=1}^d (1_{(Y_i(j)=\mu_i)} + (r+1) \times 1_{(Y_i(j)\neq\mu_i)}), \quad (7.7)$$

where  $1_e$  is the indicator function of event  $e$ . The expected number of bits communicated is given by

$$\begin{aligned} C_{\alpha,\beta} &= \mathbb{E}_\alpha \left[ \sum_{i=1}^n \beta(\alpha(X_i)) \right] \stackrel{(7.7)}{=} n\bar{r} + \sum_{i=1}^n \sum_{j=1}^d (1 - p_{ij} + (r+1)p_{ij}) \\ &= n\bar{r} + \sum_{i=1}^n \sum_{j=1}^d (1 + rp_{ij}) \end{aligned}$$

In the special case when  $p_{ij} = p > 0$  for all  $i, j$ , we get

$$C_{\alpha,\beta} = n(\bar{r} + d + pdr).$$

---

<sup>1</sup>The distinction here is because  $\mu_i$  can be chosen to be data independent, such as 0, so we don't have to communicate anything (i.e.,  $\bar{r} = 0$ )

### 7.4.3 Sparse Communication Protocol for Encoder (7.1)

We can represent  $Y_i$  as a sparse vector; that is, a list of pairs  $(j, Y_i(j))$  for which  $Y_i(j) \neq \mu_i$ . The number of bits to represent each pair is  $\lceil \log(d) \rceil + r$ . Any index not found in the list, will be interpreted by server as having value  $\mu_i$ . Additionally, we have to communicate the value of  $\mu_i$  to the server, which takes  $\bar{r}$  bits. We assume that the value  $d$ , size of the vectors, is known to the server. Hence,

$$\beta(\alpha(X_i)) = \bar{r} + \sum_{j=1}^d 1_{(Y_i(j) \neq \mu_i)} \times (\lceil \log d \rceil + r).$$

Summing up through  $i$  and taking expectations, the communication cost is given by

$$C_{\alpha,\beta} = \mathbb{E}_\alpha \left[ \sum_{i=1}^n \beta(\alpha(X_i)) \right] = n\bar{r} + (\lceil \log d \rceil + r) \sum_{i=1}^n \sum_{j=1}^d p_{ij}. \quad (7.8)$$

In the special case when  $p_{ij} = p > 0$  for all  $i, j$ , we get

$$C_{\alpha,\beta} = n\bar{r} + (\lceil \log d \rceil + r)ndp.$$

**Remark 59.** A practical improvement upon this could be to (without loss of generality) assume that the pairs  $(j, Y_i(j))$  are ordered by  $j$ , i.e., we have  $\{(j_s, Y_i(j_s))\}_{s=1}^k$  for some  $k$  and  $j_1 < j_2 < \dots < j_k$ . Further, let us denote  $j_0 = 0$ . We can then use a variant of variable-length quantity [183] to represent the set  $\{(j_s - j_{s-1}, Y_i(j_s))\}_{s=1}^k$ . With careful design one can hope to reduce the  $\log(d)$  factor in the average case. Nevertheless, this does not improve the worst case analysis we focus on in this chapter, and hence we do not delve deeper in this.

### 7.4.4 Sparse Communication Protocol for Encoder (7.4)

We now describe a sparse communication protocol compatible only with fixed length encoder defined in (7.4). Note that subset selection can be compressed in the form of a random seed, letting us avoid the  $\log(d)$  factor in (7.8). This includes the protocol defined in (7.4) but also (7.1) with uniform probabilities  $p_{ij}$ .

In particular, we can represent  $Y_i$  as a sparse vector containing the list of the values for which  $Y_i(j) \neq \mu_i$ , ordered by  $j$ . Additionally, we need to communicate the value  $\mu_i$  (using  $\bar{r}$  bits) and a random seed (using  $\bar{r}_s$  bits), which can be used to reconstruct the indices  $j$ , corresponding to the communicated values. Note that for any fixed  $k$  defining protocol (7.4), we have  $|S_i| = k$ . Hence, communication cost is deterministic:

$$C_{\alpha,\beta} = \sum_{i=1}^n \beta(\alpha(X_i)) = n(\bar{r} + \bar{r}_s) + nkr. \quad (7.9)$$

In the case of the variable-size-support encoding protocol (7.1) with  $p_{ij} = p > 0$  for all  $i, j$ , the sparse communication protocol described here yields expected communication cost

$$C_{\alpha,\beta} = \mathbb{E}_\alpha \left[ \sum_{i=1}^n \beta(\alpha(X_i)) \right] = n(\bar{r} + \bar{r}_s) + ndpr. \quad (7.10)$$

### 7.4.5 Binary

If the elements of  $Y_i$  take only two different values,  $Y_i^{\min}$  or  $Y_i^{\max}$ , we can use a *binary communication protocol*. That is, for each node  $i$ , we communicate the values of  $Y_i^{\min}$  and  $Y_i^{\max}$  (using  $2r$  bits), followed by a single bit per element of the array indicating whether  $Y_i^{\max}$  or  $Y_i^{\min}$  should be used. The resulting (deterministic) communication cost is

$$C_{\alpha,\beta} = \sum_{i=1}^n \beta(\alpha(X_i)) = n(2r) + nd. \quad (7.11)$$

### 7.4.6 Discussion

In the above, we have presented several communication protocols of different complexity. However, it is not possible to claim any of them is the most efficient one. Which communication protocol is the best, depends on the specifics of the used encoding protocol. Consider the extreme case of encoding protocol (7.1) with  $p_{ij} = 1$  for all  $i, j$ . The naive communication protocol is clearly the most efficient, as all other protocols need to send some additional information.

However, in the interesting case when we consider small communication budget, the sparse communication protocols are the most efficient. Therefore, in the following sections, we focus primarily on optimizing the performance using these protocols.

## 7.5 Examples

In this section, we highlight on several instantiations of our protocols, recovering existing techniques and formulating novel ones. We comment on the resulting trade-offs between communication cost and estimation error.

### 7.5.1 Binary Quantization

We start by recovering an existing method, which turns every element of the vectors  $X_i$  into a particular binary representation.

**Example 60.** If we set the parameters of protocol (7.1) as  $\mu_i = X_i^{\min}$  and  $p_{ij} = \frac{X_i(j) - X_i^{\min}}{\Delta_i}$ , where  $\Delta_i \stackrel{\text{def}}{=} X_i^{\max} - X_i^{\min}$  (assume, for simplicity, that  $\Delta_i \neq 0$ ), we exactly recover the quantization algorithm proposed in [171]:

$$Y_i(j) = \begin{cases} X_i^{\max} & \text{with probability } \frac{X_i(j) - X_i^{\min}}{\Delta_i}, \\ X_i^{\min} & \text{with probability } \frac{X_i^{\max} - X_i(j)}{\Delta_i}. \end{cases} \quad (7.12)$$

Using the formula (7.2) for the encoding protocol  $\alpha$ , we get

$$MSE_\alpha = \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^d \frac{X_i^{\max} - X_i(j)}{X_i(j) - X_i^{\min}} (X_i(j) - X_i^{\min})^2 \leq \frac{d}{2n} \cdot \frac{1}{n} \sum_{i=1}^n \|X_i\|^2.$$

This exactly recovers the MSE bound established in [171, Theorem 1]. Using the binary communication protocol yields the communication cost of 1 bit per element if  $X_i$ , plus a two real-valued scalars (7.11).

**Remark 61.** If we use the above protocol jointly with randomized linear encoder and decoder (see Example 51), where the linear transform is the randomized Hadamard transform, we recover the method described in [171, Section 3] which yields improved  $MSE_\alpha = \frac{2 \log d + 2}{n} \cdot \frac{1}{n} \sum_{i=1}^n \|X_i\|^2$  and can be implemented in  $\mathcal{O}(d \log d)$  time.

### 7.5.2 Sparse Communication Protocols

Now we move to comparing the communication costs and estimation error of various instantiations of the encoding protocols, utilizing the deterministic sparse communication protocol and uniform probabilities.

For the remainder of this section, let us only consider instantiations of our protocol where  $p_{ij} = p > 0$  for all  $i, j$ , and assume that the node centers are set to the vector averages, i.e.,  $\mu_i = \frac{1}{d} \sum_{j=1}^d X_i(j)$ . Denote  $R = \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^d (X_i(j) - \mu_i)^2$ . For simplicity, we also assume that  $|S| = nd$ , which is what we can in general expect without any prior knowledge about the vectors  $X_i$ .

The properties of the following examples follow from Equations (7.2) and (7.10). When considering the communication costs of the protocols, keep in mind that the trivial benchmark is  $C_{\alpha, \beta} = ndr$ , which is achieved by simply sending the vectors unmodified. Communication

| Example                  | $p$        | $C_{\alpha,\beta}$                            | $MSE_{\alpha,\gamma}$      |
|--------------------------|------------|-----------------------------------------------|----------------------------|
| Example 62 (Full)        | 1          | $ndr$                                         | 0                          |
| Example 63 (Log MSE)     | $1/\log d$ | $n(\bar{r}_s + \bar{r}) + \frac{ndr}{\log d}$ | $(\log(d) - 1)\frac{R}{n}$ |
| Example 64 (1-bit)       | $1/r$      | $n(\bar{r}_s + \bar{r}) + nd$                 | $(r - 1)\frac{R}{n}$       |
| Example 66 (below 1-bit) | $1/d$      | $n(\bar{r}_s + \bar{r}) + nr$                 | $(d - 1)\frac{R}{n}$       |

Table 7.1: Summary of achievable communication cost and estimation error, for various choices of probability  $p$ .

cost of  $C_{\alpha,\beta} = nd$  corresponds to the interesting special case when we use (on average) one bit per element of each  $X_i$ .

**Example 62** (Full communication). *If we choose  $p = 1$ , we get*

$$C_{\alpha,\beta} = n(\bar{r}_s + \bar{r}) + ndr, \quad MSE_{\alpha,\gamma} = 0.$$

*In this case, the encoding protocol is lossless, which ensures  $MSE = 0$ . Note that in this case, we could get rid of the  $n(\bar{r}_s + \bar{r})$  factor by using naive communication protocol.*

**Example 63** (Log MSE). *If we choose  $p = 1/\log d$ , we get*

$$C_{\alpha,\beta} = n(\bar{r}_s + \bar{r}) + \frac{ndr}{\log d}, \quad MSE_{\alpha,\gamma} = \frac{\log(d) - 1}{n} R.$$

*This protocol order-wise matches the  $MSE$  of the method in Remark 61. However, as long as  $d > 2^r$ , this protocol attains this error with smaller communication cost. In particular, this is on expectation less than a single bit per element of  $X_i$ . Finally, note that the factor  $R$  is always smaller or equal to the factor  $\frac{1}{n} \sum_{i=1}^n \|X_i\|^2$  appearing in Remark 61.*

**Example 64** (1-bit per element communication). *If we choose  $p = 1/r$ , we get*

$$C_{\alpha,\beta} = n(\bar{r}_s + \bar{r}) + nd, \quad MSE_{\alpha,\gamma} = \frac{r-1}{n} R.$$

*This protocol communicates on expectation single bit per element of  $X_i$  (plus additional  $\bar{r}_s + \bar{r}$  bits per client), while attaining bound on  $MSE$  of  $\mathcal{O}(r/n)$ . To the best of our knowledge, this is the first method to attain this bound without additional assumptions.*

**Example 65** (Alternative 1-bit per element communication). *If we choose  $p = \frac{d-\bar{r}_s-\bar{r}}{dr}$ , we get*

$$C_{\alpha,\beta} = nd, \quad MSE_{\alpha,\gamma} = \frac{\frac{dr}{d-\bar{r}_s-\bar{r}} - 1}{n} R.$$

*This alternative protocol attains on expectation exactly single bit per element of  $X_i$ , with (a slightly more complicated)  $\mathcal{O}(r/n)$  bound on  $MSE$ .*

**Example 66** (Below 1-bit communication). *If we choose  $p = 1/d$ , we get*

$$C_{\alpha,\beta} = n(\bar{r}_s + \bar{r}) + nr, \quad MSE_{\alpha,\gamma} = \frac{d-1}{n} R.$$

*This protocol attains the  $MSE$  of protocol in Example 60 while at the same time communicating on average significantly less than a single bit per element of  $X_i$ .*

We summarize these examples in Table 7.1.

Using the deterministic sparse protocol, there is an obvious lower bound on the communication cost —  $n(\bar{r}_s + \bar{r})$ . We can bypass this threshold by using the sparse protocol, with a data-independent choice of  $\mu_i$ , such as 0, setting  $\bar{r} = 0$ . By setting  $p = \epsilon/d(\lceil \log d \rceil + r)$ , we get arbitrarily small expected communication cost of  $C_{\alpha,\beta} = \epsilon$ , and the cost of exploding estimation error  $MSE_{\alpha,\gamma} = \mathcal{O}(1/\epsilon n)$ .

Note that all of the above examples have random communication costs. What we present is the *expected* communication cost of the protocols. All the above examples can be modified to use the encoding protocol with fixed-size support defined in (7.4) with the parameter  $k$  set to the value of  $pd$  for corresponding  $p$  used above, to get the same results. The only practical difference is that the communication cost will be deterministic for each node, which can be useful for certain applications.

## 7.6 Optimal Encoders

Here we consider  $(\alpha, \beta, \gamma)$ , where  $\alpha = \alpha(p_{ij}, \mu_i)$  is the encoder defined in (7.1),  $\beta$  is the associated sparse communication protocol, and  $\gamma$  is the averaging decoder. Recall from Lemma 7.2 and (7.8) that the mean square error and communication cost are given by:

$$MSE_{\alpha, \gamma} = \frac{1}{n^2} \sum_{i,j} \left( \frac{1}{p_{ij}} - 1 \right) (X_i(j) - \mu_i)^2, \quad C_{\alpha, \beta} = n\bar{r} + (\lceil \log d \rceil + r) \sum_{i=1}^n \sum_{j=1}^d p_{ij}. \quad (7.13)$$

Having these closed-form formulae as functions of the parameters  $\{p_{ij}, \mu_i\}$ , we can now ask questions such as:

1. Given a communication budget, which encoding protocol has the smallest mean squared error?
2. Given a bound on the mean squared error, which encoder suffers the minimal communication cost?

Let us now address the first question; the second question can be handled in a similar fashion. In particular, consider the optimization problem

$$\begin{aligned} & \text{minimize} && \sum_{i,j} \left( \frac{1}{p_{ij}} - 1 \right) (X_i(j) - \mu_i)^2 \\ & \text{subject to} && \mu_i \in \mathbb{R}, \quad i = 1, 2, \dots, n \\ & && \sum_{i,j} p_{ij} \leq B \\ & && 0 < p_{ij} \leq 1, \quad i = 1, 2, \dots, n; \quad j = 1, 2, \dots, d, \end{aligned} \quad (7.14) \quad (7.15)$$

where  $B > 0$  represents a bound on the part of the total communication cost in (7.13) which depends on the choice of the probabilities  $p_{ij}$ .

Note that while the constraints in (7.14) are convex (they are linear), the objective is not jointly convex in  $\{p_{ij}, \mu_i\}$ . However, the objective is convex in  $\{p_{ij}\}$  and convex in  $\{\mu_i\}$ . This suggests a simple *alternating minimization* heuristic for solving the above problem:

1. Fix the probabilities and optimize over the node centers,
2. Fix the node centers and optimize over probabilities.

These two steps are repeated until a suitable convergence criterion is reached. Note that the first step has a closed form solution. Indeed, the problem decomposes across the node centers to  $n$  univariate unconstrained convex quadratic minimization problems, and the solution is given by

$$\mu_i = \frac{\sum_j w_{ij} X_i(j)}{\sum_j w_{ij}}, \quad w_{ij} \stackrel{\text{def}}{=} \frac{1}{p_{ij}} - 1. \quad (7.16)$$

The second step does not have a closed form solution in general; we provide an analysis of this step in Section 7.6.1.

**Remark 67.** Note that the upper bound  $\sum_{i,j} (X_i(j) - \mu_i)^2 / p_{ij}$  on the objective is jointly convex in  $\{p_{ij}, \mu_i\}$ . We may therefore instead optimize this upper bound by a suitable convex optimization algorithm.

**Remark 68.** An alternative and a more practical model to (7.14) is to choose per-node budgets  $B_1, \dots, B_n$  and require  $\sum_j p_{ij} \leq B_i$  for all  $i$ . The problem becomes separable across the nodes, and can therefore be solved by each node independently. If we set  $B = \sum_i B_i$ , the optimal solution obtained this way will lead to MSE which is lower bounded by the MSE obtained through (7.14).

### 7.6.1 Optimal Probabilities for Fixed Node Centers

Let the node centers  $\mu_i$  be fixed. Problem (7.14) (or, equivalently, step 2 of the alternating minimization method described above) then takes the form

$$\begin{aligned} & \text{minimize} && \sum_{i,j} \frac{(X_i(j) - \mu_i)^2}{p_{ij}} \\ & \text{subject to} && \sum_{i,j} p_{ij} \leq B \\ & && 0 < p_{ij} \leq 1, \quad i = 1, 2, \dots, n, \quad j = 1, 2, \dots, d. \end{aligned} \tag{7.17}$$

Let  $S = \{(i, j) : X_i(j) \neq \mu_i\}$ . Notice that as long as  $B \geq |S|$ , the optimal solution is to set  $p_{ij} = 1$  for all  $(i, j) \in S$  and  $p_{ij} = 0$  for all  $(i, j) \notin S$ .<sup>2</sup> In such a case, we have  $MSE_{\alpha, \gamma} = 0$ . Hence, we can without loss of generality assume that  $B \leq |S|$ .

While we are not able to derive a closed-form solution to this problem, we can formulate upper and lower bounds on the optimal estimation error, given a bound on the communication cost formulated via  $B$ .

**Theorem 69** (MSE-Optimal Protocols subject to a Communication Budget). *Consider problem (7.17) and fix any  $B \leq |S|$ . Using the sparse communication protocol  $\beta$ , the optimal encoding protocol  $\alpha$  has communication complexity*

$$C_{\alpha, \beta} = n\bar{r} + (\lceil \log d \rceil + r)B, \tag{7.18}$$

and the mean squared error satisfies the bounds

$$\left(\frac{1}{B} - 1\right) \frac{R}{n} \leq MSE_{\alpha, \gamma} \leq \left(\frac{|S|}{B} - 1\right) \frac{R}{n}, \tag{7.19}$$

where  $R = \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^d (X_i(j) - \mu_i)^2 = \frac{1}{n} \sum_{i=1}^n \|X_i - \mu_i 1\|^2$ . Let  $a_{ij} = |X_i(j) - \mu_i|$  and  $W = \sum_{i,j} a_{ij}$ . If, moreover,  $B \leq \sum_{(i,j) \in S} a_{ij} / \max_{(i,j) \in S} a_{ij}$  (which is true, for instance, in the ultra-low communication regime with  $B \leq 1$ ), then

$$MSE_{\alpha, \gamma} = \frac{W^2}{n^2 B} - \frac{R}{n}. \tag{7.20}$$

*Proof.* Setting  $p_{ij} = B/|S|$  for all  $(i, j) \in S$  leads to a feasible solution of (7.17). In view of (7.13), one then has

$$MSE_{\alpha, \gamma} = \frac{1}{n^2} \left(\frac{|S|}{B} - 1\right) \sum_{(i,j) \in S} (X_i(j) - \mu_i)^2 = \left(\frac{|S|}{B} - 1\right) \frac{R}{n},$$

where  $R = \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^d (X_i(j) - \mu_i)^2 = \frac{1}{n} \sum_{i=1}^n \|X_i - \mu_i 1\|^2$ .

If we relax the problem by removing the constraints  $p_{ij} \leq 1$ , the optimal solution satisfies  $a_{ij}/p_{ij} = \theta > 0$  for all  $(i, j) \in S$ . At optimality the bound involving  $B$  must be tight, which leads to  $\sum_{(i,j) \in S} a_{ij}/\theta = B$ , whence  $\theta = \frac{1}{B} \sum_{(i,j) \in S} a_{ij}$ . So,  $p_{ij} = a_{ij}B/\sum_{(i,j) \in S} a_{ij}$ . The

---

<sup>2</sup>We interpret 0/0 as 0 and do not worry about infeasibility. These issues can be properly formalized by allowing  $p_{ij}$  to be zero in the encoding protocol and in (7.17). However, handling this singular situation requires a notational overload which we are not willing to pay.

optimal MSE therefore satisfies the lower bound

$$MSE_{\alpha,\gamma} \geq \frac{1}{n^2} \sum_{(i,j) \in S} \left( \frac{1}{p_{ij}} - 1 \right) (X_i(j) - \mu_i)^2 = \frac{1}{n^2 B} W^2 - \frac{R}{n},$$

where  $W \stackrel{\text{def}}{=} \sum_{(i,j) \in S} a_{ij} \geq \left( \sum_{(i,j) \in S} a_{ij}^2 \right)^{1/2} = (nR)^{1/2}$ . Therefore,  $MSE_{\alpha,\gamma} \geq \left( \frac{1}{B} - 1 \right) \frac{R}{n}$ . If  $B \leq \sum_{(i,j) \in S} a_{ij} / \max_{(i,j) \in S} a_{ij}$ , then  $p_{ij} \leq 1$  for all  $(i, j) \in S$ , and hence we have optimality. (Also note that, by Cauchy-Schwarz inequality,  $W^2 \leq nR|S|$ ).  $\square$

### 7.6.2 Trade-off Curves

To illustrate the trade-offs between communication cost and estimation error (MSE) achievable by the protocols discussed in this section, we present simple numerical examples in Figure 7.1, on three synthetic data sets with  $n = 16$  and  $d = 512$ . We choose an array of values for  $B$ , directly bounding the communication cost via (7.18), and evaluate the *MSE* (7.2) for three encoding protocols (we use the sparse communication protocol and averaging decoder). All these protocols have the same communication cost, and only differ in the selection of the parameters  $p_{ij}$  and  $\mu_i$ . In particular, we consider

- (i) uniform probabilities  $p_{ij} = p > 0$  with average node centers  $\mu_i = \frac{1}{d} \sum_{j=1}^d X_i(j)$  (blue dashed line),
- (ii) optimal probabilities  $p_{ij}$  with average node centers  $\mu_i = \frac{1}{d} \sum_{j=1}^d X_i(j)$  (green dotted line), and
- (iii) optimal probabilities with optimal node centers, obtained via the alternating minimization approach described above (red solid line).

In order to put a scale on the horizontal axis, we assumed that  $r = 16$ . Note that, in practice, one would choose  $r$  to be as small as possible without adversely affecting the application utilizing our distributed mean estimation method. The three plots represent  $X_i$  with entries drawn in an i.i.d. fashion from Gaussian ( $\mathcal{N}(0, 1)$ ), Laplace ( $\mathcal{L}(0, 1)$ ) and chi-squared ( $\chi^2(2)$ ) distributions, respectively. As we can see, in the case of non-symmetric distributions, it is not necessarily optimal to set the node centers to averages.

As expected, for fixed node centers, optimizing over probabilities results in improved performance, across the entire trade-off curve. That is, the curve shifts downwards. In the first two plots based on data from symmetric distributions (Gaussian and Laplace), the average node centers are nearly optimal, which explains why the red solid and green dotted lines coalesce. This can be also established formally. In the third plot, based on the non-symmetric chi-squared data, optimizing over node centers leads to further improvement, which gets more pronounced with increased communication budget. It is possible to generate data where the difference between any pair of the three trade-off curves becomes arbitrarily large.

Finally, the black cross represents performance of the quantization protocol from Example 60. This approach appears as a single point in the trade-off space due to lack of any parameters to be fine-tuned.

## 7.7 Further Considerations

In this section we outline further ideas worth consideration. However, we leave a detailed analysis to future work.

### 7.7.1 Beyond Binary Encoders

We can generalize the binary encoding protocol (7.1) to a  $k$ -ary protocol. To illustrate the concept without unnecessary notation overload, we present only the ternary (i.e.,  $k = 3$ ) case.

Let the collection of parameters  $\{p'_{ij}, p''_{ij}, \bar{X}'_i, \bar{X}''_i\}$  define an encoding protocol  $\alpha$  as follows:

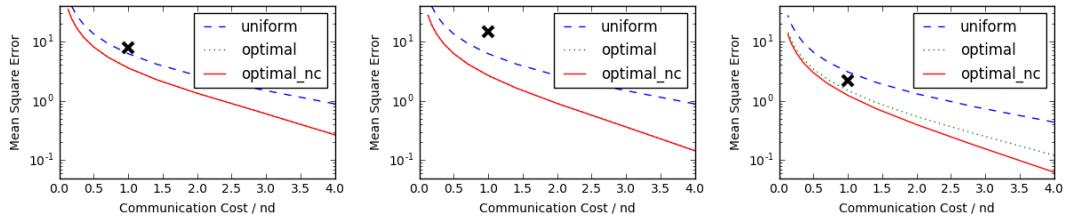


Figure 7.1: *Trade-off curves* between communication cost and estimation error (MSE) for four protocols. The plots correspond to vectors  $X_i$  drawn in an i.i.d. fashion from Gaussian, Laplace and  $\chi^2$  distributions, from left to right. The black cross marks the performance of binary quantization (Example 60).

$$Y_i(j) = \begin{cases} \bar{X}'_i & \text{with probability } p'_{ij}, \\ \bar{X}''_i & \text{with probability } p''_{ij}, \\ \frac{1}{1-p'_{ij}-p''_{ij}} (X_i(j) - p'_{ij}\bar{X}'_i - p''_{ij}\bar{X}''_i) & \text{with probability } 1 - p'_{ij} - p''_{ij}. \end{cases} \quad (7.21)$$

It is straightforward to generalize Lemmas 54 and 55 to this case. We omit the proofs for brevity.

**Lemma 70** (Unbiasedness). *The encoder  $\alpha$  defined in (7.21) is unbiased. That is,  $\mathbb{E}_\alpha [\alpha(X_i)] = X_i$  for all  $i$ . As a result,  $Y$  is an unbiased estimate of the true average:  $\mathbb{E}_\alpha [Y] = X$ .*

**Lemma 71** (Mean Squared Error). *Let  $\alpha = \alpha(p'_{ij}, p''_{ij}, \bar{X}'_i, \bar{X}''_i)$  be the protocol defined in (7.21). Then*

$$MSE_\alpha(X_1, \dots, X_n) = \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^d \left( p'_{ij} (X_i(j) - \bar{X}'_i)^2 + p''_{ij} (X_i(j) - \bar{X}''_i)^2 + (p'_{ij}\bar{X}'_i + p''_{ij}\bar{X}''_i)^2 \right).$$

We expect the  $k$ -ary protocol to lead to better (lower) MSE bounds, but at the expense of an increase in communication cost. Whether or not the trade-off offered by  $k > 2$  is better than that for the  $k = 2$  case investigated in this work is an interesting question to consider.

### 7.7.2 Preprocessing via Random Rotations

Following the idea proposed in [171], one can explore an encoding protocol  $\alpha_Q$  which arises as the composition of a random rotation,  $Q$ , applied to  $X_i$  for all  $i$ , followed by the protocol  $\alpha$  described in Section 7.3. Letting  $Z_i = QX_i$  and  $Z = \frac{1}{n} \sum_i Z_i$ , we thus have

$$Y_i = \alpha(Z_i), \quad i = 1, 2, \dots, n.$$

With this protocol we associate the decoder  $\gamma(Y_1, \dots, Y_n) = \frac{1}{n} \sum_{i=1}^n Q^{-1}Y_i$ .

Note that

$$\begin{aligned} MSE_{\alpha, \gamma} &= \mathbb{E} [\|\gamma(Y_1, \dots, Y_n) - X\|^2] \\ &= \mathbb{E} [\|Q^{-1}\gamma(Y_1, \dots, Y_n) - Q^{-1}Z\|^2] \\ &= \mathbb{E} [\|\gamma(\alpha(Z_1), \dots, \alpha(Z_n)) - Z\|^2] \\ &= \mathbb{E} [\mathbb{E} [\|\gamma(\alpha(Z_1), \dots, \alpha(Z_n)) - Z\|^2 \mid Q]]. \end{aligned}$$

This approach is motivated by the following observation: a random rotation can be identified by a single random seed, which is easy to communicate to the server without the need to communicate all floating point entries defining  $Q$ . So, a random rotation pre-processing step

implies only a minor communication overhead. However, if the preprocessing step helps to dramatically reduce the MSE, we get an improvement. Note that the inner expectation above is the formula for MSE of our basic encoding-decoding protocol, given that the data is  $Z_i = QX_i$  instead of  $\{X_i\}$ . The outer expectation is over  $Q$ . Hence, we would like to find a mapping  $Q$  which tends to transform the data  $\{X_i\}$  into new data  $\{Z_i\}$  with better MSE, in expectation.

From now on, for simplicity assume the node centers are set to the average, i.e.,  $\bar{Z}_i = \frac{1}{d} \sum_{j=1}^d Z_i(j)$ . For any vector  $x \in \mathbb{R}^d$ , define

$$\sigma(x) \stackrel{\text{def}}{=} \sum_{j=1}^d (x(j) - \bar{x})^2 = \|x - \bar{x}\|_F^2,$$

where  $\bar{x} = \frac{1}{d} \sum_j x(j)$  and  $1$  is the vector of all ones. Further, for simplicity assume that  $p_{ij} = p$  for all  $i, j$ . Then using Lemma 55, we get

$$MSE = \frac{1-p}{pn^2} \sum_{i=1}^n \mathbb{E}_Q [\|Z_i - \bar{Z}_i 1\|^2] = \frac{1-p}{pn^2} \sum_{i=1}^n \mathbb{E}_Q [\sigma(QX_i)].$$

It is interesting to investigate whether choosing  $Q$  as a random rotation, rather than identity (which is the implicit choice done in previous sections), leads to improvement in MSE, i.e., whether we can in some well-defined sense obtain an inequality of the type

$$\sum_i \mathbb{E}_Q [\sigma(QX_i)] \ll \sum_i \sigma(X_i).$$

This is the case for the quantization protocol proposed in [171], which arises as a special case of our more general protocol. This is because the quantization protocol is suboptimal within our family of encoders. Indeed, as we have shown, with a different choice of the parameter we can obtain results which improve, in theory, on the rotation + quantization approach. This suggests that perhaps combining an appropriately chosen rotation pre-processing step with our optimal encoder, it may be possible to achieve further improvements in MSE for any fixed communication budget. Finding suitable random rotations  $Q$  requires a careful study which we leave to future research.

## 7.8 Application to Federated Learning

In this Section, we experiment with applying some of these techniques in the context of Federated learning [83, 115, 82]. As discussed in previous chapter, by Federated Optimization or Learning we refer to a setting where we train a shared global model under the coordination of a central server, from a federation of participating devices. The participating devices (clients) are typically large in number and have slow or unstable internet connections. A motivating example for federated optimization arises when the training data is kept locally on users' mobile devices, and the devices are used as nodes performing computation on their local data in order to update a global model. The framework differs from conventional distributed machine learning [147, 104, 167, 199, 42, 33] due to the the large number of clients, highly unbalanced and non-i.i.d. data and unreliable network connections.

Federated learning offers distinct practical advantages compared to performing learning in the data center. The model update is generally less privacy-sensitive than the data itself, and the server never needs to store these updates. Thus, when applicable, federated learning can significantly reduce privacy and security risks by limiting the attack surface to only the device, rather than the device and the cloud. This approach also leverages the data-locality and computational power of the large number of mobile devices.

For simplicity, we consider synchronized algorithms for federated learning [115, 30], where a typical round consists of the following steps:

1. A subset of clients is selected, each of which downloads the current model.
2. Each client in the subset computes an updated model based on their local data.

3. The updated models are sent from the selected clients to the sever.
4. The server aggregates these models (typically by averaging) to construct an improved global model.

A naive implementation of the above framework requires that each client sends a full model (or a full model gradient) back to the server in each round. For large models, this step is likely to be the bottleneck of federated learning due to the asymmetric property of internet connections: the uplink is typically much slower than downlink. The US average broadband speed was 55.0Mbps download vs. 18.9Mbps upload, with some ISPs being significantly more asymmetric, e.g., Xfinity at 125Mbps down vs. 15Mbps up [1]. Cryptographic protocols used to protect individual update [15] further increase the size of the data needed to be communicated back to server. It is therefore important to investigate methods which can reduce the uplink communication cost. In this section, we study two general approaches:

- Structured updates, where we learn an update from a restricted lower-dimensional space.
- Sketched updates, where we learn a full model update, but then compress it before sending to the server.

These approaches can be combined, e.g., first learning a structured update and then sketching it; however, we do not experiment with this combination in the current work.

In the following, we formally describe the problem. The goal of federated learning is to learn a model with parameters embodied in a real matrix  $W \in \mathbb{R}^{d_1 \times d_2}$  from data stored across a large number of clients. We first provide a communication-naive version of the federated learning. In round  $t \geq 0$ , the server distributes the current model  $W_t$  to a subset  $S_t$  of  $n_t$  clients (for example, to a selected subset of clients whose devices are plugged into power, have access to broadband, and are idle). These clients independently update the model based on their local data. Let the updated local models be  $W_t^1, W_t^2, \dots, W_t^{n_t}$ , so the updates can be written as  $H_t^i := W_t^i - W_t$ , for  $i \in S_t$ . Each selected client then sends the update back to the sever, where the global update is computed by aggregating<sup>3</sup> all the client-side updates:

$$W_{t+1} = W_t + \eta_t H_t, \quad H_t := \frac{1}{n_t} \sum_{i \in S_t} H_t^i.$$

The sever chooses the learning rate  $\eta_t$  (for simplicity, we choose  $\eta_t = 1$ ). Recent works show that a careful choice of the server-side learning rate can lead to faster convergence [105, 104, 83].

In this chapter, we describe federated learning for neural networks, where we use a separate 2D matrix  $W$  to represent the parameters of each layer. We suppose that  $W$  gets right-multiplied, i.e.,  $d_1$  and  $d_2$  represent the output and input dimensions respectively. Note that the parameters of a fully connected layer are naturally represented as 2D matrices. However, the kernel of a convolutional layer is a 4D tensor of the shape #input  $\times$  width  $\times$  height  $\times$  #output. In such a case,  $W$  is reshaped from the kernel to the shape (#input  $\times$  width  $\times$  height)  $\times$  #output.

The goal of increasing communication efficiency of federated learning is to reduce the cost of sending  $H_t^i$  to the server. We propose two general strategies of achieving this, discussed next.

### 7.8.1 Structured Update

The first type of communication efficient update restricts the updates  $H_t^i$  to have a pre-specified *structure*. Two types of structures are considered:

**Low rank.** We enforce  $H_t^i \in \mathbb{R}^{d_1 \times d_2}$  to be low-rank matrices of rank at most  $k$ , where  $k$  is a fixed number. We express  $H_t^i$  as the product of two matrices:  $H_t^i = A_t^i B_t^i$ , where  $A_t^i \in \mathbb{R}^{d_1 \times k}$ ,  $B_t^i \in \mathbb{R}^{k \times d_2}$ , and  $A_t^i$  is generated randomly and fixed, and only  $B_t^i$  is optimized. Note that  $A_t^i$  can then be compressed in the form of a random seed and the clients only need to send  $B_t^i$  to the server. We also tried fixing  $B_t^i$  and training  $A_t^i$ , as well as training both  $A_t^i$  and  $B_t^i$ ; neither performed as well. Our approach seems to perform as well as the best techniques considered in [50],

---

<sup>3</sup>A weighted sum might be used to replace the average based on specific implementations.

**Random mask.** We restrict the update  $H_t^i$  to be sparse matrices, following a pre-defined random sparsity pattern (i.e., a random mask). The pattern is generated afresh in each round and for each client. Similar to the low-rank approach, the sparse pattern can be fully specified by a random seed, and therefore it is only required to send the values of the non-zeros entries of  $H_t^i$ . This strategy can be seen as the combination of the master training method and a randomized block coordinate minimization approach [143, 140].

### 7.8.2 Sketched Update

The second type of communication-efficient update, which we call *sketched*, first computes the full unconstrained  $H_t^i$ , and then encodes the update in a (lossy) compressed form before sending to the server. The server decodes the updates before doing the aggregation. Such sketching methods have application in many domains [184]. This technique corresponds to the general focus of previous sections of this chapter.

We propose two ways of performing the sketching:

**Subsampling.** Instead of sending  $H_t^i$ , each client only communicates matrix  $\hat{H}_t^i$  which is formed from a random subset of the (scaled) values of  $H_t^i$ . The server then averages the sampled updates, producing the global update  $\hat{H}_t$ . This can be done so that the average of the sampled updates is an unbiased estimator of the true average:  $\mathbb{E}[\hat{H}_t] = H_t$ . Similar to the random mask structured update, the mask is randomized independently for each client in each round, and the mask itself is stored as a synchronized seed. It was recently shown that, in a certain setting, the *expected iterates* of SGD converge to the optimal point [152]. Perturbing the iterates by a random matrix of zero mean, which is what our subsampling strategy would do, does not affect this type of convergence.

**Probabilistic quantization.** Another way of compressing the updates is by *quantizing* the weights. We first describe the algorithm of quantizing each scalar to one bit. Consider the update  $H_t^i$ , let  $h = (h_1, \dots, h_{d_1 \times d_2}) = \text{vec}(H_t^i)$ , and let  $h_{\max} = \max_j(h_j)$ ,  $h_{\min} = \min_j(h_j)$ . The compressed update of  $h$ , denoted by  $\tilde{h}$ , is generated as follows:

$$\tilde{h}_j = \begin{cases} h_{\max}, & \text{with probability } \frac{h_j - h_{\min}}{h_{\max} - h_{\min}} \\ h_{\min}, & \text{with probability } \frac{h_{\max} - h_j}{h_{\max} - h_{\min}} \end{cases}.$$

It is easy to show that  $\tilde{h}$  is an unbiased estimator of  $h$ . This method provides  $32\times$  of compression compared to a 4 byte float. One can also generalize the above to more than 1 bit for each scalar. For  $b$ -bit quantization, we first equally divide  $[h_{\min}, h_{\max}]$  into  $2^b$  intervals. Suppose  $h_i$  falls in the interval bounded by  $h'$  and  $h''$ . The quantization operates by replacing  $h_{\min}$  and  $h_{\max}$  of the above equation by  $h'$  and  $h''$ , respectively. Incremental, randomized and distributed optimization algorithms can be similarly analyzed in a quantized updates setting [144, 66, 64].

**Improving the quantization by structured random rotations.** The above 1-bit and multi-bit quantization approaches work best when the scales are approximately equal across different dimensions. For example, when  $\max = 100$  and  $\min = -100$  and most of values are 0, the 1-bit quantization will lead to large quantization error. We note that performing a random rotation of  $h$  before the quantization (multiplying  $h$  by an orthogonal matrix) will resolve this issue. In the decoding phase, the server needs to perform the inverse rotation before aggregating all the updates. Note that in practice, the dimension of  $h$  can be as high as  $d = 1e6$ , and it is computationally prohibitive to generate  $(\mathcal{O}(d^3))$  and apply  $(\mathcal{O}(d^2))$  a rotation matrix. In this work, we use a type of structured rotation matrix which is the product of a Walsh-Hadamard matrix and a binary diagonal matrix, motivated by the recent advance in this topic [190]. This reduces the computational complexity of generating and applying the matrix to  $\mathcal{O}(d)$  and  $\mathcal{O}(d \log d)$ .

### 7.8.3 Experiments

We conducted the experiments using federated learning to train deep neural networks for the CIFAR-10 image classification task [90]. There are 50,000 training examples, which we partitioned into 100 clients each containing 500 training examples. The model architecture was

taken from the TensorFlow tutorial [2], which consists of two convolutional layers followed by two fully connected layers and then a linear transformation layer to produce logits, for a total of over 1,000,000 parameters. While this model is not the state-of-the-art, it is sufficient for our needs, as our goal is to evaluate our compression methods, not achieve the best possible accuracy on this task.

We employ the Federated Averaging algorithm [115], which significantly decreases the number of rounds of communication required to train a good model. However, we expect our techniques will show a similar reduction in communication costs when applied to synchronized SGD. For Federated Averaging, on each round we select 10 clients at random, each of which performs 10 epochs of SGD with a learning rate of  $\eta$  on their local dataset using minibatches of 50 images, for a total of 100 local updates. From this updated model we compute the deltas for each layer  $H_t^i$ .

|                       | (Low) Rank       | Sampling Probabilities         | model size | reduction |
|-----------------------|------------------|--------------------------------|------------|-----------|
| Full Model (baseline) | 64, 64, 384, 192 | 1, 1, 1, 1                     | 4.075 MB   | —         |
| Medium subsampling    | 64, 64, 12, 6    | 1, 1, 0.03125, 0.03125         | 0.533 MB   | 7.6×      |
| High subsampling      | 8, 8, 12, 6      | 0.125, 0.125, 0.03125, 0.03125 | 0.175 MB   | 23.3×     |

Table 7.2: Low rank and sampling parameters for the CIFAR experiments. The Sampling Probabilities column gives the fraction of elements uploaded for the two convolutional layers and the two fully-connected layers, respectively; these parameters are used by **StructMask**, **SketchMask**, and **SketchRotMask**. The Low Rank column gives the rank restriction  $k$  for these four layers. The final softmax layer is small, so we do not compress updates to it.

We define medium and high low-rank/sampling parameterizations that result in the same compression rates for both approaches, as given in Table 7.2. The left and center columns of Figure 7.2 present non-quantized results for test-set accuracy, both as a function of the number of rounds of the algorithm, and the total number of megabytes uploaded. For all experiments, learning rates were tuned using a multiplicative grid of resolution  $\sqrt{2}$  centered at 0.15; we plot results for the learning rate with the best median accuracy over rounds 400 – 800. We used a multiplicative learning-rate decay of 0.988, which we selected by tuning only for the baseline algorithm.

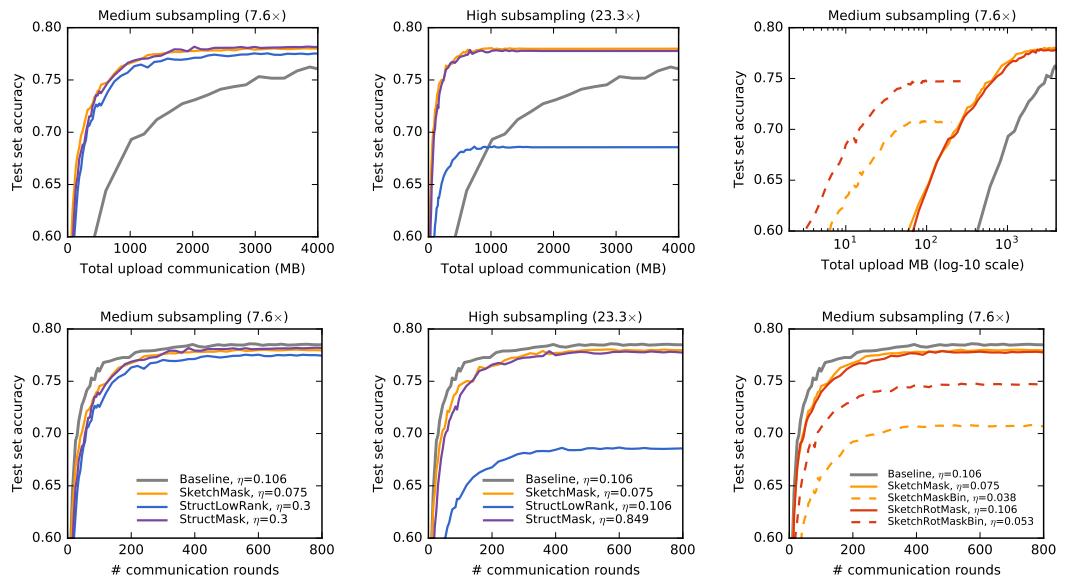


Figure 7.2: Non-quantized results (left and middle columns), and results including binary quantization (dashed lines **SketchRotMaskBin** and **SketchMaskBin**, right column). Note the  $x$ -axis is on a log scale for top-right plot. We achieve over 70% accuracy with fewer than 100MB of communication.

For medium subsampling, all three approaches provide a dramatic improvement in test

set accuracy after a fixed amount of bandwidth usage; the lower row of plots shows little loss in accuracy as a function of the number of update rounds. The exception is that the **StructLowRank** approach performs poorly for the high subsampling parameters. This may suggest that requiring a low-rank update structure for the convolution layers works poorly. Also, perhaps surprisingly, we see no advantage for **StructMask**, which optimizes for a random sparse set of coefficients, as compared to **SketchMask**, which chooses a sparse set of parameters to update *after* a full update is learned.

The right two plots in Figure 7.2 give results for **SketchMask** and **SketchRotMask**, with and without binary quantization; we consider only the medium subsampling regime which is representative. We observe that (as expected) introducing the random rotation without quantization has essentially no effect (solid red and orange lines). However, binary quantization dramatically decreases the total communication cost, and further introducing the random rotation significantly speeds convergence, and also allows us to converge to a higher level of accuracy. We are able to learn a reasonable model (70% accuracy) in only  $\sim 100\text{MB}$  of communication, two orders of magnitude less than the baseline.

## 7.9 Additional Proofs

In this section we provide proofs of Lemmas 56 and 57, describing properties of the encoding protocol  $\alpha$  defined in (7.4). For completeness, we also repeat the statements.

**Lemma 72** (Unbiasedness). *The encoder  $\alpha$  defined in (7.1) is unbiased. That is,  $\mathbb{E}_\alpha[\alpha(X_i)] = X_i$  for all  $i$ . As a result,  $Y$  is an unbiased estimate of the true average:  $\mathbb{E}_\alpha[Y] = X$ .*

*Proof.* Since  $Y(j) = \frac{1}{n} \sum_{i=1}^n Y_i(j)$  and  $X(j) = \frac{1}{n} \sum_{i=1}^n X_i(j)$ , it suffices to show that  $\mathbb{E}_\alpha[Y_i(j)] = X_i(j)$ :

$$\begin{aligned}\mathbb{E}_\alpha[Y_i(j)] &= \frac{1}{|\sigma_k(d)|} \sum_{\sigma \in \sigma_k(d)} \left[ 1_{(j \in \sigma)} \left( \frac{dX_i(j)}{k} - \frac{d-k}{k} \mu_i \right) + 1_{(j \notin \sigma)} \mu_i \right] \\ &= \binom{d}{k}^{-1} \left[ \binom{d-1}{k-1} \left( \frac{dX_i(j)}{k} - \frac{d-k}{k} \mu_i \right) + \binom{d-1}{k} \mu_i \right] \\ &= \binom{d}{k}^{-1} \left[ \binom{d-1}{k-1} \frac{d}{k} X_i(j) + \left( \binom{d-1}{k} - \binom{d-1}{k-1} \frac{d-k}{k} \right) \mu_i \right] \\ &= X_i(j)\end{aligned}$$

and the claim is proved.  $\square$

**Lemma 73** (Mean Squared Error). *Let  $\alpha = \alpha(k)$  be encoder defined as in (7.4). Then*

$$MSE_\alpha(X_1, \dots, X_n) = \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^d \frac{d-k}{k} (X_i(j) - \mu_i)^2.$$

*Proof.* Using Lemma 52, we have

$$\begin{aligned}MSE_\alpha(X_1, \dots, X_n) &= \frac{1}{n^2} \sum_{i=1}^n \mathbb{E}_\alpha [\|Y_i - X_i\|^2] \\ &= \frac{1}{n^2} \sum_{i=1}^n \mathbb{E}_\alpha \left[ \sum_{j=1}^d (Y_i(j) - X_i(j))^2 \right] \\ &= \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^d \mathbb{E}_\alpha [(Y_i(j) - X_i(j))^2].\end{aligned}\tag{7.22}$$

Further,

$$\begin{aligned}
\mathbb{E}_\alpha [(Y_i(j) - X_i(j))^2] &= \binom{d}{k}^{-1} \sum_{\sigma \in \sigma_k(d)} \left[ 1_{(j \in \sigma)} \left( \frac{dX_i(j)}{k} - \frac{d-k}{k} \mu_i - X_i(j) \right)^2 + 1_{(j \notin \sigma)} (\mu_i - X_i(j))^2 \right] \\
&= \binom{d}{k}^{-1} \left[ \binom{d-1}{k-1} \frac{(d-k)^2}{k^2} (X_i(j) - \mu_i)^2 + \binom{d-1}{k} (\mu_i - X_i(j))^2 \right] \\
&= \frac{d-k}{k} (X_i(j) - \mu_i)^2.
\end{aligned}$$

It suffices to substitute the above into (7.22). □



## **Part IV**

# **Conclusion**



## Chapter 8

# Conclusion and Future Challenges

In this thesis, we addressed the problem of minimizing a finite average of functions:

$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n f_i(w).$$

In Chapters 2 and 3, we formulated two stochastic algorithms with a variance reduction property for solving the above optimization problem. By using stochastic gradients in a particular way, relying on computation of the entire gradient occasionally, we are able to achieve progressive reduction of variance of the stochastic estimates of the gradients. We have demonstrated that these approaches vastly outperform traditional methods at the optimization task.

In Chapters 4 and 5, we described methods that can utilize parallel and distributed computing architectures. The CoCoA framework for distributed optimization addresses primarily communication efficiency — a common problem of many distributed optimization algorithms. In particular, Section 5.2 introduces a conceptual framework in which one can think of the overall efficiency of distributed optimization algorithms, and explains the usefulness of decomposing the choice of local optimization procedure from design of general local subproblems closely related to the overall optimization objective. The CoCoA framework is the first systematic realization of this approach.

In Chapters 6 and 7, we introduced the concept of Federated Optimization, which goes beyond what is usually addressed in the area of distributed optimization, by changing the often implicit assumptions on distribution of data among different computing nodes. A particular motivating example arises in the case of user-generated data. Instead of collecting the data and storing them in datacenters, one could keep the data in users' possession, and run optimization algorithms on this massively distributed collection of user devices. We have shown that this idea is conceptually feasible, and has since been experimentally deployed by Google in Android Keyboard [117].

In the following, we try to highlight some of the important challenges that are remaining open in the field.

The area of single-machine optimization algorithms for finite average of functions recently saw numerous contributions, such as [163, 158, 80, 89, 45, 5, 131]. Two major questions spanning all of these works concern the adaptability to strong convexity which is not explicitly known, and the use of stochastic higher-order information.

In general, methods relying on duality, such as SDCA [163], are easier to use in practice, compared to their primal-only alternatives. This is because for a number of problems structures, SDCA comes without the need to tune any hyperparameters. However, the construction of the dual relies on explicit knowledge of the strong convexity parameter of the objective, or a lower bound of it, which then drives the convergence speed of the algorithm, both in theory and in practice. The explicit knowledge usually comes from the use of regularizer. Nevertheless, the true strong convexity parameter can be larger, caused by for instance structure in the data or

can be larger locally near optimum. In contrast, the primal-only algorithms in general adapt to whatever is the true strong convexity, without the having to explicitly know what it is.

These benefits cause disagreements in the community over which algorithms are more useful in practice. It is likely that primal methods can be enhanced with adaptive techniques, removing the need of tuning hyperparameters, and the dual methods can be made adaptive to strong convexity without its explicit knowledge. Both of these ideas were to some extent addressed, but none of them provide a satisfactory answer. In particular, the authors of [158] provide a heuristic stepsize which ‘usually works’ for their method, but to the best of our knowledge, there is no such method supported by theory. Recent contribution in [181] shows that an existing primal-dual method can be made adaptive to the true strong convexity from data, if explicitly known, or can be estimated during the run of the algorithm. While this presents a step forward, it falls short of the simplicity of primal methods in handling this issue.

On another front, it is expected that the use of higher-order information will improve the performance of existing methods on this task. The challenge is to do so utilizing stochastic information arising from the structure of the problem, while maintaining computational stability and efficiency. There have been several attempts to do this recently, see [21, Section 3.4] for a detailed overview. However, all of them either fail to be computationally efficient and thus inferior to the existing methods, or work well only under restrictive assumptions. It is widely expected that progress in this regard will have significant influence.

An expected theoretical advance in communication-efficient distributed algorithms is the acceleration of the CoCoA<sup>+</sup> framework in the sense of [126]. It is commonly agreed that this is possible, but it has not been successfully done yet. In terms of practice, scalability and robustness are the major issues going forward. All of the existing methods to some extent either don’t scale to large number of nodes, or require assumptions about the way data is distributed across individual nodes. In particular, many method assume that each node has access to data drawn iid from the same distribution.

Assumptions on the distribution of data are problematic even in the datacenter setting, where the data are commonly partitioned ‘as is’, and reshuffling data to match the assumptions is either infeasible or very impractical. The data can be naturally clustered based on its geographical origin and/or time at which it was collected. Methods that can converge for any distribution of the data, such as CoCoA<sup>+</sup>, are naturally very useful in many practical applications. Nevertheless, the performance of CoCoA<sup>+</sup> was observed to degrade when scaled to large number of nodes such as in the context of Federated Optimization, see for instance Figure 6.2.

An interesting question bridging traditional distributed optimization with Federated Optimization, is whether the latter could serve as useful computational model for the former. Scaling distributed training of deep neural networks has been particularly problematic, generating large number of works in recent years. A recent work [31] revisits the conventional belief that synchronous methods are inferior to their asynchronous variants, showing that a synchronous alternative with backups can be superior to both. In a sense this can be thought of as a negative result for the optimization and systems community, showing the lack of sufficient robustness of methods used in practice. Nevertheless, the task of training large models very fast remain of significant interest, see for instance [70], in which the authors use up to 256 GPUs. This is considered to be a huge number, yet nowhere near the scale of convex problems in click-through rate prediction systems [116].

The concept of Federated Optimization offers a systematic way to decompose the necessary computation into more independent blocks, alleviating most of the issues related to communication necessary between individual nodes. As the concept seems stable enough to scale to support training of recurrent neural networks on phones [117], it could support scaling parallelism of training deep neural networks to many more computing nodes in a stable way. To the best of our knowledge, none experiments in this direction has been done yet.

Deployment of Federated Optimization in general opens up many new research questions. For instance, if the goal of the overall system is to make sure servers do not get any data about individual users, a natural question to ask is whether server can reconstruct what data could have caused the observed communication patterns. In this regard, recent work proposed a cryptographic protocol for secure aggregation, ensuring that the server can only see an aggregate function of individual updates, such as an average, across many participating users [16].

While the secure aggregation protocol provides an intuitive and practical obstacle for recovering the data of a participating user, it does not provide any formal guarantees. Another option is to provide quantitative guarantees for differential privacy [56]. Adding a carefully designed noise to each user’s update would likely ensure the final trained model is differentially private. While this have not been done, it is an important problem to address.

As we remarked in Section 7.8, communication is a likely candidate to be a bottleneck in practice. We proposed few techniques to reduce the amount of bits each user has to upload. However, it is not as straightforward to make all of the ideas compatible with the above secure aggregation protocol, or noise insertion commonly used to obtain differential privacy. As an example, the secure aggregation protocol expects a vector of a particular structure where on which addition can be performed directly. This structure can be obtained by naive update compression mechanisms, but not using data adaptive ones, which naturally give better performance.

Finally, if and when Federated Optimization becomes commonly used tool in machine learning practice, it will require innovation in tools for machine learning engineers. Many of the common blocks of machine learning workflow will become unavailable. For instance, it will be harder to rapidly experiment with novel network architectures if they are not trained only in a datacenter, increasing the need to get more theoretical understanding into what makes training of neural networks hard or easy, and what makes them generalize well. Further, any concerns about system stability are mostly irrelevant for machine learning engineers, as it is not an issue if a system does not train or crashes in datacenter. However, crashing every phone an experiment runs on would be a major issue. These examples and many more will drive thorough rethinking of how machine learning tools are used to ultimately design products.



# Bibliography

- [1] Speedtest market report. <http://www.speedtest.net/reports/united-states/>, August 2016.
- [2] Tensorflow convolutional neural networks tutorial. [http://www.tensorflow.org/tutorials/deep\\_cnn](http://www.tensorflow.org/tutorials/deep_cnn), 2016.
- [3] Martín Abadi, Andy Chu, Ian Goodfellow, Brendan H McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. *arXiv:1607.00133*, 2016.
- [4] Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. In *Advances in Neural Information Processing Systems 24*, pages 873–881, 2011.
- [5] Zeyuan Allen-Zhu. Katyusha: The first direct acceleration of stochastic gradient methods. *arXiv:1603.05953*, 2016.
- [6] Zeyuan Allen-Zhu, Yang Yuan, and Karthik Sridharan. Exploiting the structure: Stochastic gradient methods using raw clusters. *arXiv:1602.02151*, 2016.
- [7] Yossi Arjevani and Ohad Shamir. Communication complexity of distributed convex learning and optimization. In *Advances in Neural Information Processing Systems*, pages 1756–1764, 2015.
- [8] Maria-Florina Balcan, Avrim Blum, Shai Fine, and Yishay Mansour. Distributed learning, communication complexity and privacy. In *25th Annual Conference on Learning Theory*, pages 26.1–26.22, 2012.
- [9] Jonathan Barzilai and Jonathan M Borwein. Two-point step size gradient methods. *IMA Journal of Numerical Analysis*, 8(1):141–148, 1988.
- [10] Fabian Bastin, Cinzia Cirillo, and Philippe L Toint. Convergence theory for nonconvex stochastic programming with an application to mixed logit. *Mathematical Programming*, 108(2):207–234, 2006.
- [11] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.
- [12] Ron Bekkerman, Mikhail Bilenko, and John Langford. *Scaling up machine learning: Parallel and distributed approaches*. Cambridge University Press, 2011.
- [13] Dimitri P Bertsekas. Distributed asynchronous computation of fixed points. *Mathematical Programming*, 27(1):107–120, 1983.
- [14] Dimitri P Bertsekas and John N Tsitsiklis. *Parallel and distributed computation: numerical methods*. Prentice-Hall, Inc., 1989.
- [15] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, Brendan H McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data. *arXiv:1611.04482*, 2016.

- [16] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy preserving machine learning. *IACR Cryptology ePrint Archive*, 2017:281, 2017.
- [17] Antoine Bordes, Léon Bottou, and Patrick Gallinari. SGD-QN: Careful quasi-Newton stochastic gradient descent. *The Journal of Machine Learning Research*, 10:1737–1754, 2009.
- [18] Léon Bottou. Curiously fast convergence of some stochastic gradient descent algorithms. In *Proceedings of the Symposium on Learning and Data Science*, 2009.
- [19] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of 19th International Conference on Computational Statistics*, pages 177–186. Springer, 2010.
- [20] Léon Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, pages 421–436. Springer, 2012.
- [21] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *arXiv:1606.04838*, 2016.
- [22] Olivier Bousquet and Léon Bottou. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems 21*, pages 161–168, 2008.
- [23] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2010.
- [24] Joseph Bradley, Aapo Kyrola, Daniel Bickson, and Carlos Guestrin. Parallel coordinate descent for L1-regularized loss minimization. In *Proceedings of the 28th International Conference on Machine Learning*, pages 321–328, 2011.
- [25] Mark Braverman, Ankit Garg, Tengyu Ma, Huy L Nguyen, and David P Woodruff. Communication lower bounds for statistical estimation problems via a distributed data processing inequality. *arXiv:1506.07216*, 2015.
- [26] Richard H Byrd, Samantha L Hansen, Jorge Nocedal, and Yoram Singer. A stochastic quasi-Newton method for large-scale optimization. *SIAM Journal on Optimization*, 26(2):1008–1031, 2016.
- [27] Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.
- [28] Bob Carpenter. Lazy sparse stochastic gradient descent for regularized multinomial logistic regression. *Technical Report*, 2008.
- [29] Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12:1069–1109, 2011.
- [30] Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous SGD. In *4th International Conference on Learning Representations Workshop Track*, 2016.
- [31] Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous sgd. *arXiv:1604.00981*, 2016.
- [32] Weizhu Chen, Zhenghao Wang, and Jingren Zhou. Large-scale L-BFGS using MapReduce. In *Advances in Neural Information Processing Systems 27*, pages 1332–1340, 2014.

- [33] Trishul Chilimbi, Yutaka Suzue, Johnson Apacible, and Karthik Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 571–582, 2014.
- [34] CNN. Where (and when) do you use your smartphone: Bedroom? Church? News article <http://edition.cnn.com/2013/07/13/tech/smartphone-use-survey/>, 2013.
- [35] Patrick Louis Combettes and Jean-Christophe Pesquet. Proximal splitting methods in signal processing. In *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, volume 49, pages 185–212. Springer, 2011.
- [36] Patrick Louis Combettes and Jean-Christophe Pesquet. Stochastic quasi-fejér block-coordinate fixed point iterations with random sweeping. *SIAM Journal on Optimization*, 25(2):1221–1248, 2015.
- [37] Dominik Csiba, Zheng Qu, and Peter Richtárik. Stochastic dual coordinate ascent with adaptive probabilities. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 674–683, 2015.
- [38] Dominik Csiba and Peter Richtárik. Primal method for ERM with flexible mini-batching schemes and non-convex losses. *arXiv:1506.02227*, 2015.
- [39] Dominik Csiba and Peter Richtárik. Coordinate descent face-off: primal or dual? *arXiv:1605.08982*, 2016.
- [40] Dominik Csiba and Peter Richtárik. Importance sampling for minibatches. *arXiv:1602.02283*, 2016.
- [41] Christopher M De Sa, Ce Zhang, Kunle Olukotun, and Christopher Ré. Taming the wild: A unified analysis of hogwild-style algorithms. In *Advances in Neural Information Processing Systems 28*, pages 2656–2664, 2015.
- [42] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems 25*, pages 1223–1231, 2012.
- [43] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the Association for Computing Machinery*, 51(1):107–113, 2008.
- [44] Aaron Defazio. A simple practical accelerated method for finite sums. *arXiv:1602.02442*, 2016.
- [45] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems 27*, pages 1646–1654, 2014.
- [46] Aaron Defazio, Justin Domke, and Tiberio Caetano. Finito: A faster, permutable incremental gradient method for big data problems. In *Proceedings of the 31st International Conference on Machine Learning*, pages 1125–1133, 2014.
- [47] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal distributed online prediction using mini-batches. *The Journal of Machine Learning Research*, 13(1):165–202, 2012.
- [48] Geng Deng and Michael C Ferris. Variable-number sample-path optimization. *Mathematical Programming*, 117(1):81–109, 2009.
- [49] Wei Deng and Wotao Yin. On the global and linear convergence of the generalized alternating direction method of multipliers. *Journal of Scientific Computing*, 66(3):889–916, 2016.

- [50] Misha Denil, Babak Shakibi, Laurent Dinh, Marc’Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems 26*, pages 2148–2156, 2013.
- [51] John C Duchi, Alekh Agarwal, and Martin J Wainwright. Dual averaging for distributed optimization: convergence analysis and network scaling. *IEEE Transactions on Automatic Control*, 57(3):592–606, 2012.
- [52] John C Duchi, Sorathan Chaturapruek, and Christopher Ré. Asynchronous stochastic convex optimization. *arXiv:1508.00882*, 2015.
- [53] John C Duchi, Michael I Jordan, and Brendan H McMahan. Estimation, optimization, and parallelism when data is sparse. In *Advances in Neural Information Processing Systems 26*, pages 2832–2840, 2013.
- [54] John C Duchi, Michael I Jordan, and Martin J Wainwright. Privacy aware learning. *Journal of the Association for Computing Machinery*, 2014.
- [55] Celestine Dünner, Simone Forte, Martin Takáč, and Martin Jaggi. Primal-Dual Rates and Certificates. In *Proceedings of the 33th International Conference on Machine Learning*, 2016.
- [56] Cynthia Dwork and Aaron Roth. *The Algorithmic Foundations of Differential Privacy*. Foundations and Trends in Theoretical Computer Science. Now Publishers, 2014.
- [57] Olivier Fercoq, Zheng Qu, Peter Richtárik, and Martin Takáč. Fast distributed coordinate descent for non-strongly convex losses. In *IEEE International Workshop on Machine Learning for Signal Processing*, pages 1–6, 2014.
- [58] Olivier Fercoq and Peter Richtárik. Smooth minimization of nonsmooth functions with parallel coordinate descent methods. *arXiv:1309.5885*, 2013.
- [59] Olivier Fercoq and Peter Richtárik. Optimization in high dimensions via accelerated, parallel, and proximal coordinate descent. *SIAM Review*, 58(4):739–771, 2016.
- [60] Pedro A Forero, Alfonso Cano, and Georgios B Giannakis. Consensus-based distributed support vector machines. *Journal of Machine Learning Research*, 11:1663–1707, 2010.
- [61] The MPI Forum. MPI: A message passing interface standard, Version 3.1. Document available at <http://www.mpi-forum.org/>, 2015.
- [62] Michael P Friedlander and Mark Schmidt. Hybrid deterministic-stochastic methods for data fitting. *SIAM Journal on Scientific Computing*, 34(3):A1380–A1405, 2012.
- [63] Roy Frostig, Rong Ge, Sham M Kakade, and Aaron Sidford. Un-regularizing: approximate proximal point and faster stochastic algorithms for empirical risk minimization. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.
- [64] Mostafa El Gamal and Lifeng Lai. On randomized distributed coordinate descent with quantized updates. *arXiv:1609.05539*, 2016.
- [65] Ankit Garg, Tengyu Ma, and Huy L Nguyen. On communication cost of distributed statistical estimation and dimensionality. In *Advances in Neural Information Processing Systems 27*, pages 2726–2734, 2014.
- [66] Daniel Golovin, D Sculley, Brendan H McMahan, and Michael Young. Large-scale learning with less ram via randomization. In *Proceedings of the 30th International Conference on Machine Learning*, pages 325–333, 2013.
- [67] Robert M Gower, Donald Goldfarb, and Peter Richtárik. Stochastic block bfgs: squeezing more curvature out of data. In *Proceedings of the 33rd International Conference on Machine Learning*, pages 1869–1878, 2016.

- [68] Robert M Gower and Peter Richtárik. Randomized Iterative Methods for Linear Systems. *arXiv:1506.03296*, 2015.
- [69] Robert M Gower and Peter Richtárik. Randomized quasi-Newton updates are linearly convergent matrix inversion algorithms. *arXiv:1602.01768*, 2016.
- [70] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv:1706.02677*, 2017.
- [71] Mert Gürbüzbalaian, Asu Ozdaglar, and Pablo Parrilo. Why random reshuffling beats stochastic gradient descent. *arXiv:1510.08560*, 2015.
- [72] Per Christian Hansen. Regularization tools version 4.0 for matlab 7.3. *Numerical algorithms*, 46(2):189–194, 2007.
- [73] Filip Hanzely, Jakub Konečný, Nicolas Loizou, Peter Richtárik, and Dmitry Grishchenko. Privacy preserving randomized gossip algorithms. *arXiv:1706.07636*, 2017.
- [74] Reza Harikandeh, Mohamed Osama Ahmed, Alim Virani, Mark Schmidt, Jakub Konečný, and Scott Sallinen. Stop wasting my gradients: Practical SVRG. In *Advances in Neural Information Processing Systems 28*, pages 2251–2259, 2015.
- [75] Christina Heinze, Brian McWilliams, and Nicolai Meinshausen. DUAL-LOCO: Distributing Statistical Estimation Using Random Projections. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, pages 875–883, 2016.
- [76] Christina Heinze, Brian McWilliams, Nicolai Meinshausen, and Gabriel Krummenacher. Loco: Distributing ridge regression with random projections. *arXiv:1406.3469*, 2014.
- [77] Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 1952.
- [78] Cho-Jui Hsieh, Kai-Wei Chang, Chih-Jen Lin, S Sathiya Keerthi, and Sellamanickam Sundararajan. A dual coordinate descent method for large-scale linear svm. In *Proceedings of the 25th International Conference on Machine Learning*, pages 408–415, 2008.
- [79] Martin Jaggi, Virginia Smith, Martin Takáč, Jonathan Terhorst, Sanjay Krishnan, Thomas Hofmann, and Michael I Jordan. Communication-efficient distributed dual coordinate ascent. In *Advances in Neural Information Processing Systems 27*, pages 3068–3076, 2014.
- [80] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems 26*, pages 315–323, 2013.
- [81] Jakub Konečný, Jie Liu, Peter Richtárik, and Martin Takáč. Mini-batch semi-stochastic gradient descent in the proximal setting. *IEEE Journal of Selected Topics in Signal Processing*, 10(2):242–255, 2016.
- [82] Jakub Konečný, Brendan H McMahan, and Daniel Ramage. Federated optimization: Distributed optimization beyond the datacenter. *arXiv:1511.03575*, 2015.
- [83] Jakub Konečný, Brendan H McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv:1610.02527*, 2016.
- [84] Jakub Konečný, Zheng Qu, and Peter Richtárik. Semi-stochastic coordinate descent. *arXiv:1412.6293*, 2014.
- [85] Jakub Konečný and Peter Richtárik. Simple complexity analysis of simplified direct search. *arXiv:1410.0390*, 2014.

- [86] Jakub Konečný and Peter Richtárik. Randomized distributed mean estimation: Accuracy vs communication. *arXiv:1611.07555*, 2016.
- [87] Jakub Konečný, Jie Liu, Peter Richtárik, and Martin Takáč. mS2GD: Mini-batch semi-stochastic gradient descent in the proximal setting. *Optimization for Machine Learning workshop*, 2014.
- [88] Jakub Konečný, Brendan H McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv:1610.05492*, 2016.
- [89] Jakub Konečný and Peter Richtárik. Semi-stochastic gradient descent methods. *arXiv:1312.1666*, 2013.
- [90] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.
- [91] John Langford, Lihong Li, and Tong Zhang. Sparse online learning via truncated gradient. *Journal of Machine Learning Research*, 10:777–801, 2009.
- [92] Rémi Leblond, Fabian Pedregosa, and Simon Lacoste-Julien. ASAGA: Asynchronous parallel saga. *arXiv:1606.04809*, 2016.
- [93] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.
- [94] Ching-Pei Lee and Dan Roth. Distributed box-constrained quadratic optimization for dual linear SVM. In *Proceedings of the 32th International Conference on Machine Learning*, pages 987–996, 2015.
- [95] Jason Lee, Tengyu Ma, and Qihang Lin. Distributed stochastic variance reduced gradient methods. *arXiv:1507.07595*, 2015.
- [96] Yin Tat Lee and Aaron Sidford. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. In *IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 147–156, 2013.
- [97] Dennis Leventhal and Adrian S Lewis. Randomized methods for linear constraints: convergence rates and conditioning. *Mathematics of Operations Research*, 35(3):641–654, 2010.
- [98] Chieh-Yen Lin, Cheng-Hao Tsai, Ching-Pei Lee, and Chih-Jen Lin. Large-scale logistic regression and linear support vector machines using spark. In *IEEE International Conference on Big Data*, pages 519–528, 2014.
- [99] Hongzhou Lin, Julien Mairal, and Zaid Harchaoui. A universal catalyst for first-order optimization. In *Advances in Neural Information Processing Systems 28*, pages 3366–3374, 2015.
- [100] Dong C Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.
- [101] Ji Liu and Stephen J Wright. Asynchronous stochastic coordinate descent: Parallelism and convergence properties. *SIAM Journal on Optimization*, 25(1):351–376, 2015.
- [102] Ji Liu, Stephen J Wright, Christopher Ré, Victor Bittorf, and Srikrishna Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. *Journal of Machine Learning Research*, 16:285–322, 2015.
- [103] Anna Ma, Deanna Needell, and Aaditya Ramdas. Convergence properties of the randomized extended gauss-seidel and kaczmarz methods. *SIAM Journal on Matrix Analysis and Applications*, 36(4):1590–1604, 2015.

- [104] Chenxin Ma, Jakub Konečný, Martin Jaggi, Virginia Smith, Michael I Jordan, Peter Richtárik, and Martin Takáč. Distributed optimization with arbitrary local solvers. *arXiv:1512.04039*, 2015.
- [105] Chenxin Ma, Virginia Smith, Martin Jaggi, Michael I Jordan, Peter Richtárik, and Martin Takáč. Adding vs. averaging in distributed primal-dual optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1973–1982, 2015.
- [106] Chenxin Ma and Martin Takáč. Partitioning data on features or samples in communication-efficient distributed optimization? *arXiv:1510.06688*, 2015.
- [107] Chenxin Ma and Martin Takáč. Distributed inexact damped newton method: Data partitioning and load-balancing. *arXiv:1603.05191*, 2016.
- [108] Chenxin Ma, Rachael Tappenden, and Martin Takáč. Linear convergence of the randomized feasible descent method under the weak strong convexity assumption. *arXiv:1506.02530*, 2015.
- [109] Dhruv Mahajan, Nikunj Agrawal, S Sathiya Keerthi, Sundararajan Sellamanickam, and Léon Bottou. An efficient distributed learning algorithm based on effective local functional approximations. *arXiv:1310.8418*, 2013.
- [110] Horia Mania, Xinghao Pan, Dimitris Papailiopoulos, Benjamin Recht, Kannan Ramchandran, and Michael I Jordan. Perturbed iterate analysis for asynchronous stochastic optimization. *arXiv:1507.06970*, 2015.
- [111] Jakub Mareček, Peter Richtárik, and Martin Takáč. Distributed block coordinate descent for minimizing partially separable functions. *Numerical Analysis and Optimization 2014, Springer Proceedings in Mathematics and Statistics*, 134:261–288, 2015.
- [112] Kurt Marti and Erich Fuchs. On solutions of stochastic programming problems by descent procedures with stochastic and deterministic directions. *Methods of Operations Research*, 33:281–293, 1979.
- [113] Kurt Marti and Erich Fuchs. Rates of convergence of semi-stochastic approximation procedures for solving stochastic optimization problems. *Optimization*, 17(2):243–265, 1986.
- [114] Ryan McDonald, Mehryar Mohri, Nathan Silberman, Dan Walker, and Gideon S Mann. Efficient large-scale distributed training of conditional maximum entropy models. In *Advances in Neural Information Processing Systems 22*, pages 1231–1239, 2009.
- [115] Brendan H McMahan, Eider Moore, Daniel Ramage, and Blaise Aguera y Arcas. Federated learning of deep networks using model averaging. *arXiv:1602.05629*, 2016.
- [116] H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, et al. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1222–1230. ACM, 2013.
- [117] H. Brendan McMahan and Daniel Ramage. Federated learning: Collaborative machine learning without centralized training data. <https://research.googleblog.com/2017/04/federated-learning-collaborative.html>, 2017.
- [118] Jean-Jacques Moreau. Fonctions convexes duales et points proximaux dans un espace hilbertien. In *Reports of the Paris Academy of Sciences*, volume 255 of *A*, pages 2897–2899, 1962.
- [119] Philipp Moritz, Robert Nishihara, and Michael Jordan. A linearly-convergent stochastic L-BFGS algorithm. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 249–258, 2016.

- [120] Eric Moulines and Francis R Bach. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *Advances in Neural Information Processing Systems 24*, pages 451–459, 2011.
- [121] Ion Necoara and Dragos Clipici. Distributed coordinate descent methods for composite minimization. *arXiv:1312.5302*, 2013.
- [122] Ion Necoara and Andrei Patrascu. A random coordinate descent algorithm for optimization problems with composite objective function and linear coupled constraints. *Computational Optimization and Applications*, 57(2):307–337, 2014.
- [123] Deanna Needell. Randomized kaczmarz solver for noisy linear systems. *BIT Numerical Mathematics*, 50(2):395–403, 2010.
- [124] Deanna Needell, Nathan Srebro, and Rachel Ward. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. *Mathematical Programming*, 155(1–2):549–573, 2016.
- [125] Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19(4):1574–1609, 2009.
- [126] Yurii Nesterov. A method of solving a convex programming problem with convergence rate  $O(1/k^2)$ . *Soviet Mathematics Doklady*, 27(2):372–376, 1983.
- [127] Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Kluwer, Boston, 2004.
- [128] Yurii Nesterov. Gradient methods for minimizing composite objective function. *CORE Discussion Papers*, 2007/76.
- [129] Yurii Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22:341–362, 2012.
- [130] Jiquan Ngiam, Adam Coates, Ahbk Lahiri, Bobby Prochnow, Quoc V Le, and Andrew Y Ng. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on Machine Learning*, pages 265–272, 2011.
- [131] Lam Nguyen, Jie Liu, Katya Scheinberg, and Martin Takáč. Sarah: A novel method for machine learning problems using stochastic recursive gradient. *arXiv:1703.00102*, 2017.
- [132] Atsushi Nitanda. Stochastic proximal gradient descent with acceleration techniques. In *Advances in Neural Information Processing Systems 27*, pages 1574–1582, 2014.
- [133] Feng Niu, Benjamin Recht, Christopher Ré, and Stephen J Wright. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems 24*, pages 693–701, 2011.
- [134] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer series in Operations Research and Financial Engineering. Springer, 2. ed. edition, 2006.
- [135] Peter Oswald and Weiqi Zhou. Convergence analysis for Kaczmarz-type methods in a Hilbert space framework. *Linear Algebra and its Applications*, 478:131–161, 2015.
- [136] Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):127–239, 2014.
- [137] Dmitry Pechyony, Libin Shen, and Rosie Jones. Solving large scale linear svm with distributed block minimization. In *ACM International Conference on Information and Knowledge Management*, 2011.
- [138] Zhimin Peng, Yangyang Xu, Ming Yan, and Wotao Yin. ARock: an algorithmic framework for asynchronous parallel coordinate updates. *SIAM Journal on Scientific Computing*, 38(5):2851–2879, 2016.

- [139] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.
- [140] Zheng Qu and Peter Richtárik. Coordinate descent with arbitrary sampling I: Algorithms and complexity. *Optimization Methods and Software*, 31(5):829–857, 2016.
- [141] Zheng Qu and Peter Richtárik. Coordinate descent with arbitrary sampling II: Expected separable overapproximation. *Optimization Methods and Software*, 31(5):858–884, 2016.
- [142] Zheng Qu, Peter Richtárik, Martin Takáč, and Olivier Fercoq. SDNA: Stochastic dual newton ascent for empirical risk minimization. In *Proceedings of the 33rd International Conference on Machine Learning*, pages 1823–1832, 2016.
- [143] Zheng Qu, Peter Richtárik, and Tong Zhang. Quartz: Randomized dual coordinate ascent with arbitrary sampling. In *Advances in Neural Information Processing Systems 28*, pages 865–873, 2015.
- [144] M.G. Rabbat and R.D. Nowak. Quantized incremental algorithms for distributed optimization. *IEEE Journal on Selected Areas in Communications*, 23(4):798–808, 2005.
- [145] Sashank J Reddi, Ahmed Hefny, Suvrit Sra, Barnabás Póczós, and Alex Smola. On variance reduction in stochastic gradient descent and its asynchronous variants. In *Advances in Neural Information Processing Systems 28*, pages 2647–2655, 2015.
- [146] Sashank J Reddi, Ahmed Hefny, Suvrit Sra, Barnabás Póczós, and Alex Smola. Stochastic variance reduction for nonconvex optimization. *arXiv:1603.06160*, 2016.
- [147] Sashank J Reddi, Jakub Konečný, Peter Richtárik, Barnabás Póczós, and Alex Smola. AIDE: Fast and communication efficient distributed optimization. *arXiv:1608.06879*, 2016.
- [148] Peter Richtárik and Martin Takáč. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, 144(1–2):1–38, 2014.
- [149] Peter Richtárik and Martin Takáč. Distributed coordinate descent method for learning with big data. *Journal of Machine Learning Research*, 17(75):1–25, 2016.
- [150] Peter Richtárik and Martin Takáč. On optimal probabilities in stochastic coordinate descent methods. *Optimization Letters*, 10(6):1233–1243, 2016.
- [151] Peter Richtárik and Martin Takáč. Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, 156(1–2):433–484, 2016.
- [152] Peter Richtárik and Martin Takáč. Stochastic reformulation of linear systems and fast stochastic iterative methods. Technical report, 2016.
- [153] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.
- [154] Ralph T Rockafellar. *Convex Analysis*. Princeton University Press, 1970.
- [155] Lorenzo Rosasco, Silvia Villa, and Bang Công Vũ. Convergence of stochastic proximal gradient algorithm. *arXiv:1403.5074*, 2014.
- [156] Nicolas Le Roux, Mark Schmidt, and Francis Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *Advances in Neural Information Processing Systems 25*, pages 2663–2671, 2012.
- [157] Mark Schmidt, Reza Babanezhad, Mohamed Ahmed, Aaron Defazio, Ann Clifton, and Anoop Sarkar. Non-uniform stochastic average gradient method for training conditional random fields. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics*, pages 819–828, 2015.

- [158] Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, pages 1–30, 2013.
- [159] Shai Shalev-Shwartz. SDCA without duality. *arXiv:1502.06177*, 2015.
- [160] Shai Shalev-Shwartz. SDCA without duality, regularization, and individual convexity. *arXiv:1602.01582*, 2016.
- [161] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.
- [162] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for SVM. *Mathematical programming*, 127(1):3–30, 2011.
- [163] Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss. *Journal of Machine Learning Research*, 14(1):567–599, 2013.
- [164] Shai Shalev-Shwartz and Tong Zhang. Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization. *Mathematical Programming*, pages 1–41, 2014.
- [165] John Shalf, Sudip Dosanjh, and John Morrison. Exascale computing technology challenges. In *International Conference on High Performance Computing for Computational Science*, pages 1–25, 2010.
- [166] Ohad Shamir and Nathan Srebro. Distributed stochastic optimization and learning. In *52nd Annual Allerton Conference on Communication, Control and Computing*, pages 850–857, 2014.
- [167] Ohad Shamir, Nathan Srebro, and Tong Zhang. Communication efficient distributed optimization using an approximate Newton-type method. In *Proceedings of the 31st International Conference on Machine Learning*, 2014.
- [168] Virginia Smith, Simone Forte, Michael I Jordan, and Martin Jaggi. L<sub>1</sub>-Regularized Distributed Optimization: A Communication-Efficient Primal-Dual Framework. *arXiv:1512.04011*, 2015.
- [169] Daniel A Spielman and Shang-Hua Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *SIAM Journal on Matrix Analysis and Applications*, 35(3):835–885, 2014.
- [170] Thomas Strohmer and Roman Vershynin. A Randomized Kaczmarz Algorithm with Exponential Convergence. *Journal of Fourier Analysis and Applications*, 15(2):262–278, 2009.
- [171] Ananda Theertha Suresh, Felix X Yu, Brendan H McMahan, and Sanjiv Kumar. Distributed mean estimation with limited communication. *arXiv:1611.00429*, 2016.
- [172] Martin Takáč, Avleen Bijral, Peter Richtárik, and Nathan Srebro. Minibatch primal and dual methods for support vector machines. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.
- [173] Martin Takáč, Peter Richtárik, and Nathan Srebro. Distributed mini-batch SDCA. *arXiv:1507.08322*, 2015.
- [174] Rachael Tappenden, Peter Richtárik, and Burak Büke. Separable approximations and decomposition methods for the augmented Lagrangian. *Optimization Methods and Software*, 30(3):643–668, 2015.
- [175] Rachael Tappenden, Peter Richtárik, and Jacek Gondzio. Inexact coordinate descent: complexity and preconditioning. *Journal of Optimization Theory and Applications*, 170(1):144–176, 2016.

- [176] Rachael Tappenden, Martin Takáč, and Peter Richtárik. On the complexity of parallel coordinate descent. *arXiv:1503.03033*, 2015.
- [177] John N Tsitsiklis. Problems in decentralized decision making and computation. Technical report, DTIC Document, 1984.
- [178] Vladimir N Vapnik. *Statistical Learning Theory*, volume 1. Wiley New York, 1998.
- [179] Vladimir N Vapnik. An overview of statistical learning theory. *IEEE transactions on Neural Networks*, 10(5):988–999, 1999.
- [180] Huahua Wang and Arindam Banerjee. Randomized block coordinate descent for online and stochastic optimization. *arXiv:1407.0107*, 2014.
- [181] Jialei Wang and Lin Xiao. Exploiting strong convexity from data with primal-dual first-order algorithms. *arXiv:1703.02624*, 2017.
- [182] White House Report. Consumer data privacy in a networked world: A framework for protecting privacy and promoting innovation in the global digital economy. *Journal of Privacy and Confidentiality*, 2013.
- [183] Wikipedia. Variable-length quantity, 2016. [Online; accessed 9-Nov-2016].
- [184] David P Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends in Theoretical Computer Science*, 10(12):1–157, 2014.
- [185] Blake Woodworth and Nathan Srebro. Tight complexity bounds for optimizing composite objectives. *arXiv:1605.08003*, 2016.
- [186] Stephen J Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.
- [187] Lin Xiao and Tong Zhang. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4):2057–2075, 2014.
- [188] Tianbao Yang. Trading computation for communication: Distributed stochastic dual coordinate ascent. In *Advances in Neural Information Processing Systems 26*, pages 629–637, 2013.
- [189] Tianbao Yang, Shenghuo Zhu, Rong Jin, and Yuanqing Lin. Analysis of distributed stochastic dual coordinate ascent. *arXiv:1312.1031*, 2013.
- [190] Felix X Yu, Ananda Theertha Suresh, Krzysztof Choromanski, Daniel Holtmann-Rice, and Sanjiv Kumar. Orthogonal random features. In *Advances in Neural Information Processing Systems 29*, pages 1975–1983, 2016.
- [191] Hsiang-Fu Yu, Cho-Jui Hsieh, Kai-Wei Chang, and Chih-Jen Lin. Large linear classification when data cannot fit in memory. *ACM Transactions on Knowledge Discovery from Data*, 5(4):1–23, 2012.
- [192] Guo-Xun Yuan, Chia-Hua Ho, and Chih-Jen Lin. Recent advances of large-scale linear classification. *Proceedings of the IEEE*, 100(9):2584–2603, 2012.
- [193] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, volume 10, page 10, 2010.
- [194] C Zhang, H Lee, and K G Shin. Efficient distributed linear classification algorithms via the alternating direction method of multipliers. *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics*, 2012.
- [195] Lijun Zhang, Mehrdad Mahdavi, and Rong Jin. Linear convergence with condition number independent access of full gradients. In *Advances in Neural Information Processing Systems*, pages 980–988, 2013.

- [196] Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the 21st International Conference on Machine Learning*, pages 116–123, 2004.
- [197] Yuchen Zhang, John Duchi, Michael I Jordan, and Martin J Wainwright. Information-theoretic lower bounds for distributed statistical estimation with communication constraints. In *Advances in Neural Information Processing Systems*, pages 2328–2336, 2013.
- [198] Yuchen Zhang, John C Duchi, and Martin J Wainwright. Communication-efficient algorithms for statistical optimization. *Journal of Machine Learning Research*, 14:3321–3363, 2013.
- [199] Yuchen Zhang and Xiao Lin. DiSCO: Distributed optimization for self-concordant empirical loss. In *Proceedings of the 32th International Conference on Machine Learning*, pages 362–370, 2015.
- [200] Yuchen Zhang, Martin J. Wainwright, and John C Duchi. Communication-efficient algorithms for statistical optimization. In *Advances in Neural Information Processing Systems 25*, pages 1502–1510, 2012.
- [201] Peilin Zhao and Tong Zhang. Stochastic optimization with importance sampling for regularized loss minimization. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1–9, 2015.
- [202] Yong Zhuang, Wei-Sheng Chin, Yu-Chin Juan, and Chih-Jen Lin. Distributed newton methods for regularized logistic regression. In *Advances in Knowledge Discovery and Data Mining*, pages 690–703. 2015.
- [203] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems 23*, pages 2595–2603, 2010.
- [204] Anastasios Zouzias and Nikolaos M Freris. Randomized extended kaczmarz for solving least squares. *SIAM Journal on Matrix Analysis and Applications*, 34(2):773–793, 2013.