Asynchronous optimization for machine learning

Rémi Leblond

THÈSE DE DOCTORAT

de l'Université de recherche Paris Sciences Lettres
PSL Research University

Préparée à l'École normale supérieure

Asynchronous optimization for Machine Learning

Optimisation asynchrone pour l'apprentissage statistique

École doctorale n°386

ÉCOLE DOCTORALE DE SCIENCES MATHÉMATIQUES DE PARIS CENTRE

Spécialité INFORMATIQUE

COMPOSITION DU JURY :

Francis Bach
INRIA Paris, Directeur de thèse

Hal Daumé III
Microsoft Research, Rapporteur

John Duchi
Stanford University, Rapporteur

Alexandre Gramfort
INRIA Saclay, Membre du Jury

Simon Lacoste-Julien
MILA/UDEM, Directeur de thèse

Jean-Philippe Vert
Google Research, Président du Jury

Soutenue par Rémi Leblond
le 15.11.2018

Dirigée par **Simon Lacoste-Julien**
et **Francis Bach**

ENS ÉCOLE NORMALE SUPÉRIEURE | PSL RESEARCH UNIVERSITY PARIS

The truth is rarely pure and never simple.

– Oscar Wilde, *The importance of being Earnest*

Acknowledgements

As three years of work come to an end, it is time for me to properly thank all those who made this PhD possible and turned it into a thoroughly enriching and enjoyable experience.

First, I want to thank John Duchi and Hal Daumé III for agreeing to review my dissertation and providing detailed and insightful comments about my work. I can only imagine the amount of perseverance required to get through the whole thing, and I appreciate the dedication that is necessary to attend a defense nine time zones away!

I also want to thank Alexandre Gramfort and Jean-Philippe Vert, and all the members of the jury for listening to me drone on for (much) too long, and for the spirited conversation and thought-provoking questions that followed. I was honored that you accepted to be part of my jury, and defending in front of you has been a pleasure.

Going back a bit in time, I have been extremely fortunate to conduct my research work at INRIA. For this, I have you to thank, Francis. Even though we did not actively collaborate, you are the one who made all of it possible. You were the one who suggested a PhD in the field, you introduced me to Simon, and you even found my financing! Now that it's over I can tell it's all been worth it and I have you to thank for this incredible experience.

Simon, I am extremely grateful to you. You have taught me so many things, from the most technical optimisation details to the art of non-passive aggressive rebuttal writing, that I cannot name them all. Observing your scientific rigor, your ability to handle new research fields and your profoundly compassionate approach to teaching research has been an incredible learning experience. I also want to thank you for accommodating my demanding lifestyle, and for placing your trust in me from the very beginning, even though my early career had seen me stray a bit from academia. It's not entirely impossible that you did not realize the magnitude of the risk you were taking, but I'm very grateful either way!

INRIA provided me with not only excellent working conditions and brilliant supervisors, but also a very supportive community of fellow researchers and students.

Some of you I have worked with directly. First, Fabian – whose help was essential to getting my PhD off the ground smoothly (see what I did there?) – you've been a pleasure to work with, taught me a bunch and last but not least, never complained about my doubtful technological choices.

Anton, you provided additional guidance when Simon's presence became more virtual, shared your insights and your scientific rigor in a cheerful and encouraging manner, and you did call me out on my (always) dubious technological choices. I'm still unsure how you

pulled off rewriting our entire code base in another framework in a couple of weeks, but I'll remember it can be done!

Jean-Baptiste, I cannot thank you enough for asking me to share your office when we moved. As it turns out, this decision led us to not only start a very enjoyable collaboration, but also to share our next employer, and most importantly to become close friends. I'm impatient for our next project, whether it be extending our research or getting you to beat your wife at table tennis (this one is highly speculative)!

Others have had the misfortune to share an office with me. Nicolas, first, who reminded me on my first day of how old a PhD student I was, and helped me navigate my first few months. Jean-Baptiste and Théophile, who took the brunt of the effort and had to withstand my constantly positive and fair coverage of the 2017 election, and commentary on many more topics. Gauthier, for a brief period of time, which shall be remembered fondly; and finally Robin, who lucked out because you only had to deal with my presence for a few months.

Many of you did not share in this misfortune, but we've still had many thoughtful (if slightly acerbic) discussions, which I've enjoyed immensely. Working and interacting with you all every day has been a privilege, and I'm happy to count you amongst my friends: Relja, Dmitry, Raphaël, Piotr, Andrei, Guilhem, Lénaic, Minsu, Alexandre d'Aspremont, Alexandre Defossez, Aymeric, Christophe, Thomas E., Loïc, Nicolas, Pierre, Damien G., Pascal, Robert, Yana, Vadim, Thomas K., Sesh, Yann, Maxime, Dmitrii, Julia, Loucas, Anastasia, Antoine, Rafael, Ignacio, Vincent, Alessandro, Damien S, Tatiana, Adrien, Matthew, Federico, Gül and Sergey.

On the personal side of things, I have been fortunate to be able to count on my many friends to help me forget work when things got a little difficult. There are too many of you to cite by name, but you know who you are. Many thanks.

Finally, I want to thank my family. My father, my brother and my two elder sisters for introducing me to the research world and showing me that it was a worthwhile path; my baby sister for resisting the urge to complete the perfect full house of PhDs when I couldn't; my mother for being an incredible source of support when I needed it the most. You have all helped me more than I can say.

I also want to take this opportunity to thank Coline and Raphaël, for making this endeavour even more challenging! I think you will understand what I mean in a little while.

And of course, my wife Cécile, who has been (almost) successful in not mocking me for starting my PhD after she was done with hers. You have been my rock in these turbulent three years. I am amazed every day at the breadth of the help you provide me, from emotional support to the most technical of details. Thank you.

Abstract

The impressive breakthroughs of the last two decades in the field of machine learning can be in large part attributed to the explosion of both computing power and available data. These two quantities, which used to be the main limiting factors for practitioners, have been replaced by a new bottleneck: algorithms. The focus of this thesis is thus on introducing novel methods that can take advantage of high data quantity and computing power. We present two independent contributions in this new direction.

In Part I, we develop and analyze novel fast optimization algorithms which take advantage of the advances in parallel computing architecture and can handle vast amounts of data. We introduce a new framework of analysis for asynchronous parallel incremental algorithms, which enable correct and simple proofs. We then demonstrate its usefulness by performing the convergence and speedup analysis for several methods, including two novel algorithms.

ASAGA is a sparse asynchronous parallel variant of the variance-reduced algorithm SAGA which enjoys fast linear convergence rates on smooth and strongly convex objectives. We prove that in the right conditions, it is linearly faster than its sequential counterpart, even without sparsity assumptions (which are widely required for such properties).

PROXASAGA is an extension of ASAGA to the more general setting where the regularizer can be non-smooth – a problem often encountered in machine learning applications. We prove that PROXASAGA can also achieve a linear speedup under mild conditions.

We provide extensive experiments comparing our new algorithms to the current state-of-art. We illustrate theoretical and practical speedups, and show that PROXASAGA significantly outperforms related methods on several large-scale datasets.

In Part II, we introduce new methods for complex structured prediction tasks, where several random variables are predicted at once. We focus on recurrent neural networks (RNNs), a family of widely successful models aimed at representing sequential data.

The traditional training algorithm for RNNs – based on maximum likelihood estimation (MLE) – suffers from several issues. The associated surrogate training loss notably ignores the information contained in structured losses and introduces discrepancies between train and test times that may hurt performance.

To alleviate these problems, we propose SEARNN, a novel training algorithm for RNNs inspired by the "learning to search" approach to structured prediction. SEARNN leverages test-alike search space exploration to introduce global-local losses that are closer to the test error than the MLE objective.

We demonstrate improved performance over MLE on three challenging tasks, and provide several subsampling strategies to enable SEARNN to scale to large-scale tasks, such as machine translation. Finally, after contrasting the behavior of SEARNN models to MLE

models, we conduct an in-depth comparison of our new approach to the related work.

Keywords: optimization; parallelization; variance reduction; structured prediction; RNN.

Résumé

Les impressionnantes avancées des vingt dernières années en apprentissage automatique peuvent être attribuées en large partie aux explosions combinées de la puissance computationnelle et de la quantité de données disponibles. Les nouveaux facteurs limitants du domaine sont maintenant les algorithmes. C'est pourquoi l'objectif de cette thèse est d'introduire de nouvelles méthodes capables de tirer profit de quantités de données et de ressources computationnelles importantes. Nous présentons deux contributions indépendantes qui s'incrivent dans cette direction de recherche.

Dans la partie I, nous développons et analysons de nouveaux algorithmes d'optimisation rapides, capables d'exploiter les récentes avancées en architecture de calcul parallèle et donc de traiter des quantités massives de données. Nous introduisons un nouveau cadre d'analyse pour les algorithmes incrémentaux et parallèles asynchrones, qui nous permet de faire des preuves correctes et simples. Nous démontrons ensuite son utilité en analysant les propriétés de convergence et d'accélération de plusieurs méthodes, dont deux nouveaux algorithmes.

ASAGA est une variante parallèle asynchrone et parcimonieuse de SAGA, un algorithme à variance réduite qui a un taux de convergence linéaire rapide dans le cas d'un objectif lisse et fortement convexe. Nous prouvons que dans les conditions adéquates, ASAGA est linéairement plus rapide que son alternative séquentielle, même en l'absence de parcimonie (alors que cette propriété est d'habitude nécessaire pour obtenir ce résultat).

PROXASAGA est une extension d'ASAGA au cas plus général où le terme de régularisation n'est pas lisse, ce qui est souvent le cas en apprentissage automatique. Nous prouvons que PROXASAGA obtient aussi une accélération linéaire dans des conditions raisonnables.

Nous avons réalisé des expériences approfondies pour comparer nos algorithms à l'état de l'art. Nous illustrons les accélérations théoriques et pratiques, et montrons que PROXASAGA surpasse les méthodes apparentées sur plusieurs jeux de données de grande taille.

Dans la partie II, nous présentons de nouvelles méthodes adaptées à des tâches complexes de prédiction structurée, dans lesquelles le but est de prédire plusieurs variables aléatoires en même temps. Nous nous concentrons sur les réseaux de neurones récurrents (RNNs), une famille de modèles efficaces pour représenter des données séquentielles.

L'algorithme d'entraînement traditionnel des RNNs est basé sur le principe du maximum de vraisemblance (MLE). Malgré d'excellentes performances dans nombre d'applications, cette méthode présente plusieurs limitations. Tout d'abord, la fonction de coût associée est une approximation qui ignore l'information contenue dans les métriques structurées; de plus, elle entraîne des divergences entre l'étape d'entraînement et celle de prédiction, qui

peuvent nuire à la performance du modèle.

Afin d'éviter ces problèmes, nous proposons SEARNN, un nouvel algorithme d'entraînement des RNNs inspiré de l'approche dite "learning to search" pour la prédiction structurée. SEARNN repose sur une exploration de l'espace d'états proche de celle de l'étape de prédiction, pour définir des fonctions de coût globales-locales qui sont plus proches de la métrique d'évaluation que ne l'est l'objectif associé à MLE.

Nous démontrons que les modèles entraînés avec SEARNN ont de meilleures performances que ceux appris via MLE pour trois tâches difficiles, et fournissons plusieurs stratégies de tirage aléatoire pour permettre l'adaptation de notre algorithme à des tâches à grande échelle comme la traduction automatique. Enfin, après avoir comparé le comportement des modèles associés à SEARNN aux modèles traditionnels, nous effectuons une comparaison détaillée de notre nouvelle approche aux travaux de recherche connexes.

Mots-clés : optimisation; parallélisation; réduction de variance; prédiction structurée; RNN.

Contents

Introduction

Statistical machine learning aims at providing computers with the means of understanding phenomenons through the observation of data. This scientific field has enjoyed a strong resurgence over the last two decades, as its capabilities have dramatically improved thanks to the enormous increase in both available data and computing power. Machine learning algorithms now provide state-of-the-art results in difficult tasks such as image recognition and machine translation, and even outperform human performance in complex games such as Go.

Over the last few years, several major trends have emerged in the field. First, as traditional methods buckled under the explosion of available data, introducing novel optimization methods capable of handling previously unseen quantities of items has become necessary. Second, performance on simple tasks has reached very high levels. Consequently, researchers have turned to more and more complex problems, such as *structured prediction* tasks, where not only one but several random variables are predicted, as well as their dependencies.

Both trends stem from the same source: data collection and computing power are no longer the main issues in machine learning, so algorithms have become the new bottlenecks. The focus of this thesis is thus on introducing novel methods that can take advantage of high data quantity and computing power.

We provide contributions in both aforementioned research directions. First, since raw CPU clock speed increase has stalled in the last few years, we dedicate Part I to very fast asynchronous parallel optimization methods which can leverage the recent improvements in computer architecture. Second, in Part II, we focus on complex structured prediction tasks. We introduce new training algorithms for a class of neural networks targeted at sequential prediction, which alleviate some of the issues of their predecessors.

Part I – Asynchronous optimization for machine learning

Many supervised learning tasks involve optimizing for a *finite-sum* objective of the form:

$$\min_{x \in \Omega} f(x), \quad f(x) := \frac{1}{n} \sum_{i=1}^{n} f_i(x). \tag{1}$$

This expression naturally arises from *empirical risk minimization*, where we try to optimize the performance of a model, estimated on the training set. In this thesis, we make a few additional assumptions on the features f_i and their average f: each f_i is differentiable and convex with L-Lipschitz continuous gradients, and f is μ-strongly convex (for more details about this objective and its occurrence in machine learning, see Chapter 1).

An old problem

Although (1) does not have a closed-form solution in the general case, there have been numerical optimization algorithms capable of solving it for centuries. Methods often use first-order or second-order derivative information to incrementally update the vector of parameters x.

First order. Chief amongst the first-order approaches is *batch gradient descent*, an iterative procedure which consists in repeatedly subtracting the gradient of the objective function from the current value of the parameters, following e.g.:

$$x^+ = x - \gamma f'(x)\,. \tag{2}$$

Given the assumptions we have made on the f_i and f, this algorithm performs very nicely. Indeed, one can prove that it converges *linearly* to the optimum x^*, i.e. that each iteration decreases the distance to x^* by a fixed fraction. The computational cost of computing a gradient is $\mathcal{O}(nd)$, linear in both the number of data points n and the dimensionality of the problem d.

Second order. *Newton's method*, the first second-order algorithm, is also an iterative procedure. However, in contrast to batch gradient descent, its update rule leverages the second-order derivative (provided it exists and is invertible):

$$x^+ = x - f''(x)^{-1} f'(x)\,. \tag{3}$$

On (1), Newton's method converges *quadratically*, i.e. only $\mathcal{O}(\log\log 1/\epsilon)$ iterations are needed to reach ϵ-accuracy. From a computational perspective, each iteration has a cost of $\mathcal{O}(nd^2 + d^3)$.

While these historical methods have proven successful over many years, they have become increasingly inadequate with the recent data explosion.

New methods to handle big data

As collecting data becomes easier, both n – the quantity of data samples – and d – their dimensionality – increase accordingly, attaining millions and beyond. In this context, the complexity of Newton-like optimization algorithms quickly becomes prohibitive. As a consequence, first-order methods, with their cheaper iterations, have sparked renewed interest.

However, as n grows very large, even standard gradient descent updates can become too expensive, as the cost of computing a full gradient grows linearly with the number of data points.

Stochastic approaches. One popular alternative to full gradient computation is stochastic gradient estimation, originally proposed over 50 years ago by Robbins and Monro (1951). Their method, dubbed *stochastic gradient descent*, proposes to use one $f_i'(x)$ sampled uniformly at random in $[1; n]$ at each step instead of the full gradient $f'(x)$.

The key advantage of this method is that the computational cost of each step no longer depends on n. However, this improvement comes at the cost of introducing variance in the gradient estimator. Consequently, the method only converge linearly up to a ball around the optimum, rather than directly to the optimum x^* as gradient descent does.

Variance reduction. In order to obtain convergence to x^*, one needs the variance of the estimator to vanish. The easiest approach is to use a decreasing step size γ, rather than a constant one as we have done until now. Given an appropriate schedule, SGD does indeed reach the optimum.

Unfortunately, using such a schedule implies losing the fast linear convergence of gradient descent. In the last few years, alternative methods have been introduced that reduce the variance of the gradient estimator without resorting to decreasing step sizes (Le Roux et al., 2012; Johnson and Zhang, 2013; Defazio et al., 2014a). The basic idea behind these algorithms is to use a smarter gradient estimator, whose variance naturally disappears at the

optimum.

Asynchronous parallelism. As all the methods we have mentioned up to this point are inherently sequential, they make use of a single CPU core. Unfortunately, CPU clock speed no longer increases as fast as it used to, and most of the computing power gains nowadays come from innovation in parallel multi-core architectures. To leverage these advances, modern optimization algorithms need to be adapted to the parallel setting where multiple thread work concurrently.

Although *synchronous* parallel approaches – where cores act in a synchronized fashion – are the easiest to implement and to analyze, they are only as fast as the slowest resource they are using. This lack of robustness and adaptability to heterogeneous resources is a significant downside. In contrast, in *asynchronous* algorithms threads operate fully independently, which results in better resource usage.

Researchers have leveraged these advantages to derive very fast alternatives both to SGD itself (Niu et al., 2011) and even to some variance-reduced alternatives such as SVRG (Reddi et al., 2015; Mania et al., 2017).

However, the careful convergence analysis of asynchronous parallel methods is arduous, due to the specific difficulties associated with this setup. For instance, while iterates can usually be clearly defined as a function of the previous iterates, this is no longer the case when the method is asynchronous because several threads are writing updates concurrently. The updates are *delayed*, since between the time the parameters are read and the time the associated update is written, other updates have been completed.

Another issue is that as threads write to the parameters independently, the state in memory is *inconsistent*, with bits from different updates. Finally, as we discuss thoroughly in Chapter 2, even properly labeling the iterates with a time label t is quite intricate. As a result, practitioners often resort to strong simplifying assumptions, thus introducing large discrepancies between the actual implementations and the ones they analyze.

A more thorough introduction to Part I can be found in Chapter 1.

Contributions

In Part I of this thesis, we focus on introducing new fast asynchronous parallel incremental optimization algorithms which can handle large-scale and high-dimensional data. We also provide careful convergence analysis for our methods, taking into account the specific issues associated with asynchrony. We now detail our three main contributions to the field.

A framework for correct and easy analysis. In Chapter 2, we revisit and clarify a technical problem present in a large fraction of the literature on randomized asynchronous parallel algorithms. We then provide a novel framework we can use for rigorous and simple convergence proofs, efficiently handling the difficulties due to inconsistency and delays. This framework enables us to derive proofs under realistic assumptions, such as inconsistent reads and writes and heterogeneous computation times. We demonstrate its usefulness by analyzing HOGWILD (Niu et al., 2011), the first asynchronous SGD variant.

A fully asynchronous variance-reduced incremental method. In Chapter 3 we introduce ASAGA, a new sparse asynchronous parallel extension of the variance-reduced incremental SAGA algorithm. Using our new framework of analysis, we show that ASAGA obtains a linear speedup over SAGA under reasonable conditions, even without sparsity assumptions when $n \gg d$.

We also revisit the analysis of KROMAGNON, an asynchronous parallel variant of SVRG. In contrast to the original paper (Mania et al., 2017), we obtain fast rates of convergence while doing away with a problematic gradient bound assumption, and prove that the method can attain linear speedups under similar conditions as ASAGA.

Extension to non-smooth objectives. Finally, in Chapter 4 we propose a new asynchronous SAGA extension targeted at non-smooth regularizers, which are commonplace in machine learning workloads (e.g. box constraints, Lasso, group Lasso etc.).

As the additional proximal step induces new difficulties in the asynchronous setting (both in practice and for the analysis), we start by introducing Sparse Proximal SAGA, a novel sequential variant of the SAGA algorithm which features a reduced cost per iteration in the presence of sparse gradients and a block-separable penalty. Crucially, its sparse updates are more adapted to the parallel setting than those of the original SAGA algorithm.

Second, we present PROXASAGA, a lock-free asynchronous parallel version of the aforementioned algorithm. This is the first asynchronous parallel variance-reduced method adapted to composite optimization. We prove that this method achieves a theoretical linear speedup with respect to the sequential version under assumptions on the sparsity of gradients and block-separability of the proximal term.

Empirical results. We provide empirical results for all the algorithms we analyze, illustrating the achieved speedups as well as their limitations. In particular, we demonstrate that PROXASAGA dramatically outperforms state-of-the-art alternatives on large sparse datasets.

Part II – Improved recurrent neural network training

While we dedicated Part I to introducing new *optimization* algorithms, in Part II we focus on novel methods for complex tasks. The excellent performance obtained on simple problems (e.g. binary classification) has prompted researchers to turn to more challenging issues. Notable amongst these are structured prediction tasks, which consist in predicting several random variables at once (taking into account their dependencies). Examples of such tasks include image segmentation, dependency parsing and machine translation.

As these problems can often be solved one variable at a time, we dedicate Part II of this thesis to sequential prediction methods. More specifically, we focus on recurrent neural networks (RNNs), which have proven quite successful for structured prediction applications in the last few years. RNNs aim at representing sequential data. To do so, they recursively apply the same transformation (or *cell*) f on the sequential data and their own previous predictions, outputting a sequence of hidden states: $h_t = f(h_{t-1}, y_{t-1}, x)$, with h_0 an initial state and x an optional input.

Maximum likelihood training for recurrent neural networks

Our work concerns conditional RNNs, which have a natural probabilistic interpretation. Each cell is supposed to output the conditional probability of a token, given all previous predictions in the sequence: $p(y_t|y_1, ..., y_{t-1}, x)$. Multiplying the outputs in the sequence thus yields the joint probability of the whole sequence $y_1...y_T$, thanks to the chain rule.

Provided we replace the outputs of the model with the ground truth tokens before feeding them back to the following cell – this is known as *teacher forcing* – we can obtain the conditional probability of the ground truth sequences $p(y|x)$. This allows us to use maximum likelihood estimation (MLE) to derive a surrogate loss: we train the model by maximizing the probability of the ground truth.

Issues with MLE

While this "universal" loss has led to many successes, it does suffer from a number of limitations. This comes as no surprise as RNNs are used for a wide variety of tasks, each with their own specific validation metrics, such as BLEU score, edit distance etc.

First, as it completely ignores the actual validation metric we are interested in optimizing for, it fails to exploit the wealth of data that such structured losses can offer. MLE training only concerns itself with maximizing the probability of the ground truth, and does not distinguish a poor candidate sequence from a strong one.

Second, it introduces discrepancies between the train and test phases, such as the *exposure* or *exploration bias* (Ranzato et al., 2016). Due to teacher forcing, the model is trained to make predictions, assuming that it has not made a single mistake before. At test time though, the model does not have access to the ground truth, and thus feeds its own previous predictions to its next cell for prediction instead. Of course, errors do occur. The model then finds itself in a situation it has not encountered during training and is thus more likely to make additional mistakes.

Alternative training methods. Improving RNN training thus appears as a relevant endeavor, which has received much attention recently. In particular, ideas coming from reinforcement learning (RL), such as the REINFORCE and ACTOR-CRITIC algorithms (Ranzato et al., 2016; Bahdanau et al., 2017), have been adapted to derive training losses that are more closely related to the test error that we actually want to minimize.

In order to address the issues of MLE training, we propose instead to use ideas from the structured prediction field, in particular from the "learning to search" (L2S) approach pioneered by Daumé et al. (2009).

A more thorough introduction to Part II can be found in Chapter 6.

Contributions

In part II, we aim at proposing new training algorithms for recurrent neural networks that do not suffer from the limitations of MLE, and thus lead to improved model performance.

Traditional MLE training and its limitations. In the introductory Chapter 6, we explain the importance of recurrent neural networks to tackle complex machine learning models. We describe its most common training algorithm, MLE, and detail its limitations, clarifying the claims of the related literature. We explore its alternatives in-depth, and finally state the goals of Part II of this thesis, as well as our contributions.

SEARNN: a new training algorithm that leverages the validation metric. In Chapter 7, we propose SEARNN, a novel training algorithm inspired by the L2S approach to structured prediction. SEARNN leverages test-alike search space exploration to introduce global-local losses that are closer to the test error than the MLE surrogate.

We provide the rationale for using L2S as a basis for improved RNN training by underlining surprisingly strong similarities between this algorithm and RNNs. We investigate scaling

schemes to allow SEARNN to handle tasks with large vocabulary sizes and long sequences. We demonstrate the usefulness of our method through comprehensive experiments on three challenging datasets.

Finally, we contrast our novel approach both empirically and theoretically to MLE and the related L2S- and RL-inspired methods.

Outline

Part I is organized in five chapters.

In Chapter 1, we detail the reasons why optimizing (1) is important in many machine learning applications. We then provide some context on the new challenges that practitioners are faced with in the era of "big data", and give some insight in the new methods that are developed accordingly. Finally, we state the goals of this thesis, as well as our contributions.

In Chapter 2, we introduce a new framework of analysis, enabling correct and easy convergence proofs for asynchronous parallel incremental algorithms. After explaining how this framework alleviates the issues previous analyses ran into, we demonstrate its usefulness by analyzing HOGWILD, obtaining better bounds under more realistic assumptions than in the related literature.

This chapter covers the first half of the conference paper Leblond et al. (2017) and its extended journal version Leblond et al. (2018b).

In Chapter 3, we focus on asynchronous parallel variants of high-performing variance-reduced algorithms. We introduce a sparse variant of the incremental SAGA algorithm, as well as its asynchronous parallel adaptation, ASAGA. Using the framework introduced in Chapter 2, we show that ASAGA converges linearly and obtains a linear speedup over its sequential counterpart (under assumptions).

We also conduct the analysis of KROMAGNON, improving the known bounds for this algorithm while removing problematic assumptions. We prove that this algorithm enjoys similar speedup properties as ASAGA.

This chapter is based on the second half of the journal paper Leblond et al. (2018b).

In Chapter 4, we extend ASAGA to the composite optimization objective, where the regularizer term can be non-smooth. We introduce a variant of SAGA with sparse proximal updates, which is amenable to adaptation to the asynchronous parallel setting. We perform

the convergence analysis for this algorithm, and show that it can obtain linear speedups under sparsity assumptions. Empirical benchmarks show that PROXASAGA significantly outperforms other state-of-the-art methods on large-scale datasets.

This chapter is based on the conference paper Pedregosa et al. (2017).

Finally, in Chapter 5 we provide a summary of our contributions, discuss the limitations of our approach and how to address them in future work.

Part II is organized in three chapters.

Chapter 6 is dedicated to the formal description of the traditional training algorithm for RNNs. We explore its issues in-depth and discuss existing alternative approaches.

In Chapter 7, we introduce and analyze the behavior and performance of a novel RNN training method called SEARNN, which leverages exploration to derive global-local losses that are closer to the validation metric than the MLE surrogate. We propose scaling schemes to improve the algorithm's scaling, and we demonstrate improved performance on three challenging structured prediction tasks.

This chapter covers the conference paper Leblond et al. (2018a).

Finally, we conclude Part II in Chapter 8. We summarize our findings, list the limitations of the novel algorithm we have proposed, and offer several promising new directions of research for advancing RNN training.

Publications.

The chapters of Part I are based on three papers, all written with Fabian Pedregosa and Simon Lacoste-Julien. Part II is based on the conference paper Leblond et al. (2018a), written jointly with Jean-Baptiste Alayrac, Anton Osokin and Simon Lacoste-Julien.

- Leblond et al. (2017) was published as a conference paper at AISTATS in 2017. It introduces our novel framework of analysis for asynchronous optimization, as well as ASAGA.
- Pedregosa et al. (2017) was published as a conference paper at NIPS in 2017. In this work we focus on the non-smooth objective setting and introduce and analyze PROXASAGA.

- Leblond et al. (2018b) is to be published as a journal paper at JMLR in 2018. It is an extended version of Leblond et al. (2017), to which we have added the analysis of both HOGWILD and KROMAGNON.
- Leblond et al. (2018a) was published as a conference paper at ICLR in 2018. In this publication, we propose SEARNN, our novel RNN training algorithm, and explore its properties and performance on complex structured prediction tasks.

Part I

Asynchronous Optimization for Machine Learning

Chapter 1

Modern optimization for machine learning

1.1 Motivation

In the first part of this dissertation, we will concentrate on methods to solve efficiently an optimization objective with a specific *finite sum* structure.

$$\min_{x \in \Omega} f(x) + h(x), \quad f(x) := \frac{1}{n} \sum_{i=1}^{n} f_i(x). \tag{1.1}$$

This objective is particularly relevant for machine learning applications where it arises naturally from various tasks.

Empirical risk minimization (ERM). One salient example is the *empirical risk minimization* process, which is a standard approach in the *supervised learning* setup. Broadly speaking, we want to find a mapping ϕ between inputs z_i and outputs y_i such that the discrepancy between the mapped inputs $\phi(z_i)$ and the outputs is small. To keep computational costs under control, we specify a family of parametric functions $\phi(\cdot, x)$, and then we try to find the best candidate in this family, i.e. the candidate that minimizes the average discrepancy between its mapping of the inputs $\phi(z_i, x)$ and the actual outputs y_i. The resulting objective function is $f(x)$, also called the *data fitting* term. By adding a *regularization* term $h(x)$ we obtain (1.1).

More formally, we are given an input set $\mathbf{Z}$, an output set $\mathbf{Y}$ and a real-valued loss function $L : \mathbf{Y} \times \mathbf{Y} \to \mathbb{R}$ which measures the discrepancy between two elements in $\mathbf{Y}$.

Typically, $\mathbf{Z}$ is $\mathbb{R}^d$ and $\mathbf{Y}$ is either $\{-1, 1\}$ (in the case of a classification task) or $\mathbb{R}$ (for regression tasks). We also assume that there exist a joint probability distribution $P(z, y)$ over $\mathbf{Z}$ and $\mathbf{Y}$. We aim to learn a mapping ϕ between the inputs and the outputs such that the *risk* associated with ϕ is minimized, where risk means the expected loss: $\mathbb{E}\big(L(\phi(z), y)\big)$.

Ideally, we could look for ϕ in the whole set of mappings between $\mathbf{Z}$ and $\mathbf{Y}$. Unfortunately, such an unconstrained search would be prohibitively expensive in most cases. To circumvent this issue, we arbitrarily impose constraints on ϕ. Typically, we assume that $\phi(\cdot, x)$ is one element of a parametric family of promising candidate functions. We then seek to find the best candidate in this family, i.e. the candidate that minimizes the expected loss.

To achieve this goal, we have at our disposal a *training set*, which consists in a set of inputs $(z_i)_{i=1}^n$, each associated with an output y_i. The instances (z_i, y_i) are assumed to be drawn identically and independently from the distribution $P(z, y)$. This training set is crucial because we usually do not have access to the joint distribution $P(z, y)$, which implies that the expected loss we seek to minimize cannot be computed. The training set gives us an estimator of the expected loss, which we can minimize instead. This is the *empirical risk minimization* principle. Intuitively, for $\phi(\cdot, x)$ to be a good mapping from $\mathbf{Z}$ to $\mathbf{Y}$, we need $\phi(z_i, x)$ to be a good approximation of y_i, i.e. $L\big(\phi(z_i, x), y_i\big)$ to be small.

If we define $f_i(x) := L\big(\phi(z_i, x), y_i\big)$ as the loss associated to data point i, we obtain the optimization objective (1.1) (in the specific case where $h(x) = 0$).

Background. In this dissertation, we consider the *unconstrained* optimization problem, i.e. $\Omega = \mathbb{R}^d$. In order to design efficient methods to solve (1.1), we also impose some restrictions on f and the individual f_i:

1. each f_i is differentiable;
2. each f_i is convex, i.e.:

$$\forall x_1, x_2 \in \mathbb{R}^d, \forall t \in [0, 1] : \quad f\big(tx_1 + (1-t)x_2\big) \leq tf(x_1) + (1-t)f(x_2); \quad (1.2)$$

3. f is μ-strongly convex, i.e. $f - \frac{\mu}{2}\|x\|^2$ is convex, or equivalently:

$$\forall x_1, x_2 \in \mathbb{R}^d, \quad f(x_1) \geq f(x_2) + \langle f'(x_2), x_1 - x_2 \rangle + \frac{\mu}{2}\|x_1 - x_2\|^2. \quad (1.3)$$

4. each f_i is convex with L-lipschitz continuous gradients:

$$\forall x_1, x_2 \in \mathbb{R}^d, \quad \|f'(x_1) - f'(x_2)\| \leq L\|x_1 - x_2\|. \quad (1.4)$$

We define $\kappa := \frac{L}{\mu}$ as the *condition number* of f, and x^* as the global minimizer of the objective (as the objective is strongly convex, x^* exists and is unique).

Historical methods. In the general case, there is no closed form solution for the optimization problem (1.1), even when the additional term $h(x)$ is set to 0 (for clarity's sake, we assume $h = 0$ throughout the rest of Section 1.1). Practitioners then turn to numerical optimization methods to obtain approximate solutions.

One of the oldest optimization method which can be used to solve (1.1) is *gradient descent*, also called *batch gradient descent* or *steepest descent*, dating back to the 19^{th} century (Cauchy, 1847). This iterative method consists in repeatedly subtracting from the current iterate the gradient of the objective computed at this iterate, until convergence It follows the general scheme:

$$x_{t+1} = x_t - \gamma_t f'(x_t), \tag{1.5}$$

where γ_t is a hyper-parameter called the *step size*. Intuitively this corresponds to solving to optimality the optimization problem for a *surrogate* objective function of the following quadratic form:

$$s_t(x) = f(x_t) + f'(x_t)^\intercal (x - x_t) + \frac{1}{2\gamma}(x - x_t)^\intercal (x - x_t). \tag{1.6}$$

s_t can be seen as the first order Taylor expansion of the objective.

Gradient descent behaves very well on (1.1), where f is μ-strongly convex and L-smooth (i.e., its gradient is continuous and L-Lipschitz). Indeed, when using a constant step size, the convergence of the algorithm is *linear* (Nesterov, 2004), i.e. there exists a constant $0 < \rho < 1$ such that $f(x_t) - f(x^*) = \mathcal{O}(\rho^t)$. As the suboptimality term is decreased by a constant fraction at every iteration, the convergence is also sometimes referred to as *exponential* or *geometric*.

The number of iterations required to reach an accuracy of an arbitrary $\epsilon > 0$ is thus $\mathcal{O}\big(\kappa \log(1/\epsilon)\big)$. [1] As the computational cost of each iteration is dominated by the gradient computation cost, $\mathcal{O}(nd)$, the overall complexity of the algorithm is $\mathcal{O}\big(nd\kappa \log(1/\epsilon)\big)$.

Gradient descent is part of the family of *first-order* methods, as it only exploits information about the gradient $f'(x_t)$. This is in contrast to *second-order* or *Newton-type* methods, which also require computing the Hessian (or second derivative) of the objective.

1. κ appears in the ρ constant. As it problem-dependent, we make explicit that dependency here.

Newton's method is also an iterative procedure where the parameters are updated at each step, following a slightly different scheme:

$$x_{t+1} = x_t - \gamma[f''(x_t)]^{-1} f'(x_t), \tag{1.7}$$

assuming that f is twice differentiable and that its Hessian matrix $f''(x_t)$ is invertible. Intuitively, this corresponds to solving to optimality the optimization problem for a different quadratic surrogate objective function:

$$S_t(x) = f(x_t) + f'(x_t)^\intercal (x - x_t) + \frac{1}{2\gamma}(x - x_t)^\intercal f''(x_t)(x - x_t). \tag{1.8}$$

S_t can be seen as the second order Taylor expansion of f at x_t. Newton's method converges significantly faster than gradient descent, although it can only be used in a more restricted set of problems. Indeed, when the Hessian is invertible and Lipschitz continuous, there exists for each local optima a neighborhood such that Newton's method converges *quadratically* with step size 1. This implies that there exist a constant $\eta > 0$ such that $f(x_t) - f(x^*) = \mathcal{O}(e^{-\eta 2^t})$.

To reach ϵ-accuracy, only $\mathcal{O}(\log\log(1/\epsilon))$ iterations are needed. However, the iterations from Newton's method are more costly than those of gradient descent: their complexity is $\mathcal{O}(nd^2 + d^3)$. Consequently, which method is fastest is problem dependent.

To reduce the complexity of each iteration, a number of algorithms use an approximation of the Hessian matrix. This is the case of *quasi-Newton* methods, such as BFGS (Broyden, 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970).

These historical methods have encountered great success in the past. However, the last few years have seen significant changes in the characteristics of problem (1.1), rendering gradient descent or Newton's method increasingly inefficient.

1.2 Modern challenges

1.2.1 The big data era

Over the last few years, both the quantity and the dimensionality of data in machine learning tasks have increased dramatically. This implies that d, the dimensionality of the optimization problem (1.1), [2] as well as n, the number of data points, can nowadays easily grow to millions, and beyond. In such a setting, the computational cost of very fast,

2. Throughout this section, we discuss methods for solving (1.1) in the specific case where h is smooth. We discuss extensions to non-smooth regularizers in Section 1.2.4.

Newton-like optimization routines – which require inverting a $d \times d$ matrix [3] – becomes prohibitive.

This has prompted a renewal of interest for first-order methods, whose dependency on d is linear. However, as n grows very large, even standard gradient descent updates can become too expensive, as the cost of computing a full gradient grows linearly with the number of data points.

Stochastic gradient descent. In order to alleviate this issue, practitioners have turned to stochastic methods where cheap unbiased random estimators are used as an approximation to the true gradient. When the objective has a finite-sum structure, a very natural unbiased estimator is the gradient $f_i'(x)$ of a single random factor. The resulting algorithm is called stochastic gradient descent (SGD), and was introduced by Robbins and Monro (1951). Here is its update rule:

$$x_{t+1} = x_t - \gamma_t f_{i_t}'(x_t), \tag{1.9}$$

where the $(i_t)_{t \geq 0}$ are sampled independently and uniformly at random. One key advantage of this update scheme is that the computation cost of each iteration – $\mathcal{O}(d)$ – is independent of n, so does not scale with it.

SGD has been very successful in practice and has been proven to converge in expectation in a variety of settings, including the one we focus on (see e.g. Schmidt (2014)). However, the switch to an approximate estimator comes at a cost: using random estimates introduces non-diminishing variance in the algorithm.

The cost of randomness. One consequence of this variance is that the algorithm is not *stable*, i.e. the iterates do not stay at the global minimizer x^* once they reach it. A simple way of explaining this phenomenon is to remark that although the individual $f_i'(x^*)$ sum to 0 (since $f'(x^*) = 0$), they are not equal to 0 themselves in the general case. Another consequence is that in the strongly-convex setting, SGD with a constant step size does not converge linearly to the optimum as does gradient descent. Instead, it converges linearly up to a ball around the optimum, because the variance of the estimator does not vanish when nearing x^*.

In order for SGD to converge all the way to the optimum, the variance of the gradient estimator needs to be handled. The traditional approach has been to use vanishing step sizes. As the step size multiplies the update, the variance of the update itself goes to 0 as the

3. The cost of inverting a $d \times d$ matrix is roughly $\mathcal{O}(d^3)$.

number of iterations increases. Although this does allow SGD to converge to the optimum, unfortunately using a vanishing step size also implies losing linear convergence. Typical decreasing step size schedules result in a $\mathcal{O}(\kappa/t)$ convergence rate.

1.2.2 Variance reduction

In the last few years, novel methods to reduce the variance of the gradient estimator – that do not require using a vanishing step size – have been introduced. The resulting algorithms enjoy linear convergence on finite-sum strongly convex objectives with a constant step size. The main idea behind these methods is to use a smarter gradient estimator than that of SGD, often by adding a correction term, such that its variance vanishes at the optimum x^*. They are called *variance reduction* algorithms. This family includes examples such as SAG (Le Roux et al., 2012) [4], SDCA (Shalev-Shwartz and Zhang, 2013), SAGA (Defazio et al., 2014a) and SVRG (Johnson and Zhang, 2013), among others.

General principle. Suppose you have an estimator U with high variance, and that we have access to another random variable V, which is positively correlated with U and whose expectation can be computed efficiently. We can then introduce a new estimator $W_\beta := \beta(U - V) + \mathbb{E}V$. We have that $\mathbb{E}W_\beta = \beta\mathbb{E}U + (1 - \beta)\mathbb{E}V$, and that $\text{var}(W_\beta) = \beta^2[\text{var}(U) + \text{var}(V) - 2\text{cov}(U, V)]$. Provided than the covariance term is big enough, the variance of the new estimator is reduced compared to the variance of the initial one. This mechanism falls into the category of variance reduction.

Application to incremental gradient methods. The estimator with high variance is $f_i'(x_t)$ in this case. The usual random variable with positive correlation is the same individual gradient, taken at an earlier iterate: $f_i'(x_s)$, where the definition of $s < t$ depends on the specific algorithm. We now illustrate this principle by detailing two algorithms, SAGA (Defazio et al., 2014a) and SVRG (Johnson and Zhang, 2013).

Borrowing our notation from Hofmann et al. (2015), we first present SAGA. SAGA maintains two moving quantities to optimize (1.1): the current iterate x and a table (memory) of historical gradients $(\alpha_i)_{i=1}^n$. [5] At every iteration, the SAGA algorithm samples uniformly at random an index $i \in \{1, \dots, n\}$, and then executes the following update on x and α (for

4. It is worth noting that SAG was initially motivated as a lazy gradient evaluation method, rather than interpreted through the prism of variance reduction.

5. For linear predictor models, the memory α_i^0 can be stored as a scalar. Following Hofmann et al. (2015), α_i^0 can be initialized to any convenient value (typically 0), unlike the prescribed $f_i'(x_0)$ analyzed in (Defazio et al., 2014a).

the unconstrained optimization version):

$$x^+ = x - \gamma\big(f_i'(x) - \alpha_i + \bar{\alpha}\big); \qquad \alpha_i^+ = f_i'(x). \tag{1.10}$$

$\bar{\alpha} := {}^1\!/_n \sum_{i=1}^n \alpha_i$ can be updated efficiently in an online fashion. Let us define $\mathbf{E}$ as the conditional expectation of a random i given on all the past. By construction, $\mathbf{E}\alpha_i = \bar{\alpha}$ and thus the update direction is unbiased ($\mathbf{E}x^+ = x - \gamma f'(x)$). It can be proven (see Defazio et al. (2014a)) that under a reasonable condition on γ, the update has vanishing variance, which enables the algorithm to converge linearly with a constant step size.

The standard SVRG algorithm (Johnson and Zhang, 2013) is very similar to SAGA. The main difference is that instead of maintaining a table of historical gradients, SVRG uses a "reference" batch gradient $f'(\tilde{x})$, updated at regular intervals (typically every m iterations, where m is a hyper-parameter). SVRG is thus an epoch-based algorithm. At the beginning of every epoch, a reference iterate $\tilde{x}$ is chosen and its gradient is computed. Then, at every iteration in the epoch, the algorithm samples uniformly at random an index $i \in \{1, \dots, n\}$, and then executes the following update on x:

$$x^+ = x - \gamma\big(f_i'(x) - f_i'(\tilde{x}) + f'(\tilde{x})\big)\,. \tag{1.11}$$

As for SAGA the update direction is unbiased ($\mathbf{E}x^+ = x - \gamma f'(x)$), and it can be proven (see Johnson and Zhang (2013)) that under a reasonable condition on γ and m (the epoch size), the update has vanishing variance, again enabling the algorithm to converge linearly with a constant step size.

Compared to SAGA, SVRG trades memory for computation time. Instead of storing the past gradients, the algorithm recomputes them at a fixed iteration point $\tilde{x}$. However the convergence rates of both methods are the same, as is their overall computational complexity: $\mathcal{O}\big((n+\kappa)d\log({}^1\!/_\epsilon)\big)$.

Related methods. SAGA and SVRG are only two examples of variance-reduced incremental optimization methods. This family contains numerous other instances, such as its precursor SAG (Le Roux et al., 2012), the *stochastic dual coordinate ascent* method (Shalev-Shwartz and Zhang, 2013, SDCA), MISOμ (Mairal, 2015) or FINITO (Defazio et al., 2014b), among many others. All these methods have similar rates of convergence and iterations. Their differences lie in their interpretation and proof technique. For a more detailed comparison, we recommend Defazio et al. (2014a).

1.2.3 Asynchronous optimization

All the methods we have presented up until this point share a key characteristic: they are all inherently sequential procedures, where we define the value of the next iteration as a simple function of the current one. This implies that these algorithms typically only use a single CPU core.

Unfortunately, while the size of the datasets continues to grow very fast, the clock speed of CPU cores has stopped its dramatic increase a few years ago. Performance gains now rely on efficient parallelization over multiples cores which have access to shared memory resources. Thus to take advantage of the multi-core architecture of modern computers, the aforementioned optimization algorithms need to be adapted to the parallel setting, where multiple threads work concurrently. [6]

Synchronous approaches. The easiest way to take advantage of parallel resources is through synchronous approaches, where all cores get assigned a task, and wait for the last of them to finish before starting a new one. This suits a specific type of problem very well, that is, *embarrassingly parallel* problems which decompose naturally in parallel subtasks. One relevant example if the computation of a full gradient of an objective with a finite-sum structure. Each core can be assigned a subset of data points, compute the associated gradients and then a simple average of all sub-results yields the desired quantity.

In the context of stochastic gradient descent, the workload at each iteration cannot be easily split. Nevertheless, one way to leverage synchronous parallelism is to compute the gradient of a random mini-batch of data points, thus decreasing the variance of the estimator.

However, despite these possible improvements, synchronous approaches are not very well adapted to the incremental optimization setup. The main reason why is that as cores have to wait for each other to finish a task before they can start a new one, the overall process is only as fast as the slowest core (assuming the workload is evenly spread out). In potentially heterogeneous architectures, where cores do not have the same clock speed and random latency issues are common, this implies that a lot of computation power is wasted because cores are idle a large amount of the time. In the worst case where one core gets stuck, the whole process is simply stopped, but even under relatively mild assumptions, the slowdown can be significant. For instance, Hannah and Yin (2017) show that under reasonable assumptions the penalty for choosing a synchronous approach over a method

6. Indeed, for extremely large-scale datasets, distributed approaches without shared memory resources can be implemented. However, a large fraction of machine learning problems can still be solved by a single multi-CPU machine. Additionally, in the distributed setting, each machine in a cluster is typically multi-core. Consequently the analysis of the parallel setting is of key interest, and is the focus of this thesis.

without synchronization grows at least as $\log(p)$, where p is the amount of computing resources. This is obviously suboptimal.

Removing synchronization. An alternative to synchronizing the execution of different cores is the asynchronous paradigm, where cores operate independently and do not wait on one another. This approach avoids the waste of computational power of synchronous methods, at the cost of increased complexity. Indeed, one advantage of synchronous algorithms is that their analysis does not differ significantly from the analysis in the sequential case.

In the asynchronous setting on the other hand, as cores operate independently, a number of new difficulties appear from the perspective of the analysis. For instance, between the time when a core reads the parameters from the shared memory and the time it writes to it, updates coming from other cores have already been written to memory, implying that the gradient update was computed on *stale* iterates. This phenomenon is commonly referred to as *delayed updates*.

Another difficulty when the cores are fully independent is that writes to shared memory may happen at the same time, resulting in *inconsistent* states or even *overwriting*.

Therefore, conducting the theoretical analysis asynchronous methods is difficult. Handling the introduction of delay and inconsistency in inherently sequential algorithms either requires resorting to simplifying but unrealistic assumptions, or deriving new frameworks of analysis. The seminal textbook of Bertsekas and Tsitsiklis (1989) provides most of the foundational work for parallel and distributed optimization algorithms.

Despite these difficulties, much work has been devoted recently in proposing and analyzing asynchronous parallel variants of incremental optimization algorithms. One of the earliest examples of these new algorithms is HOGWILD, an asynchronous variant of SGD with constant step size presented by Niu et al. (2011). In HOGWILD, multiple cores have common read and write access to a shared memory. Each core runs the same iteration independently: they sample a factor uniformly at random, read the current parameter values, compute a gradient and finally apply the update to the parameters in shared memory. All cores operate completely independently.

HOGWILD is a *lock-free* algorithm, in the sense that several cores can write to the parameters at the same time. However, each coordinate update is *atomic*: while one core is writing to a single coordinate, no other writing operation to that coordinate can occur. This can be achieved with *compare-and-swap* operations which are nowadays native to CPU cores and heavily optimized. It implies that there cannot be any *overwrites*, where one update disappears because of a concurrent update. An alternative way to look at it is

that instead of vector-level locks, HOGWILD uses coordinate-level locks which are more lightweight.

In the HOGWILD implementation, both reads and writes are *inconsistent*, which means that other cores can interact with the parameter vector during these operations.

However, for the purpose of analysis the authors make a number of simplifying assumptions. First, each update is assumed to be limited to a single coordinate. Second, reads and writes are assumed to be *consistent*, i.e. all other cores cannot interact with the parameters once a core starts reading or writing to them. Third, the stochastic gradients are supposed to be uniformly bounded (which is incompatible with the strongly convex setting). Finally, an implicit homogeneity assumption on the processing times is made (we will come back to this notion later in this section).

These assumptions help alleviate some of the difficulties associated with its asynchronous analysis, and allow the authors to prove that under a strong sparsity assumption, if the delays are uniformly bounded by a small enough constant, HOGWILD can attain *linear* speedups over its sequential counterpart, SGD. Unfortunately, these assumptions are unrealistic and introduce a large discrepancy between the implemented algorithm and the analyzed version.

De Sa et al. (2015) introduces a refined analysis of the same algorithm, obtaining a relaxed condition on the sparsity assumption. However, the analysis also relies on single-coordinate updates, as well as the homogeneity of running times and the existence of a uniform gradient bound.

Mania et al. (2017) introduce a new framework of analysis, allowing them to consider inconsistent reads, full updates and heterogeneous running times, though not to remove the gradient bound assumption.

Duchi et al. (2015) analyze a variant of the algorithm, with a decreasing step size instead of a constant one, [7] targeted at stochastic optimization. An asymptotic rate of convergence is proven under improved assumptions. First, the classical uniform bound on the delay is replaced by a weaker assumption on the moments of the random delays. Second, no sparsity assumptions are required, contrary to most of the previous analyses. Finally, inconsistent reads are allowed.

In a recent development, Nguyen et al. (2018) perform the analysis of the algorithm with decreasing step size without assuming a uniform bound on the stochastic gradients, while also allowing for inconsistent reads.

Asynchronous variance reduction. As we have previously mentioned, while implementing SGD is easy it implies losing linear convergence. As a natural consequence of both

7. Note that this implies that there is a shared iteration counter to which all cores have access.

the successes of HOGWILD and variance reduction, several asynchronous variance-reduced algorithms have been introduced and analyzed.

Hsieh et al. (2015) introduces three variant of asynchronous SDCA, including a "wild" version without any locks (not even atomic updates). Although they do not provide theoretical speedup results, the authors show that this "wild" algorithm leads to convergence, albeit to a perturbed version of the initial objective.

Zhao and Li (2016) show that under an assumption on the amount of overwriting, an asynchronous variant of SVRG without any locks converges linearly. Unfortunately, as the considered SVRG update is dense, the amount of overwriting can grow to be very large. Good speedups are demonstrated experimentally, although not analyzed from a theoretical perspective.

Reddi et al. (2015) presents a hybrid algorithm called HSAG that includes SAGA and SVRG as special cases. Their asynchronous analysis is epoch-based though, and thus does not handle a fully asynchronous version of SAGA as will be presented in Chapter 3. Moreover, they require consistent reads and do not propose an efficient sparse implementation, which is key from an implementation perspective to obtain good performance.

Mania et al. (2017) uses their new framework of analysis to analyze a sparse asynchronous SVRG variant dubbed KROMAGNON. The authors show that KROMAGNON converges linearly under the gradient bound assumption, but the obtained rate of convergence is much worse than that of SVRG. This implies that no linear speedup results are provided.

Pan et al. (2016) proposes a black box mini-batch algorithm to parallelize SGD-like methods while maintaining serial equivalence through smart update partitioning. When the dataset is sparse enough, they obtain speedups over "HOGWILD" implementations of SVRG and SAGA. [8] However, these "HOGWILD" implementations appear to be quite suboptimal, as they do not leverage dataset sparsity efficiently: they try to adapt the "lazy updates" trick from Schmidt et al. (2016) to the asynchronous parallel setting – which as discussed in Appendix B.4 is extremely difficult – and end up making several approximations which severely penalize the performance of the algorithms. In particular, they have to use much smaller step sizes than in the sequential version, which makes for worse results.

Among the incremental gradient algorithms with fast linear convergence rates that can optimize (1.1) in its general form, only SVRG had had an asynchronous parallel version proposed. [9] No such adaptation had been attempted for SAGA until Leblond et al. (2017),

8. By "HOGWILD", the authors mean asynchronous parallel variants where cores independently run the sequential update rule.

9. We note that SDCA requires the knowledge of an explicit μ-strongly convex regularizer in (1.1), whereas SAG / SAGA are adaptive to any local strong convexity of f (Schmidt et al., 2016; Defazio et al., 2014a). The

even though one could argue that it is a more natural candidate as, contrarily to SVRG, it is not epoch-based and thus has no synchronization barriers at all. Leblond et al. (2017) is the publication Chapter 3 is based on.

Framework of analysis. Although asynchronous algorithms offer the best chance of fully leveraging multiple resources, their analysis is notoriously difficult. Handling delayed updates, as well as inconsistency, requires specialized approaches. Indeed, even something as seemingly innocuous as defining the iteration numbering is actually a non-trivial decision, as we will show in Chapter 2.

The framework of analysis introduced by Niu et al. (2011) has been extensively re-used and inspired most of the recent literature on asynchronous parallel optimization algorithms with convergence rates, including asynchronous variants of coordinate descent (Liu et al., 2015), SDCA (Hsieh et al., 2015), SGD for non-convex problems (De Sa et al., 2015; Lian et al., 2015), SGD for stochastic optimization (Duchi et al., 2015) and SVRG (Reddi et al., 2015; Zhao and Li, 2016).

Unfortunately, these papers make use of an unbiased gradient assumption that is not consistent with the proof technique, and thus suffer from technical problems [10] that we highlight in Section 2.2.2.

The "perturbed iterate" framework presented in Mania et al. (2017) is to the best of our knowledge the only one that does not suffer from this problem. However, the convergence proofs in this framework involve significant technical difficulties, leading to below state-of-the-art results (notably in the case of KROMAGNON, the sparse asynchronous variant of SVRG introduced in the paper). In Chapter 2, we build from their approach to propose a simpler, more convenient way to label the iterates (the "after read" framework introduced in Section 2.2.2). In Chapter 3 and 4, we show that our new framework enables correct and simpler convergence proofs for complex asynchronous algorithms. This is confirmed by the fact that recent papers, such as Nguyen et al. (2018) have adopted our "after read" framework to handle the analysis of asynchronous parallel algorithms.

This concludes our non-exhaustive review of asynchronous methods for solving (1.1) with a smooth regularizer h.

variant of SVRG from Hofmann et al. (2015) is also adaptive (we review this variant in Section 3.3.1).

10. Except (Duchi et al., 2015) that can be easily fixed by incrementing their global counter *before* sampling.

1.2.4 Non-smooth regularization

Due to their simplicity and excellent performance, parallel asynchronous variants of stochastic gradient descent have become popular methods to solve a wide range of large-scale optimization problems on multi-core architectures. Yet, despite their practical success, these methods do not support non-smooth objectives, which makes them unsuitable for many problems of interest in machine learning, such as the Lasso, group Lasso or empirical risk minimization with convex constraints. We now strive to provide a detailed – though not an exhaustive – review of the state-of-the-art approaches for solving (1.1) in the composite setting.

On the difficulty of a composite extension. Two key issues explain the paucity in the development of asynchronous incremental gradient methods for composite optimization. The first issue is related to the design of such algorithms. Asynchronous variants of SGD are most competitive when the updates are sparse and have a small overlap, that is, when each update modifies a small and different subset of the coefficients. This is typically achieved by updating only coefficients for which the partial gradient at a given iteration is nonzero, [11] but existing schemes such as the lagged updates technique (Schmidt et al., 2016) are not applicable in the asynchronous setting.

The second difficulty is related to the analysis of such algorithms. All convergence proofs crucially use the Lipschitz condition on the gradient to bound the noise terms derived from asynchrony. However, in the composite case, the gradient mapping term (Beck and Teboulle, 2009), which replaces the gradient in proximal-gradient methods, does not have a bounded Lipschitz constant. Hence, the traditional proof technique breaks down in this scenario.

Asynchronous coordinate-descent. For composite objective functions of the form (1.1), most of the existing literature on asynchronous optimization has focused on variants of coordinate descent.

Liu and Wright (2015) proposed an asynchronous variant of (proximal) coordinate descent and proved a near-linear speedup with respect to the number of cores used, given a suitable step size. This approach has been recently extended to general block-coordinate schemes by Peng et al. (2016), to greedy coordinate-descent schemes by You et al. (2016) and to non-convex problems by Davis et al. (2016).

11. Although some regularizers are sparsity inducing, large scale datasets are often extremely sparse and leveraging this property is crucial for the efficiency of the method.

All these algorithms yield linear speedups over their sequential counterpart, provided the uniform bound on the delay – which is a common assumption – is small enough. All analyses also support inconsistent reads.

Interestingly, as coordinate descent approaches typically update a single coordinate per iteration, only lightweight coordinate-wise locks should be required (i.e. each update to a coordinate is atomic). However, in all approaches (except for Peng et al. (2016)), the atomic operation consists in both the coordinate update and the proximal step, which implies global locks on all the coordinates required by the proximal step. These locks come with a heavier cost than coordinate-wise locks.

Furthermore, one has to note that while the aforementioned approaches [12] use the classical labeling scheme inherited from Niu et al. (2011), they still assume in their proof that their gradient estimators are conditionally unbiased – though this property is not verified in the general asynchronous setting.

Despite the successes obtained by these methods, they may not provide the optimal approach to solving (1.1). Indeed, as illustrated by our experiments (see e.g. Figure 4-2), in the large sample regime coordinate descent compares poorly against incremental gradient methods like SAGA. This motivates the study of asynchronous variance-reduced methods for non-smooth objectives.

Variance-reduced incremental gradient and their asynchronous variants. While initially proposed in the context of smooth optimization by Le Roux et al. (2012), variance-reduced incremental gradient methods have since been extended to minimize composite problems of the form (1.1) (see table below). As we have detailed earlier in this section, smooth variants of these methods have also recently been extended to the asynchronous setting.

Interestingly, none of these methods achieve both simultaneously, i.e. asynchronous optimization of composite problems. Since variance-reduced incremental gradient methods have shown state-of-the-art performance in both settings, this generalization is of key practical interest.

This is precisely the focus of Pedregosa et al. (2017), which extend the ASAGA algorithm presented in Chapter 3 to the proximal setting. Pedregosa et al. (2017) is the publication Chapter 4 is based on.

12. With the exception of Peng et al. (2016) which can easily be fixed by updating the global iteration counter at the *start* of each iteration, rather than at its end.

Objective	Sequential Algorithm	Asynchronous Algorithm
Smooth	SVRG (Johnson and Zhang, 2013)	SVRG (Reddi et al., 2015)
	SDCA (Shalev-Shwartz and Zhang, 2013)	PASSCODE (Hsieh et al., 2015, SDCA variant)
	SAGA (Defazio et al., 2014a)	ASAGA (Chapter 3, SAGA variant)
Composite	PROXSDCA (Shalev-Shwartz and Zhang, 2012)	
	SAGA (Defazio et al., 2014a)	PROXASAGA (Chapter 4)
	ProxSVRG (Xiao and Zhang, 2014)	

Other approaches. Recently, Meng et al. (2017); Gu et al. (2016) independently proposed a doubly stochastic method to solve the problem at hand. Following Meng et al. (2017) we refer to it as Async-PROXSVRCD. This method performs coordinate descent-like updates in which the true gradient is replaced by its SVRG approximation. It hence features a doubly-stochastic loop: at each iteration we select a random coordinate *and* a random sample. Because the selected coordinate block is uncorrelated with the chosen sample, the algorithm can be orders of magnitude slower than SAGA in the presence of sparse gradients. Appendix 4.4 contains a comparison of these methods.

1.3 Goal and contributions

This quick review of the state of the art in asynchronous optimization for solving (1.1) provides the context in which we now detail the aims of this thesis.

From a high-level perspective, we want to develop and analyze new fast and flexible optimization methods capable of handling the modern challenges of machine learning workloads – large-scale datasets, high model dimensionality, non-smooth objectives – while efficiently taking advantage of the recent advances in computer architecture. This means developing asynchronous parallel incremental optimization algorithms, as well as an adequate framework for their analysis under realistic assumptions.

At a finer level of granularity, our contributions are three-fold. [13]

A framework for correct and easy analysis. We have seen that asynchrony introduces significant new difficulties compared to the sequential paradigm, making accurate analysis arduous. In particular, taking inconsistency and delays into account and properly defining iterate labeling require considerable care. In Chapter 2, through a novel perspective, we

13. Note that these contributions will be contrasted to the related work in more details in the associated chapters.

revisit and clarify a technical problem present in a large fraction of the literature on randomized asynchronous parallel algorithms: namely, they assume unbiased gradient estimates, an assumption that is inconsistent with their proof technique without further unpractical synchronization assumptions.

We then provide a novel framework we can use for rigorous and simple convergence proofs. Our framework enables us to properly handle the difficulties of asynchronous analysis, as well as to derive proofs under realistic assumptions, such as inconsistent reads and writes and heterogeneous computation times.

We demonstrate the framework's usefulness by analyzing HOGWILD, the first asynchronous SGD variant, notably removing the classical gradient bound assumption (which is incompatible with the strong convexity assumption in the usual unconstrained setup).

A fully asynchronous variance-reduced incremental method. Variance-reduced algorithms have demonstrated improved performance over SGD, at the cost of more complex update rules and convergence proofs. Deriving asynchronous parallel extensions is thus a logical step, provided we can adapt their analysis. In Chapter 3 we introduce ASAGA, a new sparse asynchronous parallel extension of the variance-reduced incremental SAGA algorithm. We provide a tailored convergence analysis under realistic assumptions, and show that ASAGA obtains a linear speedup over SAGA under reasonable conditions – indeed, contrary to most related methods, sparsity is not a requirement in all regimes.

We also use our improved framework to revisit the analysis of KROMAGNON, an asynchronous parallel variant of SVRG. In contrast to the original paper (Mania et al., 2017), we obtain fast rates of convergence while doing away with the problematic gradient bound assumption, and prove that the method can attain linear speedups under similar conditions as ASAGA.

Finally, we provide practical implementations of these algorithms to illustrate their performance on a 40-cores architecture, showing significant improvements over HOGWILD.

Extension to non-smooth objectives. Many problems in machine learning involve a non-smooth regularizer (e.g. box constraints, Lasso, group Lasso etc.). In the sequential setup, this term is usually handled through the addition of a proximal step. In the asynchronous setting, this two-step update causes new difficulties for the analysis, notably because the objective's Lipschitz assumption cannot be used directly and because the updates are dense.

In Chapter 4, we handle these issues one by one. First, we describe Sparse Proximal SAGA, a novel variant of the SAGA algorithm which features a reduced cost per iteration in the presence of sparse gradients and a block-separable penalty. This method can be

applied to composite objectives. Like other variance-reduced methods, it enjoys a linear convergence rate under strong convexity. Crucially, as its updates are sparse, it is more adapted to the parallel setting than the original SAGA algorithm.

Second, we present PROXASAGA, a lock-free asynchronous parallel version of the aforementioned algorithm that does not require consistent reads. To our knowledge, this is the first asynchronous parallel variance-reduced method adapted to composite optimization. We prove that this method achieves a theoretical linear speedup with respect to the sequential version under assumptions on the sparsity of gradients and block-separability of the proximal term.

We report empirical benchmarks and demonstrate that this method dramatically outperforms state-of-the-art alternatives on large sparse datasets, while the empirical speedup analysis illustrates the practical gains as well as its limitations.

Chapter 2

Improved asynchronous parallel optimization analysis for stochastic incremental methods

In this chapter, we revisit and clarify a technical problem present in a large fraction of the literature on randomized asynchronous parallel algorithms (with the exception of Mania et al. (2017), which also highlights this issue): namely, they all assume unbiased gradient estimates, an assumption that is inconsistent with their proof technique without further unpractical synchronization assumptions.

To address this issue, we propose a simplification of the "perturbed iterate" framework from Mania et al. (2017) as a basis for our asynchronous convergence analysis, that enables correct and simple convergence proofs.

Finally, in order to show that our improved "after read" perturbed iterate framework can be used to revisit the analysis of other optimization routines with correct proofs that do not assume homogeneous computation, we provide the analysis of the HOGWILD algorithm, first introduced by Niu et al. (2011). Our framework allows us to remove the classic gradient bound assumption and to prove speedups in more realistic settings.

2.1 Introduction

We consider the unconstrained optimization problem of minimizing a *finite sum* of smooth convex functions:

$$\min_{x \in \mathbb{R}^d} f(x), \quad f(x) := \frac{1}{n} \sum_{i=1}^{n} f_i(x), \tag{2.1}$$

where each f_i is assumed to be convex with L-Lipschitz continuous gradient, f is μ-strongly convex and n is large (for example, the number of data points in a regularized empirical risk minimization setting). We define a condition number for this problem as $\kappa := {}^{L}/_{\mu}$, as is standard in the finite sum literature. [1]

As mentioned in Sections 1.2.1 and 1.2.2, a flurry of randomized incremental algorithms (which at each iteration select i at random and process only one gradient f_i') have recently been proposed to solve (2.1). Amongst that number, some even offer a fast [2] linear convergence rate, such as SAG (Le Roux et al., 2012), SDCA (Shalev-Shwartz and Zhang, 2013), SVRG (Johnson and Zhang, 2013) and SAGA (Defazio et al., 2014a).

However, these incremental algorithms are inherently sequential and thus are implemented and analyzed as single-core procedures. In order to take advantage of the multi-core architecture of modern computers, they need to be adapted to the asynchronous parallel setting, where multiple threads work concurrently. Much work has been devoted recently in proposing and analyzing asynchronous parallel variants of algorithms such as SGD (Niu et al., 2011).

Unfortunately, the usual frameworks for asynchronous analysis are quite intricate (see Section 2.2.2) and thus require strong simplifying assumptions. They are not well suited to the study of relatively simple algorithms such as HOGWILD, let alone the more complex algorithms we will introduce in Chapters 3 and 4.

We therefore start by introducing a novel framework of analysis, an improvement upon the newly proposed "perturbed iterate" framework of Mania et al. (2017). We then demonstrate its usefulness by conducting a thorough analysis of HOGWILD, the simplest asynchronous incremental algorithm. In Chapters 3 and 4, we show that our new approach is not limited to asynchronous SGD but can be used to investigate other algorithms and improve their existing bounds.

Contributions. In Section 2.2, we propose a simplification of the "perturbed iterate" framework from Mania et al. (2017) as a basis for our asynchronous convergence analysis. At the same time, through a novel perspective, we revisit and clarify a technical problem present in a large fraction of the literature on randomized asynchronous parallel algorithms (with the exception of Mania et al. (2017), which also highlights this issue): namely, they all assume unbiased gradient estimates, an assumption that is *inconsistent* with their proof

1. Since we have assumed that each individual f_i is L-smooth, f itself is L-smooth – but its smoothness constant L_f could be much smaller. While the more classical condition number is $\kappa_b := {}^{L_f}/_{\mu}$, our rates are in terms of this bigger ${}^{L}/_{\mu}$ in this chapter.
2. Their complexity in terms of gradient evaluations to reach an accuracy of ϵ is $O((n+\kappa)\log({}^{1}/_{\epsilon}))$, in contrast to $O(n\kappa_b \log({}^{1}/_{\epsilon}))$ for batch gradient descent in the worst case.

technique without further unpractical synchronization assumptions.

In Section 2.3, in order to show that our improved "after read" perturbed iterate framework can be used to revisit the analysis of optimization routines with correct proofs that do not assume homogeneous computation, we provide the analysis of the HOGWILD algorithm, first introduced in Niu et al. (2011). Our framework allows us to remove the classic gradient bound assumption and to prove speedups in more realistic settings.

Notation. We denote by $\mathbb{E}$ a full expectation with respect to all the randomness in the system, and by $\mathbf{E}$ the *conditional* expectation of a random i (the index of the factor f_i chosen in SGD and other algorithms), conditioned on all the past, where "past" will be clear from the context. $[x]_v$ represents the coordinate v of the vector $x \in \mathbb{R}^d$. For *sequential* algorithms, x^+ is the updated parameter vector after one algorithm iteration.

2.2 Revisiting the perturbed iterate framework for asynchronous analysis

As most recent parallel optimization contributions, we use a similar hardware model to Niu et al. (2011). We consider multiple cores which all have read and write access to a shared memory. The cores update a central parameter vector in an asynchronous and lock-free fashion. Unlike Niu et al. (2011), we *do not* assume that the vector reads are consistent: multiple cores can read and write different coordinates of the shared vector at the same time. This also implies that a full vector read for a core might not correspond to any consistent state in the shared memory at any specific point in time.

2.2.1 Perturbed Iterate Framework

We first review the "perturbed iterate" framework recently introduced by Mania et al. (2017) which will form the basis of our analysis. In the sequential setting, stochastic gradient descent and its variants can be characterized by the following update rule:

$$x_{t+1} = x_t - \gamma g(x_t, i_t)\,, \tag{2.2}$$

where i_t is a random variable independent from x_t and we have the unbiasedness condition $\mathbf{E}g(x_t, i_t) = f'(x_t)$ (recall that $\mathbf{E}$ is the relevant-past conditional expectation with respect to i_t).

Unfortunately, in the parallel setting, we manipulate stale, inconsistent reads of shared parameters and thus we do not have such a straightforward relationship. Instead, Mania et al. (2017) proposed to distinguish $\hat{x}_t$, the actual value read by a core to compute an update, from x_t, a "virtual iterate" that we can analyze and is *defined* by the update equation:

$$x_{t+1} := x_t - \gamma g(\hat{x}_t, i_t)\,. \tag{2.3}$$

We can thus interpret $\hat{x}_t$ as a noisy (perturbed) version of x_t due to the effect of asynchrony.

We formalize the precise meaning of x_t and $\hat{x}_t$ in the next section. We first note that all references mentioned in the related work section that analyzed asynchronous parallel randomized algorithms assumed that the following unbiasedness condition holds:

$$[\text{unbiasedness condition}] \quad \mathbf{E}[g(\hat{x}_t, i_t)|\hat{x}_t] = f'(\hat{x}_t). \text{ [3]} \tag{2.4}$$

This condition is at the heart of most convergence proofs for randomized optimization methods. [4] Mania et al. (2017) correctly pointed out that most of the literature thus made the often implicit assumption that i_t is independent of $\hat{x}_t$. But as we explain below, this assumption is incompatible with a non-uniform asynchronous model in the analysis approach used in most of the recent literature.

2.2.2 On the difficulty of labeling the iterates

Formalizing the meaning of x_t and $\hat{x}_t$ highlights a subtle but important difficulty arising when analyzing *randomized* parallel algorithms: what is the meaning of t? This is the problem of *labeling* the iterates for the purpose of the analysis, and this labeling can have randomness itself that needs to be taken in consideration when interpreting the meaning of an expression like $\mathbb{E}[x_t]$. In this section, we contrast three different approaches in a unified framework. We notably clarify the dependency issues that the labeling from Mania et al. (2017) resolves and propose a new, simpler labeling which allows for much simpler proof techniques.

3. We note that to be completely formal and define this conditional expectation more precisely, one would need to define another random vector that describes the entire system randomness, including all the reads, writes, delays, etc. Conditioning on $\hat{x}_t$ in (2.4) is actually a shorthand to indicate that we are conditioning on all the relevant "past" that defines both the value of $\hat{x}_t$ as well as the fact that it was the t^{th} labeled element. For clarity of exposition, we will not go into this level of technical detail, but one could define the appropriate sigma fields to condition on in order to make this equation fully rigorous.

4. A notable exception is SAG (Le Roux et al., 2012) which has biased updates and thus requires a significantly more complex convergence proof. Making SAG unbiased leads to SAGA (Defazio et al., 2014a), with a much simpler convergence proof.

We consider algorithms that execute in parallel the following four steps, where t is a global labeling that needs to be defined: [5]

1. Read the information in shared memory ($\hat{x}_t$).
2. Sample i_t.
3. Perform some computations using ($\hat{x}_t, i_t$).
4. Write an update to shared memory.

(2.5)

The "After Write" Approach. We call the "after write" approach the standard global labeling scheme used in Niu et al. (2011) and re-used in all the later papers that we mentioned in the related work section, with the notable exceptions of Mania et al. (2017) and Duchi et al. (2015). In this approach, t is a (virtual) global counter recording the number of *successful writes* to the shared memory x (incremented after step 4 in (2.5)); x_t thus represents the (true) content of the shared memory after t updates. The interpretation of the crucial equation (2.3) then means that $\hat{x}_t$ represents the (delayed) local copy value of the core that made the $(t+1)^{\text{th}}$ successful update; i_t represents the factor sampled by this core for this update. Notice that in this framework, the value of $\hat{x}_t$ and i_t is unknown at "time t"; we have to wait to the later time when the next core writes to memory to finally determine that its local variables are the ones labeled by t. We thus see that here $\hat{x}_t$ and i_t are not necessarily independent – they share dependence through the assignment of the t label. In particular, if some values of i_t yield faster updates than others, it will influence the label assignment defining $\hat{x}_t$. We provide a concrete example of this possible dependency in Figure 2-1.

The only way we can think to resolve this issue and ensure unbiasedness is to assume that the computation time for the algorithm running on a core is independent of the sample i chosen. This assumption seems overly strong in the context of potentially heterogeneous factors

5. Observe that contrary to most asynchronous algorithms, we choose to read the shared parameter vector *before* sampling the next data point. We made this design choice to emphasize that in order for $\hat{x}_t$ and i_t to be independent – which will prove crucial for the analysis – the reading of the shared parameter has to be independent of the sampled data point. Although in practice one would prefer to only read the necessary parameters *after* sampling the relevant data point, for the sake of the analysis we cannot allow this source of dependence. We note that our analysis could also handle reading the parameter first and then sampling as long as independence is ensured, but for clarity of presentation, we decided to make this independence explicit.

Mania et al. (2017) make the opposite presentation choice. In their main analysis, they explicitly assume that $\hat{x}_t$ and i_t are independent, although they explain that it is not the case in practical implementations. The authors then propose a scheme to handle the dependency directly in their appendix. However, this "fix" can only be applied in a restricted setup: only for the HOGWILD algorithm, with the assumption that the norm of the gradient is uniformly bounded. Furthermore, even in this restricted setup, the scheme leads to worsened theoretical results (the bound on τ is κ^2 worse). Applying it to a more complex algorithm such as KROMAGNON or ASAGA would mean overcoming several significant hurdles and is thus still an open problem.

For lack of a better option, we choose to enforce the independence of $\hat{x}_t$ and i_t with our modified steps ordering.

	f_1	f_2		f_1	f_2		f_1	f_2		f_1	f_2
core 1	$\times$		core 1	$\times$		core 1		$\times$	core 1		$\times$
core 2	$\times$		core 2		$\times$	core 2	$\times$		core 2		$\times$
$f'_{i_0}(\hat{x}_0)$	$f'_1(\hat{x}_0)$			$f'_1(\hat{x}_0)$			$f'_1(\hat{x}_0)$			$f'_2(\hat{x}_0)$	

Figure 2-1 – Suppose that we have two cores and that f has two factors: f_1 which has support on only one variable, and f_2 which has support on 10^6 variables and thus yields a gradient step that is significantly more expensive to compute. x_0 is the initial content of the memory, and we do not know yet whether $\hat{x}_0$ is the local copy read by the first core or the second core, but we are sure that $\hat{x}_0 = x_0$ as no update can occur in shared memory without incrementing the counter. There are four possibilities for the next step defining x_1 depending on which index i was sampled on each core. If any core samples $i = 1$, we know that $x_1 = x_0 - \gamma f'_1(x_0)$ as it will be the first (much faster update) to complete. This happens in 3 out of 4 possibilities; we thus have that $\mathbb{E}x_1 = x_0 - \gamma(\frac{3}{4}f'_1(x_0) + \frac{1}{4}f'_2(x_0))$. We see that this analysis scheme *does not* satisfy the crucial unbiasedness condition (2.4). To understand this subtle point better, note that in this very simple example, i_0 and i_1 are not independent. We can show that $P(i_1 = 2 \mid i_0 = 2) = 1$. They share dependency *through the labeling assignment*.

f_i's, and is thus a fundamental flaw for analyzing non-uniform asynchronous computation that has mostly been ignored in the recent asynchronous optimization literature. [6]

The "Before Read" Approach. Mania et al. (2017) address this issue by proposing instead to increment the global t counter just *before* a new core starts to *read* the shared memory (before step 1 in (2.5)). In their framework, $\hat{x}_t$ represents the (inconsistent) read that was made by this core in this computational block, and i_t represents the chosen sample. The update rule (2.3) represents a *definition* of the meaning of x_t, which is now a "virtual iterate" as it does not necessarily correspond to the content of the shared memory at any point. The real quantities manipulated by the algorithm in this approach are the $\hat{x}_t$'s, whereas x_t is used only for the analysis – consequently, the critical quantity we want to see vanish is $\mathbb{E}\|\hat{x}_t - x^*\|^2$. The independence of i_t with $\hat{x}_t$ can be simply enforced in this approach by making sure that the way the shared memory x is read does not depend on i_t (e.g. by reading all its coordinates in a fixed order). Note that this implies that we have to read all of x's coordinates, regardless of the size of f_{i_t}'s support. This is a much weaker condition than the assumption that all the computation in a block does not depend on i_t as required by the

6. We note that Bertsekas and Tsitsiklis (1989) briefly discussed this issue (see Section 7.8.3), stressing that their analysis for SGD required that the scheduling of computation was independent from the randomness from SGD, but they did not offer any solution if this assumption was not satisfied. Both the "before read" labeling from Mania et al. (2017) and our proposed "after read" labeling resolve this issue.

"after write" approach, and is thus more reasonable.

A New Global Ordering: the "After Read" Approach. The "before read" approach gives rise to the following complication in the analysis: $\hat{x}_t$ can depend on i_r for $r > t$. This is because t is a global time ordering only on the assignment of computation to a core, not on when $\hat{x}_t$ was finished being read. This means that we need to consider both the "future" and the "past" when analyzing x_t.[7] To simplify the analysis, we thus propose a third way to label the iterates that we call "after read": $\hat{x}_t$ represents the $(t+1)^{\text{th}}$ *fully completed read* (t incremented after step 1 in (2.5)). As in the "before read" approach, we can ensure that i_t is independent of $\hat{x}_t$ by ensuring that how we read does not depend on i_t. But unlike in the "before read" approach, t here now does represent a global ordering on the $\hat{x}_t$ iterates – and thus we have that i_r is independent of $\hat{x}_t$ for $r > t$. Again using (2.3) as the definition of the virtual iterate x_t as in the perturbed iterate framework, we then have a very simple form for the value of x_t and $\hat{x}_t$ (assuming atomic writes, see Property 4 below):

$$\begin{aligned} x_t &= x_0 - \gamma \sum_{u=0}^{t-1} g(\hat{x}_u, i_u)\,; \\ [\hat{x}_t]_v &= [x_0]_v - \gamma \sum_{\substack{u=0 \\ \text{u s.t. coordinate } v \text{ was written} \\ \text{for } u \text{ before } t}}^{t-1} [g(\hat{x}_u, i_u)]_v\,. \end{aligned} \tag{2.6}$$

This improvement proved crucial to obtain better bounds for HOGWILD, and even more so to conduct the convergence proofs for the more complex algorithms introduced in Chapters 3 and 4.

The main idea of the perturbed iterate framework is to use this handle on $\hat{x}_t - x_t$ to analyze the convergence for x_t. As x_t is a virtual quantity, Mania et al. (2017) supposed that there exists an index T such that x_T lives in shared memory (T is a pre-set final iteration number after which all computation is completed, which means $x_T = \hat{x}_T$) and gave their convergence result for this x_T.

In this chapter (and indeed in the rest of this thesis), we instead state the convergence results directly in terms of $\hat{x}_t$, thus avoiding the need for an unwieldy pre-set final iteration counter, and also enabling guarantees during the entire course of the algorithm.

Remark 1. *There is another subtle difference between the "before read" and "after read" approaches. While in* (2.5)*, we have opted to read the parameters first and then sample,*

7. Note that this is also the case when using the "after write" framework and inconsistent writes, since theoretically one core can start writing updates and not finish before another one performs a full iteration. Again, the iterates of the second core $\hat{x}_t$ depend on the iterates of the first core $\hat{x}_r$, with $r > t$.

Algorithm 1 HOGWILD (analysis)

Initialize shared variable x
keep doing in parallel
  $\hat{x} =$ inconsistent read of x
  Sample i uniformly in $\{1, ..., n\}$
  Let S_i be f_i's support
  $[\delta x]_{S_i} := -\gamma f_i'(\hat{x})$
  for v **in** S_i **do**
    $[x]_v \leftarrow [x]_v + [\delta x]_v$ // atomic
  end for
end parallel loop

Algorithm 2 HOGWILD (implementation)

Initialize shared x
keep doing in parallel
  Sample i uniformly in $\{1, ..., n\}$
  Let S_i be f_i's support
  $[\hat{x}]_{S_i} =$ inconsistent read of x on S_i
  $[\delta x]_{S_i} = -\gamma(f_i'([\hat{x}]_{S_i}))$
  for v **in** S_i **do**
    $[x]_v \leftarrow [x]_v + [\delta x]_v$ // atomic
  end for
end parallel loop

Mania et al. (2017) do the opposite: first sample and then read. This enables them to read only the relevant subset of the parameters that are needed for the current iteration – although as previously mentioned this setup violates the independence assumption between $\hat{x}_t$ and i_t.

We can thus consider that their approach is "after sampling" rather than "before read". If we take this view, then to obtain something equivalent to our "after read" approach we have to switch the order of the reading and sampling operations.

2.3 HOGWILD analysis

In order to show that our improved "after read" perturbed iterate framework can be used to revisit the analysis of other optimization routines with correct proofs that do not assume homogeneous computation, we now provide the analysis of the HOGWILD algorithm (i.e. asynchronous parallel constant step size SGD) first introduced in Niu et al. (2011).

We start by describing HOGWILD in Algorithm 1. We then describe a few relevant properties of the algorithm, and finally give our theoretical convergence and speedup results and their proofs. Note that our framework allows us to easily remove the classical bounded gradient assumption, which is used in one form or another in most of the literature (Niu et al., 2011; De Sa et al., 2015; Mania et al., 2017) – although it is inconsistent with strong convexity in the unconstrained regime. This allows for better bounds where the uniform bound on $\|f_i'(x)\|^2$ is replaced by its variance at the optimum.

2.3.1 Useful properties and assumptions

Before stating our convergence result, we highlight some properties of Algorithm 1 and make one central assumption.

Property 2 (independence)**.** *Given the "after read" global ordering, i_r is independent of $\hat{x}_t$ $\forall r \geq t$.*

The independence property for $r = t$ is assumed in most of the parallel optimization literature, even though it is not verified in case the "after write" labeling is used. We emulate Mania et al. (2017) and *enforce* this independence in Algorithm 1 by having the core read all the shared data parameters and historical gradients before starting their iterations. Although this is too expensive to be practical if the data is sparse, this is required by the theoretical Algorithm 1 that we can analyze. The independence for $r > t$ is a consequence of using the "after read" global ordering instead of the "before read" one.

Property 3 (unbiased estimator)**.** *The update, $g_t := g(\hat{x}_t, i_t)$, is an unbiased estimator of the true gradient at $\hat{x}_t$ (i.e. (2.3) yields (2.4) in conditional expectation).*

This property is crucial for the analysis, as in most related literature. It follows by the independence of i_t with $\hat{x}_t$.

Property 4 (atomicity)**.** *The shared parameter coordinate update of $[x]_v$ on line 11 is atomic.*

Since our updates are additions, there are no overwrites, even when several cores compete for the same resources. In practice, this is enforced by using *compare-and-swap* semantics, which are heavily optimized at the processor level and have minimal overhead. Our experiments with non-thread safe algorithms (i.e. where this property is not verified, see Figure B-1 of Appendix B.5) show that compare-and-swap is necessary to optimize to high accuracy.

Finally, as is standard in the literature, we make an assumption on the maximum delay that asynchrony can cause – this is the *partially asynchronous* setting as defined in Bertsekas and Tsitsiklis (1989):

Assumption 5 (bounded overlaps)**.** *We assume that there exists a uniform bound, called τ, on the maximum number of iterations that can overlap together. We say that iterations r and t overlap if at some point they are processed concurrently. One iteration is being processed from the start of the reading of the shared parameters to the end of the writing of its update. The bound τ means that iterations r cannot overlap with iteration t for $r \geq t + \tau + 1$, and*

thus that every coordinate update from iteration t is successfully written to memory before the iteration $t + \tau + 1$ starts.

Our result will give us conditions on τ subject to which we have linear speedups. τ is usually seen as a proxy for p, the number of cores (which lowerbounds it). However, though τ appears to depend linearly on p, it actually depends on several other factors (notably the data sparsity distribution) and can be orders of magnitude bigger than p in real-life experiments. We can upper bound τ by $(p-1)R$, where R is the ratio of the maximum over the minimum iteration time (which encompasses theoretical aspects as well as hardware overhead). More details can be found in Section 3.4.7.

Explicit effect of asynchrony. By using the overlap Assumption 5 in the expression (2.6) for the iterates, we obtain the following explicit effect of asynchrony that is crucially used in our proof:

$$\hat{x}_t - x_t = \gamma \sum_{u=(t-\tau)_+}^{t-1} G_u^t g(\hat{x}_u, i_u), \tag{2.7}$$

where G_u^t are $d \times d$ diagonal matrices with terms in $\{0, +1\}$. From our definition of t and x_t, it is clear that every update in $\hat{x}_t$ is already in x_t – this is the 0 case. Conversely, some updates might be late: this is the $+1$ case. $\hat{x}_t$ may be lacking some updates from the "past" in some sense, whereas given our global ordering definition, it cannot contain updates from the "future".

2.3.2 Convergence and speedup results

We now state the theoretical results of our analysis of HOGWILD with inconsistent reads and writes in the "after read framework". We give an outline of the proof in Section 2.3.3 and its full details in Appendix A.

We start with two useful definitions of quantities that will appear in our results.

Definition 6. *Let $\sigma^2 = \mathbf{E}\|f_i'(x^*)\|^2$ be the variance of the gradient estimator at the optimum.*

Definition 7 (Sparsity)**.** *As in Niu et al. (2011), we introduce $\Delta_r := \max_{v=1..d} |\{i : v \in S_i\}|$. Δ_r is the maximum right-degree in the bipartite graph of the factors and the dimensions, i.e., the maximum number of data points with a specific feature. For succinctness, we also define $\Delta := \Delta_r/n$. We have $1 \leq \Delta_r \leq n$, and hence $1/n \leq \Delta \leq 1$.*

For reference, we start by giving the rate of convergence of serial SGD (see e.g. (Schmidt, 2014)).

Theorem 8 (Convergence guarantee and rate of SGD)**.** *Let* $a \leq \frac{1}{2}$*. Then for any step size* $\gamma = \frac{a}{L}$*,* SGD *converges in expectation to* b*-accuracy at a geometric rate of at least:* $\rho(a) = a/\kappa$*, i.e.,* $\mathbb{E}\|x_t - x^*\|^2 \leq (1-\rho)^t \|x_0 - x^*\|^2 + b$*, where* $b = 2\frac{\gamma\sigma^2}{\mu}$*.*

As SGD only converges linearly up to a ball around the optimum, to make sure we reach ϵ-accuracy, it is necessary that $\frac{2\gamma\sigma^2}{\mu} \leq \epsilon$, i.e. $\gamma \leq \frac{\epsilon\mu}{2\sigma^2}$. All told, in order to get linear convergence to ϵ-accuracy, serial SGD requires $\gamma \leq \min\left\{\frac{1}{2L}, \frac{\epsilon\mu}{2\sigma^2}\right\}$. The proof can be found in Appendix A.5.

Theorem 9 (Convergence guarantee and rate of HOGWILD)**.** *Let*

$$a^*(\tau) := \frac{1}{5\left(1 + 2\tau\sqrt{\Delta}\right)\xi(\kappa,\Delta,\tau)} \qquad \text{where } \xi(\kappa,\Delta,\tau) := \sqrt{1 + \frac{1}{2\kappa}\min\{\frac{1}{\sqrt{\Delta}}, \tau\}}$$
$$\text{(note that } \xi(\kappa,\Delta,\tau) \approx 1 \text{ unless } \kappa < 1/\sqrt{\Delta}\ (\leq \sqrt{n})\text{).} \tag{2.8}$$

For any step size $\gamma = \frac{a}{L}$ *with* $a \leq \min\left\{a^*(\tau), \frac{\kappa}{\tau}\right\}$*, the inconsistent read iterates of Algorithm 1 converge in expectation to* b*-accuracy at a geometric rate of at least:* $\rho(a) = a/\kappa$*, i.e.,* $\mathbb{E}\|\hat{x}_t - x^*\|^2 \leq (1-\rho)^t(2\|x_0 - x^*\|^2) + b$*, where* $b = (\frac{8\gamma(C_1 + \tau C_2)}{\mu} + 4\gamma^2 C_1 \tau)\sigma^2$*,* $C_1 = 1 + \sqrt{\Delta}\tau$ *and* $C_2 = \sqrt{\Delta} + \gamma\mu C_1$*.* [8]

This result is quite close to the one obtained for serial SGD. Note that we recover this exact condition (up to a small constant factor) if we set $\tau = 0$, i.e. if we force our asynchronous algorithm to be serial.

The condition $a \leq \frac{\kappa}{\tau}$ is equivalent to $\gamma\mu\tau \leq 1$ and should be thought of as a condition on τ. We will see that it is always verified in the regime we are interested in, that is the linear speed-up regime (where more stringent conditions are imposed on τ).

We now investigate the conditions under which HOGWILD is linearly faster than SGD. Note that to derive these conditions we need not only compare their respective convergence rates, but also the size of the ball around the optimum to which both algorithms converge. These quantities are provided in Theorems 8 and 9.

Corollary 10 (Speedup condition)**.** *Suppose* $\tau = \mathcal{O}(\min\{\frac{1}{\sqrt{\Delta}}, \kappa\})$*. Then for any step size* $\gamma \leq \frac{a^*(\tau)}{L} = \mathcal{O}(\frac{1}{L})$ *(i.e., any allowable step size for* SGD*),* HOGWILD *converges*

8. Note that C_2 depends on γ. In the rest of this chapter, we write $C_2(\gamma)$ instead of C_2 when we want to draw attention to that dependency.

geometrically to a ball of radius $r_h = \mathcal{O}(\frac{\gamma\sigma^2}{\mu})$ *with rate factor* $\rho = \frac{\gamma\mu}{2}$ *(similar to* SGD*), and is thus linearly faster than its sequential counterpart up to a constant factor.*

Moreover, a universal step size of $\Theta(\frac{1}{L})$ *can be used for* HOGWILD *to be adaptive to local strong convexity with a similar rate to* SGD *(i.e., knowledge of* κ *is not required).*

If $\gamma = \mathcal{O}(1/L)$, HOGWILD obtains the same convergence rate as SGD and converges to a ball of equivalent radius. Since the maximum step size guaranteeing linear convergence for SGD is also $\mathcal{O}(1/L)$, HOGWILD is linearly faster than SGD for any reasonable step size – under the condition that $\tau = \mathcal{O}(\min\{\frac{1}{\sqrt{\Delta}}, \kappa\})$. We also remark that since $\gamma \leq 1/L$ and $\tau \leq \kappa$, we have $\gamma\mu\tau \leq 1$, which means the condition $a \leq \frac{\kappa}{\tau}$ is superseded by $a \leq a^*(\tau)$ in Theorem 9.

Function values results. Our results are derived directly on iterates, that is, we bound the distance between $\hat{x}_t$ and x^*. We can easily obtain results on function values to bound $\mathbb{E}f(\hat{x}_t) - f(x^*)$ by adapting the classical smoothness inequality:[9] $\mathbb{E}f(x_t) - f(x^*) \leq \frac{L}{2}\mathbb{E}\|x_t - x^*\|^2$ to the asynchronous parallel setting.

Convergence to ϵ-accuracy. As noted in Mania et al. (2017), for our algorithm to converge to ϵ-accuracy for some $\epsilon > 0$, we require an additional bound on the step size to make sure that the radius of the ball to which we converge is small enough. For SGD, this means using a step size $\gamma = \mathcal{O}(\frac{\epsilon\mu}{\sigma^2})$ (see Appendix A.5). We can also prove that under the conditions that $\tau = \mathcal{O}(\frac{1}{\sqrt{\Delta}})$ and $\gamma\mu\tau \leq 1$, HOGWILD requires the same bound on the step size to converge to ϵ-accuracy (see Appendix A.6).

If ϵ is small enough, the active upper bound on the step size is $\gamma = \mathcal{O}(\frac{\epsilon\mu}{\sigma^2})$ for both algorithms. In this regime, we obtain a relaxed condition on τ for a linear speedup. The condition $\tau \leq \kappa$ which came from comparing maximum allowable step sizes is removed. Instead, we enforce $\gamma\mu\tau \leq 1$, which gives us the weaker condition $\tau = \mathcal{O}(\frac{\sigma^2}{\epsilon\mu^2})$. Our condition on the overlap is then: $\tau = \mathcal{O}(\min\{\frac{1}{\sqrt{\Delta}}, \frac{\sigma^2}{\epsilon\mu^2}\})$. We see that this is similar to the condition obtained by Mania et al. (2017, Theorem 4) in their HOGWILD analysis, although we have the variance at the optimum σ^2 instead of a squared global bound on the gradient.

Comparison to related work.

- We give the first convergence analysis for HOGWILD with no assumption on a global bound on the gradient (M). This allows us to replace the usual dependence in M^2 by

9. See e.g. Moulines and Bach (2011).

a term in σ^2 which is potentially significantly smaller. This means improved upper bounds on the step size and the allowed overlap.

- We obtain the same condition on the step size for linear convergence to ϵ-accuracy of HOGWILD as previous analysis for serial SGD (e.g. Needell et al. (2014)) – given $\tau \leq 1/\gamma\mu$.
- In contrast to the HOGWILD analysis from Niu et al. (2011); De Sa et al. (2015), our proof technique handles inconsistent reads and a non-uniform processing speed across f_i's. Further, Corollary 10 gives a better dependence on the sparsity than in Niu et al. (2011), where $\tau \leq \mathcal{O}(\Delta^{-1/4})$, and does not require various bounds on the gradient assumptions.
- In contrast to the HOGWILD analysis from Mania et al. (2017, Thm. 3), removing their gradient bound assumption enables us to get a (potentially) significantly better upper bound condition on τ for a linear speedup. We also give our convergence guarantee on $\hat{x}_t$ *during* the algorithm, whereas they only bound the error for the "last" iterate x_T.

2.3.3 Proof outlines

We give here an extended outline of the proof. We detail key lemmas in Section 2.3.4.

Initial recursive inequality. Let $g_t := g(\hat{x}_t, i_t)$. By expanding the update equation (2.3) defining the virtual iterate x_{t+1} and introducing $\hat{x}_t$ in the inner product term, we obtain:

$$\begin{aligned}\|x_{t+1} - x^*\|^2 &= \|x_t - \gamma g_t - x^*\|^2 \\ &= \|x_t - x^*\|^2 + \gamma^2\|g_t\|^2 - 2\gamma\langle x_t - x^*, g_t\rangle \\ &= \|x_t - x^*\|^2 + \gamma^2\|g_t\|^2 - 2\gamma\langle \hat{x}_t - x^*, g_t\rangle + 2\gamma\langle \hat{x}_t - x_t, g_t\rangle\,. \end{aligned} \tag{2.9}$$

Note that we introduce $\hat{x}_t$ in the inner product because g_t is a function of $\hat{x}_t$, not x_t.

In the sequential setting, we require i_t to be independent of x_t to obtain unbiasedness. In the perturbed iterate framework, we instead require that i_t is independent of $\hat{x}_t$ (see Property 2). This crucial property enables us to use the unbiasedness condition (2.4) to write: $\mathbb{E}\langle \hat{x}_t - x^*, g_t\rangle = \mathbb{E}\langle \hat{x}_t - x^*, f'(\hat{x}_t)\rangle$. Taking the expectation of (2.9) and using this unbiasedness condition we obtain an expression that allows us to use the μ-strong convexity

of f: [10]

$$\langle \hat{x}_t - x^*, f'(\hat{x}_t) \rangle \geq f(\hat{x}_t) - f(x^*) + \frac{\mu}{2} \|\hat{x}_t - x^*\|^2. \tag{2.10}$$

With further manipulations on the expectation of (2.9), including the use of the standard inequality $\|a+b\|^2 \leq 2\|a\|^2 + 2\|b\|^2$ (see Appendix A.1), we obtain our basic recursive contraction inequality:

$$a_{t+1} \leq (1 - \frac{\gamma\mu}{2}) a_t + \gamma^2 \mathbb{E}\|g_t\|^2 \underbrace{+\gamma\mu \mathbb{E}\|\hat{x}_t - x_t\|^2 + 2\gamma \mathbb{E}\langle \hat{x}_t - x_t, g_t \rangle}_{\text{additional asynchrony terms}} - 2\gamma e_t \,, \tag{2.11}$$

where $a_t := \mathbb{E}\|x_t - x^*\|^2$ and $e_t := \mathbb{E} f(\hat{x}_t) - f(x^*)$.

Inequality (2.11) is a midway point between the one derived in the proof of Lemma 1 in Hofmann et al. (2015) and Equation (2.5) in Mania et al. (2017), because we use the tighter strong convexity bound (2.10) than in the latter (giving us the important extra term $-2\gamma e_t$).

In the sequential setting, one crucially uses the negative suboptimality term $-2\gamma e_t$ to cancel the variance term $\gamma^2 \mathbb{E}\|g_t\|^2$ (thus deriving a condition on γ). In our setting, we need to bound the additional asynchrony terms using the same negative suboptimality in order to prove convergence and speedup for our parallel algorithm – this will give stronger constraints on the maximum step size.

The rest of the proof is as follows:

1. By using the expansion (2.7) for $\hat{x}_t - x_t$, we can bound the additional asynchrony terms in (2.11) in terms of the past updates $(\mathbb{E}\|g_u\|^2)_{u \leq t}$. This gives Lemma 11 below.
2. We then bound the updates $\mathbb{E}\|g_t\|^2$ in terms of the suboptimality e_t and variance term σ (see Lemma 14 below).
3. By substituting Lemma 14 into Lemma 11, we get a master contraction inequality (2.22) in terms of a_{t+1}, a_t, $(e_u)_{u \leq t}$ and σ^2.
4. We then unroll this master inequality and cleverly regroup terms to obtain a contraction inequality (2.23) between a_t, a_0 and σ^2.
5. By using that $\|\hat{x}_t - x^*\|^2 \leq 2a_t + 2\|\hat{x}_t - x_t\|^2$, we obtain a contraction inequality directly on the "real" iterates (as opposed to the "virtual" iterates as in Mania et al. (2017)), subject to a maximum step size condition on γ. This finishes the proof for

10. Note that here is our departure point with Mania et al. (2017) who replaced the $f(\hat{x}_t) - f(x^*)$ term with the lower bound $\frac{\mu}{2}\|\hat{x}_t - x^*\|^2$ in this relationship (see their Equation (2.4)), thus yielding an inequality too loose afterwards to get the fast rates for SVRG.

Theorem 9.

6. Finally, we only have to derive the conditions on γ and τ under which HOGWILD converges with a similar convergence rate to a ball with a similar radius than serial SGD to finish the proof for Corollary 10.

2.3.4 Key Lemmas

We list the key lemmas below with their proof sketch, and pointers to the relevant parts of Appendix A for detailed proofs.

Lemma 11 (Inequality in terms of $g_t := g(\hat{x}_t, i_t)$)**.** *For all $t \geq 0$:*

$$a_{t+1} \leq (1 - \frac{\gamma\mu}{2})a_t + \gamma^2 C_1 \mathbb{E}\|g_t\|^2 + \gamma^2 C_2 \sum_{u=(t-\tau)_+}^{t-1} \mathbb{E}\|g_u\|^2 - 2\gamma e_t \,, \tag{2.12}$$

$$\textit{where} \quad C_1 := 1 + \sqrt{\Delta}\tau \quad \textit{and} \quad C_2 := \sqrt{\Delta} + \gamma\mu C_1 \,. \text{[11]} \tag{2.13}$$

To prove this lemma we need to bound both $\mathbb{E}\|\hat{x}_t - x^*\|^2$ and $\mathbb{E}\langle \hat{x}_t - x_t, g_t\rangle$ with respect to $(\mathbb{E}\|g_u\|^2)_{u\leq t}$. We achieve this by crucially using Equation (2.7), together with the following proposition, which we derive by a combination of Cauchy-Schwartz and our sparsity definition (see Section A.2).

Proposition 12. *For any $u \neq t$,*

$$\mathbb{E}|\langle g_u, g_t\rangle| \leq \frac{\sqrt{\Delta}}{2}(\mathbb{E}\|g_u\|^2 + \mathbb{E}\|g_t\|^2) \,. \tag{2.14}$$

To derive this essential inequality for both the right-hand-side terms of Eq. (2.11), we start by proving a relevant property of Δ. We reuse the sparsity constant introduced in Reddi et al. (2015) and relate it to the one we have defined earlier, Δ_r:

Remark 13. *Let D be the smallest constant such that:*

$$\mathbf{E}\|x\|_i^2 = \frac{1}{n}\sum_{i=1}^{n} \|x\|_i^2 \leq D\|x\|^2 \quad \forall x \in \mathbb{R}^d, \tag{2.15}$$

where $\|.\|_i$ is defined to be the ℓ_2-norm restricted to the support S_i of f_i. We have:

$$D = \frac{\Delta_r}{n} = \Delta \,. \tag{2.16}$$

11. Note that C_2 depends on γ. In the rest of this thesis, we write $C_2(\gamma)$ instead of C_2 when we want to draw attention to that dependency.

Proof. We have:

$$\mathbf{E}\|x\|_i^2 = \frac{1}{n}\sum_{i=1}^{n}\|x\|_i^2 = \frac{1}{n}\sum_{i=1}^{n}\sum_{v\in S_i}[x]_v^2 = \frac{1}{n}\sum_{v=1}^{d}\sum_{i|v\in S_i}[x]_v^2 = \frac{1}{n}\sum_{v=1}^{d}\delta_v[x]_v^2\,, \tag{2.17}$$

where $\delta_v := |(i \mid v \in S_i)|$. This implies:

$$D \geq \frac{1}{n}\sum_{v=1}^{d}\delta_v\frac{[x]_v^2}{\|x\|^2}\,. \tag{2.18}$$

Since D is the minimum constant satisfying this inequality, we have:

$$D = \max_{x\in\mathbb{R}^d}\frac{1}{n}\sum_{v=1}^{d}\delta_v\frac{[x]_v^2}{\|x\|^2}\,. \tag{2.19}$$

We need to find x such that it maximizes the right-hand side term. Note that the vector $([x]_v^2/\|x\|^2)_{v=1..d}$ is in the unit probability simplex, which means that an equivalent problem is the maximization over all convex combinations of $(\delta_v)_{v=1..d}$. This maximum is found by putting all the weight on the maximum δ_v, which is Δ_r by definition.

This implies that $\Delta = \Delta_r/n$ is indeed the smallest constant satisfying (2.15). □

Proof of Proposition 12 Let $u \neq t$. Without loss of generality, $u < t$. [12] Then:

$$\begin{aligned}
\mathbb{E}|\langle g_u, g_t\rangle| &\leq \mathbb{E}\|g_u\|_{i_t}\|g_t\| && \text{(Sparse inner product; support of } g_t \text{ is } S_{i_t})\\
&\leq \sqrt{\mathbb{E}\|g_u\|_{i_t}^2}\sqrt{\mathbb{E}\|g_t\|^2} && \text{(Cauchy-Schwarz for expectations)}\\
&\leq \sqrt{\Delta\mathbb{E}\|g_u\|^2}\sqrt{\mathbb{E}\|g_t\|^2} && \text{(Remark 13 and } i_t \perp\!\!\!\perp g_u, \forall u < t)\\
&\leq \frac{\sqrt{\Delta}}{2}(\mathbb{E}\|g_u\|^2 + \mathbb{E}\|g_t\|^2)\,. && \text{(AM-GM inequality)}
\end{aligned}$$

All told, we have:

$$\mathbb{E}|\langle g_u, g_t\rangle| \leq \frac{\sqrt{\Delta}}{2}(\mathbb{E}\|g_u\|^2 + \mathbb{E}\|g_t\|^2)\,. \tag{2.20}$$

Lemma 14 (Suboptimality bound on $\mathbb{E}\|g_t\|^2$)**.** *For all* $t \geq 0$,

$$\mathbb{E}\|g_t\|^2 \leq 4Le_t + 2\sigma^2\,. \tag{2.21}$$

12. One only has to switch u and t if $u > t$.

We give the proof in Appendix A.3. Contrary to what will be the case for variance-reduced methods (see Lemmas 20 and 26), the second term does not vanish as t grows. This reflects the fact that constant step size SGD does not converge to the optimum but rather to a ball around it. However, this simpler form allows us to simply unroll the resulting master inequality to get our convergence result.

Master inequality. We plug (2.21) in Lemma 11, which gives us that (see Appendix A.4):

$$a_{t+1} \leq (1 - \frac{\gamma\mu}{2})a_t + (4L\gamma^2 C_1 - 2\gamma)e_t + 4L\gamma^2 C_2 \sum_{u=(t-\tau)_+}^{t-1} e_u + 2\gamma^2\sigma^2(C_1 + \tau C_2)\,. \quad (2.22)$$

Contraction inequality on x_t. As the term in σ^2 does not vanish, we simply unroll Equation (2.22) all the way to $t = 0$. This gives us (see Appendix A.4):

$$a_{t+1} \leq (1 - \frac{\gamma\mu}{2})^{t+1}a_0 + (4L\gamma^2 C_1 + 8L\gamma^2\tau C_2 - 2\gamma)\sum_{u=0}^{t}(1 - \frac{\gamma\mu}{2})^{t-u}e_u + \frac{4\gamma\sigma^2}{\mu}(C_1 + \tau C_2)\,. \quad (2.23)$$

Contraction inequality on $\hat{x}_t$. We now use that $\|\hat{x}_t - x^*\|^2 \leq 2a_t + 2\|\hat{x}_t - x_t\|^2$ together with our previous bound (A.5). Together with (2.23), we get (see Appendix A.4):

$$\mathbb{E}\|\hat{x}_t - x^*\|^2 \leq (1 - \frac{\gamma\mu}{2})^{t+1}2a_0 + (24L\gamma^2 C_1 + 16L\gamma^2\tau C_2 - 4\gamma)\sum_{u=0}^{t}(1 - \frac{\gamma\mu}{2})^{t-u}e_u + \Big(\frac{8\gamma(C_1 + \tau C_2)}{\mu} + 4\gamma^2 C_1\tau\Big)\sigma^2\,. \quad (2.24)$$

To get our final contraction inequality, we need to safely remove all the e_u terms, so we enforce $16L\gamma^2 C_1 + 16L\gamma^2\tau C_2 - 4\gamma \leq 0$. This leads directly to Theorem 9.

Convergence rate and ball-size comparison. To prove Corollary 10, we simply show that under the condition $\tau = \mathcal{O}(\min\{\frac{1}{\sqrt{\Delta}}, \kappa\})$, the biggest allowable step size for HOGWILD to converge linearly is $\mathcal{O}(1/L)$, as is also the case for SGD; and that the size of the ball to which both algorithms converge is of the same order. The proof is finished by remarking that for both algorithms, the rates of convergence are directly proportional to the step size.

Chapter 3

Asynchronous parallel variance reduction

In this chapter, we introduce ASAGA, a novel asynchronous parallel version of the incremental gradient algorithm SAGA that enjoys fast linear convergence rates. Using the "after read" framework of Chapter 2, we perform the convergence analysis of both this new method and KROMAGNON, an asynchronous SVRG variant presented by Mania et al. (2017). We are able to both remove problematic assumptions and obtain better theoretical results. Notably, we prove that ASAGA and KROMAGNON can obtain a theoretical linear speedup on multi-core systems even without sparsity assumptions.

We present results of an implementation on a 40-core architecture illustrating the practical speedups as well as the hardware overhead.

Finally, we investigate the overlap constant, an ill-understood but central quantity for the theoretical analysis of asynchronous parallel algorithms. We find that it encompasses much more complexity than suggested in previous work, and often is order-of-magnitude bigger than traditionally thought.

3.1 Introduction

In Sections 1.2.1 and 1.2.2, we have discussed the limitations of the stochastic gradient descent algorithm (SGD) in terms of convergence speed, as well as the rise of variance-reduced randomized incremental algorithms solving (2.1) with a fast [1] linear convergence rate. Examples include SAG (Le Roux et al., 2012), SDCA (Shalev-Shwartz and Zhang,

1. Their complexity in terms of gradient evaluations to reach an accuracy of ϵ is $O((n+\kappa)\log(1/\epsilon))$, in contrast to $O(n\kappa_b \log(1/\epsilon))$ for batch gradient descent in the worst case.

2013), SVRG (Johnson and Zhang, 2013) and SAGA (Defazio et al., 2014a). These algorithms can be interpreted as variance-reduced versions of SGD, and they have demonstrated both theoretical and practical improvements over SGD (for the *finite sum* optimization problem (2.1)).

Despite impressive achievements, these initial variance-reduced algorithms are inherently sequential methods implemented as single CPU core procedures. In order to take advantage of the multi-core architecture of modern computers, the aforementioned algorithms need to be adapted to the asynchronous parallel setting, where multiple threads work concurrently.

Much work has been devoted recently in proposing and analyzing asynchronous parallel variants of algorithms such as SGD (Niu et al., 2011), SDCA (Hsieh et al., 2015) and SVRG (Reddi et al., 2015; Mania et al., 2017; Zhao and Li, 2016). Among the incremental gradient algorithms with fast linear convergence rates that can optimize (2.1) in its general form, only SVRG had had an asynchronous parallel version proposed. [2] No such adaptation had been attempted for SAGA until Leblond et al. (2017), even though one could argue that it is a more natural candidate as, contrarily to SVRG, it is not epoch-based and thus has no synchronization barriers at all. This chapter is based on Leblond et al. (2018b), an extended journal version of the conference paper from Leblond et al. (2017).

The analysis of ASAGA provides us with a good opportunity to showcase the "after read" framework introduced in Chapter 2. Indeed, it allows us to obtain state-of-the-art convergence rates for KROMAGNON, contrary to the paper which first introduced this algorithm (Mania et al., 2017).

Contributions. In Section 3.2.1, we present a novel sparse variant of SAGA that is more adapted to the parallel setting than the original SAGA algorithm. In Section 3.2.2, we present ASAGA, a lock-free asynchronous parallel version of Sparse SAGA that does not require consistent read or write operations. We give a tailored convergence analysis for ASAGA. Our main result states that ASAGA obtains the same geometric convergence rate per update as SAGA when the overlap bound τ (which scales with the number of cores) satisfies $\tau \leq \mathcal{O}(n)$ and $\tau \leq \mathcal{O}(\frac{1}{\sqrt{\Delta}} \max\{1, \frac{n}{\kappa}\})$, where $\Delta \leq 1$ is a measure of the sparsity of the problem. This notably implies that a linear speedup is theoretically possible even without sparsity in the well-conditioned regime where $n \gg \kappa$. This result is in contrast to previous analysis which always required some sparsity assumptions.

In Section 3.3, we revisit the asynchronous variant of SVRG from Mania et al. (2017),

2. We note that SDCA requires the knowledge of an explicit μ-strongly convex regularizer in (2.1), whereas SAG / SAGA are adaptive to any local strong convexity of f (Schmidt et al., 2016; Defazio et al., 2014a). The variant of SVRG from Hofmann et al. (2015) is also adaptive (we review this variant in Section 3.3.1).

KROMAGNON, while removing their gradient bound assumption (which was inconsistent with the strongly convex setting). [3] We prove that the algorithm enjoys the same fast rates of convergence as SVRG under similar conditions as ASAGA – whereas the original paper only provided analysis for slower rates (in both the sequential and the asynchronous case), and thus less meaningful speedup results.

In Section 3.4, we provide a practical implementation of ASAGA and illustrate its performance on a 40-core architecture, showing improvements compared to asynchronous variants of SVRG and SGD. We also present experiments on the overlap bound τ, showing that it encompasses much more complexity than suggested in previous work.

Related Work. As we have detailed in Section 1.2.3, in the recent past a few asynchronous variance-reduced algorithms have been introduced, including asynchronous variants of SDCA (Hsieh et al., 2015) and SVRG (Reddi et al., 2015; Zhao and Li, 2016). These papers make use of an unbiased gradient assumption that is not consistent with the proof technique, and thus suffers from technical problems.

The analysis of the KROMAGNON (asynchronous sparse SVRG) presented by Mania et al. (2017) is to the best of our knowledge the only one that does not suffer from this problem. However the "perturbed iterate" framework that they introduce entails complex proofs techniques, ultimately resulting in below state-of-the-art bounds under problematic assumptions. In particular, the authors assumed that f was both strongly convex and had a bound on the gradient, two *inconsistent* assumptions in the unconstrained setting that they analyzed. We overcome these difficulties by using tighter inequalities that remove the requirement of a bound on the gradient, and by using our more convenient way to label the iterates proposed in Section 2.2.2. The sparse version of SAGA that we propose is also inspired from the sparse version of SVRG proposed by Mania et al. (2017).

Reddi et al. (2015) presents a hybrid algorithm called HSAG that includes SAGA and SVRG as special cases. Their asynchronous analysis is epoch-based though, and thus does not handle a fully asynchronous version of SAGA as we do. Moreover, they require consistent reads and do not propose an efficient sparse implementation for SAGA, in contrast to ASAGA.

Pan et al. (2016) proposes a black box mini-batch algorithm to parallelize SGD-like methods while maintaining serial equivalence through smart update partitioning. When the

3. Although the authors mention that this gradient bound assumption can be enforced through the use of a thresholding operator, they do not explain how to handle the interplay between this non-linear operator and the asynchrony of the algorithm. Their theoretical analysis relies on the linearity of the operations (e.g. to derive (Mania et al., 2017, Eq. (2.6))), and thus this claim is not currently supported by theory (note that a strongly convex function over an unbounded domain always has unbounded gradients).

dataset is sparse enough, they obtain speedups over "HOGWILD" implementations of SVRG and SAGA. [4] However, these "HOGWILD" implementations appear to be quite suboptimal, as they do not leverage dataset sparsity efficiently: they try to adapt the "lazy updates" trick from Schmidt et al. (2016) to the asynchronous parallel setting – which as discussed in Appendix B.4 is extremely difficult – and end up making several approximations which severely penalize the performance of the algorithms. In particular, they have to use much smaller step sizes than in the sequential version, which makes for worse results.

3.2 Asynchronous Parallel Sparse SAGA

We start by presenting Sparse SAGA, a sparse variant of the SAGA algorithm that is more adapted to the asynchronous parallel setting. We then introduce ASAGA, the asynchronous parallel version of Sparse SAGA. Finally, we state both convergence and speedup results for ASAGA and give an outline of their proofs.

3.2.1 Sparse SAGA

Borrowing our notation from Hofmann et al. (2015), we first present the original SAGA algorithm and then describe our novel sparse variant.

Original SAGA Algorithm. The standard SAGA algorithm (Defazio et al., 2014a) maintains two moving quantities to optimize (2.1): the current iterate x and a table (memory) of historical gradients $(\alpha_i)_{i=1}^n$. [5] At every iteration, the SAGA algorithm samples uniformly at random an index $i \in \{1, \dots, n\}$, and then executes the following update on x and α (for the unconstrained optimization version):

$$x^+ = x - \gamma\big(f_i'(x) - \alpha_i + \bar{\alpha}\big); \qquad \alpha_i^+ = f_i'(x), \tag{3.1}$$

where γ is the step size and $\bar{\alpha} := {}^1\!/_n \sum_{i=1}^n \alpha_i$ can be updated efficiently in an online fashion. Crucially, $\mathbf{E}\alpha_i = \bar{\alpha}$ and thus the update direction is unbiased ($\mathbf{E}x^+ = x - \gamma f'(x)$). Furthermore, it can be proven (see Defazio et al. (2014a)) that under a reasonable condition on γ, the update has vanishing variance, which enables the algorithm to converge linearly with a constant step size.

4. By "HOGWILD", the authors mean asynchronous parallel variants where cores independently run the sequential update rule.

5. For linear predictor models, the memory α_i^0 can be stored as a scalar. Following Hofmann et al. (2015), α_i^0 can be initialized to any convenient value (typically 0), unlike the prescribed $f_i'(x_0)$ analyzed in (Defazio et al., 2014a).

Motivation for a Variant. In its current form, every SAGA update is dense even if the individual gradients are sparse due to the historical gradient ($\bar{\alpha}$) term. Schmidt et al. (2016) introduced an implementation technique denoted lagged updates in which each iteration has a cost proportional to the size of the support of $f_i'(x)$. However, this technique involves keeping track of past updates and is not easily adaptable to the parallel setting (see Appendix B.4). We therefore introduce Sparse SAGA, a novel variant which explicitly takes sparsity into account and is easily parallelizable.

Sparse SAGA Algorithm. As in the Sparse SVRG algorithm proposed in Mania et al. (2017), we obtain Sparse SAGA by a simple modification of the parameter update rule in (3.1) where $\bar{\alpha}$ is replaced by a sparse version equivalent in expectation:

$$x^+ = x - \gamma(f_i'(x) - \alpha_i + D_i\bar{\alpha}), \tag{3.2}$$

where D_i is a diagonal matrix that makes a weighted projection on the support of f_i'. More precisely, let S_i be the support of the gradient f_i' function (i.e., the set of coordinates where f_i' can be nonzero). Let D be a $d \times d$ diagonal reweighting matrix, with coefficients $1/p_v$ on the diagonal, where p_v is the probability that dimension v belongs to S_i when i is sampled uniformly at random in $\{1, ..., n\}$. We then define $D_i := P_{S_i}D$, where P_{S_i} is the projection onto S_i. The reweighting by D ensures that $\mathbf{E}D_i\bar{\alpha} = \bar{\alpha}$, and thus that the update is still unbiased despite the sparsifying projection.

Convergence Result for (Serial) Sparse SAGA. For clarity of exposition, we model our convergence result after the simple form of Hofmann et al. (2015, Corollary 3). Note that the rate we obtain for Sparse SAGA is the same as the one obtained in the aforementioned reference for SAGA.

Theorem 15. *Let $\gamma = \frac{a}{5L}$ for any $a \leq 1$. Then* Sparse SAGA *converges geometrically in expectation with a rate factor of at least $\rho(a) = \frac{1}{5}\min\left\{\frac{1}{n}, a\frac{1}{\kappa}\right\}$, i.e., for x_t obtained after t updates, we have $\mathbb{E}\|x_t - x^*\|^2 \leq (1-\rho)^t C_0$, where $C_0 := \|x_0 - x^*\|^2 + \frac{1}{5L^2}\sum_{i=1}^n \|\alpha_i^0 - f_i'(x^*)\|^2$.*

Proof outline. We reuse the proof technique from Hofmann et al. (2015), in which a combination of classical strong convexity and Lipschitz inequalities is used to derive the

inequality (Hofmann et al., 2015, Lemma 1):

$$\mathbf{E}\|x^+ - x^*\|^2 \leq (1-\gamma\mu)\|x-x^*\|^2 + 2\gamma^2\mathbf{E}\|\alpha_i - f_i'(x^*)\|^2 \\ + (4\gamma^2 L - 2\gamma)\big(f(x) - f(x^*)\big). \tag{3.3}$$

This gives a contraction term. A Lyapunov function is then defined to control the two other terms. To ensure our variant converges at the same rate as regular SAGA, we only need to prove that the above inequality (Hofmann et al., 2015, Lemma 1) is still verified. To prove this, we derive close variants of equations (6) and (9) in their paper. The rest of the proof can be reused without modification. The full details can be found in Appendix B.1.1.

Comparison with Lagged Updates. The lagged updates technique in SAGA is based on the observation that the updates for component $[x]_v$ need not be applied until this coefficient needs to be accessed, that is, until the next iteration t such that $v \in S_{i_t}$. We refer the reader to Schmidt et al. (2016) for more details.

Interestingly, the expected number of iterations between two steps where a given dimension v is in the support of the partial gradient is p_v^{-1}, where p_v is the probability that v is in the support of the partial gradient at a given step. p_v^{-1} is precisely the term which we use to multiply the update to $[x]_v$ in Sparse SAGA. Therefore one may see the updates in Sparse SAGA as *anticipated* updates, whereas those in the Schmidt et al. (2016) implementation are *lagged*.

The two algorithms appear to be very close, even though Sparse SAGA uses an expectation to multiply a given update whereas the lazy implementation uses a random variable (with the same expectation). Sparse SAGA therefore uses a slightly more aggressive strategy, which may explain the result of our experiments (see Section 3.4.3): both Sparse SAGA and SAGA with lagged updates had similar convergence in terms of number of iterations, with the Sparse SAGA scheme being slightly faster in terms of runtime.

Although Sparse SAGA requires the computation of the p_v probabilities, this can be done during a first pass throughout the data (during which constant step size SGD may be used) at a negligible cost.

Bonus: Sparse SAGA with non-uniform sampling. Although uniform randomization schemes have brought great success to first-order optimization methods, recent works have shown that adapting the sampling to the specific optimization objective can bring additional speed improvements. This is the case for SGD (Nesterov, 2012) or stochastic coordinate descent (Needell et al., 2014, SCD).

The intuition is that the quicker the gradients evolve, the more they should be sampled. The standard way to implement this non-uniform sampling scheme is to sample according to the Lipschitz constant L_i of each data point i. This results in methods that depend on the average Lipschitz constant $\bar{L} := \frac{1}{n}\sum_{i=1}^{n} L_i$, instead of the bigger maximum Lipschitz constant L.

The original SAGA algorithm has already benefited from such an adaptation (Schmidt et al., 2015). We now introduce a non-uniform sampling scheme for Sparse SAGA, and show that we obtain the same improvements than the dense version. The update rule of this new algorithm is as follows:

$$x^+ = x - \frac{\gamma}{nq_i}\left(f_i'(x) - \alpha_i\right) + \tilde{D}_i\bar{\alpha}; \qquad \alpha_j^+ = f_j'(x), \tag{3.4}$$

where $q_i := \frac{L_i}{\sum_{i=1}^{n} L_i}$, i is sampled according to the $(q_i)_{i=1}^n$ distribution, and j is sampled uniformly at random in $\{1, ..., n\}$. $\tilde{D}$ is again a $d \times d$ diagonal reweighting matrix, whose purpose is to ensure that $\mathbf{E}\tilde{D}_i\bar{\alpha} = \bar{\alpha}$ and thus that the update is still unbiased despite the sparsifying projection. As we are not sampling uniformly at random anymore, $\tilde{D}$ has coefficients $1/\tilde{p}_v$ on the diagonal, where $\tilde{p}_v$ is the probability that dimension v belongs to S_i when i is sampled according to $(q_i)_{i=1}^n$.

Note that compared to SGD or SCD, there is an additional difficulty in the case of SAGA, where the sampled data point i is used both to compute the gradient update to the parameters and to update the historical gradients table. However, to adapt SAGA to non-uniform sampling, one has to decouple these two updates and use separate data points (respectively i and j in (3.4)), sampled from two different probability distributions. Here, the gradient update data point is sampled from a non-uniform scheme, while the data point for the historical gradients table update is sampled uniformly at random.

Theorem 16. *Let $\gamma = \frac{a}{5\bar{L}}$ for any $a \leq 1$. Then* Sparse SAGA *with non-uniform sampling converges geometrically in expectation with a rate factor of at least $\rho(a) = \frac{1}{5}\min\left\{\frac{1}{n}, a\frac{1}{\kappa}\right\}$, i.e., for x_t obtained after t updates, we have $\mathbb{E}\|x_t - x^*\|^2 \leq (1-\rho)^t C_0^{nus}$, where $C_0^{nus} := \|x_0 - x^*\|^2 + \frac{1}{5\bar{L}^2}\sum_{i=1}^{n}\|\alpha_i^0 - f_i'(x^*)\|^2$.*

We see that this result is almost exactly the same as Theorem 15. The only difference is that the maximum Lipschitz constant L has been replaced with the average Lipschitz constant $\bar{L}$ in the maximum allowable stepsize (and in the constant C_0^{nus}), which means that non-uniform sampling allows larger stepsizes, ultimately leading to even faster linear convergence.

Algorithm 3 ASAGA (analyzed algorithm)

Initialize shared variables x and $(\alpha_i)_{i=1}^n$
keep doing in parallel
  $\hat{x} =$ inconsistent read of x
  $\forall j$, $\hat{\alpha}_j =$ inconsistent read of α_j
  Sample i uniformly in $\{1, ..., n\}$
  Let S_i be f_i's support
  $[\bar{\alpha}]_{S_i} = {}^1/_n \sum_{k=1}^n [\hat{\alpha}_k]_{S_i}$

  $[\delta x]_{S_i} = -\gamma(f_i'(\hat{x}) - \hat{\alpha}_i + D_i[\bar{\alpha}]_{S_i})$
  for v **in** S_i **do**
    $[x]_v \leftarrow [x]_v + [\delta x]_v$ // atomic
    $[\alpha_i]_v \leftarrow [f_i'(\hat{x})]_v$
    // '$\leftarrow$' denotes a shared memory update
  end for
end parallel loop

Algorithm 4 ASAGA (implementation)

Initialize shared x, $(\alpha_i)_{i=1}^n$ and $\bar{\alpha}$
keep doing in parallel
  Sample i uniformly in $\{1, ..., n\}$
  Let S_i be f_i's support
  $[\hat{x}]_{S_i} =$ inconsistent read of x on S_i
  $\hat{\alpha}_i =$ inconsistent read of α_i
  $[\bar{\alpha}]_{S_i} =$ inconsistent read of $\bar{\alpha}$ on S_i
  $[\delta\alpha]_{S_i} = f_i'([\hat{x}]_{S_i}) - \hat{\alpha}_i$
  $[\delta x]_{S_i} = -\gamma([\delta\alpha]_{S_i} + D_i[\bar{\alpha}]_{S_i})$
  for v **in** S_i **do**
    $[x]_v \leftarrow [x]_v + [\delta x]_v$ // atomic
    $[\alpha_i]_v \leftarrow [\alpha_i]_v + [\delta\alpha]_v$ // atomic
    $[\bar{\alpha}]_v \leftarrow [\bar{\alpha}]_v + {}^1/_n[\delta\alpha]_v$ // atomic
  end for
end parallel loop

Proof outline. We can again reuse the proof technique from Hofmann et al. (2015). Again, proving that a similar inequality to (Hofmann et al., 2015, Lemma 1) is verified will be enough to finish the proof. This is because we have made it so that each α_j has the same probability of being updated in the historical gradients table at each iteration, which is key to derive (Hofmann et al., 2015, Lemma 2).

The updated inequality we have to demonstrate is very similar to (3.3). Notably, L has been replaced by $\bar{L}$:

$$\mathbf{E}\|x^+ - x^*\|^2 \leq (1-\gamma\mu)\|x-x^*\|^2 + 2\gamma^2\mathbf{E}\|\frac{\alpha_i - f_i'(x^*)}{nq_i}\|^2 + (4\gamma^2\bar{L} - 2\gamma)\big(f(x) - f(x^*)\big). \tag{3.5}$$

To prove it, we again derive close variants of equations (6) and (9) in their paper and reuse the rest of the proof without modification. The full details can be found in Appendix B.1.2.

3.2.2 Asynchronous Parallel Sparse SAGA

We describe ASAGA, a sparse asynchronous parallel implementation of Sparse SAGA, in Algorithm 3 in the theoretical form that we analyze, and in Algorithm 4 as its practical implementation. We state our convergence result and analyze our algorithm using the improved perturbed iterate framework.

In the specific case of (Sparse) SAGA, we have to add the additional read memory

argument $\hat{\alpha}^t$ to our perturbed update (2.3):

$$\begin{aligned} x_{t+1} &:= x_t - \gamma g(\hat{x}_t, \hat{\alpha}^t, i_t); \\ g(\hat{x}_t, \hat{\alpha}^t, i_t) &:= f'_{i_t}(\hat{x}_t) - \hat{\alpha}^t_{i_t} + D_{i_t}\left(1/n \sum_{i=1}^n \hat{\alpha}^t_i\right). \end{aligned} \tag{3.6}$$

This results in a slightly different unbiasedness property.

Property 17 (unbiased estimator)**.** *The update, $g_t := g(\hat{x}_t, \hat{\alpha}^t, i_t)$, is an unbiased estimator of the true gradient at $\hat{x}_t$, i.e.*

$$\mathbf{E}[g(\hat{x}_t, \hat{\alpha}^t, i_t)|\hat{x}_t] = f'(\hat{x}_t). \tag{3.7}$$

As in the HOGWILD analysis performed in Section 2.3, this property is crucial for the analysis. It follows by the independence of i_t with $\hat{x}_t$ and from the computation of $\bar{\alpha}$ on line 7 of Algorithm 3, which ensures that $\mathbb{E}\hat{\alpha}_i = 1/n \sum_{k=1}^n [\hat{\alpha}_k]_{S_i} = [\bar{\alpha}]_{S_i}$, making the update unbiased. In practice, recomputing $\bar{\alpha}$ is not optimal, but storing it instead introduces potential bias issues in the proof (as detailed in Appendix B.5.3).

Equation (2.6), which governs the effect of asynchrony also require some adaptation.

$$\begin{aligned} x_t &= x_0 - \gamma \sum_{u=0}^{t-1} g(\hat{x}_u, \hat{\alpha}^u, i_u)\,; \\ [\hat{x}_t]_v &= [x_0]_v - \gamma \sum_{\substack{u=0 \\ \text{u s.t. coordinate } v \text{ was written} \\ \text{for } u \text{ before } t}}^{t-1} [g(\hat{x}_u, \hat{\alpha}^u, i_u)]_v\,. \end{aligned} \tag{3.8}$$

So does Equation (2.7), which makes explicit the difference between the actual iterate $\hat{x}_t$ and the virtual one x_t.

$$\hat{x}_t - x_t = \gamma \sum_{u=(t-\tau)_+}^{t-1} G^t_u g(\hat{x}_u, \hat{\alpha}^u, i_u). \tag{3.9}$$

3.2.3 Convergence and speedup results

We now state our main theoretical results. We give a detailed outline of the proof in Section 3.2.3 and its full details in Appendix B.2.

Convergence and speedup statements

Theorem 18 (Convergence guarantee and rate of ASAGA)**.** *Suppose* $\tau < n/10$.[6] *Let*

$$a^*(\tau) := \frac{1}{32\left(1+\tau\sqrt{\Delta}\right)\xi(\kappa,\Delta,\tau)} \quad \text{where } \xi(\kappa,\Delta,\tau) := \sqrt{1+\frac{1}{8\kappa}\min\{\frac{1}{\sqrt{\Delta}},\tau\}}$$
$$\text{(note that } \xi(\kappa,\Delta,\tau) \approx 1 \text{ unless } \kappa < 1/\sqrt{\Delta}\ (\leq \sqrt{n})\text{).} \tag{3.10}$$

For any step size $\gamma = \frac{a}{L}$ *with* $a \leq a^*(\tau)$*, the inconsistent read iterates of Algorithm 3 converge in expectation at a geometric rate of at least:* $\rho(a) = \frac{1}{5}\min\left\{\frac{1}{n}, a\frac{1}{\kappa}\right\}$*, i.e.,* $\mathbb{E}f(\hat{x}_t) - f(x^*) \leq (1-\rho)^t\,\tilde{C}_0$*, where* $\tilde{C}_0$ *is a constant independent of* t ($\approx \frac{n}{\gamma}C_0$ *with* C_0 *as defined in Theorem 15).*

This result is very close to SAGA's original convergence theorem, but with the maximum step size divided by an extra $1+\tau\sqrt{\Delta}$ factor[7]. Referring to Hofmann et al. (2015) and our own Theorem 15, the rate factor for SAGA is $\min\{1/n, a/\kappa\}$ up to a constant factor. Comparing this rate with Theorem 18 and inferring the conditions on the maximum step size $a^*(\tau)$, we get the following conditions on the overlap τ for ASAGA to have the same rate as SAGA (comparing upper bounds).

Corollary 19 (Speedup condition)**.** *Suppose* $\tau \leq \mathcal{O}(n)$ *and* $\tau \leq \mathcal{O}(\frac{1}{\sqrt{\Delta}}\max\{1, \frac{n}{\kappa}\})$*. Then using the step size* $\gamma = a^*(\tau)/L$ *from* (3.10)*,* ASAGA *converges geometrically with rate factor* $\Omega(\min\{\frac{1}{n}, \frac{1}{\kappa}\})$ *(similar to* SAGA*), and is thus linearly faster than its sequential counterpart up to a constant factor. Moreover, if* $\tau \leq \mathcal{O}(\frac{1}{\sqrt{\Delta}})$*, then a universal step size of* $\Theta(\frac{1}{L})$ *can be used for* ASAGA *to be adaptive to local strong convexity with a similar rate to* SAGA *(i.e., knowledge of* κ *is not required).*

Interestingly, in the well-conditioned regime ($n > \kappa$), ASAGA enjoys the same rate as SAGA even in the non-sparse regime ($\Delta = 1$) for $\tau < \mathcal{O}(n/\kappa)$. This is in contrast to the previous work on asynchronous incremental gradient methods which required some kind of sparsity to get a theoretical linear speedup over their sequential counterpart (see Niu et al. (2011); Mania et al. (2017) or Chapter 2).

This discrepancy with HOGWILD can be explained by the fact that SAGA has a composite rate factor which is not directly proportional to the step size. As a result, in the well-conditioned setting it enjoys a range of step sizes that all give the same contraction rate. This allows its asynchronous variant to use smaller step sizes while maintaining a linear

6. ASAGA can actually converge for any τ, but the maximum step size then has a term of $\exp(\tau/n)$ in the denominator with much worse constants. See Appendix B.2.5.

7. Recall that Δ is a measure of the sparsity of the dataset, given in Definition 7.

speedup.[8] SGD, on the other hand, has a rate factor that is directly proportional to its step size, hence the more restrictive condition on τ.

In the ill-conditioned regime ($\kappa > n$), sparsity is required for a linear speedup, with a bound on τ of $\mathcal{O}(\sqrt{n})$ in the best-case (though degenerate) scenario where $\Delta = 1/n$.

The proof for Corollary 19 can be found in Appendix B.2.7.

Comparison to related work.

- We give the first convergence analysis for an asynchronous parallel version of SAGA (note that Reddi et al. (2015) only covers an epoch based version of SAGA with random stopping times, a fairly different algorithm).
- Theorem 18 can be directly extended to the parallel extension of the SVRG version from Hofmann et al. (2015) which is adaptive to the local strong convexity with similar rates (see Section 3.3.2).
- In contrast to the parallel SVRG analysis from Reddi et al. (2015, Thm. 2), our proof technique handles inconsistent reads and a non-uniform processing speed across f_i's. Our bounds are similar (noting that Δ is equivalent to theirs), except for the adaptivity to local strong convexity: ASAGA does not need to know κ for optimal performance, contrary to parallel SVRG (see Section 3.3 for more details).
- In contrast to the SVRG analysis from Mania et al. (2017, Thm. 14), we obtain a better dependence on the condition number in our rate ($1/\kappa$ vs. $1/\kappa^2$ on their work) and on the sparsity (they obtain $\tau \leq \mathcal{O}(\Delta^{-1/3})$), while we furthermore remove their gradient bound assumption. We also give our convergence guarantee on $\hat{x}_t$ *during* the algorithm, whereas they only bound the error for the "last" iterate x_T.

Proof outline of Theorem 18

We give here an extended outline of the proof. We detail key lemmas in Section 3.2.4.

Our proof technique begins as our HOGWILD analysis, with Properties 2, 3 and 4 also verified for ASAGA.[9] As in our HOGWILD analysis, we make Assumption 5. Defining $g_t := g(\hat{x}_t, \hat{\alpha}^t, i_t)$ yields no meaningful difference over the beginning of our HOGWILD analysis (except in the proof of Property 17, the equivalent of 3, which we have already provided). Consequently, the basic recursive contraction inequality (2.11) and Lemma 11 also hold.

8. It should be noted that SVRG has the same characteristic. Indeed, using a finer analysis than in previous attempts, we show in Section 3.3 that KROMAGNON also exhibits a non-sparse linear speedup regime.

9. Once again, in our analysis the ASAGA algorithm reads the parameters before sampling, so that Property 2 is verified for $r = t$.

The rest of the proof then proceeds as follows:

1. We bound the updates $\mathbb{E}\|g_t\|^2$ in terms of past suboptimalities $(e_v)_{v\leq u}$ by using standard SAGA inequalities and carefully analyzing the update rule for α_i^+ (3.1) in expectation. This gives Lemma 20 below.
2. By applying Lemma 20 to the result of Lemma 11, we obtain a master contraction inequality (3.13) in terms of a_{t+1}, a_t and $(e_u)_{u\leq t}$.
3. We define a novel Lyapunov function $\mathcal{L}_t = \sum_{u=0}^{t}(1-\rho)^{t-u}a_u$ and manipulate the master inequality to show that $\mathcal{L}_t$ is bounded by a contraction, subject to a maximum step size condition on γ (given in Lemma 21 below).
4. Finally, we unroll the Lyapunov inequality to get the convergence Theorem 18.

3.2.4 Key Lemmas

We list the key lemmas below with their proof sketch, and pointers to the relevant parts of Appendix B.2 for detailed proofs.

Lemma 20 (Suboptimality bound on $\mathbb{E}\|g_t\|^2$)**.** *For all $t \geq 0$,*

$$\mathbb{E}\|g_t\|^2 \leq 4Le_t + \frac{4L}{n}\sum_{u=1}^{t-1}(1-\frac{1}{n})^{(t-2\tau-u-1)_+}e_u + 4L(1-\frac{1}{n})^{(t-\tau)_+}\tilde{e}_0\,, \tag{3.11}$$

where $\tilde{e}_0 := \frac{1}{2L}\mathbb{E}\|\alpha_i^0 - f_i'(x^*)\|^2$. [10]

From our proof of convergence for Sparse SAGA we know that (see Appendix B.1.1):

$$\mathbb{E}\|g_t\|^2 \leq 2\mathbb{E}\|f_{i_t}'(\hat{x}_t) - f_{i_t}'(x^*)\|^2 + 2\mathbb{E}\|\hat{\alpha}_{i_t}^t - f_{i_t}'(x^*)\|^2. \tag{3.12}$$

We can handle the first term by taking the expectation over a Lipschitz inequality (Hofmann et al. (2015, Equations (7) and (8)). All that remains to prove the lemma is to express the $\mathbb{E}\|\hat{\alpha}_{i_t}^t - f_{i_t}'(x^*)\|^2$ term in terms of past suboptimalities. We note that it can be seen as an expectation of past first terms with an adequate probability distribution which we derive and bound.

From our algorithm, we know that each dimension of the memory vector $[\hat{\alpha}_i]_v$ contains a partial gradient computed at some point in the past $[f_i'(\hat{x}_{u_{i,v}^t})]_v$ [11] (unless $u=0$, in which case we replace the partial gradient with α_i^0). We then derive bounds on $P(u_{i,v}^t = u)$

10. We introduce this quantity instead of e_0 so as to be able to handle the arbitrary initialization of the α_i^0.
11. More precisely: $\forall t, i, v\ \ \exists u_{i,v}^t < t$ s.t. $[\hat{\alpha}_i^t]_v = [f_i'(\hat{x}_{u_{i,v}^t})]_v$.

and sum on all possible u. Together with clever conditioning, we obtain Lemma 20 (see Section B.2.1).

We note that this is a much more complicated result than its equivalent in the HOGWILD analysis, Lemma 14. However, all terms in the right-hand side go to 0 as we near the optimum, which will enable us to prove convergence to the optimum rather than to a fixed-size ball surrounding it.

Master inequality. Let H_t be defined as $H_t := \sum_{u=1}^{t-1}(1-\frac{1}{n})^{(t-2\tau-u-1)_+}e_u$. Then, by setting (3.11) into Lemma 11, we get (see Section B.2.3):

$$\begin{aligned} a_{t+1} \leq & (1-\frac{\gamma\mu}{2})a_t - 2\gamma e_t + 4L\gamma^2 C_1\big(e_t + (1-\frac{1}{n})^{(t-\tau)_+}\tilde{e}_0\big) + \frac{4L\gamma^2 C_1}{n}H_t \\ & + 4L\gamma^2 C_2 \sum_{u=(t-\tau)_+}^{t-1}\big(e_u + (1-\frac{1}{n})^{(u-\tau)_+}\tilde{e}_0\big) + \frac{4L\gamma^2 C_2}{n}\sum_{u=(t-\tau)_+}^{t-1} H_u\,. \end{aligned} \tag{3.13}$$

Lyapunov function and associated recursive inequality. We now have the beginning of a contraction with additional positive terms which all converge to 0 as we near the optimum, as well as our classical negative suboptimality term. This is not unusual in the variance reduction literature. One successful approach in the sequential case is then to define a Lyapunov function which encompasses all terms and is a true contraction (see Defazio et al. (2014a); Hofmann et al. (2015)). We emulate this solution here. However, while all terms in the sequential case only depend on the current iterate, t, in the parallel case we have terms "from the past" in our inequality. To resolve this issue, we define a more involved Lyapunov function which also encompasses past iterates:

$$\mathcal{L}_t = \sum_{u=0}^{t}(1-\rho)^{t-u}a_u, \quad 0<\rho<1, \tag{3.14}$$

where ρ is a target contraction rate that we define later.

Using the master inequality (3.13), we get (see Appendix B.2.4):

$$\begin{aligned} \mathcal{L}_{t+1} &= (1-\rho)^{t+1}a_0 + \sum_{u=0}^{t}(1-\rho)^{t-u}a_{u+1} \\ &\leq (1-\rho)^{t+1}a_0 + (1-\frac{\gamma\mu}{2})\mathcal{L}_t + \sum_{u=1}^{t} r_u^t e_u + r_0^t\tilde{e}_0\,. \end{aligned} \tag{3.15}$$

The aim is to prove that $\mathcal{L}_t$ is bounded by a contraction. We have two promising terms at the beginning of the inequality, and then we need to handle the last term. Basically, we can

rearrange the sums in (3.13) to expose a simple sum of e_u multiplied by factors r_u^t.

Under specific conditions on ρ and γ, we can prove that r_u^t is negative for all $u \geq 1$, which coupled with the fact that each e_u is positive means that we can safely drop the sum term from the inequality. The r_0^t term is a bit trickier and is handled separately.

In order to obtain a bound on e_t directly rather than on $\mathbb{E}\|\hat{x}_t - x^*\|^2$, we then introduce an additional γe_t term on both sides of (3.15). The bound on γ under which the modified $r_t^t + \gamma$ is negative is then twice as small (we could have used any multiplier between 0 and 2γ, but chose γ for simplicity's sake). This condition is given in the following Lemma.

Lemma 21 (Sufficient condition for convergence)**.** *Suppose $\tau < n/10$ and $\rho \leq 1/4n$. If*

$$\gamma \leq \gamma^* = \frac{1}{32L(1+\sqrt{\Delta}\tau)\sqrt{1+\frac{1}{8\kappa}\min(\tau, \frac{1}{\sqrt{\Delta}})}} \tag{3.16}$$

then for all $u \geq 1$, the coefficients r_u^t from (3.15) are negative. Furthermore, we have $r_t^t + \gamma \leq 0$ and thus:

$$\gamma e_t + \mathcal{L}_{t+1} \leq (1-\rho)^{t+1} a_0 + (1 - \frac{\gamma\mu}{2})\mathcal{L}_t + r_0^t \tilde{e}_0 \,. \tag{3.17}$$

We obtain this result after carefully deriving the r_u^t terms. We find a second-order polynomial inequality in γ, which we simplify down to (3.16) (see Appendix B.2.5).

We can then finish the argument to bound the suboptimality error e_t. We have:

$$\mathcal{L}_{t+1} \leq \gamma e_t + \mathcal{L}_{t+1} \leq (1 - \frac{\gamma\mu}{2})\mathcal{L}_t + (1-\rho)^{t+1}(a_0 + A\tilde{e}_0) \,. \tag{3.18}$$

We have two linearly contracting terms. The sum contracts linearly with the worst rate between the two (the smallest geometric rate factor). If we define $\rho^* := \nu \min(\rho, \gamma\mu/2)$, with $0 < \nu < 1$, [12] then we get:

$$\gamma e_t + \mathcal{L}_{t+1} \leq (1 - \frac{\gamma\mu}{2})^{t+1}\mathcal{L}_0 + (1-\rho^*)^{t+1}\frac{a_0 + A\tilde{e}_0}{1-\eta} \tag{3.19}$$

$$\gamma e_t \leq (1-\rho^*)^{t+1}\big(\mathcal{L}_0 + \frac{1}{1-\eta}(a_0 + A\tilde{e}_0)\big) \,, \tag{3.20}$$

where $\eta := \frac{1-M}{1-\rho^*}$ with $M := \max(\rho, \gamma\mu/2)$. Our geometric rate factor is thus ρ^* (see Appendix B.2.6).

12. ν is introduced to circumvent the problematic case where ρ and $\gamma\mu/2$ are too close together.

3.3 Asynchronous Parallel SVRG with the "After Read" labeling

ASAGA vs. asynchronous SVRG. There are several scenarios in which ASAGA can be practically advantageous over its closely related cousin, asynchronous SVRG (note though that "asynchronous" SVRG still requires one synchronization step per epoch to compute a full gradient).

First, while SAGA trades memory for less computation, in the case of generalized linear models the memory cost can be reduced to $\mathcal{O}(n)$, compared to $\mathcal{O}(d)$ for SVRG (Johnson and Zhang, 2013). This is of course also true for their asynchronous counterparts.

Second, as ASAGA does not require any synchronization steps, it is better suited to heterogeneous computing environments (where cores have different clock speeds or are shared with other applications).

Finally, ASAGA does not require knowing the condition number κ for optimal convergence in the sparse regime. It is thus adaptive to local strong convexity, whereas SVRG is not. Indeed, SVRG and its asynchronous variant require setting an additional hyper-parameter – the epoch size m – which needs to be at least $\Omega(\kappa)$ for convergence but yields a slower effective convergence rate than ASAGA if it is set much bigger than κ. SVRG thus requires tuning this additional hyper-parameter or running the risk of either slower convergence (if the epoch size chosen is much bigger than the condition number) or even not converging at all (if m is chosen to be much smaller than κ). [13]

Motivation for analyzing asynchronous SVRG. Despite the advantages that we have just listed, in the case of complex models, the storage cost of SAGA may become too expensive for practical use. SVRG (Johnson and Zhang, 2013) trades off more computation for less storage and does not suffer from this drawback. It can thus be applied to cases where SAGA cannot (e.g. deep learning models, see Reddi et al. (2016)).

Another advantage of KROMAGNON is that the historical gradient term $f'(\tilde{x})$ is fixed during an epoch, while its ASAGA equivalent, $\bar{\alpha}$, has to be updated at each iteration, either by recomputing if from the $\hat{\alpha}$ – which is costly – or by updating a maintained quantity – which is cheaper but may ultimately result in introducing some bias in the update (see Appendix B.5.3 for more details on this subtle issue).

13. Note that as SAGA (and contrary to the original SVRG) the SVRG variant from Hofmann et al. (2015) does not require knowledge of κ and is thus adaptive to local strong convexity, which carries over to its asynchronous adaptation that we analyze in Section 3.3.2.

It is thus worthwhile to carry out the analysis of KROMAGNON (Mania et al., 2017), [14] the asynchronous parallel version of SVRG, although it has to be noted that since SVRG requires regularly computing batch gradients, KROMAGNON will present regular synchronization steps as well as coordinated computation – making it less attractive for the asynchronous parallel setting.

We first extend our ASAGA analysis to analyze the convergence of a variant of SVRG presented in Hofmann et al. (2015), obtaining exactly the same bounds. This variant improves upon the initial algorithm because it does not require tuning the epoch size hyperparameter and is thus adaptive to local strong convexity (see Section 3.3.1). Furthermore, it allows for a cleaner analysis where – contrary to SVRG – we do not have to replace the final parameters of an epoch by one of its random iterates.

Then, using our "after read" labeling, we are also able to derive a convergence and speedup proof for KROMAGNON, with comparable results to our ASAGA analysis. In particular, we prove that as for ASAGA in the "well-conditioned" regime KROMAGNON can achieve a linear speedup even without sparsity assumptions.

3.3.1 SVRG algorithms

We start by describing the original SVRG algorithm, the variant given in Hofmann et al. (2015) and the sparse asynchronous parallel adaptation, KROMAGNON.

Original SVRG algorithm. The standard SVRG algorithm (Johnson and Zhang, 2013) is very similar to SAGA. The main difference is that instead of maintaining a table of historical gradients, SVRG uses a "reference" batch gradient $f'(\tilde{x})$, updated at regular intervals (typically every m iterations, where m is a hyper-parameter). SVRG is thus an epoch-based algorithm, where at the beginning of every epoch a reference iterate $\tilde{x}$ is chosen and its gradient is computed. Then, at every iteration in the epoch, the algorithm samples uniformly at random an index $i \in \{1, \dots, n\}$, and then executes the following update on x:

$$x^+ = x - \gamma\left(f_i'(x) - f_i'(\tilde{x}) + f'(\tilde{x})\right). \tag{3.21}$$

As for SAGA the update direction is unbiased ($\mathbf{E}x^+ = x - \gamma f'(x)$) and it can be proven (see Johnson and Zhang (2013)) that under a reasonable condition on γ and m (the epoch size), the update has vanishing variance, which enables the algorithm to converge linearly

14. The speedup analysis presented in Mania et al. (2017) is not fully satisfactory as it does not achieve state-of-the-art convergence results for either SVRG or KROMAGNON. Furthermore, we are able to remove their uniform gradient bound assumption, which is inconsistent with strong convexity.

with a constant step size.

Hofmann's SVRG variant. Hofmann et al. (2015) introduce a variant where the size of the epoch is a random variable. At each iteration t, a first Bernoulli random variable B_t with $p = 1/n$ is sampled. If $B_t = 1$, then the algorithm updates the reference iterate, $\tilde{x} = x_t$ and computes its full gradient as its new "reference gradient". If $B_t = 0$, the algorithm executes the normal SVRG inner update. Note that this variant is adaptive to local strong convexity, as it does not require the inner loop epoch size $m = \Omega(\kappa)$ as a hyperparameter. In that respect it is closer to SAGA than the original SVRG algorithm which is not adaptive.

KROMAGNON KROMAGNON, introduced in Mania et al. (2017) is obtained by using the same sparse update technique as Sparse SAGA, and then running the resulting algorithm in parallel (see Algorithm 5).

3.3.2 Extension to the SVRG variant from Hofmann et al. (2015)

We introduce AHSVRG – a sparse asynchronous parallel version for the SVRG variant from Hofmann et al. (2015) – in Algorithm 6. Every core runs stochastic updates independently as long as they are all sampling inner updates, and coordinate whenever one of them decides to do a batch gradient computation. The one difficulty of this approach is that each core needs to be able to communicate to every other core that they should stop doing inner updates and start computing a synchronized batch gradient instead.

To this end, we introduce a new shared variable, s, which represents the "state" of the computation. This variable is checked by each core c before each update. If $s = 1$, then another core has called for a batch gradient computation and core c starts computing its allocated part of this computation. If $s = 0$, core c proceeds to sample a first random variable. Then it either samples and performs an inner update and keeps going, or it samples a full gradient computation, in which case it updates s to 1 and starts computing its allocated part of the computation. Once a full gradient is computed, s is set to 0 once again and every core resume their loop.

Our ASAGA convergence and speedup proofs can easily be adapted to accommodate AHSVRG since it is closer to SAGA than the initial SVRG algorithm. To prove convergence, all one has to do is to modify Lemma 20 very slightly (the only difference is that the $(t - 2\tau - u - 1)_+$ exponent is replaced by $(t - \tau - u - 1)_+$ and the rest of the proof can be used as is). The justification for this small tweak is that the batch steps in SVRG are fully synchronized. More details can be found in Appendix B.2.2.

Algorithm 5 KROMAGNON (Mania et al., 2017)

Initialize shared x and x_0
while True **do**
  Compute in parallel $g = f'(x_0)$ (synchronously)
  for $i = 1..m$ **do in parallel (asynchronously)**
    Sample i uniformly in $\{1, ..., n\}$
    Let S_i be f_i's support
    $[\hat{x}]_{S_i}$ = inconsistent read of x on S_i
    $[\delta x]_{S_i} = -\gamma\big([f_i'(\hat{x}_t) - f_i'(x_0)]_{S_i} + D_i[g]_{S_i}\big)$
    for v **in** S_i **do**
      $[x]_v = [x]_v + [\delta x]_v$ // atomic
    end for
  end parallel loop
  $x_0 = x$
end while

Algorithm 6 AHSVRG

Initialize shared x, s and x_0
while True **do**
  Compute in parallel $g = f'(x_0)$ (synchronously)
  $s = 0$
  while $s = 0$ **do in parallel (asynchronously)**
    Sample B with $p = 1/n$
    if $B = 1$ **then**
      $s = 1$
    else
      Sample i uniformly in $\{1, ..., n\}$
      Let S_i be f_i's support
      $[\hat{x}]_{S_i}$ = inconsistent read of x on S_i
      $[\delta x]_{S_i} = -\gamma([f_i'(\hat{x}_t) - f_i'(x_0)]_{S_i} + D_i[g]_{S_i})$
      for v **in** S_i **do**
        $[x]_v = [x]_v + [\delta x]_v$ // atomic
      end for
    end if
  end parallel loop
  $x_0 = x$
end while

3.3.3 Fast convergence and speedup rates for KROMAGNON

We now state our main theoretical results. We give a detailed outline of the proof in Section 3.3.4 and its full details in Appendix B.3.

Theorem 22 (Convergence guarantee and rate of KROMAGNON)**.** *Suppose the step size γ and epoch size m are chosen such that the following condition holds:*

$$0 < \theta := \frac{\frac{1}{\mu\gamma m} + 2L(1 + 2\sqrt{\Delta}\tau)(\gamma + \tau\mu\gamma^2)}{1 - 2L(1 + 2\sqrt{\Delta}\tau)(\gamma + \tau\mu\gamma^2)} < 1\,. \tag{3.22}$$

Then the inconsistent read iterates of KROMAGNON *converge in expectation at a geometric rate, i.e.*

$$\mathbb{E}f(\tilde{x}_k) - f(x^*) \leq \theta^t(f(x_0) - f(x^*))\,, \tag{3.23}$$

where $\tilde{x}_k$ is the initial iterate for epoch k, which is obtained by choosing uniformly at random among the inconsistent read iterates from the previous epoch.

This result is similar to the theorem given in the original SVRG paper (Johnson and Zhang, 2013). Indeed, if we remove the asynchronous part (i.e. if we set $\tau = 0$), we get exactly the same rate and condition. It also has the same form as the one given in Reddi et al. (2015), which was derived for dense asynchronous SVRG in the easier setting of consistent read and writes (and in the flawed "after write" framework), and gives essentially the same conditions on γ and m.

In the canonical example presented in most SVRG papers, with $\kappa = n$, $m = \mathcal{O}(n)$ and $\gamma = 1/10L$, SVRG obtains a convergence rate of 0.5. Reddi et al. (2015) get the same rate by setting $\gamma = 1/20\max(1,\sqrt{\Delta}\tau)L$ and $m = \mathcal{O}\big(n(1 + \sqrt{\Delta}\tau)\big)$. Following the same line of reasoning (setting $\gamma = 1/20\max(1,\sqrt{\Delta}\tau)L$, $\tau = \mathcal{O}(n)$ and $\theta = 0.5$ and computing the resulting condition on m), these values for γ and m also give us a convergence rate of 0.5. Therefore, as in Reddi et al. (2015), when $\kappa = n$ we get a linear speedup for $\tau < 1/\sqrt{\Delta}$ (which can be as big as $\sqrt{n}$ in the degenerate case where no data points share any feature with each other). Note that this is the same speedup condition as ASAGA in this regime.

SVRG theorems are usually similar to Theorem 22, which does not give an optimal step size or epoch size. This makes the analysis of a parallel speedup difficult, prompting authors to compare rates in specific cases with most parameters fixed, as we have just done. In order to investigate the speedup and step size conditions more precisely and thus derive a more general theorem, we now give SVRG and KROMAGNON results modeled on Theorem 18.

Corollary 23 (Convergence guarantee and rate for serial SVRG)**.** *Let $\gamma = \frac{a}{4L}$ for any $a \leq \frac{1}{4}$ and $m = \frac{32\kappa}{a}$. Then* SVRG *converges geometrically in expectation with a rate factor per gradient computation of at least $\rho(a) = \frac{1}{4}\min\left\{\frac{1}{n}, \frac{a}{64\kappa}\right\}$, i.e.*

$$\mathbb{E}f(\tilde{x}_k) - f(x^*) \leq (1 - \rho)^{k(2m+n)}\left(f(x_0) - f(x^*)\right) \qquad \forall k \geq 0\,. \tag{3.24}$$

Due to SVRG's special structure, we cannot write $\mathbb{E}f(x_t) - f(x^*) \leq (1-\rho)^t \left(f(x_0) - f(x^*)\right)$ for all $t \geq 0$. However, expressing the convergence properties of this algorithm in terms of a rate factor per gradient computation (of which there are $2m+n$ per epoch) makes it easier to compare convergence rates, either to similar algorithms such as SAGA or to its parallel variant KROMAGNON – and thus to study the speedup obtained by parallelizing SVRG.

Compared to SAGA, this result is very close. The main difference is that the additional hyper-parameter m has to be set and requires knowledge of μ. This illustrates the fact that SVRG is not adaptive to local strong convexity, whereas both SAGA and Hofmann's SVRG are.

Corollary 24 (Simplified convergence guarantee and rate for KROMAGNON)**.** *Let*

$$a^*(\tau) = \frac{1}{4(1+2\sqrt{\Delta}\tau)(1+\frac{\tau}{16\kappa})}\,. \tag{3.25}$$

For any step size $\gamma = \frac{a}{4L}$ with $a \leq a^*(\tau)$ and $m = \frac{32\kappa}{a}$, KROMAGNON *converges geometrically in expectation with a rate factor per gradient computation of at least $\rho(a) = \frac{1}{4}\min\left\{\frac{1}{n}, \frac{a}{64\kappa}\right\}$, i.e.*

$$\mathbb{E}f(\tilde{x}_k) - f(x^*) \leq (1-\rho)^{k(2m+n)}\left(f(x_0) - f(x^*)\right) \qquad \forall k \geq 0\,. \tag{3.26}$$

This result is again quite close to Corollary 23 derived in the serial case. We see that the maximum step size is divided by an additional $(1+2\tau\sqrt{\Delta})$ term, while the convergence rate is the same. Comparing the rates and the maximum allowable step sizes in both settings give us the sufficient condition on τ to get a linear speedup.

Corollary 25 (Speedup condition)**.** *Suppose $\tau \leq \mathcal{O}(n)$ and $\tau \leq \mathcal{O}(\frac{1}{\sqrt{\Delta}}\max\{1, \frac{n}{\kappa}\})$. If $n \geq \kappa$, also suppose $\tau \leq \sqrt{n\Delta^{-1/2}}$. Then using the step size $\gamma = {}^{a^*(\tau)}/_{L}$ from (3.25),* KROMAGNON *converges geometrically with rate factor $\Omega(\min\{\frac{1}{n}, \frac{1}{\kappa}\})$ (similar to* SVRG*), and is thus linearly faster than its sequential counterpart up to a constant factor.*

This result is almost the same as ASAGA, with the additional condition that $\tau \leq \mathcal{O}(\sqrt{n})$ in the well-conditioned regime. We see that in this regime KROMAGNON can also get the same rate as SVRG even without sparsity, which had not been observed in previous work.

Furthermore, one has to note that τ is generally smaller for KROMAGNON than for ASAGA since it is reset to 0 at the beginning of each new epoch (where all cores are synchronized once more).

Comparison to related work.

- Corollary 23 provides a rate of convergence *per gradient computation* for SVRG, contrary to most of the literature on this algorithm (including the seminal paper (Johnson and Zhang, 2013)). This result allows for easy comparison with SAGA and other algorithms (in contrast, Konečný and Richtárik (2013) is more involved).
- In contrast to the SVRG analysis from Reddi et al. (2015, Thm. 2), our proof technique handles inconsistent reads and a non-uniform processing speed across f_i's. While Theorem 22 is similar to theirs, Corollary 23 and 24 are more precise results. They enable a finer analysis of the speedup conditions (Corollary 25) – including the possible speedup without sparsity regime.
- In contrast to the KROMAGNON analysis from Mania et al. (2017, Thm. 14), Theorem 22 gives a better dependence on the condition number in the rate ($1/\kappa$ vs. $1/\kappa^2$ for them) and on the sparsity (they get $\tau \leq \mathcal{O}(\Delta^{-1/3})$), while we remove their gradient bound assumption. Our results are state-of-the-art for SVRG (contrary to theirs) and so our speedup comparison is more meaningful. Finally, Theorem 22 gives convergence guarantees on $\hat{x}_t$ *during* the algorithm, whereas they only bound the error for the "last" iterate x_T.

3.3.4 Proof outline

We now give a detailed outline of the proof. Its full details can be found in Appendix B.3.

Once again, our proof technique begins as our HOGWILD analysis. In particular, Properties 2, 17, 4 are also verified for KROMAGNON [15], and as in our HOGWILD and ASAGA analysis, we make Assumption 5 (bounded overlaps). Consequently, the basic recursive contraction inequality (2.11) and Lemma 11 also hold. As for ASAGA the proof diverges when we derive the equivalent of Lemma 14. We get a slightly different form from the ASAGA equivalent, which prompts a difference in the rest of the proof technique.

Lemma 26 (Suboptimality bound on $\mathbb{E}\|g_t\|^2$)**.**

$$\mathbb{E}\|g_t\|^2 \leq 4Le_t + 4L\tilde{e}_k \qquad \forall k \geq 0, km \leq t \leq (k+1)m\,, \tag{3.27}$$

where $\tilde{e}_k := \mathbb{E}f(\tilde{x}_k) - f(x^*)$ *and* $\tilde{x}_k$ *is the initial iterate for epoch* k*.*

We give the proof in Appendix B.3.1. To derive both terms, we use the same technique as for the first term of Lemma 20. Although this is a much simpler result than Lemma 20

15. Note that similarly to ASAGA, the KROMAGNON algorithm which we analyze reads the parameters first and then samples. This is necessary in order for Property 2 to be verified at $r = t$, although not practical when it comes to actual implementation.

in the case of ASAGA, two key differences prevent us from reusing the same Lyapunov function proof technique. First, the e_0 term in Lemma 20 is replaced by $\tilde{e}_k$ which depends on the epoch number. Second, this term is not multiplied by a geometrically decreasing quantity, which means the $-2\gamma e_0$ term is not sufficient to cancel out all of the e_0 terms coming from subsequent inequalities. To solve this issue, we go to more traditional SVRG techniques.

The rest of the proof is as follows:

1. By substituting Lemma 26 into Lemma 11, we get a master contraction inequality (3.28) in terms of a_{t+1}, a_t and $e_u, u \leq t$.
2. As in Johnson and Zhang (2013), we sum the master contraction inequality over a whole epoch, and then use the same randomization trick (3.30) to derive a relationship between $(e_t)_{km \leq t \leq (k+1)m-1}$ and $\tilde{e}_k$.
3. We thus obtain a contraction inequality between $\tilde{e}_k$ and $\tilde{e}_{k-1}$, which finishes the proof for Theorem 22.
4. We then only have to derive the conditions on γ, τ and m under which we contractions and compare convergence rates to finish the proofs for Corollary 23, Corollary 24 and Corollary 25.

We list the key points below with their proof sketch, and give the detailed proof in Appendix B.3.

Master inequality. As in our ASAGA analysis, we apply (3.27) to the result of Lemma 11, which gives us that for all $k \geq 0, km \leq t \leq (k+1)m - 1$ (see Appendix B.3.2):

$$a_{t+1} \leq (1 - \frac{\gamma\mu}{2})a_t + (4L\gamma^2 C_1 - 2\gamma)e_t + 4L\gamma^2 C_2 \sum_{u=\max(km, t-\tau)}^{t-1} e_u + (4L\gamma^2 C_1 + 4L\gamma^2 \tau C_2)\tilde{e}_k \,. \tag{3.28}$$

Contraction inequality. As we previously mentioned, the term in $\tilde{e}_k$ is not multiplied by a geometrically decreasing factor, so using the same Lyapunov function as for ASAGA cannot work. Instead, we apply the same method as in the original SVRG paper (Johnson and Zhang, 2013): we sum the master contraction inequality over a whole epoch. This gives us (see Appendix B.3.2):

$$a_{(k+1)m} \leq a_{km} + (4L\gamma^2 C_1 + 4L\gamma^2 \tau C_2 - 2\gamma) \sum_{t=km}^{(k+1)m-1} e_t + m(4L\gamma^2 C_1 + 4L\gamma^2 \tau C_2)\tilde{e}_k \,. \tag{3.29}$$

To cancel out the $\tilde{e}_k$ term, we only have one negative term on the right-hand side of (3.29): $-2\gamma \sum_{t=km}^{(k+1)m-1} e_t$. This means we need to relate $\sum_{t=km}^{(k+1)m-1} e_t$ to $\tilde{e}_k$. We can do it using the same randomization trick as in Johnson and Zhang (2013): instead of choosing the last iterate of the k^{th} epoch as $\tilde{x}_k$, we pick one of the iterates of the epoch uniformly at random. This means we get:

$$\tilde{e}_k = \mathbb{E} f(\tilde{x}_k) - f(x^*) = \frac{1}{m} \sum_{t=(k-1)m}^{km-1} e_t \tag{3.30}$$

We now have: $\sum_{t=km}^{(k+1)m-1} e_t = m\tilde{e}_{k+1}$. Combined with the fact that $a_{km} \leq \frac{2}{\mu}\tilde{e}_k$ and that we can remove the positive $a_{(k+1)m}$ term from the left-hand-side of (3.29), this gives us our final recursion inequality:

$$\big(2\gamma m - 4L\gamma^2 C_1 m - 4L\gamma^2 \tau C_2 m\big)\tilde{e}_{k+1} \leq \big(\frac{2}{\mu} + 4L\gamma^2 C_1 m + 4L\gamma^2 \tau C_2 m\big)\tilde{e}_k \tag{3.31}$$

Replacing C_1 and C_2 by their values (defined in (2.13)) in (3.31) directly leads to Theorem 22.

3.4 Empirical results

We now present the results of our experiments. We first compare our new sequential algorithm, Sparse SAGA, to its existing alternative, SAGA with lagged updates and to the original SAGA algorithm as a baseline. We then move on to our main results, the empirical comparison of ASAGA, KROMAGNON and HOGWILD. Finally, we present additional results, including convergence and speedup figures with respect to the number of iteration (i.e. "theoretical speedups") and measures on the τ constant.

3.4.1 Experimental setup

Models. Although ASAGA can be applied more broadly, we focus on logistic regression, a model of particular practical importance. The associated objective function takes the following form:

$$\frac{1}{n}\sum_{i=1}^{n} \log\left(1 + \exp(-b_i a_i^\intercal x)\right) + \frac{\mu}{2}\|x\|^2, \tag{3.32}$$

where $a_i \in \mathbb{R}^d$ and $b_i \in \{-1, +1\}$ are the data samples.

Datasets. We consider two sparse datasets: RCV1 (Lewis et al., 2004) and URL (Ma et al., 2009); and a dense one, Covtype (Collobert et al., 2002), with statistics listed in the table below. As in Le Roux et al. (2012), Covtype is standardized, thus 100% dense. Δ is $\mathcal{O}(1)$ in all datasets, hence not very insightful when relating it to our theoretical results. Deriving a less coarse sparsity bound remains an open problem.

Table 3.1 – Basic dataset statistics.

	n	d	density	L
RCV1	697,641	47,236	0.15%	0.25
URL	2,396,130	3,231,961	0.004%	128.4
Covtype	581,012	54	100%	48428

3.4.2 Implementation details

Regularization. Following Schmidt et al. (2016), the amount of regularization used was set to $\mu = 1/n$. In each update, we project the gradient of the regularization term (we multiply it by D_i as we also do with the vector $\bar{\alpha}$) to preserve the sparsity pattern while maintaining an unbiased estimate of the gradient. For squared ℓ_2, the Sparse SAGA updates becomes:

$$x^+ = x - \gamma(f_i'(x) - \alpha_i + D_i\bar{\alpha} + \mu D_i x). \tag{3.33}$$

Comparison with the theoretical algorithm. The algorithm we used in the experiments is fully detailed in Algorithm 4. There are two differences with Algorithm 3. First, in the implementation we choose i_t at random *before* we read the feature vector a_{i_t}. This enables us to only read the necessary data for a given iteration (i.e. $[\hat{x}_t]_{S_i}, [\hat{\alpha}_i^t], [\bar{\alpha}^t]_{S_i}$). Although this violates Property 2, it still performs well in practice.

Second, we maintain $\bar{\alpha}^t$ in memory. This saves the cost of recomputing it at every iteration (which we can no longer do since we only read a subset data). Again, in practice the implemented algorithm enjoys good performance. But this design choice raises a subtle point: the update is not guaranteed to be unbiased in this setup (see Appendix B.5.3 for more details).

Step sizes. For each algorithm, we picked the best step size among 10 equally spaced values in a grid, and made sure that the best step size was never at the boundary of this interval. For Covtype and RCV1, we used the interval $[\frac{1}{10L}, \frac{10}{L}]$, whereas for URL we used the interval $[\frac{1}{L}, \frac{100}{L}]$ as it admitted larger step sizes. It turns out that the best step size

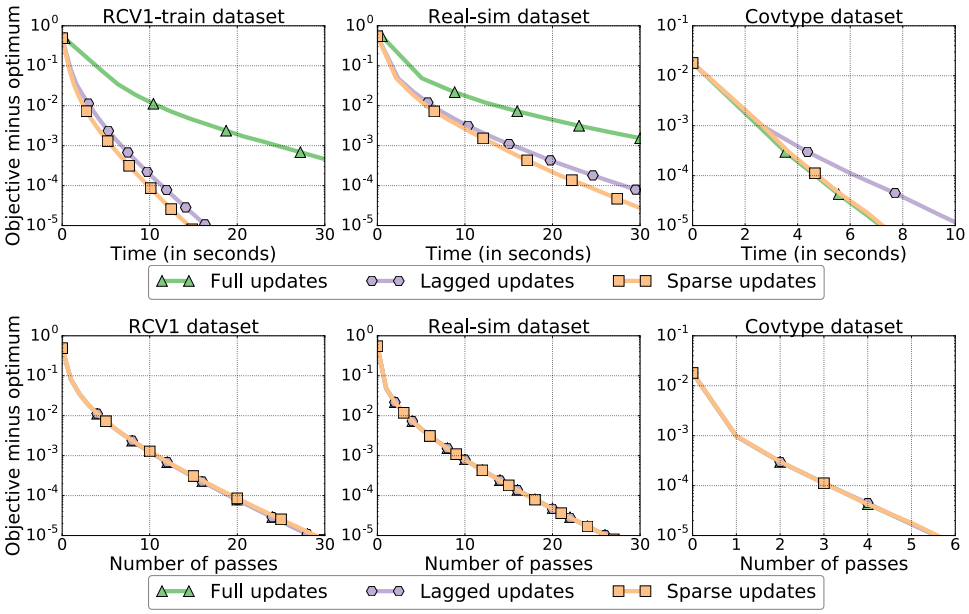


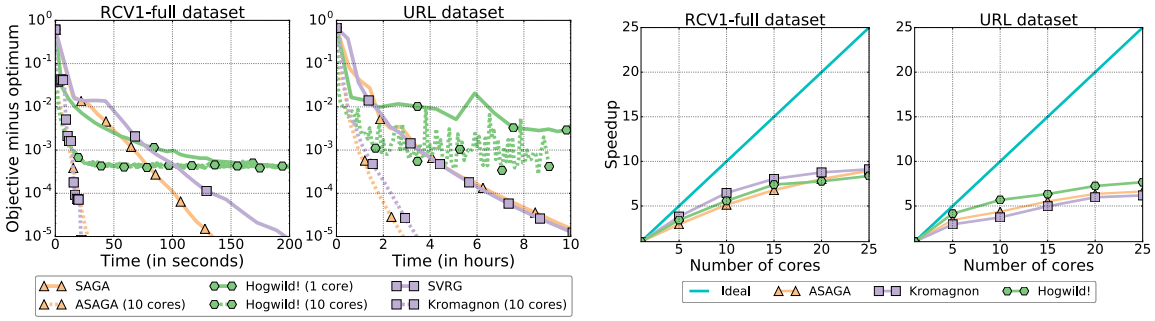
Figure 3-1 – **Lagged vs Sparse SAGA updates**. Suboptimality with respect to time for different SAGA update schemes on various datasets. First row: suboptimality as a function of time. Second row: suboptimality as a the number of passes over the dataset. For sparse datasets (RCV1 and Real-sim), lagged and sparse updates have a lower cost per iteration which result in faster convergence.

was fairly constant for different number of cores for both ASAGA and KROMAGNON, and both algorithms had similar best step sizes (0.7 for RCV1, 0.05 for URL and 5×10^{-5} for Covtype).

Hardware and software. Experiments were run on a 40-core machine with 384GB of memory. All algorithms were implemented in Scala. We chose this high-level language despite its typical 20x slowdown compared to C (when using standard libraries, see Appendix B.5.2) because our primary concern was that the code may easily be reused and extended for research purposes (to this end, we have made all our code available at `http://www.di.ens.fr/sierra/research/asaga/`).

3.4.3 Comparison of sequential algorithms: Sparse SAGA vs Lagged updates

We compare the Sparse SAGA variant proposed in Section 3.2.1 to two other approaches: the naive (i.e., dense) update scheme and the lagged updates implementation described in Defazio et al. (2014a). Note that we use different datasets from the parallel experiments,



(a) Suboptimality as a function of time.

(b) Speedup as a function of the number of cores

Figure 3-2 – **Convergence and speedup for asynchronous stochastic gradient descent methods**. We display results for RCV1 and URL. Results for Covtype can be found in Section 3.4.6.

including a subset of the RCV1 dataset and the Realsim dataset (see description in Appendix B.5.1). Figure 3-1 reveals that sparse and lagged updates have a lower cost per iteration than their dense counterpart, resulting in faster convergence for sparse datasets. Furthermore, while the two approaches had similar convergence in terms of number of iterations, the Sparse SAGA scheme is slightly faster in terms of runtime (and as previously pointed out, sparse updates are better adapted for the asynchronous setting). For the dense dataset (Covtype), the three approaches exhibit similar performance.

3.4.4 ASAGA vs. KROMAGNON vs. HOGWILD

We compare three different asynchronous variants of stochastic gradient methods on the aforementioned datasets: ASAGA, presented in this work, KROMAGNON, the asynchronous sparse SVRG method described in Mania et al. (2017) and HOGWILD (Niu et al., 2011). Each method had its step size chosen so as to give the fastest convergence (up to a suboptimality of 10^{-3} in the special case of HOGWILD). The results can be seen in Figure 3-2a: for each method we consider its asynchronous version with both one (hence sequential) and ten processors. This figure reveals that the asynchronous version offers a significant speedup over its sequential counterpart.

We then examine the speedup relative to the increase in the number of cores. The speedup is measured as time to achieve a suboptimality of 10^{-5} (10^{-3} for HOGWILD) with one core divided by time to achieve the same suboptimality with several cores, averaged over 3 runs. Again, we choose step size leading to fastest convergence [16] (see Appendix B.5.2 for information about the step sizes). Results are displayed in Figure 3-2b.

16. Although we performed grid search on a large interval, we observed that the best step size was fairly constant for different number of cores, and similar for ASAGA and KROMAGNON.

As predicted by our theory, we observe linear "theoretical" speedups (i.e. in terms of number of iterations, see Section 3.4.6). However, with respect to running time, the speedups seem to taper off after 20 cores. This phenomenon can be explained by the fact that our hardware model is by necessity a simplification of reality. As noted in Duchi et al. (2015), in a modern machine there is no such thing as *shared memory*. Each core has its own levels of cache (L1, L2, L3) in addition to RAM. These faster pools of memory are fully leveraged when using a single core. Unfortunately, as soon as several cores start writing to common locations, cache coherency protocols have to be deployed to ensure that the information is consistent across cores. These protocols come with computational overheads. As more and more cores are used, the shared information goes lower and lower in the memory stack, and the overheads get more and more costly. It may be the case that on much bigger datasets, where the cache memory is unlikely to provide benefits even for a single core (since sampling items repeatedly becomes rare), the running time speedups actually improve. More experimentation is needed to quantify these effects and potentially increase performance.

3.4.5 Effect of sparsity

Sparsity plays an important role in our theoretical results, where we find that while it is necessary in the "ill-conditioned" regime to get linear speedups, it is not in the "well-conditioned" regime. We confront this to real-life experiments by comparing the convergence and speedup performance of our three asynchronous algorithms on the Covtype dataset, which is fully dense after standardization. The results appear in Figure 3-3.

While we still see a significant improvement in speed when increasing the number of cores, this improvement is smaller than the one we observe for sparser datasets. The speedups we observe are consequently smaller, and taper off earlier than on our other datasets. However, since the observed "theoretical" speedup is linear (see Section 3.4.6), we can attribute this worse performance to higher hardware overhead. This is expected because each update is fully dense and thus the shared parameters are much more heavily contended for than in our sparse datasets.

One thing we notice when computing the Δ constant for our datasets is that it often fails to capture the full sparsity distribution, being essentially a maximum: for all three datasets, we obtain $\Delta = \mathcal{O}(1)$. This means that Δ can be quite big even for very sparse datasets. Deriving a less coarse bound remains an open problem.

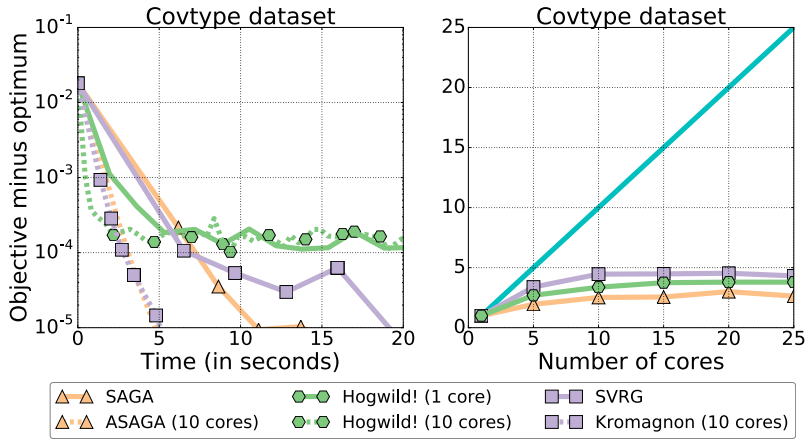


Figure 3-3 – Comparison on the Covtype dataset. Left: suboptimality. Right: speedup. The number of cores in the legend only refers to the left plot.

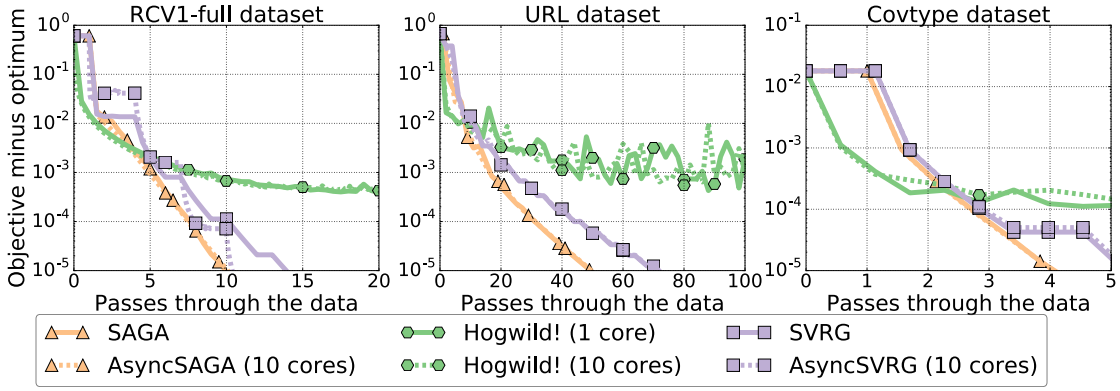


Figure 3-4 – **Theoretical speedups**. Suboptimality with respect to number of iterations for ASAGA, KROMAGNON and HOGWILD with 1 and 10 cores. Curves almost coincide, which means the theoretical speedup is almost the number of cores p, hence linear.

3.4.6 Theoretical speedups

In the previous experimental sections, we have shown experimental speedup results where suboptimality was a function of the running time. This measure encompasses both theoretical algorithmic properties and hardware overheads (such as contention of shared memory) which are not taken into account in our analysis.

In order to isolate these two effects, we now plot our convergence experiments where suboptimality is a function of the number of iterations; thus, we abstract away any potential hardware overhead. [17] The experimental results can be seen in Figure 3-4.

17. To do so, we implement a global counter which is sparsely updated (every 100 iterations for example) in order not to modify the asynchrony of the system. This counter is used only for plotting purposes and is not needed otherwise.

For all three algorithms and all three datasets, the curves for 1 and 10 cores almost coincide, which means that we are indeed in the "theoretical linear speedup" regime. Indeed, when we plotted the amount of iterations required to converge to a given accuracy as a function of the number of cores, we obtained straight horizontal lines for our three algorithms.

The fact that the speedups we observe in running time are less than linear can thus be attributed to various hardware overheads, including shared variable contention – the compare-and-swap operations are more and more expensive as the number of competing requests increases – and cache effects as mentioned in Section 3.4.4.

3.4.7 A closer look at the τ constant

Theory

In the parallel optimization literature, τ is often referred to as a proxy for the number of cores. However, intuitively as well as in practice, it appears that there are a number of other factors that can influence this quantity. We will now attempt to give a few qualitative arguments as to what these other factors might be and how they relate to τ.

Number of cores. The first of these factors is indeed the number of cores. If we have p cores, $\tau \geq p - 1$. Indeed, in the best-case scenario where all cores have exactly the same execution speed for a single iteration, $\tau = p - 1$.

Length of an iteration. To get more insight into what τ really encompasses, let us now try to define the worst-case scenario in the preceding example. Consider 2 cores. In the worst case, one core runs while the other is stuck. Then the overlap is t for all t and eventually grows to $+\infty$. If we assume that one core runs twice as fast as the other, then $\tau = 2$. If both run at the same speed, $\tau = 1$.

It appears then that a relevant quantity is R, the ratio between the fastest execution time and the slowest execution time for a single iteration. We have $\tau \leq (p-1)R$, which can be arbitrarily bigger than p.

There are several factors at play in R itself. These include:

- the speed of execution of the cores themselves (i.e. clock time).
- the data matrix itself. Different support sizes for f_i means different gradient computation times. If one f_i has support of size n while all the others have support of size 1 for example, R may eventually become very big.

- the length of the computation itself. The longer our algorithm runs, the more likely it is to explore the potential corner cases of the data matrix.

The overlap is then upper bounded by the number of cores multiplied by the ratio of the maximum iteration time over the minimum iteration time (which is linked to the sparsity distribution of the data matrix). This is an upper bound, which means that in some cases it will not really be useful. For example, if one factor has support size 1 and all others have support size d, the probability of the event which corresponds to the upper bound is exponentially small in d. We conjecture that a more useful indicator could be ratio of the maximum iteration time over the expected iteration time.

To sum up this preliminary theoretical exploration, the τ term encompasses much more complexity than is usually implied in the literature. This is reflected in the experiments we ran, where the constant was orders of magnitude bigger than the number of cores.

Experimental results

In order to verify our intuition about the τ variable, we ran several experiments on all three datasets, whose characteristics are reminded in Table 3.2. δ_l^i is the support size of f_i.

Table 3.2 – Density measures including minimum, average and maximum support size δ_l^i of the factors.

	n	d	density	$\max(\delta_l^i)$	$\min(\delta_l^i)$	$\bar{\delta}_l$	$\max(\delta_l^i)/\bar{\delta}_l$
RCV1	697,641	47,236	0.15%	1,224	4	73.2	16.7
URL	2,396,130	3,231,961	0.003%	414	16	115.6	3.58
Covtype	581,012	54	100%	12	8	11.88	1.01

To estimate τ, we compute the average overlap over 100 iterations, i.e. the difference in labeling between the end of the hundredth iteration and the start of the first iteration on, divided by 100. This quantity is a lower bound on the actual overlap (which is a maximum, not an average). We then take its maximum observed value. The reason why we use an average is that computing the overlap requires using a global counter, which we do not want to update every iteration since it would make it a heavily contentious quantity susceptible of artificially changing the asynchrony pattern of our algorithm.

The results we observe are order of magnitude bigger than p, indicating that τ can indeed not be dismissed as a mere proxy for the number of cores, but has to be more carefully analyzed.

First, we plot the maximum observed τ as a function of the number of cores (see Figure 3-5). We observe that the relationship does indeed seem to be roughly linear with

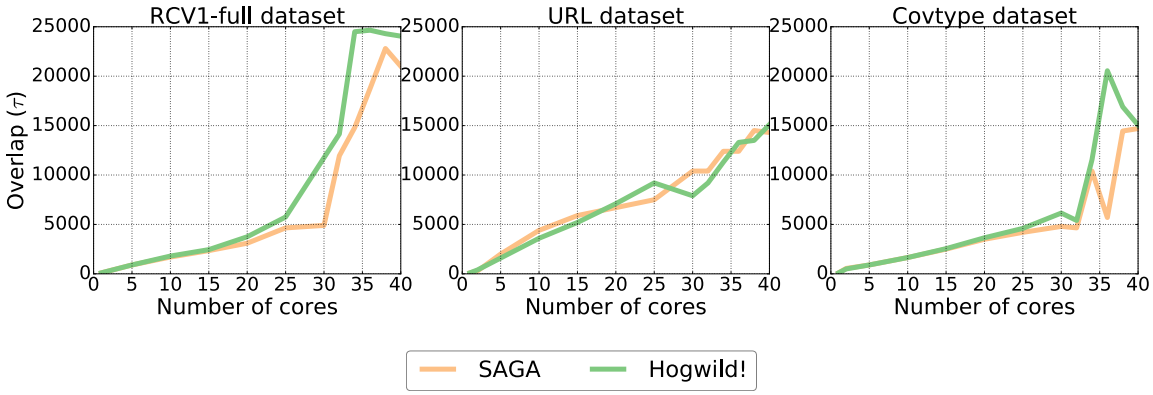


Figure 3-5 – **Overlap**. Overlap as a function of the number of cores for both ASAGA and HOGWILD on all three datasets.

respect to the number of cores until 30 cores. After 30 cores, we observe what may be a phase transition where the slope increases significantly.

Second, we measured the maximum observed τ as a function of the number of epochs. We omit the figure since we did not observe any dependency; that is, τ does not seem to depend on the number of epochs. We know that it must depend on the number of iterations (since it cannot be bigger, and is an increasing function with respect to that number for example), but it appears that a stable value is reached quite quickly (before one full epoch is done).

If we allowed the computations to run forever, we would eventually observe an event such that τ would reach the upper bound mentioned in Section 3.4.7, so it may be that τ is actually a very slowly increasing function of the number of iterations.

3.5 Conclusion and future work

Building on the recently proposed "perturbed iterate" framework, we have proposed a novel perspective to clarify an important technical issue present in a large fraction of the recent convergence rate proofs for asynchronous parallel optimization algorithms. To resolve it, we have introduced a novel "after read" framework and demonstrated its usefulness by analyzing three asynchronous parallel incremental optimization algorithms, including ASAGA, a novel sparse and fully asynchronous variant of the incremental gradient algorithm SAGA. Our proof technique accommodates more realistic settings than is usually the case in the literature (such as inconsistent reads and writes and an unbounded gradient); we obtain tighter conditions than in previous work. In particular, we show that ASAGA is linearly faster than SAGA under mild conditions, and that sparsity is not always necessary to get

linear speedups. Our empirical benchmarks confirm speedups up to $10\times$.

Schmidt et al. (2016) have shown that SAG enjoys much improved performance when combined with non-uniform sampling and line-search. We have also noticed that our Δ_r constant (being essentially a maximum) sometimes fails to accurately represent the full sparsity distribution of our datasets. Finally, while our algorithm can be directly ported to a distributed master-worker architecture, its communication pattern would have to be optimized to avoid prohibitive costs. Limiting communications can be interpreted as artificially increasing the delay, yielding an interesting trade-off between delay influence and communication costs.

These constitute interesting directions for future analysis, as well as a further exploration of the τ term, which we have shown encompasses more complexity than previously thought.

Chapter 4

Asynchronous composite optimization

In this chapter, we turn to a more general objective than in Chapters 2 and 3: we consider the case where the regularizer can be non-smooth. This problem is of interest in many machine learning applications, such as the Lasso, group Lasso or empirical risk minimization with convex constraints.

We introduce PROXASAGA, a fully asynchronous sparse method inspired by SAGA and its asynchronous parallel variant ASAGA. The proposed algorithm is easy to implement and significantly outperforms the state of the art on several non-smooth, large-scale problems.

Once again using the "after read" framework of asynchronous convergence analysis introduced in Chapter 2, we prove that our method achieves a theoretical linear speedup with respect to the sequential version under assumptions on the sparsity of gradients and block-separability of the proximal term. Empirical benchmarks on a multi-core architecture illustrate practical speedups of up to $12\times$ on a 20-core machine.

4.1 Introduction

As we have seen in Section 1.2.3, the widespread availability of multi-core computers motivates the development of parallel methods adapted for these architectures. One of the most popular approaches is HOGWILD (Niu et al., 2011), an asynchronous variant of stochastic gradient descent (SGD). In this algorithm, multiple threads run the update rule of SGD asynchronously in parallel. As SGD, it only requires visiting a small batch of random examples per iteration, which makes it ideally suited for large scale machine learning problems. Due to its simplicity and excellent performance, this parallelization approach has recently been extended to other variants of SGD with better convergence properties, such as SVRG (Johnson and Zhang, 2013) and SAGA (Defazio et al., 2014a).

Despite their practical success, existing parallel asynchronous variants of SGD are limited to smooth objectives, making them inapplicable to many problems in machine learning and signal processing. In this work, we develop a sparse variant of the SAGA algorithm and consider its parallel asynchronous variants for general *composite* optimization problems of the form:

$$\arg\min_{\boldsymbol{x}\in\mathbb{R}^d} f(\boldsymbol{x}) + h(\boldsymbol{x}), \qquad \text{with } f(\boldsymbol{x}) := \frac{1}{n}\sum_{i=1}^{n} f_i(\boldsymbol{x}) \quad , \tag{4.1}$$

where each f_i is convex with L-Lipschitz gradient, the average function f is μ-strongly convex and h is convex but potentially nonsmooth. We further assume that h is "simple" in the sense that we have access to its proximal operator, and that it is block-separable, that is, it can be decomposed block coordinate-wise as $h(\boldsymbol{x}) = \sum_{B\in\mathcal{B}} h_B([\boldsymbol{x}]_B)$, where $\mathcal{B}$ is a partition of the coefficients into subsets which will call *blocks* and h_B only depends on coordinates in block B.

Note that there is no loss of generality in this last assumption as a unique block covering all coordinates is a valid partition, though in this case, our sparse variant of the SAGA algorithm reduces to the original SAGA algorithm and no gain from sparsity is obtained.

This template models a broad range of problems arising in machine learning and signal processing: the finite-sum structure of f includes the least squares or logistic loss functions; the proximal term h includes penalties such as the ℓ_1 or group lasso penalty. Furthermore, this term can be extended-valued, thus allowing for convex constraints through the indicator function.

Contributions. This work presents two main contributions. First, in Section 4.2 we describe Sparse Proximal SAGA, a novel variant of the SAGA algorithm which features a reduced cost per iteration in the presence of sparse gradients and a block-separable penalty. Like other variance-reduced methods, it enjoys a linear convergence rate under strong convexity.

Second, in Section 4.3 we present PROXASAGA, a lock-free asynchronous parallel version of the aforementioned algorithm that does not require consistent reads. Our main results states that PROXASAGA obtains (under assumptions) a theoretical linear speedup with respect to its sequential version. Empirical benchmarks reported in Section 4.4 show that this method dramatically outperforms state-of-the-art alternatives on large sparse datasets, while the empirical speedup analysis illustrates the practical gains as well as its limitations.

4.1.1 Related work

Asynchronous coordinate-descent. For composite objective functions of the form (4.1), most of the existing literature on asynchronous optimization has focused on variants of coordinate descent. Liu and Wright (2015) proposed an asynchronous variant of (proximal) coordinate descent and proved a near-linear speedup in the number of cores used, given a suitable step size. This approach has been recently extended to general block-coordinate schemes by Peng et al. (2016), to greedy coordinate-descent schemes by You et al. (2016) and to non-convex problems by Davis et al. (2016). However, as illustrated by our experiments, in the large sample regime coordinate descent compares poorly against incremental gradient methods like SAGA.

Variance reduced incremental gradient and their asynchronous variants. Initially proposed in the context of smooth optimization by Le Roux et al. (2012), variance-reduced incremental gradient methods have since been extended to minimize composite problems of the form (4.1) (see table below). Smooth variants of these methods have also recently been extended to the asynchronous setting, where multiple threads run the update rule asynchronously and in parallel.

Interestingly, none of these methods achieve both simultaneously, i.e. asynchronous optimization of composite problems. Since variance-reduced incremental gradient methods have shown state of the art performance in both settings, this generalization is of key practical interest.

Objective	Sequential Algorithm	Asynchronous Algorithm
Smooth	SVRG (Johnson and Zhang, 2013)	SVRG (Reddi et al., 2015)
	SDCA (Shalev-Shwartz and Zhang, 2013)	PASSCODE (Hsieh et al., 2015, SDCA variant)
	SAGA (Defazio et al., 2014a)	ASAGA (Chapter 3, SAGA variant)
Composite	PROXSDCA (Shalev-Shwartz and Zhang, 2012)	This work: PROXASAGA
	SAGA (Defazio et al., 2014a)	
	ProxSVRG (Xiao and Zhang, 2014)	

On the difficulty of a composite extension. Two key issues explain the paucity in the development of asynchronous incremental gradient methods for composite optimization. The first issue is related to the design of such algorithms. Asynchronous variants of SGD are most competitive when the updates are sparse and have a small overlap, that is, when each update modifies a small and different subset of the coefficients. This is typically achieved

by updating only coefficients for which the partial gradient at a given iteration is nonzero, [1] but existing schemes such as the lagged updates technique (Schmidt et al., 2016) are not applicable in the asynchronous setting.

The second difficulty is related to the analysis of such algorithms. All convergence proofs crucially use the Lipschitz condition on the gradient to bound the noise terms derived from asynchrony. However, in the composite case, the gradient mapping term (Beck and Teboulle, 2009), which replaces the gradient in proximal-gradient methods, does not have a bounded Lipschitz constant. Hence, the traditional proof technique breaks down in this scenario.

Other approaches. Recently, Meng et al. (2017); Gu et al. (2016) independently proposed a doubly stochastic method to solve the problem at hand. Following Meng et al. (2017) we refer to it as Async-ProxSvrcd. This method performs coordinate descent-like updates in which the true gradient is replaced by its Svrg approximation. It hence features a doubly-stochastic loop: at each iteration we select a random coordinate *and* a random sample. Because the selected coordinate block is uncorrelated with the chosen sample, the algorithm can be orders of magnitude slower than Saga in the presence of sparse gradients. Section 4.4 contains a comparison of these methods.

4.1.2 Definitions and notations

By convention, we denote vectors and vector-valued functions in lowercase boldface (e.g. $\boldsymbol{x}$) and matrices in uppercase boldface (e.g. $\boldsymbol{D}$). The proximal operator of a convex lower semicontinuous function h is defined as $\mathbf{prox}_h(\boldsymbol{x}) := \arg\min_{\boldsymbol{z}\in\mathbb{R}^d}\{h(\boldsymbol{z}) + \frac{1}{2}\|\boldsymbol{x}-\boldsymbol{z}\|^2\}$. A function f is said to be L-smooth if it is differentiable and its gradient is L-Lipschitz continuous. A function f is said to be μ-strongly convex if $f - \frac{\mu}{2}\|\cdot\|^2$ is convex. We use the notation $\kappa := L/\mu$ to denote the condition number for an L-smooth and μ-strongly convex function. [2]

$\boldsymbol{I}_d$ denotes the d-dimensional identity matrix, $\mathbb{1}\{\text{cond}\}$ the characteristic function, which is 1 if cond evaluates to true and 0 otherwise. The average of a vector or matrix is denoted $\overline{\boldsymbol{\alpha}} := \frac{1}{n}\sum_{i=1}^n \boldsymbol{\alpha}_i$. We use $\|\cdot\|$ for the Euclidean norm. For a positive semi-definite matrix $\boldsymbol{D}$, we define its associated distance as $\|\boldsymbol{x}\|_{\boldsymbol{D}}^2 := \langle \boldsymbol{x}, \boldsymbol{D}\boldsymbol{x}\rangle$. We denote by $[\,\boldsymbol{x}\,]_b$ the b-th coordinate in $\boldsymbol{x}$. This notation is overloaded so that for a collection of blocks $T = \{B_1, B_2, \ldots\}$, $[\boldsymbol{x}]_T$

1. Although some regularizers are sparsity inducing, large scale datasets are often extremely sparse and leveraging this property is crucial for the efficiency of the method.
2. Since we have assumed that each individual f_i is L-smooth, f itself is L-smooth – but it could have a smaller smoothness constant. Our rates are in terms of this bigger L/μ, as is standard in the Saga literature.

denotes the vector $\boldsymbol{x}$ restricted to the coordinates in the blocks of T. For convenience, when T consists of a single block B we use $[\boldsymbol{x}]_B$ as a shortcut of $[\boldsymbol{x}]_{\{B\}}$.

Finally, we distinguish $\mathbb{E}$, the full expectation taken with respect to all the randomness in the system, from $\mathbf{E}$, the conditional expectation of a random i_t (the random feature sampled at each iteration by SGD-like algorithms) conditioned on all the "past", which the context will clarify.

4.2 Sparse Proximal SAGA

Original SAGA algorithm. The original SAGA algorithm (Defazio et al., 2014a) maintains two moving quantities: the current iterate $\boldsymbol{x}$ and a table (memory) of historical gradients $(\boldsymbol{\alpha}_i)_{i=1}^n$. At every iteration, it samples an index $i \in \{1, \dots, n\}$ uniformly at random, and computes the next iterate $(\boldsymbol{x}^+, \boldsymbol{\alpha}^+)$ according to the following recursion:

$$\boldsymbol{u}_i = \nabla f_i(\boldsymbol{x}) - \boldsymbol{\alpha}_i + \overline{\boldsymbol{\alpha}}\,; \quad \boldsymbol{x}^+ = \mathbf{prox}_{\gamma h}\big(\boldsymbol{x} - \gamma \boldsymbol{u}_i\big); \quad \boldsymbol{\alpha}_i^+ = \nabla f_i(\boldsymbol{x})\,. \tag{4.2}$$

On each iteration, this update rule requires to visit all coefficients even if the partial gradients ∇f_i are sparse. Sparse partial gradients arise in a variety of practical scenarios: for example, in generalized linear models the partial gradients inherit the sparsity pattern of the dataset. Given that large-scale datasets are often sparse, [3] leveraging this sparsity is crucial for the success of the optimizer.

Sparse Proximal SAGA algorithm. We will now describe an algorithm that leverages sparsity in the partial gradients by only updating those blocks that intersect with the support of the partial gradients. Since in this update scheme some blocks might appear more frequently than others, we will need to counterbalance this undersirable effect with a well-chosen block-wise reweighting of the average gradient and the proximal term.

In order to make precise this block-wise reweighting, we define the following quantities. We denote by T_i the *extended support* of ∇f_i, which is the set of blocks that intersect the support of ∇f_i, formally defined as $T_i := \{B : \mathrm{supp}(\nabla f_i) \cap B \neq \varnothing,\ B \in \mathcal{B}\}$. For totally separable penalties such as the ℓ_1 norm, the blocks are individual coordinates and so the extended support covers the same coordinates as the support. Let $d_B := n/n_B$, where $n_B := \sum_i \mathbb{1}\{B \in T_i\}$ is the number of times that $B \in T_i$. For simplicity we assume $n_B > 0$, as otherwise the problem can be reformulated without block B.

3. For example, in the `LibSVM` datasets suite, 8 out of the 11 datasets (as of May 2017) with more than a million samples have a density between 10^{-4} and 10^{-6}.

The update rule in (4.2) requires computing the proximal operator of h, which involves a full pass on the coordinates. In our proposed algorithm, we replace h in (4.2) with the function $\varphi_i(\boldsymbol{x}) := \sum_{B \in T_i} d_B h_B(\boldsymbol{x})$, whose form is justified by the following three properties. First, this function is zero outside T_i, allowing for sparse updates. Second, because of the block-wise reweighting d_B, the function φ_i is an unbiased estimator of h (i.e., $\mathbf{E}\, \varphi_i = h$), property which will be crucial to prove the convergence of the method. Third, φ_i inherits the block-wise structure of h and its proximal operator can be computed from that of h as $[\mathbf{prox}_{\gamma\varphi_i}(\boldsymbol{x})]_B = [\mathbf{prox}_{(d_B\gamma)h_B}(\boldsymbol{x})]_B$ if $B \in T_i$ and $[\mathbf{prox}_{\gamma\varphi_i}(\boldsymbol{x})]_B = [\boldsymbol{x}]_B$ otherwise.

As we did for ASAGA in Chapter 3, we will also replace the dense gradient estimate $\boldsymbol{u}_i$ by the sparse estimate $\boldsymbol{v}_i := \nabla f_i(\boldsymbol{x}) - \boldsymbol{\alpha}_i + \boldsymbol{D}_i \overline{\boldsymbol{\alpha}}$, where $\boldsymbol{D}_i$ is the diagonal matrix defined block-wise as $[\boldsymbol{D}_i]_{B,B} = d_B \mathbb{1}\{B \in T_i\} \boldsymbol{I}_{|B|}$. It is easy to verify that the vector $\boldsymbol{D}_i \overline{\boldsymbol{\alpha}}$ is a weighted projection onto the support of T_i and $\mathbf{E}\, \boldsymbol{D}_i \overline{\boldsymbol{\alpha}} = \overline{\boldsymbol{\alpha}}$, making $\boldsymbol{v}_i$ an unbiased estimate of the gradient.

We now have all necessary elements to describe the Sparse Proximal SAGA algorithm. As the original SAGA algorithm, it maintains two moving quantities: the current iterate $\boldsymbol{x} \in \mathbb{R}^p$ and a table of historical gradients $(\boldsymbol{\alpha}_i)_{i=1}^n$, $\boldsymbol{\alpha}_i \in \mathbb{R}^d$. At each iteration, the algorithm samples an index $i \in \{1, \dots, n\}$ and computes the next iterate $(\boldsymbol{x}^+, \boldsymbol{\alpha}^+)$ as:

$$\boldsymbol{v}_i = \nabla f_i(\boldsymbol{x}) - \boldsymbol{\alpha}_i + \boldsymbol{D}_i \overline{\boldsymbol{\alpha}}\,;\; \boldsymbol{x}^+ = \mathbf{prox}_{\gamma\varphi_i}\big(\boldsymbol{x} - \gamma \boldsymbol{v}_i\big)\,;\; \boldsymbol{\alpha}_i^+ = \nabla f_i(\boldsymbol{x})\,, \tag{4.3}$$

where in a practical implementation the vector $\overline{\boldsymbol{\alpha}}$ is updated incrementally at each iteration.

The above algorithm is sparse in the sense that it only requires to visit and update blocks in the extended support: if $B \notin T_i$, by the sparsity of $\boldsymbol{v}_i$ and $\mathbf{prox}_{\varphi_i}$, we have $[\boldsymbol{x}^+]_B = [\boldsymbol{x}]_B$. Hence, when the extended support T_i is sparse, this algorithm can be orders of magnitude faster than the naive SAGA algorithm. The extended support is sparse for example when the partial gradients are sparse and the penalty is separable, as is the case of the ℓ_1 norm or the indicator function over a hypercube, or when the the penalty is block-separable in a way such that only a small subset of the blocks overlap with the support of the partial gradients. Initialization of variables and a reduced storage scheme for the memory are discussed in Section 4.4.1.

Relationship with existing methods. This algorithm can be seen as a generalization of both the Standard SAGA algorithm and the Sparse SAGA algorithm of Chapter 3. When the proximal term is not block-separable, then $d_B = 1$ (for a unique block B) and the algorithm defaults to the Standard (dense) SAGA algorithm. In the smooth case (i.e., $h = 0$), the

algorithm defaults to the Sparse SAGA method. Hence we note that the sparse gradient estimate $\boldsymbol{v}_i$ in our algorithm is the same as the one proposed in Chapter 3.

However, we emphasize that a straightforward combination of this sparse update rule with the proximal update from the Standard SAGA algorithm results in a nonconvergent algorithm: the block-wise reweighting of h is a surprisingly simple but crucial change. We now give the convergence guarantees for this algorithm.

Theorem 27. *Let $\gamma = \frac{a}{5L}$ for any $a \leq 1$ and f be μ-strongly convex ($\mu > 0$). Then Sparse Proximal SAGA converges geometrically in expectation with a rate factor of at least $\rho = \frac{1}{5}\min\{\frac{1}{n}, a\frac{1}{\kappa}\}$. That is, for $\boldsymbol{x}_t$ obtained after t updates, we have the following bound:*

$$\mathbb{E}\|\boldsymbol{x}_t - \boldsymbol{x}^*\|^2 \leq (1-\rho)^t C_0\,, \quad \textit{with } C_0 := \|\boldsymbol{x}_0 - \boldsymbol{x}^*\|^2 + \tfrac{1}{5L^2}\textstyle\sum_{i=1}^n \|\boldsymbol{\alpha}_i^0 - \nabla f_i(\boldsymbol{x}^*)\|^2\,.$$

Remark. For the step size $\gamma = 1/5L$, the convergence rate is $(1 - 1/5\min\{1/n, 1/\kappa\})$. We can thus identify two regimes: the "big data" regime, $n \geq \kappa$, in which the rate factor is bounded by $1/5n$, and the "ill-conditioned" regime, $\kappa \geq n$, in which the rate factor is bounded by $1/5\kappa$. This rate roughly matches the rate obtained by Defazio et al. (2014a).

While the step size bound of $1/5L$ is slightly smaller than the $1/3L$ one obtained in that work, this can be explained by their stronger assumptions: each f_i is strongly convex whereas they are strongly convex only on average in this work.

Key differences in the proof technique compared to Sparse SAGA (Section 3.2.1). All proofs for this section can be found in Appendix C.2. Although the general structure is similar to the one of Sparse SAGA, there are several significant differences.

- First, as the update rule is different (notably with a proximal term which depends on the sampled data point i_t), one has to prove that the gradient estimator is still unbiased, which we do in Lemma 35.
- Second, because we are in the composite setup (4.1), we do not have $f'(x^*) = 0$ and thus we need an alternative characterization of the solutions of (4.1) in terms of f and φ. We provide it in Lemma 36:

$$\boldsymbol{x}^* = \mathbf{prox}_{\gamma\varphi}\big(\boldsymbol{x}^* - \gamma \boldsymbol{D}\nabla f(\boldsymbol{x}^*)\big)\,, \tag{4.4}$$

  i.e. $\boldsymbol{x}^*$ is a fixed point of the expected update.
- Third, again because we do not have $f'(\boldsymbol{x}^*) = 0$ (as in the simpler case analyzed in Chapter 3), we have to use the Bregman divergence term $B_f(\boldsymbol{x}_t, \boldsymbol{x}^*) := f(\boldsymbol{x}_t) -$

$f(\boldsymbol{x}^*) - \langle \nabla f(\boldsymbol{x}^*), \boldsymbol{x}_t - \boldsymbol{x}^* \rangle$ instead of a simple suboptimality term $f(x_t) - f(x^*)$ throughout the proof.

- Finally, instead of a gradient estimator term, a gradient mapping term $\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{v}, i) := \frac{1}{\gamma}\big(\boldsymbol{x} - \mathbf{prox}_{\gamma\varphi_i}(\boldsymbol{x} - \gamma \boldsymbol{v})\big)$ appears in the analysis. This quantity is crucially not Lipschitz-smooth, so we have to derive a specialized inequality linking it to the other terms that arise in the optimization. We do this in Lemma 37.

4.3 Asynchronous Sparse Proximal SAGA

We introduce ProxASaga – the asynchronous parallel variant of Sparse Proximal Saga. In this algorithm, multiple cores update a central parameter vector using the Sparse Proximal Saga introduced in the previous section, and updates are performed asynchronously. The algorithm parameters are read and written without vector locks, i.e., the vector content of the shared memory can potentially change while a core is reading or writing to main memory coordinate by coordinate. These operations are typically called *inconsistent* (at the vector level).

The full algorithm is described in Algorithm 7 for its theoretical version (on which our analysis is built) and in Algorithm 8 for its practical implementation. The practical implementation differs from the analyzed algorithm in three points. First, in the implemented algorithm, index i is sampled before reading the coefficients to minimize memory access since only the extended support needs to be read. Second, since our implementation targets generalized linear models, the memory $\boldsymbol{\alpha}_i$ can be compressed into a single scalar in line 20 (see Section 4.4.1). Third, $\overline{\boldsymbol{\alpha}}$ is stored in memory and updated incrementally instead of recomputed at each iteration.

The rest of the section is structured as follows: we start by describing our framework of analysis; we then derive essential properties of ProxASaga along with a classical delay assumption. Finally, we state our main convergence and speedup result.

4.3.1 Analysis framework

As in most of the recent asynchronous optimization literature, we build on the hardware model introduced by Niu et al. (2011), with multiple cores reading and writing to a shared memory parameter vector. These operations are asynchronous (lock-free) and *inconsistent*: [4] $\hat{\boldsymbol{x}}_t$, the local copy of the parameters of a given core, does not necessarily correspond to a consistent iterate in memory.

4. This is an extension of the framework of Niu et al. (2011), where consistent updates were assumed.

Algorithm 7 PROXASAGA (analyzed)

Initialize shared variables $\boldsymbol{x}$ and $(\boldsymbol{\alpha}_i)_{i=1}^n$
keep doing in parallel
  $\hat{\boldsymbol{x}} =$ inconsistent read of $\boldsymbol{x}$
  $\hat{\boldsymbol{\alpha}} =$ inconsistent read of $\boldsymbol{\alpha}$
  Sample i uniformly in $\{1, ..., n\}$
  $S_i :=$ support of ∇f_i
  $T_i :=$ extended support of ∇f_i in $\mathcal{B}$
  $[\,\overline{\boldsymbol{\alpha}}\,]_{T_i} = {}^1\!/_n \sum_{j=1}^n [\,\hat{\boldsymbol{\alpha}}_j\,]_{T_i}$
  $[\,\delta\boldsymbol{\alpha}\,]_{S_i} = [\nabla f_i(\hat{\boldsymbol{x}})]_{S_i} - [\hat{\boldsymbol{\alpha}}_i]_{S_i}$
  $[\,\hat{\boldsymbol{v}}\,]_{T_i} = [\,\delta\boldsymbol{\alpha}\,]_{T_i} + [\boldsymbol{D}_i\overline{\boldsymbol{\alpha}}\,]_{T_i}$
  $[\,\delta\boldsymbol{x}\,]_{T_i} = [\mathbf{prox}_{\gamma\varphi_i}(\hat{\boldsymbol{x}} - \gamma\hat{\boldsymbol{v}})]_{T_i} - [\hat{\boldsymbol{x}}]_{T_i}$
  for B **in** T_i **do**
    for $b \in B$ **do**
      $[\,\boldsymbol{x}\,]_b \leftarrow [\,\boldsymbol{x}\,]_b + [\,\delta\boldsymbol{x}\,]_b$
      if $b \in S_i$ **then**
        $[\boldsymbol{\alpha}_i]_b \leftarrow [\nabla f_i(\hat{\boldsymbol{x}})]_b$
      end if
    end for
  end for
  // '$\leftarrow$' denotes atomic shared memory update
end parallel loop

Algorithm 8 PROXASAGA (implemented)

Initialize shared variables $\boldsymbol{x}$, $(\boldsymbol{\alpha}_i)_{i=1}^n$, $\overline{\boldsymbol{\alpha}}$
keep doing in parallel
  Sample i uniformly in $\{1, ..., n\}$
  $S_i :=$ support of ∇f_i
  $T_i :=$ extended support of ∇f_i in $\mathcal{B}$
  $[\,\hat{\boldsymbol{x}}\,]_{T_i} =$ inconsistent read of $\boldsymbol{x}$ on T_i
  $\hat{\boldsymbol{\alpha}}_i =$ inconsistent read of $\boldsymbol{\alpha}_i$
  $[\,\overline{\boldsymbol{\alpha}}\,]_{T_i} =$ inconsistent read of $\overline{\boldsymbol{\alpha}}$ on T_i
  $[\,\delta\boldsymbol{\alpha}\,]_{S_i} = [\nabla f_i(\hat{\boldsymbol{x}})]_{S_i} - [\hat{\boldsymbol{\alpha}}_i]_{S_i}$
  $[\,\hat{\boldsymbol{v}}\,]_{T_i} = [\delta\boldsymbol{\alpha}\,]_{T_i} + [\,\boldsymbol{D}_i\overline{\boldsymbol{\alpha}}\,]_{T_i}$
  $[\,\delta\boldsymbol{x}\,]_{T_i} = [\mathbf{prox}_{\gamma\varphi_i}(\hat{\boldsymbol{x}} - \gamma\hat{\boldsymbol{v}})]_{T_i} - [\hat{\boldsymbol{x}}]_{T_i}$
  for B **in** T_i **do**
    for b **in** B **do**
      $[\,\boldsymbol{x}\,]_b \leftarrow [\,\boldsymbol{x}\,]_b + [\,\delta\boldsymbol{x}\,]_b$
      if $b \in S_i$ **then**
        $[\,\overline{\boldsymbol{\alpha}}\,]_b \leftarrow [\overline{\boldsymbol{\alpha}}]_b + {}^1\!/_n[\delta\boldsymbol{\alpha}]_b$
      end if
    end for
  end for
  $\boldsymbol{\alpha}_i \leftarrow \nabla f_i(\hat{\boldsymbol{x}})$ (scalar update)
end parallel loop

"Perturbed" iterates. To handle this additional difficulty, contrary to most contributions in this field, we choose the "after read" framework proposed in Chapter 2. We now give a brief summary of this framework's characteristics. It is useful to analyze variants of SGD which obey the update rule:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \gamma \boldsymbol{v}(\boldsymbol{x}_t, i_t)\,, \text{ where } \boldsymbol{v} \text{ verifies the unbiasedness condition: } \mathbf{E}\,\boldsymbol{v}(\boldsymbol{x}, i_t) = \nabla f(\boldsymbol{x})$$

and the expectation is computed with respect to i_t. In the asynchronous parallel setting, cores are reading inconsistent iterates from memory, which we denote $\hat{\boldsymbol{x}}_t$. As these inconsistent iterates are affected by various delays induced by asynchrony, they cannot easily be written as a function of their previous iterates. To alleviate this issue, we introduce an additional quantity for the purpose of the analysis:

$$\boldsymbol{x}_{t+1} := \boldsymbol{x}_t - \gamma \boldsymbol{v}(\hat{\boldsymbol{x}}_t, i_t), \tag{4.5}$$

the "virtual iterate" – which is never actually computed. Note that this equation is the *definition* of this new quantity $\boldsymbol{x}_t$. This virtual iterate is useful for the convergence analysis

and makes for much easier proofs than in the related literature.

"After read" labeling. How we choose to define the iteration counter t to label an iterate $\boldsymbol{x}_t$ matters in the analysis. In this chapter, we again follow the "after read" labeling proposed in Chapter 2, in which we update our iterate counter, t, as each core *finishes reading* its copy of the parameters (in the specific case of ProxASAGA, this includes both $\hat{\boldsymbol{x}}_t$ and $\hat{\boldsymbol{\alpha}}^t$). This means that $\hat{\boldsymbol{x}}_t$ is the $(t+1)^{th}$ fully completed read. One key advantage of this approach compared to the classical choice of Niu et al. (2011) – where t is increasing after each successful update – is that it guarantees both that the i_t are uniformly distributed and that i_t and $\hat{\boldsymbol{x}}_t$ are independent. This property is not verified when using the "after write" labeling of Niu et al. (2011), although it is still implicitly assumed in the papers using this approach, see Section 2.2.2 for a discussion of issues related to the different labeling schemes.

Generalization to composite optimization. Although the perturbed iterate framework was designed for gradient-based updates, we can extend it to proximal methods by remarking that in the sequential setting, proximal stochastic gradient descent and its variants can be characterized by the following similar update rule:

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \gamma \boldsymbol{g}(\boldsymbol{x}_t, \boldsymbol{v}_{i_t}, i_t)\,, \qquad \text{with } \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{v}, i) := \tfrac{1}{\gamma}\big(\boldsymbol{x} - \mathbf{prox}_{\gamma\varphi_i}(\boldsymbol{x} - \gamma \boldsymbol{v})\big)\,, \tag{4.6}$$

where as before $\boldsymbol{v}$ verifies the unbiasedness condition $\mathbf{E}\,\boldsymbol{v} = \nabla f(\boldsymbol{x})$. The Proximal Sparse SAGA iteration can be easily written within this template by using φ_i and $\boldsymbol{v}_i$ as defined in Section 4.2. Using this definition of $\boldsymbol{g}$, we can define ProxASAGA virtual iterates as:

$$\boldsymbol{x}_{t+1} := \boldsymbol{x}_t - \gamma \boldsymbol{g}(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{v}}^t_{i_t}, i_t)\,, \qquad \text{with } \hat{\boldsymbol{v}}^t_{i_t} = \nabla f_{i_t}(\hat{\boldsymbol{x}}_t) - \hat{\boldsymbol{\alpha}}^t_{i_t} + \boldsymbol{D}_{i_t}\overline{\boldsymbol{\alpha}}^t\,, \tag{4.7}$$

where as in the sequential case, the memory terms are updated as $\hat{\boldsymbol{\alpha}}^t_{i_t} = \nabla f_{i_t}(\hat{\boldsymbol{x}}_t)$. Our theoretical analysis of ProxASAGA will be based on this definition of the virtual iterate $\boldsymbol{x}_{t+1}$.

4.3.2 Properties and assumptions

Having adapted the "after read" labeling for proximal methods in Eq. (4.7), we can leverage this framework to derive the same essential properties for the analysis of ProxASAGA, as those we have done for HOGWILD, ASAGA and KROMAGNON in Section 2.3.1. We describe below these three useful properties arising from the definition of Algorithm 7, and

then re-state the central (but standard) assumption that the delays induced by the asynchrony are uniformly bounded.

Independence. Due to the "after read" global ordering, i_r is independent of $\hat{\boldsymbol{x}}_t$ for all $r \geq t$. We enforce the independence for $r = t$ by having the cores read all the shared parameters before their iterations.

Unbiasedness. The term $\hat{\boldsymbol{v}}_{i_t}^t$ is an unbiased estimator of the gradient of f at $\hat{\boldsymbol{x}}_t$. This property is a consequence of the independence between i_t and $\hat{\boldsymbol{x}}_t$.

Atomicity. The shared parameter coordinate update of $[\boldsymbol{x}]_b$ on line 14 is atomic. This means that there are no overwrites for a single coordinate even if several cores compete for the same resources. Most modern processors have support for atomic operations with minimal overhead.

Bounded overlap assumption. We assume that there exists a uniform bound, τ, on the maximum number of overlapping iterations. This means that every coordinate update from iteration t is successfully written to memory before iteration $t + \tau + 1$ starts. Our result will give us conditions on τ to obtain linear speedups.

Bounding $\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t$. The delay assumption of the previous paragraph allows to express the difference between real and virtual iterate using the gradient mapping $\boldsymbol{g}_u := \boldsymbol{g}(\hat{\boldsymbol{x}}_u, \hat{\boldsymbol{v}}_{i_u}^u, i_u)$, following the model (3.9):

$$\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t = \gamma \sum_{u=(t-\tau)_+}^{t-1} \boldsymbol{G}_u^t \boldsymbol{g}_u \,, \tag{4.8}$$

where $\boldsymbol{G}_u^t$ are $d \times d$ diagonal matrices with terms in $\{0, +1\}$. 0 represents instances where both $\hat{\boldsymbol{x}}_u$ and $\boldsymbol{x}_u$ have received the corresponding updates. $+1$, on the contrary, represents instances where $\hat{\boldsymbol{x}}_u$ has not yet received an update that is already in $\boldsymbol{x}_u$ by definition. This bound will prove essential to our analysis, as was the case in previous chapters.

4.3.3 Theoretical results

In this section, we state our convergence and speedup results for PROXASAGA. The outline of the analysis can be found in Section 4.3.4 and its full details in Appendix C.3. As in Chapters 2 and 3, we introduce a sparsity measure that will appear in our results. Note

that compared to the quantity given in Definition 7, this quantity has been generalized to the composite setting.

Definition 28. *Let* $\Delta := \max_{B \in \mathcal{B}} |\{i : T_i \ni B\}|/n$. *This is the normalized maximum number of times that a block appears in the extended support. For example, if a block is present in all* T_i, *then* $\Delta = 1$. *If no two* T_i *share the same block, then* $\Delta = 1/n$. *We always have* $1/n \leq \Delta \leq 1$.

Theorem 29 (Convergence guarantee of PROXASAGA)**.** *Suppose* $\tau \leq \frac{1}{10\sqrt{\Delta}}$. *For any step size* $\gamma = \frac{a}{L}$ *with* $a \leq a^*(\tau) := \frac{1}{36}\min\{1, \frac{6\kappa}{\tau}\}$, *the inconsistent read iterates of Algorithm 7 converge in expectation at a geometric rate factor of at least:* $\rho(a) = \frac{1}{5}\min\left\{\frac{1}{n}, a\frac{1}{\kappa}\right\}$, *i.e.* $\mathbb{E}\|\hat{\boldsymbol{x}}_t - \boldsymbol{x}^*\|^2 \leq (1-\rho)^t \tilde{C}_0$, *where* $\tilde{C}_0$ *is a constant independent of* t ($\approx \frac{n\kappa}{a} C_0$ *with* C_0 *as defined in Theorem 27).*

This last result is similar to the original SAGA convergence result and our own Theorem 27, with both an extra condition on τ and on the maximum allowable step size. In the best sparsity case, $\Delta = 1/n$ and we get the condition $\tau \leq \sqrt{n}/10$. We now compare the geometric rate above to the one of Sparse Proximal SAGA to derive the necessary conditions under which PROXASAGA is linearly faster.

Corollary 30 (Speedup)**.** *Suppose* $\tau \leq \frac{1}{10\sqrt{\Delta}}$. *If* $\kappa \geq n$, *then using the step size* $\gamma = 1/36L$, PROXASAGA *converges geometrically with rate factor* $\Omega(\frac{1}{\kappa})$. *If* $\kappa < n$, *then using the step size* $\gamma = 1/36n\mu$, PROXASAGA *converges geometrically with rate factor* $\Omega(\frac{1}{n})$. *In both cases, the convergence rate is the same as Sparse Proximal* SAGA. *Thus* PROXASAGA *is linearly faster than its sequential counterpart up to a constant factor. Note that in both cases* the step size does not depend on τ.

Furthermore, if $\tau \leq 6\kappa$, *we can use a universal step size of* $\Theta(1/L)$ *to get a similar rate for* PROXASAGA *than Sparse Proximal* SAGA, *thus making it adaptive to local strong convexity since the knowledge of* κ *is not required.*

4.3.4 Proof outline

In this section we give the outline of the proofs for the convergence and speedup results, underlining the similarities and the differences to the proof technique developed for ASAGA (Section 3.2.3). The technical details of the proofs can be found in Appendix C.3.

Notation. Throughout this section, we use the following shorthand for the gradient mapping: $\boldsymbol{g}_t := \boldsymbol{g}(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{v}}^t_{i_t}, i_t)$.

As in the smooth case ($h = 0$), we start by using the definition of $\boldsymbol{x}_{t+1}$ in Eq. (4.7) to relate the distance to the optimum in terms of its previous iterates:

$$\|\boldsymbol{x}_{t+1} - \boldsymbol{x}^*\|^2 = \|\boldsymbol{x}_t - \boldsymbol{x}^*\|^2 + 2\gamma\langle\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t, \boldsymbol{g}_t\rangle + \gamma^2\|\boldsymbol{g}_t\|^2 - 2\gamma\langle\hat{\boldsymbol{x}}_t - \boldsymbol{x}^*, \boldsymbol{g}_t\rangle\,. \tag{4.9}$$

However, in this case $\boldsymbol{g}_t$ is not a gradient estimator but a gradient mapping, so we cannot continue as is customary – by using the unbiasedness of the gradient in the $\langle\hat{\boldsymbol{x}}_t - \boldsymbol{x}^*, \boldsymbol{g}_t\rangle$ term together with the strong convexity of f (see Section 2.3.3).

To circumvent this difficulty, we derive a tailored inequality for the gradient mapping (Lemma 37 in Appendix C.2), which in turn allows us to use the classical unbiasedness and strong convexity arguments to get the following inequality:

$$a_{t+1} \leq (1 - \frac{\gamma\mu}{2})a_t + \gamma^2\mathbb{E}\|\boldsymbol{g}_t\|^2 - 2\gamma\mathbb{E}B_f(\hat{\boldsymbol{x}}_t, \boldsymbol{x}^*) + \underbrace{\gamma\mu\mathbb{E}\|\hat{\boldsymbol{x}}_t - \boldsymbol{x}\|^2 + 2\gamma\mathbb{E}\langle\boldsymbol{g}_t, \hat{\boldsymbol{x}}_t - \boldsymbol{x}_t\rangle}_{\text{additional asynchrony terms}}$$
$$\underbrace{+\gamma^2(\beta - 2)\mathbb{E}\|\boldsymbol{g}_t\|^2 + \frac{\gamma^2}{\beta}\mathbb{E}\|\hat{\boldsymbol{v}}_{i_t}^t - \boldsymbol{D}_{i_t}\nabla f(\boldsymbol{x}^*)\|^2}_{\text{additional proximal and variance terms}}\,, \tag{4.10}$$

where $a_t := \mathbb{E}\|\boldsymbol{x}_t - \boldsymbol{x}^*\|^2$. Note that since f is strongly convex, $B_f(\hat{\boldsymbol{x}}_t, \boldsymbol{x}^*) \geq \frac{\mu}{2}\|\hat{\boldsymbol{x}}_t - \boldsymbol{x}^*\|^2$.

In the smooth setting, one first expresses the additional asynchrony terms as linear combinations of past gradient variance terms $(\mathbb{E}\|\boldsymbol{g}_u\|^2)_{0\leq u\leq t}$. Then one crucially uses the negative Bregman divergence term to control the variance terms. However, in our current setting, we cannot relate the norm of the gradient mapping $\mathbb{E}\|\boldsymbol{g}_t\|^2$ to the Bregman divergence (from which h is absent). Instead, we use the negative term $\gamma^2(\beta - 1)\mathbb{E}\|\boldsymbol{g}_t\|^2$ to control all the $(\mathbb{E}\|\boldsymbol{g}_u\|^2)_{0\leq u\leq t}$ terms that arise from asynchrony.

The rest of the proof consists in:

i) expressing the additional asynchrony terms as linear combinations of $(\mathbb{E}\|\boldsymbol{g}_u\|^2)_{0\leq u\leq t}$, following Lemma 11;

ii) expressing the last variance term, $\|\hat{\boldsymbol{v}}_{i_t}^t - D_{i_t}\nabla f(\boldsymbol{x}^*)\|^2$, as a linear combination of past Bregman divergences (Lemma 38 in Appendix C.2 and Lemma 14);

iii) defining a Lyapunov function, $\mathcal{L}_t := \sum_{u=0}^t (1-\rho)^{t-u}a_u$, and proving that it is bounded by a contraction given conditions on the maximum step size and delay.

4.3.5 Comparison to related work

In this section, we relate our theoretical results and proof technique with the related literature.

Speedups. Our speedup regimes are comparable with the best ones obtained in the smooth case, including Niu et al. (2011); Reddi et al. (2015), even though unlike these papers, we support inconsistent reads and nonsmooth objective functions. The one exception is Leblond et al. (2017), where the authors prove that their algorithm, ASAGA, can obtain a linear speedup even without sparsity in the well-conditioned regime. In contrast, PROXASAGA always requires some sparsity. Whether this property for smooth objective functions could be extended to the composite case remains an open problem.

Coordinate Descent. We compare our approach for composite objective functions to its most natural competitor: ASYSPCD (Liu and Wright, 2015), an asynchronous stochastic coordinate descent algorithm. While ASYSPCD also exhibits linear speedups, subject to a condition on τ, one has to be especially careful when trying to compare these conditions.

First, while in theory the iterations of both algorithms have the same cost, in practice various tricks are introduced to save on computation, yielding different costs per updates. [5] Second, the bound on τ for the coordinate descent algorithm depends on d, the dimensionality of the problem, whereas ours involves n, the number of data points. Third, a more subtle issue is that τ is not affected by the same quantities for both algorithms. [6] See Appendix C.4 for a more detailed explanation of the differences between the bounds.

In the best case scenario (where the components of the gradient are uncorrelated, a somewhat unrealistic setting), ASYSPCD can get a near-linear speedup for τ as big as $\sqrt[4]{d}$. Our result states that $\tau = \mathcal{O}(1/\sqrt{\Delta})$ is necessary for a linear speedup. This means in case $\Delta \leq 1/\sqrt{d}$ our bound is better than the one obtained for ASYSPCD.

Recalling that $1/n \leq \Delta \leq 1$, it appears that PROXASAGA is favored when n is bigger than $\sqrt{d}$ whereas ASYSPCD may have a better bound otherwise, though this comparison should be taken with a grain of salt given the assumptions we had to make to arrive at comparable quantities.

Furthermore, one has to note that while Liu and Wright (2015) use the classical labeling scheme inherited from Niu et al. (2011), they still assume in their proof that the i_t are uniformly distributed and that their gradient estimators are conditionally unbiased – though neither property is verified in the general asynchronous setting. Finally, we note that ASYSPCD (as well as its incremental variant Async-PROXSVRCD) assumes that the computation and assignment of the proximal operator is an atomic step, while we do not

5. For PROXASAGA the relevant quantity becomes the average number of features per data point. For ASYSPCD it is rather the average number of data points per feature. In both cases the tricks involved are not covered by the theory.

6. To make sure τ is the same quantity for both algorithms, we have to assume that the iteration costs are homogeneous.

make such assumption.

SVRG. The Async-ProxSVRG algorithm of Meng et al. (2017) also exhibits theoretical linear speedups subject to the same condition as ours. However, the analyzed algorithm uses dense updates and consistent read and writes. Although they make the analysis easier, these two factors introduce costly bottlenecks and prevent linear speedups in running time. Furthermore, here again the classical labeling scheme is used together with the unverified conditional unbiasedness condition.

Doubly stochastic algorithms. The Async-PROXSVRCD algorithm from Meng et al. (2017); Gu et al. (2016) has a maximum allowable stepsize [7] that is in $\mathcal{O}(1/dL)$, whereas the maximum step size for PROXASAGA is in $\Omega(1/L)$, so can be up to d times bigger. Consequently, PROXASAGA enjoys much faster theoretical convergence rates. Unfortunately, we could not find a condition for linear speedups to compare to. We also note that their algorithm is not appropriate in a sparse features setting. This is illustrated in an empirical comparison in Section 4.4 where we see that their convergence in number of iterations is orders of magnitude slower than appropriate algorithms like SAGA or PROXASAGA.

4.4 Experiments

In this section, we perform a detailed comparison of Sparse Proximal SAGA and PROXASAGA with related methods on different datasets. Although these algorithms can be applied more broadly, we focus on $\ell_1+\ell_2$-regularized logistic regression, a model of particular practical importance. The objective function takes the form

$$\frac{1}{n}\sum_{i=1}^{n}\log\left(1+\exp(-b_i\boldsymbol{a}_i^\intercal\boldsymbol{x})\right)+\tfrac{\lambda_1}{2}\|\boldsymbol{x}\|_2^2+\lambda_2\|\boldsymbol{x}\|_1\,, \tag{4.11}$$

where $\boldsymbol{a}_i\in\mathbb{R}^d$ and $b_i\in\{-1,+1\}$ are the data samples. Following Defazio et al. (2014a), we set $\lambda_1=1/n$. The amount of ℓ_1 regularization (λ_2) is selected to give an approximate $1/10$ nonzero coefficients. We chose the 3 datasets described in Table 4.1, which were all downloaded from the LibSVM dataset suite. [8]

7. To the best of our understanding, noting that extracting an interpretable bound from the given theoretical results was difficult. Furthermore, it appears that the proof technique may still have significant issues: for example, the "fully lock-free" assumption of Gu et al. (2016) allows for overwrites, and is thus incompatible with their framework of analysis, in particular their Eq. (8).

8. https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

Table 4.1 – Description of datasets.

Dataset	n	d	density	L	Δ
KDD 2010 (Yu et al., 2010)	19,264,097	1,163,024	10^{-6}	28.12	0.15
KDD 2012 (Juan et al., 2016)	149,639,105	54,686,452	2×10^{-7}	1.25	0.85
Criteo (Juan et al., 2016)	45,840,617	1,000,000	4×10^{-5}	1.25	0.89

4.4.1 Implementation details

We start by given an in-depth description of our implementation. We use most of the same settings as in Chapter 3, though we chose the C++ language instead of Scala for these experiments.

Initialization. In the Sparse Proximal SAGA algorithm and its asynchronous variant, PROXASAGA, the vector $\boldsymbol{x}$ can be initialized arbitrarily. The memory terms $\boldsymbol{\alpha}_i$ can be initialized to any vector that verifies $\text{supp}(\boldsymbol{\alpha}_i) = \text{supp}(\nabla f_i)$. In practice, as in Chapter 3 we found that the initialization $\boldsymbol{\alpha}_i = \boldsymbol{0}$ is very fast to set up and often outperforms more costly initializations.

With this initialization, the gradient approximation before the first update of the memory terms becomes $\nabla f_i(\boldsymbol{x}) + \boldsymbol{D}_i \overline{\boldsymbol{\alpha}}$. Since most of the values in $\boldsymbol{\alpha}$ are zero, $\overline{\boldsymbol{\alpha}}$ will tend to be small compared to $\nabla f_i(\boldsymbol{x})$, and so the gradient estimate is very close to the SGD estimate $\nabla f_i(\boldsymbol{x})$. The SGD approximation is known to have a very fast initial convergence (which, in light of Figure 4-2, our method inherits) and has even been used as a heuristic to use during the first epoch of variance-reduced methods (Schmidt et al., 2016).

The initialization of coefficients $\boldsymbol{x}_0$ was always set to zero.

Regularization. Computing the gradient of a smooth regularization such as the squared ℓ_2 penalty of Eq. (4.11) is independent of n and so we can use the exact regularizer in the update of the coefficients instead of storing it in $\boldsymbol{\alpha}$, which would also invalidate the compressed storage of the memory terms described below. In practice, as in Chapter 3 we use this "exact regularization", multiplied by $\boldsymbol{D}_i$ to preserve the sparsity pattern.

Assuming a squared ℓ_2 regularization term of the form $\frac{\lambda}{2}$, the gradient estimate in (4.3) becomes (note the extra $\lambda \boldsymbol{x}$)

$$\boldsymbol{v}_i = \nabla f_i(\boldsymbol{x}) - \boldsymbol{\alpha}_i + \boldsymbol{D}_i(\overline{\boldsymbol{\alpha}} + \lambda \boldsymbol{x}) \,. \tag{4.12}$$

Hyperparameters. The ℓ_1-regularization parameter λ_2 was chosen as to give around 10% of non-zero features. The exact chosen values are the following: $\lambda_2 = 10^{-11}$ for KDD 2010,

$\lambda_2 = 10^{-16}$ for KDD 2012 and $\lambda_2 = 4 \times 10^{-12}$ for Criteo.

Storage of memory terms. The storage requirements for this method is in the worst case a table of size $n \times d$. However, as for SAG and SAGA, for linearly parametrized loss functions of the form $f_i(\boldsymbol{x}) = \ell(\boldsymbol{a}_i^T \boldsymbol{x})$, where ℓ is some real-valued function and $(\boldsymbol{a}_i)_{i=1}^n$ are samples associated with the learning problem, this can be reduced to a table of size n (Schmidt et al., 2016, §4.1). This includes popular linear models such as least squares or logistic regression with ℓ the squared or logistic function, respectively.

The reduce storage comes from the fact that in this case the partial gradients have the structure

$$\nabla f_i(\boldsymbol{x}) = \boldsymbol{a}_i \underbrace{\ell'(\boldsymbol{a}_i^T \boldsymbol{x})}_{\text{scalar}} \quad . \tag{4.13}$$

Since $\boldsymbol{a}_i$ is independent of $\boldsymbol{x}$, we only need to store the scalar $\ell'(\boldsymbol{a}_i^T \boldsymbol{x})$. This decomposition also explains why ∇f_i inherits the sparsity pattern of $\boldsymbol{a}_i$.

Atomic updates. Most modern processors have support for atomic operations with minimal overhead. In our case, we implemented a double-precision atomic type using the `C++11` atomic features (`std::atomic<double>`). This type implements atomic operations through the compare and swap semantics.

Empirically, we have found it necessary to implement atomic operations at least in the vector $\boldsymbol{\alpha}$ and $\overline{\boldsymbol{\alpha}}$ to reach arbitrary precision. If non-atomic operations are used, the method converges only to a limited precision (around normalized function suboptimality of 10^{-3}), which might be sufficient for some machine learning applications but which we found not satisfying from an optimization point of view.

AsySPCD. Following (Peng et al., 2016) we keep the vector $(\boldsymbol{a}_i^T \boldsymbol{x})_{i=1}^n$ in memory and update it at each iteration using atomic updates.

Hardware and software. All experiments were run on a Dell PowerEdge 920 machine with 4 Intel Xeon E7-4830v2 processors with 10 2.2GHz cores each and 384GB 1600 Mhz RAM. The PROXASAGAand ASYSPCD code was implemented on C++ and binded in Python. The FISTA code is implemented in pure Python using NumPY and SciPy for matrix computations (in this case the bottleneck is in large sparse matrix-vector operations for which efficient BLAS routines were used). Our PROXASAGA implementation can be downloaded from `http://github.com/fabianp/ProxASAGA`.

4.4.2 Comparison of Sparse Proximal SAGA with sequential methods

We now provide a comparison between the Sparse Proximal SAGA and related methods in the sequential case. We compare against two algorithms: the MRBCD method of Zhao et al. (2014) (which forms the basis of Async-PROXSVRCD) and the vanilla implementation of SAGA (Defazio et al., 2014a), which does not have the ability to perform sparse updates.

We report performance both in terms of both passes through the data (epochs) and running time. We use the same step size for all methods $(1/3L)$. Due to the slow convergence of some methods, we use a smaller dataset than the ones used in Table 4.1. Dataset RCV1 has $n = 697,641, d = 47,236$ and a density of 0.15, while Covtype is a dense dataset with $n = 581,012, d = 54$.

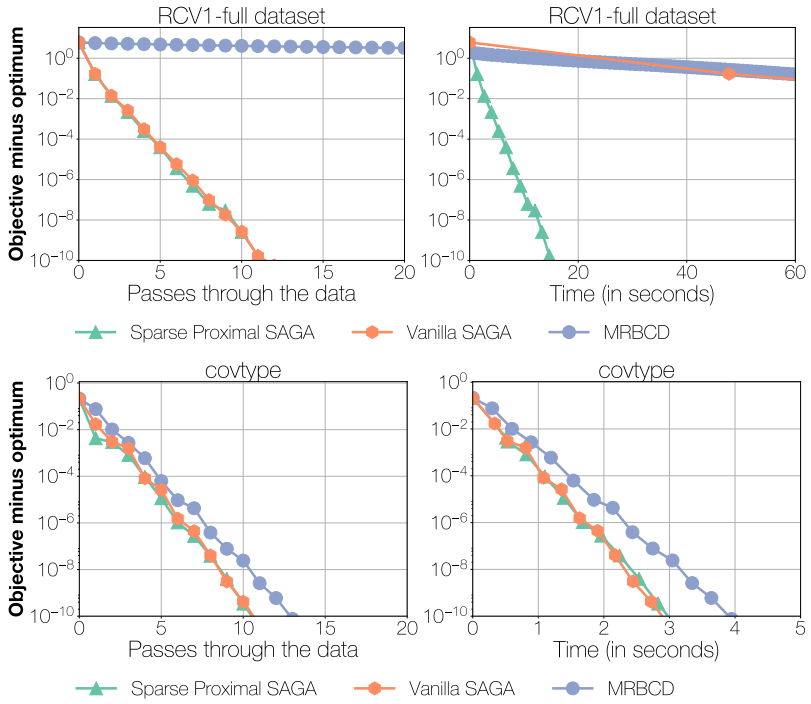


Figure 4-1 – Suboptimality of different sequential algorithms. Each marker represents one pass through the dataset.

We observe that for the convergence behavior in terms of number of passes, Sparse Proximal SAGA performs as well as vanilla SAGA, though the latter requires dense updates at every iteration (Fig. 4-1 top left). On the other hand, in terms of running time, our implementation of Sparse Proximal SAGA is much more efficient than the other methods for sparse input (Fig. 4-1 top right). In the case of dense input (Fig. 4-1 bottom), the three methods perform similarly.

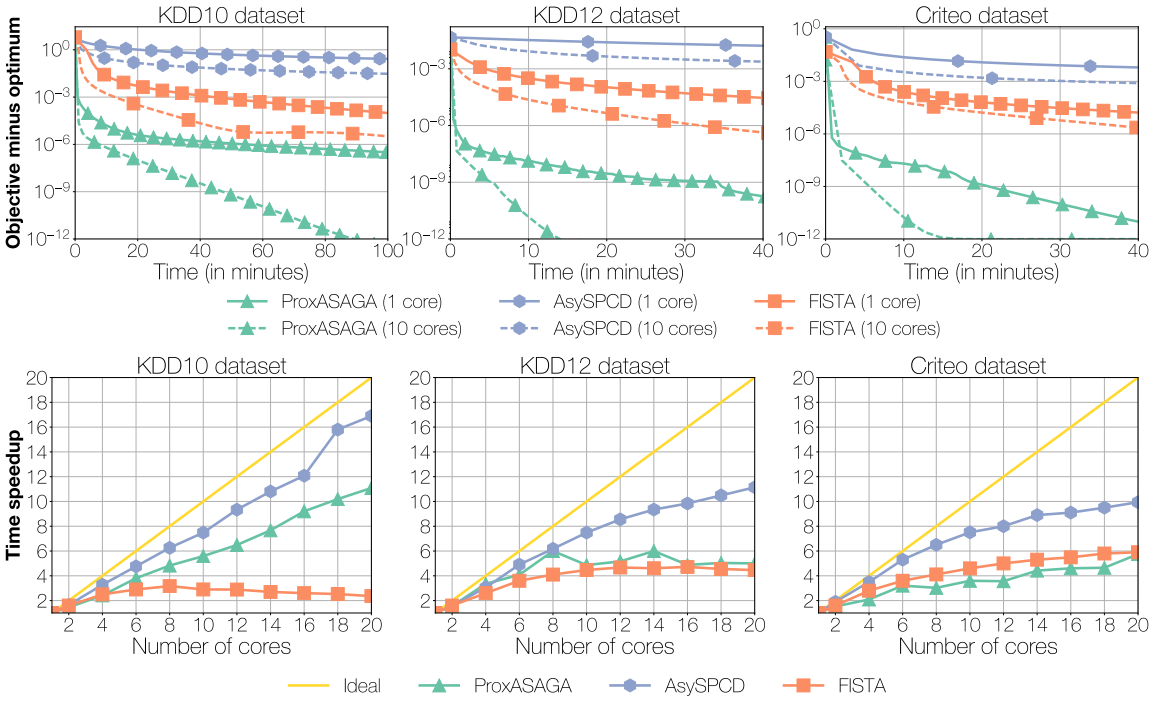


Figure 4-2 – **Convergence for asynchronous stochastic methods for $\ell_1 + \ell_2$-regularized logistic regression**. Top: Suboptimality as a function of time for different asynchronous methods using 1 and 10 cores. Bottom: Running time speedup as function of the number of cores. PROXASAGA achieves significant speedups over its sequential version while being orders of magnitude faster than competing methods. ASYSPCD achieves the highest speedups but it also the slowest overall method.

A note on the performance of MRBCD. It may appear surprising that Sparse Proximal SAGA outperforms MRBCD so dramatically on sparse datasets. However, one should note that MRBCD is a doubly stochastic algorithm where both a random data point and a random coordinate are sampled for each iteration. If the data matrix is very sparse, then the probability that the sampled coordinate is in the support of the sampled data point becomes very low. This means that the gradient estimator term only contains the reference gradient term of SVRG, which only changes once per epoch. As a result, this estimator becomes very coarse and produces a slower empirical convergence.

This is reflected in the theoretical results given in Zhao et al. (2014), where the epoch size needed to get linear convergence are k times bigger than the ones required by plain SVRG, where k is the size of the set of blocks of coordinates.

4.4.3 Comparison of PROXASAGA with asynchronous methods

We compare three parallel asynchronous methods on the datasets of Table 4.1: PROXASAGA (this work), [9] ASYSPCD, the asynchronous proximal coordinate descent method of Liu and Wright (2015) and the (synchronous) FISTA algorithm (Beck and Teboulle, 2009), in which the gradient computation is parallelized by splitting the dataset into equal batches.

We aim to benchmark these methods in the most realistic scenario possible; to this end we use the following step size: $1/2L$ for PROXASAGA, $1/L_c$ for ASYSPCD, where L_c is the coordinate-wise Lipschitz constant of the gradient, while FISTA uses backtracking line-search.

The results can be seen in Figure 4-2 (top) with both one (thus sequential) and ten processors. Two main observations can be made from this figure. First, PROXASAGA is significantly faster on these problems. Second, its asynchronous version offers a significant speedup over its sequential counterpart.

In Figure 4-2 (bottom) we present speedup with respect to the number of cores, where speedup is computed as the time to achieve a suboptimality of 10^{-10} with one core divided by the time to achieve the same suboptimality using several cores. While our *theoretical speedups* (with respect to the number of iterations) are almost linear as our theory predicts (see Section 4.4), we observe a different story for our *running time* speedups. This can be attributed to memory access overhead, which our model does not take into account. As predicted by our theoretical results, we observe a high correlation between the Δ dataset sparsity measure and the empirical speedup: KDD 2010 ($\Delta = 0.15$) achieves a $11\times$ speedup, while in Criteo ($\Delta = 0.89$) the speedup is never above $6\times$.

Note that although competitor methods exhibit similar or sometimes better speedups, they remain orders of magnitude slower than PROXASAGA in running time for large sparse problems. In fact, our method is between $5\times$ and $80\times$ times faster (in time to reach 10^{-10} suboptimality) than FISTA and between $13\times$ and $290\times$ times faster than ASYSPCD (see Section 4.4.5).

4.4.4 Theoretical speedups

In Section 4.4.3, we have shown experimental speedup results where suboptimality was a function of the running time. This measure encompasses both theoretical algorithmic optimization properties and hardware overheads (such as contention of shared memory) which are not taken into account in our analysis.

9. A reference C++/Python implementation of is available at https://github.com/fabianp/ProxASAGA

In order to isolate these two effects, we now plot our speedup results in Figure 4-3 where suboptimality is a function of the number of iterations; thus, we abstract away any potential hardware overhead. To do so, we implement a global counter which is sparsely updated (every 100 iterations for example) in order not to modify the asynchrony of the system. This counter is used only for plotting purposes and is not needed otherwise. Specifically, we define the theoretical speedup as:

$$\text{theoretical speedup} := (\text{number of cores}) \frac{\text{number of iterations for sequential algorithm}}{\text{total number of iterations for parallel algorithm}}.$$

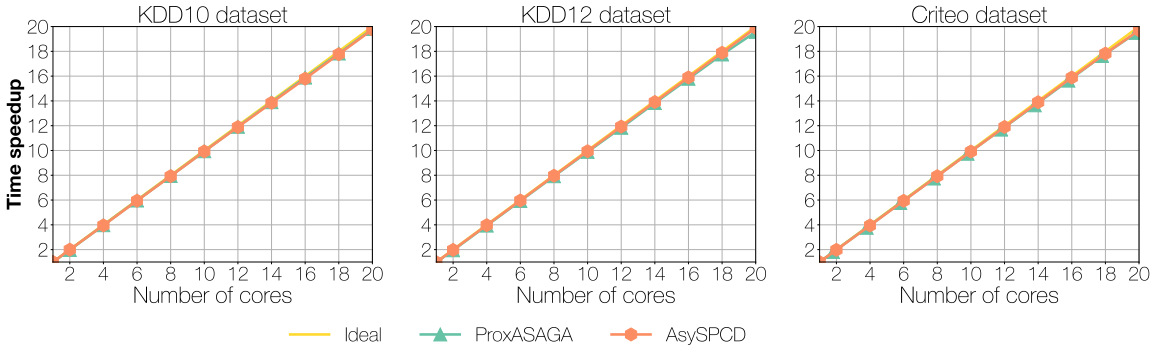


Figure 4-3 – **Theoretical optimization speedups for ℓ_1+ℓ_2-regularized logistic regression**. Speedup as measured by the number of iterations required to reach 10^{-5} suboptimality for PROXASAGA and ASYSPCD. In FISTA the iterates are the same with different cores and so matches the "ideal" speedup.

We see clearly that the theoretical speedups obtained by both PROXASAGA and ASYSPCD are linear (i.e. ideal). As we observe worse results in running time, this means that the hardware overheads of asynchronous methods are quite significant.

4.4.5 Timing benchmarks

Finally, we provide the time it takes for the different methods with 10 cores to reach a suboptimality of 10^{-10}. All results are in hours.

Dataset	PROXASAGA	ASYSPCD	FISTA
KDD 2010	1.01	13.3	5.2
KDD 2012	0.09	26.6	8.3
Criteo	0.14	33.3	6.6

4.5 Conclusion and future work

In this chapter, we have described PROXASAGA, an asynchronous variance-reduced algorithm with support for composite objective functions. This method builds upon a novel sparse variant of the (proximal) SAGA algorithm that takes advantage of sparsity in the individual gradients. We have proven that this algorithm is linearly convergent under a condition on the step size and that it is linearly faster than its sequential counterpart given a bound on the delay. Empirical benchmarks show that PROXASAGA is orders of magnitude faster than existing state-of-the-art methods.

This work can be extended in several ways. First, we have focused on the SAGA method as the basic iteration loop, but this approach can likely be extended to other proximal incremental schemes such as SGD or ProxSVRG. Second, as mentioned in Section 4.3.3, it is an open question whether it is possible to obtain convergence guarantees without any sparsity assumption, as was done for ASAGA.

Chapter 5

Conclusion

The continued growth in dataset and model size in the field of machine learning imply using more and more computational resources. While the CPU clock speed improvements used to be sufficient to cover this increase, this is no longer the case, as we approach the limits of reasonably-costed processor miniaturization. This implies that new ways of increasing computing power have become necessary, as well as limiting the amount of computation required by developing algorithms that converge very fast.

Asynchronous parallel algorithms are well-positioned in this context since they allow practitioners to leverage the recent advances in multi-core architectures. Consequently, the focus of Part I of this thesis has been the introduction of new fast asynchronous algorithms, and the analysis of their theoretical and empirical performance.

Unfortunately, conducting the analysis of such algorithms is typically quite difficult. Indeed, we cannot reuse the techniques deployed in the sequential case without significant adjustments, in order to take into account the effects of asynchrony (such as delayed updates and inconsistency). Handling these difficulties has often meant relying on simplistic assumptions or even flawed reasoning (see for instance Section 2.2.2).

Chapter 2. Therefore, the first focus of this thesis has been to understand the flaws of the current approaches and to derive a new framework of analysis to enable correct and simple proofs. Building on the recently introduced "perturbed iterate" framework, we have proposed a novel perspective to clarify an important technical issue present in a large fraction of the recent convergence rate proofs for asynchronous parallel optimization algorithms. To resolve it, we have introduced our novel "after read" framework.

As a first token towards demonstrating its usefulness, we have conducted the analysis of the simplest of asynchronous parallel incremental optimization algorithms: HOGWILD (Section 2.3). Our proof technique accommodates more realistic settings than is usually

the case in the literature (such as inconsistent reads and writes and an unbounded gradient). Furthermore, we obtain tighter conditions than in previous work.

Chapter 3. We have seen that one key direction to handle the new challenges of optimization for machine learning is to reduce the amount of necessary computation through the use of fast converging algorithms. Combining this idea with leveraging modern multi-core architectures, we have introduced ASAGA, a novel sparse and fully asynchronous variant of the fast variance-reduced incremental gradient algorithm SAGA.

Using our "after read" framework, we have conducted the convergence analysis of ASAGA under realistic settings, thus demonstrating the applicability of the framework to complex algorithms (Section 3.2.2). We obtain stronger results than is typically the case in the related literature. In particular, we show that ASAGA is linearly faster than SAGA under mild conditions, and that sparsity is not always necessary to get linear speedups.

We have also conducted the analysis of a related method, KROMAGNON. Using the "after read" framework yielded much improved bounds, enabling us to show that this algorithm can also attain a linear speedup without sparsity in the right conditions.

Our experiments are consistent with our theoretical results, as the algorithms exhibit linear speedups in terms of number of iterations, and strong but sub-linear speedups in running time (up to $10\times$).

Chapter 4. The methods analyzed in Chapters 2 and 3 are only amenable to smooth objectives. However, many applications in machine learning require optimizing for a non-smooth objective (e.g. box constraints, $\ell 1$-regularization...).

We have therefore introduced PROXASAGA, an asynchronous variance-reduced algorithm with support for composite objective functions. This method builds upon a novel sparse variant of the (proximal) SAGA algorithm that takes advantage of sparsity in the individual gradients.

Again using the "after read" framework, we have proven that this algorithm is linearly convergent under a condition on the step size and that it is linearly faster than its sequential counterpart given a bound on the delay. Unfortunately, our current results do not exhibit a setting in which linear speedups are attainable without sparsity as was the case for ASAGA. Whether this interesting property can be recovered is still open research.

Empirical benchmarks show that PROXASAGA is orders of magnitude faster than existing state-of-the-art methods on large-scale datasets.

Limitations and future work. Despite the success we have encountered using the "after read" framework, it is not exempt of limitations. The biggest issue with this approach is that there subsists a discrepancy between the algorithms we analyze, which read the whole parameter vector at the start of each iteration, and their actual implementation where only the relevant parameters are read. The reason for this is to enforce the independence between the iterates $\hat{x}_t$ and the sampled factor i_t. An interesting direction of future research would be to remove the discrepancy and handle the dependency directly in the proof.

Another suboptimal aspect of our results is the definition of the sparsity constant Δ_r. As it is essentially a maximum, we have noticed that it often fails to accurately reflect the full sparsity distribution of our datasets. Replacing this maximum with a smarter quantity would yield more realistic results and is still open research.

Further exploration of the τ term is also a promising direction, as this ubiquitous quantity is still poorly understood. Indeed, our preliminary experiments (see Section 3.4.7) indicate that it encompasses more complexity than previously thought. A more thorough understanding would result in more meaningful and interpretable convergence and speedup results, potentially opening up other research directions.

Finally, while our algorithms can be directly ported to a distributed master-worker architecture, their communication pattern would have to be optimized to avoid prohibitive costs. Limiting communications can be interpreted as artificially increasing the delay, yielding an interesting trade-off between delay influence and communication costs (underlying once again the importance of a thorough understanding of the delay mechanisms).

Part II

Improving Recurrent Neural Networks Training through Global-Local Losses

Chapter 6

A brief introduction to recurrent neural networks

Many modern machine learning tasks can be modeled as sequential prediction, where multiple random variables are predicted one at a time. This is notably the case of machine translation for instance, or caption generation, whose output structure is naturally a sequence. Even for more complex structures, there are often ways to transform problems such that they can be solved using sequential prediction: a good example of this adaptation is dependency parsing.

Many approaches have been proposed to tackle the difficulties of sequential prediction. In the last few years, recurrent neural networks (RNNs) – a family of neural network models specialized for sequential prediction – have shown particular promise in this field, performing successfully in structured prediction applications such as machine translation (Sutskever et al., 2014), parsing (Ballesteros et al., 2016) or caption generation (Vinyals et al., 2015).

The second part of this thesis will be focused on RNN training algorithms. We first detail how RNNs are built (Section 6.1) and explain how they are typically trained (Section 6.2.1). We then detail the limitations of the current training approach (Section 6.2.2) and their impact on model quality. Finally, in Chapter 7 we propose SEARNN, a new training algorithm inspired from the "learning to search" (L2S) framework for structured prediction, and demonstrate improved behavior and performance.

6.1 What is a recurrent neural network (RNN)?

RNNs are a large family of neural network models aimed at representing sequential data. To do so, they produce a sequence of states $(h_1, ..., h_T)$ by recursively applying the same

transformation (or *cell*) f on sequences of data:

$$h_t = f(h_{t-1}, y_{t-1}, x)\,, \tag{6.1}$$

with h_0 an initial state, $(y_t)_{0\leq t\leq T}$ the sequence of output tokens and x an optional input.

The output tokens $(y_t)_{0\leq t\leq T}$ are generated from the corresponding hidden states. As each prediction takes into account all previous predictions, the RNN cell learns to output the next token conditioned on the previous ones.

The key feature of RNNs is that they implement a tailored *parameter sharing* approach, since they use the same transition function f at every time step. This is an efficient way of representing similar random variables, for the following reasons. First, as we are learning a transition function from state to state, [1] the input and output sizes are always the same, so the parametrization of f is fixed. Second, it implies learning a single shared model, applicable to any time step (and hence to sequences of any size). Finally, training such a shared model can be done with much fewer training examples than otherwise. Without parameter sharing, we would have to learn one model per time step, thus requiring many more training items, even though we could not hope to generalize to sequence lengths unseen during training time.

These advantages are compelling, but they do come at a cost. First of all, although RNNs sport compact representations, the optimization process for the smaller number of parameters may be quite difficult. Second, as a single model is learned, this approach is more suited to sequences of similar output tokens. Machine translation, whose output is a sequence of words, is a nicely suited example, whereas trying to predict a sequence containing both words and real numbers is less so. Finally, the implicit assumption of using a shared transition model is that the dependency of the variables at a given time step t – conditioned on the previous variables – is stationary, i.e. it does not depend on t. [2]

6.1.1 A concrete example

There are many flavors of RNN which fit the description given by (6.1). We now detail one of them, a conditional RNN with identical input and output sizes, depicted in Figure 6-1.

In this specific case we can rewrite the update as:

$$h_t = \sigma(b + Ux_t + Wh_{t-1} + Ry_{t-1})\,, \tag{6.2}$$

1. And optionally input to state and state to output.
2. As noted by Goodfellow et al. (2016), in theory it is possible to add t as an additional input at each time step. Whether this is enough for the model to learn possible time dependencies is unclear though.

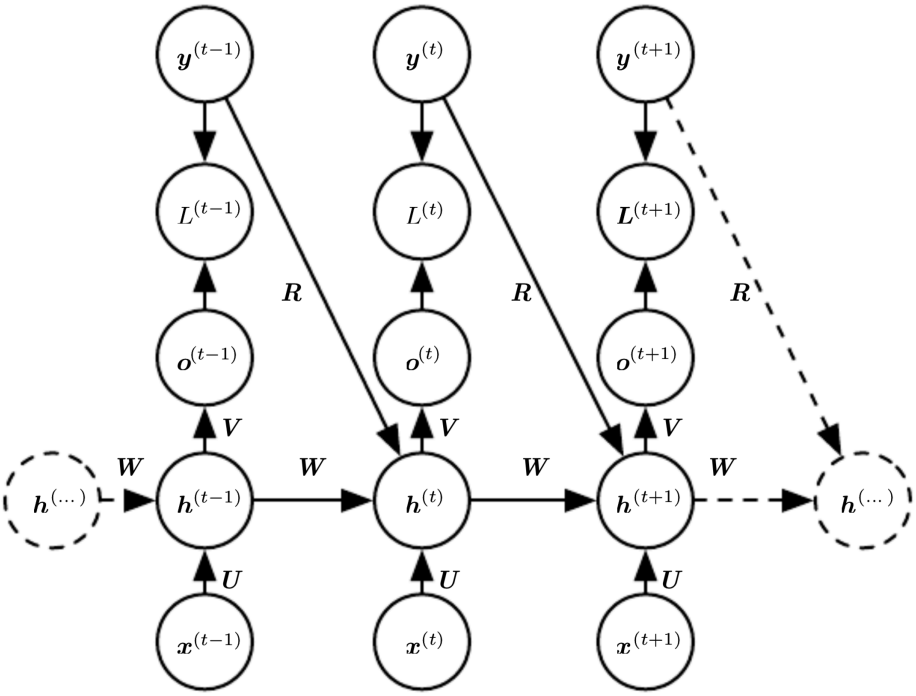


Figure 6-1 – Reproduced from Figure 10.10 in (Goodfellow et al., 2016). This example is a conditional RNN with learned input-to-state, state-to-state and output-to-state connections. It can map input sequences of variable size to output sequences of the same size.

where b is a bias term, U, W, R are the weight matrices parameterizing the input-to-state, state-to-state and output-to-state connections (respectively) and σ is a non-linear activation function such as the $\tanh$ function.

The output is derived from the hidden state according to the following transformation:

$$o_t = c + Vh_t\,, \tag{6.3}$$

where c is another bias term, and V is a learned matrix which projects the fixed-size hidden state into the (often larger) output space.

This RNN has some specific characteristics. First, it has an input x connected through the input-to-state matrix U. It is therefore a conditional RNN. Second, it also has connections between its outputs [3] and its hidden states through matrix R. Third, the output is fed one item at a time, and there is one corresponding output per input token. As a consequence, when using this model the output size has to match the input size.

3. Technically, the connection here is between the *ground truth* outputs and the hidden states, rather than between the model outputs and the hidden states. We explain this discrepancy in details in Section 6.2.1.

6.1.2 Interpretation as a graphical model

RNN models come with a natural graphical model interpretation. Using the chain rule (with assumptions depending on the architecture), we can model the joint probability distribution of the output sequence. The specific model presented in Figure 6-1 has the following properties.

If we normalize the output so that it sums to 1 (say, through the use of a softmax layer – an exponential normalizer), we can model the behavior of the output y conditioned on (part of) the input x: $P(y_t|y_1, ..., y_{t-1}, x_1, ..., x_{t-1})$.

However, we cannot simply model the joint probability of y conditioned on x (as we will with the encoder-decoder architecture presented in Section 6.1.3), unless we explicitly assume that $P(y|x)$ factorizes as:

$$\prod_{t=1}^{T} P(y_t|y_1, ..., y_{t-1}, x_1, ..., x_t) \,. \tag{6.4}$$

This is equivalent to assuming that the y_t only depend on the input through the tokens which come before t. Note that if the model did not have output-to-state connection, we could only model $P(y_t|x_1, ..., x_t)$. In order to apply the chain rule and recover the joint sequence probability, we would also have to assume that the y_t are conditionally independent.

6.1.3 The encoder-decoder architecture

We have seen that the model presented in Figure 6-1 can handle sequences with variable size. However, it does suffer from a significant drawback: it can only be applied to tasks where the output sequence is of the same length as the input sequence. Unfortunately, this property does not hold in the general case. In machine translation, for instance, the number of words in a sentence is not constant across languages. General sequence prediction thus requires a more general architecture.

A solution to this issue was found by Cho et al. (2014) and Sutskever et al. (2014), based on the idea that there exists RNN architectures that transform variable-sized inputs into fixed-sized outputs, and conversely fixed-sized inputs into variable-sized outputs. Thus, to go from variable-sizes inputs to (potentially different) variable-sized outputs, one can combine two RNNs.

The first one, called the *encoder*, takes a variable-sized sequence as input and transforms it into a fixed-sized *context* vector (typically a function of its last hidden state). This context vector is then fed to the second RNN, called the *decoder*, typically as its initial state h_0.

An interesting consequence of this architecture is that the decoder RNN's outputs are conditioned on both the previous outputs and the entire input, through the context vector. Provided the decoder contains output-to-state connections, this implies that the joint probability $P(y|x)$ factorizes naturally as the product of the outputs of the RNN, without having to resort to any independence assumption.

6.1.4 Decoding

As we have detailed in the previous paragraphs, the decoder RNN gives us access to the conditional probability of an output sequence y given an input sequence x. How can we use such a model for prediction?

We can compute $\arg\max_{y \in \mathcal{Y}} p(y|x)$ exactly. However, this comes at a prohibitive computational cost, as the cardinality of $\mathcal{Y}$ grows exponentially with the sequence length. We thus need to resort to approximate decoding methods. As we have decomposed the joint probability in a product of conditional probabilities, we usually leverage *search-based* approaches.

Greedy search. The most common one is called *greedy search* or *greedy decoding*. As indicated by its name, it simply consists in greedily picking the token with maximum probability at each time step. The decoded sequence is the sequence of greedy predictions $\hat{y}_t := \arg\max_{a \in \mathcal{A}} p(a|\hat{y}_1, \ldots, \hat{y}_{t-1}, x)$.

Of course, this can be suboptimal: picking the token with highest probability does not guarantee that the overall product of conditional probabilities will be maximal (the following probability terms might be smaller than those obtained by picking another token). The key advantage of this method is that it is computationally inexpensive.

Beam search. If we want to trade off additional computation for improved performance, we can turn to an alternate method called *beam search*. This procedure also operates step by step.

We start by choosing a beam size k and adding the empty sequence to the beam. Then, at each step t, for each prefix sequence p_{t-1}^i in the beam and for each of the corresponding k best tokens a_t^j at cell t (bearing in mind that the distribution at cell t depends on the prefix sequence), we compute the score of $p_{t-1}^i : a_t^j$. We end up with at most k^2 new prefix sequences, with size increased by one. As the beam can only contain k prefixes, we end each step by pick the k new prefixes with the highest scores.

The procedure ends either once a given length has been reached or once all the sequences

in the beam finish with a special "end-of-sequence" (<EOS>) character.[4] The last step consists in picking the candidate with the highest score out of the beam.

While beam search usually leads to improved decoded sequence scores, this does not automatically imply better performance on the validation metric (which is typically not likelihood-based). Furthermore, compared to greedy search, the cost of beam search grows linearly with the beam size.

This short introduction on recurrent neural networks is of course incomplete. For more thorough information on the subject, we point the reader to Graves (2012) or Goodfellow et al. (2016, Chapter 10).

6.2 Traditional RNN training and its limitations

6.2.1 Maximum likelihood training (MLE)

The standard training loss for RNNs is derived from maximum likelihood estimation (MLE): we consider that the cell outputs a probability distribution at each step in the sequence, and we seek to maximize the probability of the ground truth tokens.

We will now explain this process in more details for the case of the decoder RNN, the specific architecture our work is focused on. This subset of the RNN family is particularly adapted to structured prediction, where we want to model the *joint probability* of a target sequence $(y_1, \dots, y_{T_x}) \in \mathcal{A}^{T_x}$ given an input x. Here $\mathcal{A}$ is the alphabet of output tokens and T_x is the length of the output sequence associated with input x (although T_x may take different values, in the following we drop the dependency in x and use T for simplicity).

To achieve this modeling, we feed h_t through a projection layer (i.e. a linear classifier) to obtain a vector of scores s_t over all possible tokens $a \in \mathcal{A}$, and normalize these with a softmax layer (an exponential normalizer) to obtain a distribution o_t over tokens:

$$h_t = f(h_{t-1}, y_{t-1}, x)\,; \qquad s_t = \text{proj}(h_t)\,; \qquad o_t = \text{softmax}(s_t) \qquad \forall\, 1 \leq t \leq T\,. \quad (6.5)$$

The vector o_t is interpreted as the predictive conditional distribution for the t^{th} token given by the RNN model, i.e. $p(a|y_1, \dots, y_{t-1}, x) := o_t(a)$ for $a \in \mathcal{A}$. Multiplying the values $o_t(y_t)$ together thus yields the joint probability of the sequence y defined by the RNN (thanks to

4. Note that the beam only contains sequences of the size of the current step minus one, or sequences finishing with <EOS>.

the chain rule, see Section 6.1.2):

$$p(y_1, ..., y_T|x) = p(y_1|x)p(y_2|y_1, x) ... p(y_T|y_1, ..., y_{T-1}, x) := \Pi_{t=1}^{T} o_t(y_t) \,. \tag{6.6}$$

As pointed by Goodfellow et al. (2016), the underlying structure of these RNNs as graphical models is thus a complete graph, and there is no conditional independence assumption to simplify the difficult prediction task of computing $\arg\max_{y \in \mathcal{Y}} p(y|x)$. We have detailed the two usual approximate approaches to solve this problem in Section 6.1.4.

Teacher forcing. At decoding time, the decoder RNN has a connection between its output and the next hidden state. However, at training time, in order to obtain the joint probability of a ground truth input sequence, we need the output probability distribution at each cell – the $p(y_t|y_1, ..., y_{t-1}, x)$ terms – to be conditioned on the *ground truth* tokens. This is why the output-to-state connection in Figure 6-1 goes from the ground truth tokens to the hidden states, rather than from the outputs to the hidden state.

Using the ground truth tokens rather than the greedy predictions of the RNN as inputs to the RNN cell is called *teacher forcing*. If we use such a regimen, we can recover the probability of each ground truth sequence according to the RNN model. We can then use MLE to derive a loss to train the RNN.

$$\mathcal{L}(\theta) = -\sum_{i=1}^{N} \sum_{t=1}^{T} o_t(a_{i,t}^{gt}; \theta) \,, \tag{6.7}$$

where $a_{i,t}^{gt}$ is the ground truth token of the i^{th} training sequence at step t, N is the number of training examples and θ are the parameters of the model.

One should note here that although the individual output probabilities are at the token level, the MLE loss involves the joint probability (computed via the chain rule) and is thus at the *sequence level*.

6.2.2 Limitations

While this maximum likelihood style of training has been very successful in various applications, it suffers from several known issues, especially for structured prediction problems.

The first one is called *exposure* or *exploration bias* (Ranzato et al., 2016). During training (with teacher forcing), the model learns the probabilities of the next tokens conditioned on the ground truth. But at test time, the model does not have access to the ground truth and

outputs probabilities are conditioned on its own previous predictions instead. Therefore if the predictions differ from the ground truth, the model has to continue based on an exploration path it has not seen during training, which means that it is less likely to make accurate predictions. This phenomenon, which is typical of sequential prediction tasks (Kääriäinen, 2006; Daumé et al., 2009) can lead to a compounding of errors, where mistakes in prediction accumulate and prevent good performance.

The second major issue is the discrepancy between the training loss and the various test errors associated with the tasks for which RNNs are used (e.g. edit distance, F1 score...). Of course, a single surrogate is not likely to be a good approximation for all these errors. One salient illustration of that fact is that MLE ignores the information contained in structured losses. As it only focuses on maximizing the probability of the ground truth, it does not distinguish between a prediction that is very close to the ground truth and one that is very far away. Thus, most of the information given by a structured loss is not leveraged when using this approach.

Local vs. sequence-level. Some recent papers (Ranzato et al., 2016; Wiseman and Rush, 2016) also point out the fact that since RNNs output next token predictions, their loss is local instead of sequence-level, contrary to the error we typically want to minimize. This claim seems to contradict the standard RNN analysis, which postulates that the underlying graphical model is the complete graph: that is, the RNN outputs the probability of the next tokens conditioned on all the previous predictions. Thanks to the chain rule, one recovers the probability of the whole sequence. Thus the maximum likelihood training loss is indeed a *sequence level* loss, even though we can decompose it in a product of local losses at each cell.

However, if we assume that the RNN outputs are only conditioned on the last few predictions (instead of all previous ones), then we can indeed consider the MLE loss as local. In this setting, the underlying graphical model obeys Markovian constraints (as in maximum entropy Markov models (MEMMs)) rather than being the complete graph; this corresponds to the assumption that the information from the previous inputs is imperfectly carried through the network to the cell, preventing the model from accurately representing long-term dependencies.

6.3 Alternative training approaches

Given all these limitations, exploring novel ways of training RNNs appears to be a worthy endeavor, and this field has attracted a lot of interest in the past few years.

6.3.1 Imitation learning approaches

One first source of inspiration has been a flavor of imitation learning called "learning to search" (L2S), which we present in more details in Section 7.2.

Several papers have tried using L2S-like ideas for better RNN training, starting with Bengio et al. (2015) which introduces "scheduled sampling" to avoid the exposure bias problem. The idea is to start with training under the teacher forcing regimen and to gradually mix in model predictions instead of ground truth tokens as inputs to the RNN cell as training progresses. This idea already appears in the foundational L2S paper (Daumé et al., 2009), albeit not in the specific context of RNN training.

Wiseman and Rush (2016, BSO) adapt one of the early variants of the L2S framework: the "Learning A Search Optimization" approach of Daumé and Marcu (2005, LASO) to train RNNs (which, it should be noted, is quite different from the more modern SEARN family of algorithms pioneered by (Daumé et al., 2009)). BSO's training loss is defined by violations in the beam-search procedure: during training, a beam of sequences is maintained. The model receives training signal each time the ground truth sequence falls off this beam. BSO's ad hoc surrogate objective provides very sparse sequence-level training signal, as mentioned by their authors, thus requiring warm-starting from an MLE trained model.

Other approaches are developed for specific tasks where an optimal policy can be computed for free, starting from a given prefix sequence (this is the case for some flavors of dependency parsing for instance). Ballesteros et al. (2016) use a loss that is similar to MLE, although they replace the ground truth token with the best token according to the optimal policy. Sun et al. (2017) introduces new gradient procedures to incorporate neural classifiers in the AGGREVATE (Ross and Bagnell, 2014) variant of L2S. [5]

6.3.2 RL-inspired approaches

Reinforcement learning (RL) has proven to be another particularly rich source of inspiration for improved RNN training. The basic idea is to adapt existing RL algorithms – which operate with an always available reward function as source of training signal – to the supervised learning setup for structured prediction tasks – where ground truth trajectories are available at training (but not a decoding) time. As ground truth sequences can be leveraged to derive a reward signal, the setup we are studying is "richer" at training time (with access both to a reward and ground truth tokens) and "poorer" at test time (where we do not have access to any signal besides the input). Adapting traditional RL algorithms such as

5. Sun et al. (2017)'s algorithm simply replaces the classifier in AGGREVATE with a neural network. As it is trained on an ever growing dataset, a natural gradient update is required to make the algorithm tractable.

REINFORCE or ACTOR-CRITIC to leverage this additional information has been a major research direction (Ranzato et al., 2016; Bahdanau et al., 2017).

One of the key advantages of the RL-inspired approaches is that they can leverage the validation metric to define their reward function, ultimately deriving training losses that are more closely related to the quantity that we actually want to minimize.

The most common approach in RL-augmented training consists in optimizing for the expectation of the validation metric directly (under π_θ, the stochastic policy parameterized by the RNN):

$$\mathcal{L}(\theta) = -\sum_{i=1}^{N} \mathbb{E}_{(y_1^i,..,y_T^i)\sim\pi(\theta)} r(y_1^i,..,y_T^i)\,, \tag{6.8}$$

where N is the amount of training samples and r is the reward function.

Since we are taking an expectation over all possible structured outputs, the only term that depends on the parameters is the probability term (the tokens in the error term are fixed). This allows this loss function to support non-differentiable test errors, which is another key advantage.

Of course, actually computing the expectation over an exponential number of possibilities is computationally intractable. To circumvent this issue, Shen et al. (2016) subsample trajectories according to the learned policy, while Ranzato et al. (2016); Rennie et al. (2017) use the REINFORCE algorithm, which essentially approximates the expectation with a single trajectory sample. Instead of approximating the gradient term via sampling, Bunel et al. (2018) compute its exact value on a simplified probability distribution with reduced support (basically the k best samples returned by beam search).[6] Bahdanau et al. (2017) adapt the ACTOR-CRITIC algorithm, where a second *critic* network is trained to approximate the expectation.[7] For a more complete bibliography of RL methods applied to sequence-to-sequence training, we refer the reader to Keneshloo et al. (2018).

Issues with RL-based approaches. Despite successfully improving upon MLE baselines, most attempts at adapting RL algorithms to RNN training come with a number of limitations.

First, while maximizing the expected reward allows the RL approaches to use gradient descent even when the test error is not differentiable, this comes at the cost of approximating said gradient through trajectory sampling according to the model policy. This introduces a discrepancy between training and testing. Indeed, at test time, one does not decode by sampling from the stochastic policy. Instead, one selects the "best" sequence according to a

6. Note that unlike in the other papers referenced in this paragraph, the resulting estimator is biased.
7. All the papers mentioned in this paragraph tackle machine translation, except for Rennie et al. (2017), which targeted at image captioning, and Bunel et al. (2018), targeted at program synthesis.

search algorithm, e.g. greedy or beam search (see Section 6.1.4).

Second, these approaches are usually more complex than MLE, be it because they involve training additional models (e.g. baselines in REINFORCE) or because they require deploying standard RL tricks such as target networks (Bahdanau et al., 2017).

Finally, while all these approaches report significant improvement on various tasks, one trait they share is that they only work when initialized from a good pre-trained model. This phenomenon is often explained by the sparsity of the information contained in "sequence-level" losses. Indeed, in the case of REINFORCE, no distinction is made between the tokens that form a sequence: depending on whether the sampled trajectory is above a global baseline, all tokens are pushed up or down by the gradient update. This means good tokens are sometimes penalized and bad tokens rewarded.

This distinction is quite relevant, because warm-starting means initializing in a specific region of parameter space which may be hard to escape. Exploration is less constrained when starting from scratch as in MLE training.

6.3.3 Other methods

RAML and variants. RAML (Norouzi et al., 2016) is another RL-inspired approach. Here, in order to mitigate the 0/1 aspect of MLE training, the authors introduce noise in the target outputs at each iteration. The amount of random noise is determined according to the associated reward (target outputs with a lot of noise obtain lower rewards and are thus sampled with lower probability). This idea is linked to the label smoothing technique (Szegedy et al., 2016), where the target distribution at each step is the addition of a Dirac (the usual MLE target) and a uniform distribution.

RAML has spawned several variants. Dai et al. (2018) establish a theoretical link between RAML and entropy-regularized RL, and leverage this insight to propose both an improved RAML alternative as well as an improved ACTOR-CRITIC algorithm. Elbayad et al. (2018) on the other hand extend the sequence-level RAML approach to the token-level, improving performance over the initial algorithm.

Structured prediction approaches. A final paper worth mentioning is Edunov et al. (2018), which studies various classical structured prediction and RL-inspired losses on machine translation and abstractive summarization tasks, obtaining state-of-the-art results (although their experiments are run using convolutional neural networks, they are in principle adaptable to RNNs).

This section is by no means a complete bibliography of RNN training algorithms. In Section 7.8, we present a more thorough comparison, detailing the pros and cons compared to SEARNN, the novel training algorithm we introduce in Chapter 7.

6.4 Goal and contributions

We have detailed the most common approach to training RNNs in Section 6.2.1. The known deficiencies of this method, presented in Section 6.2.2, provide the motivation for our aim in the second part of this thesis.

Our goal is to provide new training algorithms for recurrent neural networks that do not suffer from the limitations of MLE, and thus lead to improved model performance. In contrast to most of the alternative training approaches presented in Section 6.3, we do not try to adapt RL algorithms to the supervised setting. Instead, we propose to use ideas from the structured prediction field, in particular from the "learning to search" approach introduced by Daumé et al. (2009) and later refined by Ross and Bagnell (2014) and Chang et al. (2015) among others.

In Chapter 7, we propose SEARNN, a novel training algorithm for recurrent neural networks inspired by the L2S approach to structured prediction. SEARNN leverages test-alike search space exploration to introduce global-local losses that are closer to the test error than the MLE surrogate. Compared to other L2S-inspired approaches, SEARNN is derived from the more modern SEARN family of algorithms, and is more generally applicable.

We provide the rationale for using SEARN as a basis for improved RNN training in Section 7.2, by underlining surprisingly strong similarities between this algorithm and RNNs.

We demonstrate the usefulness of our method through comprehensive experiments on three challenging datasets (see Sections 7.3 and 7.5). We investigate scaling schemes to allow SEARNN to handle tasks with large vocabulary sizes and long sequences (Sections 7.4 and 7.7).

We investigate the behavior of our algorithm compared to MLE in details (Section 7.6). Finally, we contrast our novel approach to the related L2S- and RL-inspired methods in Section 7.8.

Chapter 7 covers and extends the conference paper Leblond et al. (2018a).

Chapter 7

SEARNN

In this Chapter, we propose SEARNN, a novel training algorithm for recurrent neural networks (RNNs) inspired by the "learning to search" (L2S) approach to structured prediction.

RNNs have been widely successful in structured prediction applications such as machine translation or parsing, and are commonly trained using maximum likelihood estimation (MLE). Unfortunately, this training loss is not always an appropriate surrogate for the test error: by only maximizing the ground truth probability, it fails to exploit the wealth of information offered by structured losses. Further, it introduces discrepancies between training and predicting (such as exposure bias) that may hurt test performance.

Instead, SEARNN leverages test-alike search space exploration to introduce global-local losses that are closer to the test error. We first demonstrate improved performance over MLE on two different tasks: OCR and spelling correction. Then, we propose a subsampling strategy to enable SEARNN to scale to large vocabulary sizes. This allows us to validate the benefits of our approach on a machine translation task.

Finally, after contrasting the behavior of SEARNN models to MLE models, we provide an in-depth comparison of our new approach to the related work.

An open-source implementation of SEARNN and all the experiments in this chapter is available at: https://github.com/RemiLeblond/SeaRNN-open.

7.1 Introduction

Recurrent neural networks (RNNs) have been quite successful in structured prediction applications such as machine translation (Sutskever et al., 2014), parsing (Ballesteros et al., 2016) or caption generation (Vinyals et al., 2015). These models use the same repeated

cell (or unit) to output a sequence of tokens one by one. As each prediction takes into account all previous predictions, this cell learns to output the next token conditioned on the previous ones. The standard training loss for RNNs is derived from maximum likelihood estimation (MLE): we consider that the cell outputs a probability distribution at each step in the sequence, and we seek to maximize the probability of the ground truth tokens.

Unfortunately, this training loss is not a particularly close surrogate to the various test errors we want to minimize. A striking example of discrepancy is that the MLE loss is close to 0/1: it makes no distinction between candidates that are close or far away from the ground truth (with respect to the structured test error), thus failing to exploit valuable information. Another example of train/test discrepancy is called *exposure* or *exploration bias* (Ranzato et al., 2016): in traditional MLE training the cell learns the conditional probability of the next token, based on the previous ground truth tokens – this is often referred to as *teacher forcing*. However, at test time the model does not have access to the ground truth, and thus feeds its own previous predictions to its next cell for prediction instead. For more information on MLE training, see Section 6.2.1.

Improving RNN training thus appears as a relevant endeavor, which has received much attention recently. In particular, ideas coming from reinforcement learning (RL), such as the REINFORCE and ACTOR-CRITIC algorithms (Ranzato et al., 2016; Bahdanau et al., 2017), have been adapted to derive training losses that are more closely related to the test error that we actually want to minimize.

In order to address the issues of MLE training, we propose instead to use ideas from the structured prediction field, in particular from the "learning to search" (L2S) approach introduced by Daumé et al. (2009) and later refined by Ross and Bagnell (2014) and Chang et al. (2015) among others.

Contributions. In Section 7.2, we present the "learning to search" approach and make explicit its strong links with RNNs.

In Section 7.3, we present SEARNN, a novel training algorithm for RNNs, using ideas from L2S to derive a *global-local* loss that is much closer to the test error than MLE. We demonstrate that this novel approach leads to significant improvements on two difficult structured prediction tasks, including a spelling correction problem recently introduced in Bahdanau et al. (2017).

As this algorithm is quite costly, we investigate scaling solutions in Sections 7.4 and 7.7. We explore subsampling strategies that allows us to considerably reduce training times, while maintaining improved performance compared to MLE. We apply this new algorithm to machine translation and report significant improvements in Section 7.5.

In Section 7.6, we compare the behavior of SEARNN and MLE models, demonstrating that the former do not suffer from the same overconfidence issues as the latter.

Finally, we contrast our novel approach to the related L2S and RL-inspired methods in Section 7.8.

7.2 Links between RNNs and learning to search

The L2S approach to structured prediction was first introduced by Daumé et al. (2009). The main idea behind it is a *learning reduction* (Beygelzimer et al., 2016): transforming a complex learning problem (structured prediction) into a simpler one that we know how to solve (multiclass classification). To achieve this, Daumé et al. (2009) propose in their SEARN algorithm to train a shared local classifier to predict each token *sequentially* (conditioned on all inputs and all past decisions), thus searching greedily step by step in the big combinatorial space of structured outputs. The idea that tokens can be predicted one at a time, conditioned on their predecessors, is central to this approach.

The training procedure is iterative: at the beginning of each round, one uses the current model (or policy [1]) to build an intermediate dataset to train the shared classifier on. The specificity of this new dataset is that each new sample is accompanied by a cost vector containing one entry per token in the output vocabulary $\mathcal{A}$.

To obtain these cost vectors, one starts by applying a *roll-in* policy to predict all the tokens up to T, thus building one trajectory (or exploration path) in the search space per sample in the initial dataset. Then, at each time step t, one picks arbitrarily each possible token (diverging from the roll-in trajectory) and continues predicting to finish the modified trajectory using a *roll-out* policy. One finally computes the cost of all the obtained sequences, and ends up with T vectors (one per time step) of size $|\mathcal{A}|$ (the number of possible tokens) for every sample. This process is described by Figure 7-1.

One then extracts features from the "context" at each time step t (which encompasses the full input and the previous tokens predicted up to t during the roll-in). [2] Combining the cost vectors to these features yields the new intermediary dataset. The original problem is thus reduced to multi-class *cost-sensitive* classification. Once the shared classifier has been fully trained on this new dataset, the policy is updated for the next round. The algorithm is described more formally in Algorithm 9. Theoretical guarantees for various policy updating

1. Note that the vocabulary used in this literature is slightly different from that of RNNs: tokens are rather referenced as actions, predictions as decisions and models as policies.
2. This is often referred to as "search state" in the L2S literature, but we prefer calling it context to avoid confusion with the RNN hidden state.

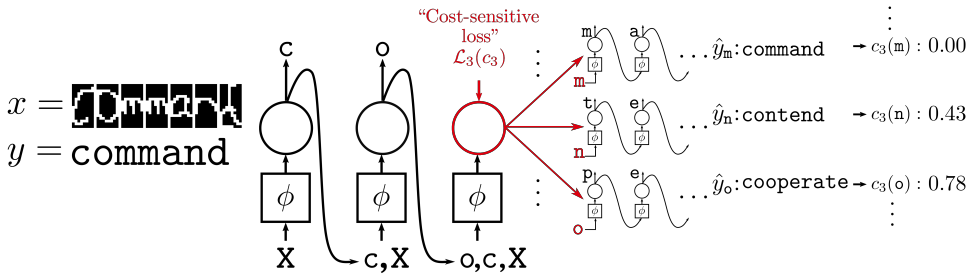


Figure 7-1 – Illustration of the roll-in/roll-out mechanism used in SEARN. The goal is to obtain one vector of costs for each time step in order to define a *cost-sensitive loss* to train the shared classifier. These vectors have one entry per possible token. Here, we show how to obtain the vector of costs associated with the third decision.
First, we use a *roll-in* policy to make the sequence of decisions leading to the decision of interest. Each decision is taken by extracting features from the context and feeding these features to the model. The context initially contains only the original input, but it is augmented at each time step by adding the associated decision. Note that the extractor is not learned in SEARN, and that the model is not deep: the process by which we put decisions in the context is not differentiated through.
Second, we proceed to the *roll-out* phase. For each possible token a (here in red), we ask the model – alternatively, a policy of reference – to finish predicting as if this token had been chosen at this time step (this is as easy as adding the token to the context). We thus obtain one predicted sequence $\hat{y}_a$ per token. Comparing it to the ground truth sequence y yields the associated cost $c(a)$.

rules are provided by e.g. Daumé et al. (2009) and Chang et al. (2015).

Roll-in and roll-out policies. The policies used to create the intermediate datasets fulfill different roles. The *roll-in* policy controls what part of the search space the algorithm explores, while the *roll-out* policy determines how the cost of each token is computed. The main possibilities for both roll-in and roll-out are explored by Chang et al. (2015).

The *reference* policy tries to pick the optimal token based on the ground truth. During the roll-in, it corresponds to picking the ground truth. For the roll-out phase, while it is easy to compute an optimal policy in some cases (e.g. for the Hamming loss where simply copying the ground truth is also optimal), it is often too expensive (e.g. for BLEU score). One then uses a heuristic (in our experiments the reference policy is to copy the ground truth for both roll-in and roll-out unless indicated otherwise).

The *learned policy* simply uses the current model instead, and the *mixed* policy stochastically combines both. According to Chang et al. (2015), the best combination when the reference policy is poor is to use a learned roll-in and a mixed roll-out.

Algorithm 9 SEARN algorithm (adapted from Daumé et al. (2009), Figure 1.)

Initialize a policy h with the reference policy π.
for i in 1 to N **do**
    # Start of round i.
    Initialize the set of cost-sensitive examples $S \leftarrow \emptyset$.
    # Create the intermediate dataset for round i.
    for (x, y) in the ground truth input/output structured pairs **do**
        # Perform the roll-in (actually only run once).
        Compute predictions under the current policy, $(\hat{y}_1, ..., \hat{y}_{T_x}) \sim h, x$.
        for t in 1 to T_x **do**
            Compute input features $\phi(s_t)$ for context $s_t = (x, \hat{y}_1, ..., \hat{y}_t)$.
            Initialize a cost vector $c_t = \langle\rangle$.
            # Perform the roll-outs for each action to fill the cost vector.
            for each possible token $a \in \mathcal{A}$ **do**
                Get a full sequence $\hat{y}_t(a)$ by applying an expert policy, starting from $(x, \hat{y}_{1..t}, a)$.
                Collect the cost $c_t(a)$ by comparing $\hat{y}_t(a)$ and y.
            end for
            Add cost-sensitive example (ϕ, c) to S
        end for
    end for
    Learn a classifier h' on S.
    Interpolate $h \leftarrow \beta h' + (1 - \beta)h$.
end for
Return h.

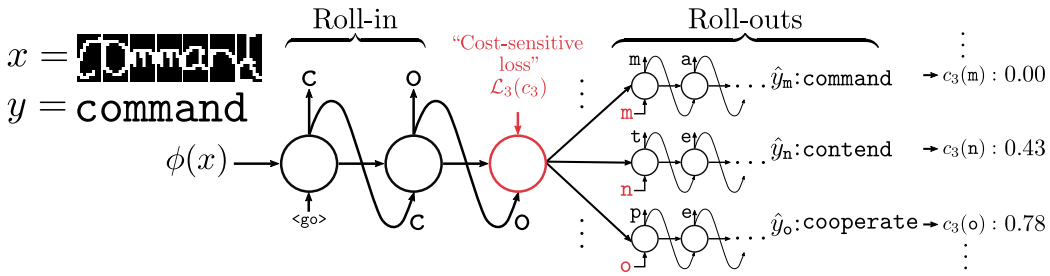


Figure 7-2 – Illustration of the roll-in/roll-out mechanism used in SEARNN. The goal is to obtain a vector of costs for each cell of the RNN in order to define a *cost-sensitive loss* to train the network. These vectors have one entry per possible token. Here, we show how to obtain the vector of costs for the red cell. First, we use a *roll-in* policy to predict until the cell of interest. We highlight here the *learned* policy where the network passes its own prediction to the next cell. Second, we proceed to the *roll-out* phase. We feed every possible token (illustrated by the red letters) to the next cell and let the model predict the full sequence. For each token a, we obtain a predicted sequence $\hat{y}_a$. Comparing it to the ground truth sequence y yields the associated cost $c(a)$.

Links to RNNs. One can identify the following interesting similarities between a greedy approach to RNNs and L2S. Both models handle sequence labeling problems by outputting tokens recursively, conditioned on past decisions. Further, the RNN "cell" is shared at each time step and can thus also be seen as a shared local classifier that is used to make structured predictions, as in the L2S framework. In addition, there is a clear equivalent to the choice of roll-in policy in RNNs. Indeed, teacher forcing (conditioning the outputs on the ground truth) can be seen as the roll-in *reference policy* for the RNN. Instead, if one conditions the outputs on the previous predictions of the model, then we obtain a roll-in *learned policy*.

Despite these connections, many differences remain. Amongst them, the fact that no roll-outs are involved in standard RNN training. We thus consider next whether ideas coming from L2S could mitigate the limitations of MLE training for RNNs. In particular, one key property of L2S worth porting over to RNN training is that the former fully leverages structured losses information, contrarily to MLE as previously noted in Section 6.2.2.

7.3 Improving RNN training with L2S

Since we are interested in leveraging structured loss information, we can try to obtain it in the same fashion as L2S. The main tool that L2S uses in order to construct a cost-sensitive dataset is the roll-out policy. In many classical structured prediction use cases, one does not need to follow through with a policy because the "cost-to-go" that the roll-out yields is either free or easily computable from the ground truth. We are however also interested in

cases where this information is unavailable, and roll-outs are needed to approximate it (e.g. for machine translation). This leads to several questions. How can we integrate roll-outs in a RNN model? How do we use this additional information, i.e. what loss do we use to train the model on? How do we make it computationally tractable?

7.3.1 The SEARNN Algorithm

The basic idea of the SEARNN algorithm is quite simple: we borrow from L2S the idea of using a *global* loss for each *local* cell of the RNN. As in L2S, we first compute a *roll-in* trajectory, following a specific roll-in policy. Then, at each step t of this trajectory, we compute the costs $c_t(a)$ associated with each possible token a. To do so we pick a at this step and then follow a *roll-out* policy to finish the output sequence $\hat{y}_a$. We then compare $\hat{y}_a$ with the ground truth using the test error itself, rather than a surrogate. By repeating this for the T steps we obtain T cost vectors. We use this information to derive one *cost-sensitive* training loss for each cell, which allows us to compute an update for the parameters of the model. The full process for one cell is illustrated in Figure 7-2.

Our losses are *global-local*, in the sense that they appear at the local level but all contain sequence-level information. Our final loss is the sum over the T local losses. We provide the pseudo-code for SEARNN in Algorithm 10.

7.3.2 Adaptation to RNNs

SEARNN appears quite similar to L2S, but there are a few key differences that merit more explanation.

Choosing a multi-class classifier. As the RNN cell can serve as a multi-class classifier, in SEARNN we could pick the cell as a (shallow) shared classifier, whose input are features extracted from the full context by the previous cells of the RNN. Instead, we pick the RNN itself, thus getting a (deep) shared classifier that also learns the features directly from the context. The difference between the two options is more thoroughly detailed in Appendix D.2. Arbitrarily picking a token a during the roll-out phase can then be done by emulating the teacher forcing technique: if predicted tokens are fed back to the model (say if the roll-out policy requires it), we use a for the next cell (instead of the prediction the cell would have output). We also use a in the output sequence before computing the cost.

Choosing a cost-sensitive loss. We now also explain our choice for the training loss function derived from the cost vectors. One popular possibility from L2S is to go the full

Algorithm 10 SEARNN algorithm (for a simple encoder-decoder network)

Initialize the weights ω of the RNN network.
for i in 1 to N **do**
    Sample B ground truth input/output structured pairs $\{(x^1, y^1), \cdots, (x^B, y^B)\}$
    # Perform the roll-in/roll-outs to get the costs. This step can be heavily parallelized.
    for b in 1 to B **do**
        Compute input features $\phi(x^b)$
        # Roll-in.
        Run the RNN until cell t with $\phi(x^b)$ as initial state, following the roll-in policy
        (see Appendix D.1 for details in the case of reference roll-in policy)
        Store the sequence of hidden states in order to perform several roll-outs
        for t in 1 to T **do**
            # Roll-outs for all actions in order to collect the cost vector at the t^{th} cell.
            for a in 1 to A **do**
                Pick a decoding method (e.g. greedy or beam search)
                Run the RNN from the t^{th} cell to the end by first enforcing action a at cell t, and then following the decoding method.
                Collect the cost $c_t^b(a)$ by comparing the obtained output sequence $\hat{y}_t^b(a)$ to y^b
            end for
        end for
    end for
    Derive a loss for each cell from the collected costs
    Update the parameters of the network ω by doing a single gradient step
end for

reduction route down to binary classification. However, this technique involves creating multiple new datasets (which is hard to implement as part of a neural network), as well as training $|\mathcal{A}|^2$ binary classifiers. Instead, we simply work with the multi-class classifier encoded by the RNN cell with training losses defined next.

In the following, each loss is defined at the cell level. The global loss is the sum of all T losses. $s_t(a)$ refers to the score output by cell t for token a.

Expected costs. The first loss we tried is a simple conditional expected loss: as we have access to both the costs of each token and their conditional probability according to the model (we can obtain this quantity, denoted p_t, by applying a softmax layer on top of the vector of scores s_t), we can compute the expected cost at each cell.

$$\mathcal{L}_t(s_t; c_t) = \mathbb{E}_{p_t} c_t = \frac{\sum_{i=1}^{A} e^{s_t(i)} c_t(i)}{\sum_{i=1}^{A} e^{s_t(i)}}. \tag{7.1}$$

In this case we were not able to successfully train the model as the loss saturated extremely fast. It appears this behavior is often observed when trying to apply policy gradient algorithms on top of softmax layers, which is consistent with the fact that other RL-inspired methods usually combine their RL objective with the classical MLE one, or rely on warm-starting. Ding and Soricut (2017) detail this issue and propose a work-around: one needs to add an exponential and a logarithm in the right places of the loss. We have toyed around with this idea but have not been successful with it as of yet.

Structured hinge loss (SHL). Our next step was to try classical structured prediction losses. We considered the (cost-sensitive) structured hinge loss commonly used for structured SVMs (Tsochantaridis et al., 2005):

$$\mathcal{L}_t(s_t; c_t) = \max_{a \in \mathcal{A}} (s_t(a) + c_t(a)) - s_t(a^\star) \text{ where } a^\star = \arg\min_{a \in \mathcal{A}} c_t(a)\,. \tag{7.2}$$

While this loss did enable the RNNs to learn, the overall performance was actually slightly worse than that of MLE. This may be due to the fact that RNNs have a harder time optimizing the resulting objective, compared to others more similar to the traditional MLE objective (which they have been tuned to train well on).

Consistent loss. This third loss is also inspired from traditional structured prediction. Following Lee et al. (2004), we define:

$$\mathcal{L}_t(c_t) = \sum_{a \in \mathcal{A}} c_t(a) \ln(1 + \exp(\tilde{s}_t(a))) \text{ where } \tilde{s}_t(a) = s_t(a) - \frac{1}{A} \sum_{a \in \mathcal{A}} s_t(a)\,. \tag{7.3}$$

Unfortunately, we encountered optimization issues and could not get significant improvements over the MLE baseline.

Log-loss (LL). We now introduce two of the more successful losses we used. A central idea in L2S is to learn the target tokens the model should aim for. This is more meaningful than blindly imposing the ground truth as target, in particular when the model has deviated from the ground truth trajectory. Golberg and Nivre (2012) refer to this technique as using *dynamic oracles*. In the context of RNN training, we call this approach *target learning*.

Our first loss is thus a simple log-loss with the minimal cost token as target:

$$\mathcal{L}_t(s_t; c_t) = -\log \left(e^{s_t(a^\star)} \Big/ \sum_{i=1}^{A} e^{s_t(i)} \right) \text{ where } a^\star = \arg\min_{a \in \mathcal{A}} c_t(a)\,. \tag{7.4}$$

It is structurally similar to MLE. The only difference is that instead of maximizing the probability of the ground truth action, we maximize the probability of the best performing action with respect to the cost vector. This similarity is a significant advantage from an optimization perspective: as RNNs have mostly been trained using MLE, this allows us to leverage decades of previous work. Note that when the reference policy is to simply copy the ground truth (which is sometimes optimal, e.g. when the test error is the Hamming loss), $a^\star$ is always the ground truth token. LL with reference roll-in and roll-out is in this case *equivalent* to MLE.

Kullback-Leibler divergence (KL). The log-loss approach appears to be relatively wasteful with the structured information we have access to since we are only using the minimal cost value. To exploit this information more meaningfully, we consider the following approach: we convert each cost vector into a probability distribution (e.g. through a softmax operator) and then minimize a divergence between the current model distribution P_M and the "target distribution" P_C derived from the costs. As the MLE objective itself can be expressed as the KL divergence between D_{gt} (a Dirac distribution with full mass on the ground truth) and P_M, we also choose to minimize the KL divergence between P_C and P_M. Since the costs are considered fixed with respect to the parameters of the model, our loss is

equivalent to the cross-entropy between P_C and P_M.

$$\mathcal{L}_t(s_t; c_t) = -\sum_{a=1}^{A} \Big(P_C(a) \log \big(P_M(a) \big) \Big) \quad \begin{array}{l} \text{where } P_C(a) = e^{-\alpha c_t(a)} \big/ \sum_{i=1}^{A} e^{-\alpha c_t(i)} \\ \text{and } P_M(a) = e^{s_t(a)} \big/ \sum_{i=1}^{A} e^{s_t(i)}. \end{array} \tag{7.5}$$

α is a scaling parameter that controls how peaky the target distributions are. It can be chosen using a validation set. The associated gradient update discriminates between tokens based on their costs. Compared to LL, KL leverages the structured loss information more directly and thus mitigates the 0/1 nature of MLE better.

Log-loss with cost-augmented softmax (LLCAS). LLCAS is another attempt at leveraging the structured information we have access to more meaningfully, through a slight modification of LL. We add information about the full costs in the exponential, following e.g. Pletscher et al. (2010); Gimpel and Smith (2010); Hazan and Urtasun (2010).

$$\mathcal{L}_t(s_t; c_t) = -\log \left(e^{s_t(a^\star) + \alpha c_t(a^\star)} \big/ \sum_{i=1}^{A} e^{s_t(i) + \alpha c_t(i)} \right) \tag{7.6}$$
$$\text{where } a^\star = \arg\min_{a \in \mathcal{A}} c_t(a)\,.$$

α is a scaling parameter that ensures that the scores of the model and the costs are not too dissimilar, and can be chosen using a validation set. The associated gradient update discriminates between tokens based on their costs. Interestingly, this loss can be seen as a smooth version of the SHL loss.

Although it leverages the structured loss information more directly and thus should in principle mitigate the 0/1 nature of MLE better, we did not observe any significant improvements over LL, even after tuning the scaling parameter α.

Optimization. Another difference between SEARN and RNNs is that RNNs are typically trained using stochastic gradient descent, whereas SEARN is a batch method. In order to facilitate training, we decide to adapt the optimization process of LOLS, an online variant of SEARN introduced by Chang et al. (2015). At each round, we select a random mini-batch of samples, and then take a single gradient step on the parameters with the associated loss (contrary to SEARN where the reduced classifier is fully trained at each round).

Note that we do not need the test error to be differentiable, as our costs $c_t(a)$ are fixed when we minimize our training loss. This corresponds to defining a different loss at each round, which is the way it is done in L2S. In this case our gradient is unbiased. However, if instead we consider that we define a single loss for the whole procedure, then the costs

Dataset		A	T	Cost	MLE	AC [4]		LL			KL		
							roll-in	learned	reference	learned	learned	reference	learned
							roll-out	mixed	learned	learned	mixed	learned	learned
OCR		26	15	Hamming	2.8	–		1.9	2.5	1.8	**1.0**	1.4	1.1
Spelling	0.3	43	10	edit	19.3	18.7		**17.7**	19.5	17.8	**17.7**	19.5	**17.7**
	0.5				41.9	37.4		**37.1**	43.2	37.6	38.1	43.2	**37.1**

Table 7.1 – Comparison of the SEARNN algorithm with MLE for different cost-sensitive losses and roll-in/roll-out policies. We provide the number of actions A and the maximum sequence length T. Note that we use 0.5 as the mixing probability for the mixed roll-out policy. We ran the ACTOR-CRITIC algorithm from Bahdanau et al. (2017) on our data splits for the spelling task and report the results in the AC column (the results reported in Bahdanau et al. (2017) were not directly comparable as they used a different random test dataset each time).

depend on the parameters of the model and we effectively compute an approximation of the gradient. Whether it is possible not to fix the costs and to backpropagate through the roll-in and roll-out remains an open problem.

7.3.3 Expected and empirical benefits

Expected benefits. SEARNN can improve performance because of a few key properties. First, our losses leverage the test error, leading to potentially much better surrogates than MLE.

Second, all of our training losses (even plain LL) leverage the structured information that is contained in the computed costs. This is much more satisfactory than MLE which does not exploit this information and ignores nuances between good and bad candidate predictions. Indeed, our hypothesis is that the more complex the error is, the more SEARNN can improve performance.

Third, the exploration bias we find in teacher forcing can be mitigated by using a "learned" roll-in policy, which may be the best roll-in policy for L2S applications according to Chang et al. (2015).

Fourth, the loss at each cell is *global*, in the sense that the computed costs contain information about full sequences. This may help with the classical vanishing gradients problem that is prevalent in RNN training and motivated the introduction of specialized cells such as LSTMs (Hochreiter and Schmidhuber, 1997) or GRUs (Cho et al., 2014).

Experiments. In order to validate these theoretical benefits, we ran SEARNN on two datasets and compared its performance against that of MLE. For a fair comparison, we use the same optimization routine for all methods. We pick the one that performs best for the

MLE baseline. Note that in all the experiments of this chapter, we use greedy decoding, both for our cost computation and for evaluation. Furthermore, whenever we use a mixed roll-out we always use 0.5 as our mixin parameter, following Chang et al. (2015). [3]

The first dataset is the optical character recognition (OCR) dataset introduced in Taskar et al. (2003). The task is to output English words given an input sequence of handwritten characters. We use an encoder-decoder model with GRU cells (Cho et al., 2014) of size 128. For all runs, we use SGD with constant step-size 0.5 and batch size of 64. The cost used in the SEARNN algorithm is the Hamming error. We report the total Hamming error, normalized by the total number of characters on the test set.

The second dataset is the Spelling dataset introduced in Bahdanau et al. (2017). The task is to recover correct text from a corrupted version. This dataset is synthetically generated from a text corpus (One Billion Word dataset): for each character, we decide with some fixed probability whether or not to replace it with a random one. The total number of tokens A is 43 (alphabet size plus a few special characters) and the maximum sequence length T is 10 (sentences from the corpus are clipped). We provide results for two sub-datasets generated with the following replacement probabilities: 0.3 and 0.5. For this task, we follow Bahdanau et al. (2017) and use the edit distance as our cost. It is defined as the edit distance between the predicted sequence and the ground truth sequence divided by the ground truth length. We reuse the attention-based encoder-decoder model with GRU cells of size 100 described in (Bahdanau et al., 2017). For all runs, we use the Adam optimizer (Kingma and Ba, 2015) with learning rate 0.001 and batch size of 128. Results are given in Table 7.1, including ACTOR-CRITIC (Bahdanau et al., 2017) runs on our data splits as an additional baseline.

Key takeaways. First, SEARNN outperforms MLE by a significant margin on the two different tasks and datasets, which confirms our intuition that taking structured information into account enables better performance. Second, we observed that the best performing losses were those structurally close to MLE – LL and KL – whereas others did not improve results for a variety of reasons. This might be explained by the fact that RNN architectures and optimization techniques have been evolving for decades with MLE training in mind. Third, the best roll-in/out strategy appears to be combining a learned roll-in and a mixed roll-out, which is consistent with the claims from Chang et al. (2015). Fourth, although we expect SEARNN to make stronger improvements over MLE on hard tasks (where a simplistic roll-out policy – akin to MLE – is suboptimal), we do get improvements even when outputting the ground truth (regardless of the current trajectory) is the optimal policy.

3. We confirmed their finding empirically: the performance of SEARNN is not sensitive to this parameter.

7.4 Scaling up SEARNN

While SEARNN does provide significant improvements on the two tasks we have tested it on, it comes with a rather heavy price, since a large number of roll-outs (i.e. forward passes) have to be run in order to compute the costs. This number, $|\mathcal{A}|T$, is proportional both to the length of the sequences, and to the number of possible tokens. SEARNN is therefore not directly applicable to tasks with large output sequences or vocabulary size (such as machine translation) where computing so many forward passes becomes a computational bottleneck. Even though forward passes can be parallelized more heavily than backward ones (because they do not require maintaining activations in memory), their asymptotic cost remains in $\mathcal{O}(dT)$, where d is the number of parameters of the model.

There are a number of ways to mitigate this issue. In this chapter, we focus on subsampling both the cells and the tokens when computing the costs. That is, instead of computing a cost vector for each cell, we only compute them for a subsample of all cells. Similarly, we also compute these costs only for a small portion of all possible tokens. The speedups we can expect from this strategy are large, since the total number of roll-outs is proportional to both the quantities we are decreasing.

Sampling strategies. First, we need to decide how we select the steps and tokens that we sample. We have chosen to sample steps uniformly when we do not take all of them. On the other hand, we have explored several different possibilities for token sampling. The first is indeed the uniform sampling strategy. The 3 alternative samplings we tried use the current state of our model: stochastic current policy sampling (where we use the current state of the stochastic policy to pick at random), a biased version of current policy sampling where we boost the scores of the low-probability tokens, and finally a *top-k* strategy where we take the top k tokens according to the current policy. Note that the latter strategy (*top-k*) can be seen as a simplified variant of *targeted sampling* (Goodman et al., 2016), another smarter strategy introduced to help L2S methods scale (for more details, see Section 7.7). Finally, in all strategies we always sample the ground truth action to make sure that our performance is at least as good as MLE.

Adapting our losses to sampling. Our losses require computing the costs of all possible tokens at a given step. One could still use LL by simply making the assumption that the token with minimum cost is always sampled. However this is a rather strong assumption and it means pushing down the scores of tokens that were not even sampled and hence could not compete with the others. To alleviate this issue, we replace the full softmax by a layer

Dataset		MLE	LL	KL	sLL				sKL			
					uni.	pol.	bias.	top-k	uni.	pol.	bias.	top-k
OCR		2.8	1.9	1.0	1.7	1.8	1.8	1.5	1.2	1.2	**0.9**	1.4
Spelling	0.3	19.3	17.7	17.7	17.6	17.7	17.7	**17.6**	18.4	17.7	17.7	18.2
	0.5	41.9	37.1	38.1	37.0	37.1	36.6	**36.6**	37.8	37.6	37.1	38.0

Table 7.2 – Comparison of the SEARNN algorithm with MLE for different datasets using the sampling approach. sLL and sKL are respectively the subsampled version of the LL and the KL losses. All experiments were run with a learned roll-in and a mixed roll-out.

applied only on the tokens that were sampled (Jean et al., 2015). While the target can still only be in the sampled tokens, the unsampled tokens are left alone by the gradient update, at least for the first order dependency. This trick is even more needed for KL, which otherwise requires a "default" score for unsampled tokens, adding a difficult to tune hyperparameter. We refer to these new losses as sLL and sKL.

Experiments. The main goal of these experiments is to assess whether or not combining subsampling with the SEARNN algorithm is a viable strategy. To do so we ran the method on the same two datasets that we used in the previous section. We decided to only focus on subsampling tokens as the vocabulary size is usually the blocking factor rather than the sequence length. Thus we sampled all cells. We evaluate different sampling strategies and training losses. For all experiments, we use the learned policy for roll-in and the mixed one for roll-out and we sample 5 tokens per cell. Finally, we use the same optimization techniques than in the previous experiment.

Key takeaways. Results are given in Table 7.2. The analysis of this experiment yields interesting observations. First, and perhaps most importantly, subsampling appears to be a viable strategy to obtain a large part of the improvements of SEARNN while keeping computational costs under control. Indeed, we recover all of the improvements of the full method while only sampling a fraction of all possible tokens. Second, it appears that the best strategy for token sampling depends on the chosen loss. In the case of sLL, the *top-k* strategy performs best, whereas sKL favors the biased current policy. Third, it also seems like the best performing loss is task-dependent. Finally, this sampling technique yields a 5× running time speedup, therefore validating our scaling approach.

7.5 Neural Machine Translation

Having introduced a cheaper alternative SEARNN method enables us to apply it to a large-scale structured prediction task and to thus investigate whether our algorithm also improves upon MLE in more challenging real-life settings.

7.5.1 Experimental results

We choose neural machine translation as out task, and the German-English translation track of the IWSLT 2014 campaign (Cettolo et al., 2014) as our dataset, as it was used in several related papers and thus allows for easier comparisons. We reuse the pre-processing of Ranzato et al. (2016), obtaining training, validation and test datasets of roughly 153k, 7k and 7k sentence pairs respectively with vocabularies of size 22822 words for English and 32009 words for German.

For fair comparison to related methods, we use similar architectures. To compare with BSO and ACTOR-CRITIC, we use an encoder-decoder model with GRU cells of size 256, with a bidirectional encoder and single-layer RNNs. For the specific case of MIXER, we replace the recurrent encoder with a convolutional encoder as in Ranzato et al. (2016) . We use Adam as our optimizer, with an initial learning rate of 10^{-3} gradually decreasing to 10^{-5}, and a batch size of 64. We select the best models on the validation set and report results both without and with dropout (0.3).

Regarding the specific settings of SEARNN, we use a reference roll-in and a mixed roll-out. Additionally, we sample 25 tokens at each cell, following a mixed sampling strategy (detailed in Appendix D.3). We use the best performing loss on the validation set, i.e. the KL loss with scaling parameter 200.

The traditional evaluation metric for such tasks is the BLEU score (Papineni et al., 2002). As we cannot use this corpus-wide metric to compute our sentence-level intermediate costs, we adopt the alternative smoothed BLEU score of Bahdanau et al. (2017) as our cost. We use a custom reference policy (detailed in Appendix D.3). We report the corpus-wide BLEU score on the test set in Table 7.3.

Key takeaways. First, the significant improvements SEARNN obtains over MLE on this task (2 BLEU points without dropout) show that the algorithm can be profitably applied to large-scale, challenging structured prediction tasks at a reasonable computational cost.

Second, our performance is on par or better than those of related methods with comparable baselines. Our performance using a convolutional encoder is similar to that of MIXER. Compared to BSO (Wiseman and Rush, 2016), our baseline, absolute performance and

MLE*	MIXER*	SEARNN (conv)	MLE†	BSO†	MLE'	AC'	MLE	SEARNN	MLE (dropout)	SEARNN (dropout)
17.7	20.7	20.5	22.5	23.8	25.8	27.5	24.8	26.8	27.4	**28.2**

Table 7.3 – Comparison of SEARNN with MIXER (Ranzato et al., 2016), BSO (Wiseman and Rush, 2016) and ACTOR-CRITIC (Bahdanau et al., 2017) on the IWSLT 14 German to English machine translation dataset. The asterisk (*), dagger (†) and apostrophy (') indicate results reproduced from Ranzato et al. (2016), Wiseman and Rush (2016) and Bahdanau et al. (2017), respectively. We use a reference roll-in and a mixed roll-out for SEARNN, along with the subsampled version of the KL loss and a scaling factor of 200. SEARNN (conv) indicates that we used a convolutional encoder instead of a recurrent one for fair comparison with MIXER.

improvements are all stronger. While SEARNN presents similar improvements to ACTOR-CRITIC, the absolute performance is slightly worse. This can be explained in part by the fact that SEARNN requires twice less parameters during training.

7.5.2 In-depth analysis

Roll-in policy. The learned roll-in policy performed poorly for this specific task, so we used instead a reference roll-in. While this observation seems to go against the L2S analysis from Chang et al. (2015), it is consistent with another experiment we ran: we tried applying scheduled sampling (Bengio et al., 2015) – which uses a schedule of mixed roll-ins – on this dataset, but did not succeed in obtaining any improvements, despite using a careful schedule as proposed by their authors in private communications.

One potential explicative factor is that our reference policy is not good enough to yield valuable signal when starting from a poor roll-in. Another possibility is that the underlying optimization problem becomes harder when using a learned rather than a reference roll-in.

Alternatively, this result may illustrate a gap in the standard reduction theory from the L2S framework. Indeed, the standard reduction analysis (Daumé et al., 2009; Chang et al., 2015) guarantees that the level of performance of the classifier on the reduced problem translates to overall performance on the initial problem. However, this does not take into account the fact that the reduced problem may be harder or easier, depending on the choice of roll-in/roll-out combination. In this case, it appears that using a learned roll-in may have lead to a harder reduced problem and thus ultimately worse overall performance.

One final element of explanation is that there is a subtle interaction between learned roll-in policies and tasks with both variable size outputs and specific losses, which implies that models trained with a learned roll-in benefit from a lot less training signal than those trained with a reference roll-in. We now detail the mechanisms behind this observation.

Counting the roll-outs. Consider what happens when we use SEARNN with a learned roll-in and a learned roll-out. At the beginning of training, the weights of the model are random, and so are its predictions. In particular, the probability of outputting the <EOS> symbol indicating the end of a sequence is inversely proportional to the size of the output vocabulary. In case the output vocabulary is big – say, for NMT – this implies that there are essentially no <EOS> symbols in the completed trajectories (the one exception is when the arbitrary token is itself <EOS>). The associated costs are consequently very high, and at many cells the best performing token will be <EOS>, which cuts the sequence short.

As a consequence, we observe that over the first few gradient steps, SEARNN-trained models (with both a learned roll-in and a learned roll-out) develop a very strong bias towards outputting <EOS>. As the roll-in follows the model policy, after just a few iterations most roll-ins then consist in sequences containing only <EOS>.

Now, we note that we do not perform any roll-outs for cells that come after the first <EOS> symbol in the roll-in. As tokens after <EOS> are irrelevant, all costs computed from sequences that differ after <EOS> would be equal, and so these costs would not provide informative training signal.

The combination of these two facts result in a lot fewer costs being computed per iteration when using a learned roll-out policy compared to a reference one. Indeed, after just a few gradient steps, the roll-ins contain very early <EOS> symbols, which implies that only very few roll-outs are done. So the models start off prediction mainly <EOS>, and then they only have a fraction of the training signal to improve their performance.

These observations are borne out empirically: on NMT for instance, we see that when training a model with a learned roll-in, the amount of costs computed over the first 1000 iterations is twice less than when using a reference roll-in. Over the first 10000 iterations, that fraction grows to 75%. Although it continues growing slowly afterwards, it never does quite catch up.

Although the models eventually escape the regime where they output mainly <EOS>, this process is slow and costly, and may be one of the reasons why the performance we observed for the learned roll-in policy was poor for NMT.

Now, in our example the model was trained with both a learned roll-in and a learned roll-out. With a reference roll-out, only the roll-ins are random, so completed trajectories do contain <EOS> symbols, and thus our explanation for the initial bias towards <EOS> does not hold. However, somewhat surprisingly, we still observe this very strong bias empirically. The explanation for this phenomenon lies in the structure of the BLEU loss. Indeed, to maximize this loss when starting from a random roll-in, it's often better to stop predicting by outputting <EOS> than to finish with the ground truth suffix. We believe this property to

be fairly specific to BLEU.

The discrepancy in terms of training signal may explain in part the poor performance of the learned roll-in policy (note that as a combination of reference and learned roll-out, the mixed rollout is also covered by our analysis).

Ghost signal. The issue with models trained with a learned roll-in is that no training signal is received after the end of the roll-in, which is susceptible to end prematurely. The cells after <EOS> in the roll-in but before the full length of the ground truth sequence are not taken into account in the loss (and hence in the gradient updates). One possible solution to alleviate this problem is to use *ghost signal* to compensate for the absence of information. The idea is quite simple: we use the ground truth to derive suitable targets for the problematic cells.

Let us focus on the log loss for now. We can simply use the ground truth tokens as targets whenever we do not have other signal. This can be achieved quite easily by adding a small bonus to the costs of the ground truth tokens everywhere. This bonus is only relevant when all costs are equal, i.e. in the case of problematic cells.

In some sense, this technique can be seen as complementing the SEARNN objective with the MLE objective whenever the SEARNN objective is uninformative. Of course, from a theoretical perspective we are trying to teach our model to make decisions after the end of sequence token, which should not be useful. However, we observe marked improvements empirically. The models trained with a learned roll-in and ghost signal perform a lot better than without, although they still do not attain quite the same performance as the models with reference roll-ins.

How to use this technique combined with the KL loss is less direct, as one needs to leverage the ground truth to construct full ghost distributions rather than simply targets. We have not yet investigated this possibility.

7.6 Model confidence and beam search

On the overconfidence of MLE trained models. One known issue with MLE-trained RNN models that we have not mentioned until know is that these models' predictions tend to be extremely confident, even when they are wrong (see e.g. Pereyra et al. (2017)). This is of course problematic, as even if we cannot expect models to output perfect predictions, we would at least prefer wrong predictions to come with weak confidence. More generally, low entropy output distributions are often a sign of overfitting.

Another reason why this is suboptimal is that the output probabilities for wrong tokens are also indicators of how well a model generalizes: the probabilities of tokens related to the ground truth tokens should be bigger than the probabilities associated with random tokens. Overconfident models tend to output Dirac-like distributions which do not differentiate tokens based on their proximity to the ground truth. In this sense, it's not surprising that MLE training, which effectively ignores similarity in favor of exactness, does not lead to distributions that reflect lexical proximity.

Finally, some specific applications actually benefit from non-negligible policy variance. For instance, in reinforcement learning high entropy policies lead to better exploration and overall performance (Williams and Peng, 1991).

As model overconfidence is linked to overfitting, practitioners have sought to alleviate it by applying regularization techniques. One of the most prominent of these techniques is called *label smoothing* (Pereyra et al., 2017). It consists in adding a uniform distribution term to the Dirac target distribution obtained by MLE; equivalently, instead of always using the ground truth token as a target, for a fixed fraction of examples a token chosen uniformly at random is used.

Are SEARNN models overconfident? As SEARNN training differs in many ways from MLE training, it is worth wondering whether SEARNN-trained model suffer from this undesirable property or not. Indeed, compared to MLE, a few key elements can give us hope. First, SEARNN training does not exclusively focus on maximizing the probability of the ground truth. Second, SEARNN relies on comparisons between the performance of different tokens. Finally, when using the KL loss, we give the whole cost distribution as a target for the output distribution.

In order to answer our question, we have looked at the output distributions of our final MLE and SEARNN models for NMT. We find that on average the highest probability token in the MLE model output is 0.65 (and in the $[0.63; 0.67]$ range over 95% of the time). This level of confidence is very high, considering that the vocabulary size for this task exceeds 20000.

In contrast, when using SEARNN training and the log loss (LL), this average highest probability is only around 0.065 (in the $[0.055; 0.075]$ interval in over 95% instances). When using the KL loss, this quantity goes down to around 0.02 (in the $[0.018; 0.022]$ interval in over 95% instances). This difference of more than an order of magnitude implies that SEARNN models have much higher entropy than MLE ones, and thus suffer less from overconfidence.

We now detail a few experiments we ran to rule out some possible explanations for this

behavior. First, we were able to confirm these findings (with sometimes slightly reduced but still significant discrepancies) with or without dropout. Second, to make sure these observation were not the result of our sub-sampling scheme (where the gradient step only flows through the scores of the sampled tokens, thus leaving untouched the scores of the vast majority of tokens at the first order), we trained an MLE model with sub-sampling – that is, with gradient steps flowing only through the same amount of sampled tokens as in our SEARNN runs. The resulting model exhibited the same overconfidence level as regularly-trained MLE models, although somewhat surprisingly slightly improved performance. Finally, we also observed the phenomenon on our other datasets, although the differences were not as dramatic. We conjecture that when the performance of the model reaches very high levels of performance (as is the case for the OCR dataset for instance), even the SEARNN models become quite confident.

Link to label smoothing. We have mentioned label smoothing, a technique designed explicitly to avoid the overconfidence issues of RNNs. Interestingly, we can draw a parallel between this approach and SEARNN with the KL loss – indeed, we will show in the next paragraph that in specific settings both are equivalent. In both cases, we use non-Dirac probability distributions as target (with an additional uniform distribution in the case of label smoothing and a distribution computed from the costs vectors for SEARNN). The difference is that in the case of label smoothing, an arbitrary additional distribution is used to mitigate the one-hot aspect of the target distribution, while in our case we use a distribution that is based on costs computed using the model itself. In this sense, we can view SEARNN with the KL loss as *learned* label smoothing. This perspective offers a way of understanding why our KL models are less confident.

Strict equivalence with the Hamming loss. We have seen that when the loss function is the Hamming loss, the reference policy is to simply output the ground truth. In this case, LL with a reference roll-in and roll-out is equivalent to MLE. Interestingly, in the same setup KL is also strictly equivalent to label smoothing. Indeed, the vector of costs can be written as a vector with equal coordinates minus a one-hot vector with all its mass on the ground truth token. After transformation through a softmax operator, this yields the same target distribution as in label smoothing.

When using LL instead of KL, the target distribution does have a Dirac shape and so there is no direct equivalence. However, the targets depend on the costs, which themselves depend on the model. As a consequence, as training progresses, the targets can change and are thus not as concentrated as in MLE training.

β	1.0	1.2	1.4	1.6	1.8	2.0	2.2	2.4	2.6	2.8	3.0	4.0	10	100
	27.59	28.53	29.20	29.73	30.09	30.26	30.39	30.42	30.34	30.28	30.23	29.97	29.65	29.53

Table 7.4 – Evolution of performance with the beam rescaling factor for our best SEARNN model. The performance is measured on the validation set. $\beta = 2.4$ appears to be the optimal parameter in the grid search.

Beam-search performance. As we have explained in Section 6.1.4, several search procedures can be used to compute the sequence with maximum score according to a model, including greedy search and beam search. Using beam search is linearly more computationally expensive than greedy search, as it requires maintaining k trajectories, but it typically leads to improved performance.

We do observe this phenomenon on our MLE models: when using a beam size of 10, performance increases by 1.4 BLEU points to 26.2 for the model without dropout and by 1.2 points for the model trained with dropout.

Surprisingly, for SEARNN models we see the opposite behavior: rather than increasing performance, using beam search makes it worse. This is in fact due to a negative interaction between beam search and high entropy models: as beam search allows more exploration and many tokens have similar probabilities, bad performing tokens can make it in. This does not happen with low entropy models, where token scores are highly delineated.

In this instance, it appears as though lack of confidence is detrimental. Fortunately, we were able to implement an easy workaround: when using beam search we introduce a rescaling parameter β which we apply to our output distributions to reduce their entropy. This parameter can be calibrated for on a validation set. Empirically, we found that $\beta = 2.4$ worked best (see Table 7.4): the performance of our SEARNN model with beam search then increased by 1.0 BLEU points to 27.8 without dropout, and by 0.8 points with dropout. Interestingly, this level of rescaling leads to roughly equivalent levels of confidence (about 0.6 on average for the highest probability token) than traditional MLE models.

We present the evolution of the performance for increasing values of β in Table 7.4 and the performance using beam size 10 for both MLE and SEARNN models in table 7.5. [4]

7.7 Scaling SEARNN up further

The cost of the full version of SEARNN is prohibitive for tasks with large vocabularies and long sequences. In Section 7.4, we've introduced a simple sub-sampling scheme and

4. For completeness' sake, we also tried to rescale the output distributions of our MLE models to decrease their confidence. We did not observe any improvement in performance; quite the opposite in fact.

	MLE				SEARNN			
Dropout	0.0	0.0	0.3	0.3	0.0	0.0	0.3	0.3
Beam size	1	10	1	10	1	10	1	10
	24.8	26.2	27.4	28.6	26.8	27.8	28.2	29.0

Table 7.5 – Comparison of beam search improvements for MLE and SEARNN models on the IWSLT 14 German to English machine translation dataset. We use the best-performing models for both algorithms. For the SEARNN model, the rescaling factor β is set to 2.4 (this value was derived through experimentation on the validation set, as reported in Table 7.4).

demonstrated that it significantly reduced the amount of computation without negatively impacting performance. In Section 7.5, this scheme has allowed us to apply SEARNN to a challenging machine translation task, sampling only one thousandth of available tokens at each step.

However, our sampling scheme seems quite arbitrary, and does not leverage model information fully. Further, it does not alleviate SEARNN's quadratic dependency on the sequence length. In order to remedy this state of affairs, we have explored two techniques introduced by Goodman et al. (2016): *focused costing* and *targeted sampling*.

Focused costing. Focused costing is a mixed roll-out policy, where a fixed number of learned steps are taken before resorting to the reference policy. The number of learned steps usually obeys a growing schedule. This method can lead to significant computational gains in the specific case where the reference policy is significantly cheaper than the learned policy (in extreme cases the reference policy is essentially free). Indeed, as the number of learned steps is fixed, the technique lifts the quadratic dependency on the sequence length. [5]

We have implemented this approach, which is fully compatible with sub-sampling. Using a basic schedule, we obtained similar results on the machine translation task to those presented in Section 7.5 at reduced computational cost (the training was $2.5\times$ faster on average). Optimizing the scheduler could potentially yield improvements, but remains open research for now.

In some sense, focused costing is similar to the approach taken by the MIXER algorithm (Ranzato et al., 2016), except that the learned steps in MIXER rollouts are positioned at the very end of the sequence. One interesting future research direction would be to adapt focused costing to put the learned steps at the end of the sequence rather than at the beginning.

5. That is, unless the schedule grows asymptotically to the sequence length.

Targeted sampling. Targeted sampling, as its name indicates, is a sub-sampling strategy. The main idea is to try to be sample-efficient. This method focuses on cells where the model predictions are uncertain or differ from the ground truth, considering that when the model is sure of itself and agrees with the ground truth there is little to gain by further exploration.

To enforce this principle, only tokens whose probability is within a fixed threshold of the best rated token are sampled (as well as the ground truth token). This means that whenever the model agrees with the ground truth and does not score any other token highly, no further exploration is performed. Computational power is deployed for cases where the model disagrees with the ground truth or rates several tokens highly.

We have implemented targeted sampling as an alternative the more naive samplings introduced in Section 7.4. We were able to obtain similar results on the machine translation task after basic tuning of the threshold hyper-parameter.

We conjecture that using a schedule for this threshold quantity rather than a fixed value could improve performance, although we have not experimented with this option.

7.8 Discussion and related work

We now contrast SEARNN to several related algorithms, including traditional L2S approaches (which are not adapted to RNN training), and RNN training methods inspired by L2S and RL.

7.8.1 Traditional L2S approaches

Although SEARNN is heavily inspired by SEARN, it is actually closer to LOLS (Chang et al., 2015), another L2S algorithm. As LOLS, SEARNN is a meta-algorithm where roll-in/roll-out strategies are customizable (we explored most combinations in our experiments). Our findings are in agreement with those of Chang et al. (2015): we advocate using the same combination, that is, a learned roll-in and a mixed roll-out. The one exception to this rule of thumb is when the associated reduced problem is too hard (as seems to be the case for machine translation), in which case we recommend switching to a reference roll-in.

Moreover, as noted in Section 7.3, SEARNN adapts the optimization process of LOLS (the one difference being that our method is stochastic rather than online): each intermediate dataset is only used for a single gradient step. This means the policy interpolation is of a different nature than in SEARN where intermediate datasets are optimized for fully and the resulting policy is mixed with the previous one.

However, despite the similarities we have just underlined, SEARNN presents significant differences from these traditional L2S algorithms. First off, and most importantly, SEARNN is a full integration of the L2S ideas to RNN training, whereas previous methods cannot be used for this purpose directly. Second, in order to achieve this adaptation we had to modify several design choices, including:

- the intermediate dataset construction, which significantly differs from traditional L2S; [6]
- the careful choice of a classifier (those used in the L2S literature do not fit RNNs well);
- the design of tailored surrogate loss functions that leverage cost information while being easy to optimize in RNNs.

7.8.2 L2S-inspired approaches

Several other papers have tried using L2S-like ideas for better RNN training, starting with Bengio et al. (2015) which introduces "scheduled sampling" to avoid the exposure bias problem. The idea is to start with teacher forcing and to gradually use more and more model predictions instead of ground truth tokens during training. This is akin to a mixed roll-in – an idea which also appears in (Daumé et al., 2009).

Wiseman and Rush (2016, BSO) adapt one of the early variants of the L2S framework: the "Learning A Search Optimization" approach of Daumé and Marcu (2005, LASO) to train RNNs. However LASO is quite different from the more modern SEARN family of algorithms that we focus on: it does not include either local classifiers or roll-outs, and has much weaker theoretical guarantees. Additionally, BSO's training loss is defined by violations in the beam-search procedure, yielding a very different algorithm from SEARNN. Furthermore, BSO requires being able to compute a meaningful loss on partial sequences, and thus does not handle general structured losses unlike SEARNN. Finally, its ad hoc surrogate objective provides very sparse sequence-level training signal, as mentioned by their authors, thus requiring warm-start.

Ballesteros et al. (2016) use a loss that is similar to LL for parsing, a specific task where cost-to-go are essentially free. This property is also a requirement for Sun et al. (2017), in which new gradient procedures are introduced to incorporate neural classifiers in the

6. The feature extraction is fully integrated in the model and thus learnable instead of being hand-crafted. Moreover, arbitrarily picking a token a during the roll-out phase to compute the associated costs requires feeding them back to the RNN (as opposed to simply adding the decision to the context before extracting features).

AGGREVATE (Ross and Bagnell, 2014) variant of L2S. [7] In contrast, SEARNN can be used on tasks without a free cost-to-go oracle.

7.8.3 RL-inspired approaches

In structured prediction tasks, we have access to ground truth trajectories, i.e. a lot more information than in traditional RL. One major direction of research has been to adapt RL techniques to leverage this additional information. The main idea is to try to optimize the expectation of the test error directly (under the stochastic policy parameterized by the RNN):

$$\mathcal{L}(\theta) = -\sum_{i=1}^{N} \mathbb{E}_{(y_1^i,..,y_T^i)\sim\pi(\theta)} r(y_1^i,..,y_T^i)\,. \tag{7.7}$$

Since we are taking an expectation over all possible structured outputs, the only term that depends on the parameters is the probability term (the tokens in the error term are fixed). This allows this loss function to support non-differentiable test errors, which is a key advantage. Of course, actually computing the expectation over an exponential number of possibilities is computationally intractable.

To circumvent this issue, Shen et al. (2016) subsample trajectories according to the learned policy, while Ranzato et al. (2016); Rennie et al. (2017) use the REINFORCE algorithm, which essentially approximates the expectation with a single trajectory sample. Instead of approximating the gradient term via sampling, Bunel et al. (2018) compute its exact value on a simplified probability distribution with reduced support (basically the k best samples returned by beam search). [8] Bahdanau et al. (2017) adapt the ACTOR-CRITIC algorithm, where a second *critic* network is trained to approximate the expectation.

While all these approaches report significant improvement on various tasks, one trait they share is that they only work when initialized from a good pre-trained model. This phenomenon is often explained by the sparsity of the information contained in "sequence-level" losses. Indeed, in the case of REINFORCE, no distinction is made between the tokens that form a sequence: depending on whether the sampled trajectory is above a global baseline, all tokens are pushed up or down by the gradient update. This means good tokens are sometimes penalized and bad tokens rewarded.

In contrast, SEARNN uses "global-local" losses, with a local loss attached to each step, which contains global information since the costs are computed on full sequences. To do so,

7. Sun et al. (2017)'s algorithm simply replaces the classifier in AGGREVATE with a neural network. As it is trained on an ever growing dataset, a natural gradient update is required to make the algorithm tractable.
8. Note that unlike in the other papers referenced in this paragraph, this estimator is biased.

we have to "sample" more trajectories through our roll-in/roll-outs. As a result, SEARNN does not require warm-starting to achieve good experimental performance. This distinction is quite relevant, because warm-starting means initializing in a specific region of parameter space which may be hard to escape. Exploration is less constrained when starting from scratch, leading to potentially larger gains over MLE.

RL-based methods often involve optimizing additional models (baselines for REINFORCE and the critic for ACTOR-CRITIC), introducing more complexity (e.g. target networks). SEARNN does not.

Finally, while maximizing the expected reward allows the RL approaches to use gradient descent even when the test error is not differentiable, it introduces another discrepancy between training and testing. Indeed, at test time, one does not decode by sampling from the stochastic policy. Instead, one selects the "best" sequence (according to a search algorithm, e.g. greedy or beam search). SEARNN avoids this averse effect by computing costs using deterministic roll-outs – the same decoding technique as the one used at test time – so that its loss is even closer to the test loss. The associated price is that we approximate the gradient by fixing the costs, although they do depend on the parameters.

7.8.4 Other methods

RAML and variants. RAML (Norouzi et al., 2016) is another RL-inspired approach. Though quite different from the previous papers we have cited, it is also related to SEARNN. Here, in order to mitigate the 0/1 aspect of MLE training, the authors introduce noise in the target outputs at each iteration. The amount of random noise is determined according to the associated reward (target outputs with a lot of noise obtain lower rewards and are thus less sampled). This idea is linked to the label smoothing technique (Szegedy et al., 2016), where the target distribution at each step is the addition of a Dirac (the usual MLE target) and a uniform distribution.

In this sense, when using the KL loss SEARNN can be viewed as doing *learned* label smoothing, where we compute the target distribution from the intermediate costs rather than arbitrarily adding the uniform distribution.

RAML has spawned several variants. Dai et al. (2018) establish a theoretical link between RAML and entropy regularized RL, and leverage this insight to propose both an improved RAML alternative as well as an improved ACTOR-CRITIC algorithm. Elbayad et al. (2018) on the other hand extend the sequence-level RAML approach to the token-level, improving performance over the initial algorithm.

Alternative losses. Yu et al. (2017) introduces an interesting approach to train generative (non-conditional) RNNs as part of a generative adversarial network (Goodfellow et al., 2014, GAN) architecture. Here a *discriminator* classifier (first pre-trained against an MLE trained generator) acts as a learned loss on full sequences. The algorithm then alternates between training the generative RNN using policy gradient updates and training the discriminator. Interestingly, in order to obtain token-level training signal for the generator RNN, the algorithm uses Monte Carlo roll-outs to compute finished sequences the discriminator can be applied to.

A final paper worth mentioning is Edunov et al. (2018), which studies various classical structured prediction and RL-inspired losses on machine translation and abstractive summarization tasks, obtaining state-of-the-art results (although their experiments are run using convolutional neural networks, they are in principle adaptable to RNNs).

7.8.5 An unexpected cousin: AlphaGo Zero.

AlphaGo Zero, introduced by Silver David et al. (2017), is a reinforcement learning algorithm aimed at playing the game of Go. It operates in purely RL setup – that is, it does not have access to expert demonstrations or policies.

The algorithm relies on training a model, which given a game state outputs a probability distribution on the possible actions, as well as a prediction of which player will win the game. At each iteration, the algorithm starts by playing a game against itself. The decisions are made according to search probabilities obtained through Monte Carlo tree search (MCTS) simulations guided by the current state of the model.

When the game is done, states and decisions are extracted. The model is then trained so as to maximize the similarity of its output distribution to the search distribution given by MCTS (and the accuracy of its game-winner predictions).

Interestingly, the game of self-play can be construed of as a roll-in trajectory. Given this insight, let us compare AlphaGo Zero to SEARNN with a learned roll-in, a learned or mixed roll-out and the KL loss. Both algorithms start with a roll-in guided by the current version of the model. At each step, both algorithms run a search procedure guided by the current policy, giving them access to a distribution over possible tokens that contains additional exploration information and is computed based on the current version of the model. Finally, both algorithms attempt to minimize the divergence of the model's output distribution and the "smarter" distribution thus obtained.

Of course, despite these interesting similarities many differences subsist. First, the search procedure is embedded in the roll-in phase in AlphaGo Zero, whereas the two are distinct for

SEARNN. Second, AlphaGo Zero uses the probabilistic MCTS as search procedure, while SEARNN relies on a deterministic approach. Third, AlphaGo Zero learns values which it uses to perform MCTS, whereas SEARNN uses roll-outs. Finally, AlphaGo Zero operates in the RL setup, where a reward function is available both at training and at test time, though no expert demonstrations or policy. Conversely, SEARNN relies on ground truth information during training (which can be used to compute a "reward" function in the RL sense), but does not have access to either ground truth or reward at test time. Consequently, while AlphaGo Zero plays games in the same fashion both at train and test time, SEARNN cannot leverage exploration at test time.

These differences provide us with interesting directions for future research. For instance, roll-outs are quite expensive in SEARNN. Replacing them with a value network as in AlphaGo Zero would mean using significantly less computational resources, hence potentially applying SEARNN to even larger scale tasks. Another idea would be to add a cost prediction module in SEARNN and to include the associated loss in the general loss, as is done in AlphaGo Zero.

Chapter 8

Conclusion and future work

Machine learning tasks are evolving towards more and more complex problems, hence the renewed interest in structured prediction, where we seek to predict several random variables as well as their dependencies. Often, structured tasks can be handled through sequential prediction, focusing on random variables one at a time.

Recurrent neural networks (RNNs), a family of neural networks specifically targeted at sequential tasks, have enjoyed great success in structured prediction applications such as machine translation (Sutskever et al., 2014), parsing (Ballesteros et al., 2016) or caption generation (Vinyals et al., 2015).

Chapter 6. These models are commonly trained using a surrogate loss derived from maximum likelihood estimation. Of course, relying on a single surrogate loss to optimize for very diverse metrics is not optimal. It can indeed lead to problematic situations, as we have detailed in Section 6.2.2.

First, as the MLE loss only seeks to maximize the probability of the ground truth, it ignores the wealth of information that structured losses can offer.

Second, on a related point, MLE has a strong 0/1 flavor, in the sense that it does not distinguish candidates based on their similarity to the ground truth, foregoing potentially valuable training signal.

Finally, MLE introduces discrepancies between the training and testing phases, notably because of exposure bias: during training the model only learns to make decisions based on the assumption that all its previous decisions were correct. In contrast, at test time the model makes decisions conditioned on its own previous decisions, which may differ from the ground truth (to which the model does not have access). Consequently the trained model can find itself in states it has never encountered during training, and is therefore more likely to fail.

The issues associated with the traditional training approach of RNNs, together with the increasing importance of sequential prediction, imply that introducing alternative means of RNN training is a relevant research direction.

Chapter 7. This field of research has received much attention lately. A popular approach is to adapt reinforcement learning algorithms to the supervised setting, in order to derive training losses that can exploit the structured metrics of interest (Ranzato et al. (2016); Bahdanau et al. (2017) are based on the REINFORCE and ACTOR-CRITIC algorithms, for instance).

In this thesis, we have instead revisited ideas from the structured prediction field, and more precisely from the "learning to search" (L2S) approach pioneered by Daumé et al. (2009). As it happens, this family of algorithms has strong links with RNNs, which we have detailed in Section 7.2.

In Section 7.3, we have described SEARNN, a novel algorithm that uses core ideas from the L2S framework in order to alleviate the known limitations of MLE training for RNNs. By leveraging structured cost information obtained through strategic exploration, we have defined global-local losses that are much closer to the test error than MLE. These losses provide a *global* feedback related to the structured task at hand, distributed *locally* within the cells of the RNN. This alternative procedure enables training RNNs from scratch and outperforms MLE on two challenging structured prediction tasks.

As this algorithm is quite costly, we have investigated scaling solutions in Sections 7.4 and 7.7. These schemes have allowed us to considerably reduce training times and thus to apply SEARNN on structured tasks for which the output vocabulary is very large, such as neural machine translation. We have shown in Section 7.5 that on this difficult task, SEARNN also significantly outperforms MLE training.

Finally, we have given some insights into how the training approach affects model behavior in Section 7.6: for instance, MLE models tend to be overconfident, while SEARNN models avoid that pitfall.

Despite the promising results we have showcased in Chapter 7, SEARNN still comes with a number of limitations, and several potential extensions seem promising.

Improving scalability. One issue we have already worked on addressing is the scalability of the algorithm, so that SEARNN may accommodate more complex tasks. Although there are many different ways of approaching this problem, two main leads are of particular interest.

The first one is to use smarter sampling strategies than the simple ones we have tried. Ideas include hierarchical sampling (Goodman, 2001) – where we first predict a class and then precise elements of the class – and tuned targeted sampling (Goodman et al., 2016) where we only sample promising actions when the model is unsure of itself.

The second lead is to devise smarter methods of cost approximation than doing roll-outs at every iteration. Here, we can adapt ideas from L2S such as focused costing (Goodman et al., 2016) – a mixed roll-out policy where a fixed number of learned steps are taken before resorting to the reference policy – which would help us lift the quadratic dependency of SEARNN on the sequence length. Adapting "bandit" L2S alternatives (Chang et al., 2015) would also reduce computational costs significantly, as well as allow us to apply SEARNN to tasks where only a single trajectory may be observed at any given point (so trying every possible token is not possible).

Alternatively, we can look to reinforcement learning for inspiration. One promising direction is to emulate the ACTOR-CRITIC architecure and train a model to approximate the costs of each token instead of computing full roll-outs. Such a model could be trained using the results of full roll-outs computed only on a fraction of all (cell, token) pairs.

Improving robustness. The second limitation of SEARNN is its robustness. We have seen for instance that for specific tasks we cannot use a learned roll-in, as this policy does not perform well enough. Here again, there are two main leads.

First, we could – as many related methods do – introduce schedules for the roll-in and/or the roll-out, so as to reduce the dependency of the algorithm on the learned policy at the beginning of training, where said policy is suboptimal. The price to pay would be an accrued dependency on the reference policy, thus limiting exploration.

Second, we could try to alleviate the problems linked to the abundance of noise in the costs – because the learned roll-out policy can be uninformative – by training a model to interpret these results, similarly to imagination-augmented agents (Racanière et al., 2017, I2As).

SEARNN as combination of imitation and reinforcement learning. As noted by Ross and Bagnell (2014), L2S can be seen as a combination of imitation (when doing reference roll-outs) and reinforcement learning (with learned roll-outs). Following this analogy, we could explore how SEARNN performs in a data-rich context where we have access to both a reward function and some expert demonstrations. The hope is to combine the best of both worlds in this setup: state-of-the-art results of advanced RL methods at the reduced sampling cost of IL.

One particularly interesting use case for this integration is neural combinatorial optimization (NCO), where leveraging decades of advances in operations research through heuristics seems natural. Another advantage of NCO is that although the metrics involved are often heavily non-linear, they usually have a strong underlying structure. As MLE does not exploit this information, it typically performs poorly with respect to these complex losses. However, SEARNN does leverage structure, and thus we conjecture that it is on this type of tasks that it can offer the biggest improvements over MLE.

Appendix A

HOGWILD analysis using the "After read" framework

A.1 Initial recursive inequality derivation

We start by proving Equation (2.11). Let $g_t := g(\hat{x}_t, \hat{\alpha}^t, i_t)$. From (2.3), we get:

$$\begin{aligned}\|x_{t+1} - x^*\|^2 &= \|x_t - \gamma g_t - x^*\|^2 \\ &= \|x_t - x^*\|^2 + \gamma^2\|g_t\|^2 - 2\gamma\langle x_t - x^*, g_t\rangle \\ &= \|x_t - x^*\|^2 + \gamma^2\|g_t\|^2 - 2\gamma\langle \hat{x}_t - x^*, g_t\rangle + 2\gamma\langle \hat{x}_t - x_t, g_t\rangle.\end{aligned}$$

In order to prove Equation (2.11), we need to bound the $-2\gamma\langle \hat{x}_t - x^*, g_t\rangle$ term. Thanks to Property 2, we can write:

$$\mathbb{E}\langle \hat{x}_t - x^*, g_t\rangle = \mathbb{E}\langle \hat{x}_t - x^*, \mathbf{E}g_t\rangle = \mathbb{E}\langle \hat{x}_t - x^*, f'(\hat{x}_t)\rangle\,.$$

We can now use a classical strong convexity bound as well as a squared triangle inequality to get:

$$\begin{aligned}-\langle \hat{x}_t - x^*, f'(\hat{x}_t)\rangle &\leq -\big(f(\hat{x}_t) - f(x^*)\big) - \frac{\mu}{2}\|\hat{x}_t - x^*\|^2 && \text{(Strong convexity bound)} \\ -\|\hat{x}_t - x^*\|^2 &\leq \|\hat{x}_t - x_t\|^2 - \frac{1}{2}\|x_t - x^*\|^2 && (\|a+b\|^2 \leq 2\|a\|^2 + 2\|b\|^2) \\ -2\gamma\mathbb{E}\langle \hat{x}_t - x^*, g_t\rangle &\leq -\frac{\gamma\mu}{2}\mathbb{E}\|x_t - x^*\|^2 + \gamma\mu\mathbb{E}\|\hat{x}_t - x_t\|^2 - 2\gamma\big(\mathbb{E}f(\hat{x}_t) - f(x^*)\big)\,.\end{aligned}$$

Putting it all together, we get the initial recursive inequality (2.11), rewritten here explicitly:

$$a_{t+1} \leq (1 - \frac{\gamma\mu}{2})a_t + \gamma^2\mathbb{E}\|g_t\|^2 + \gamma\mu\mathbb{E}\|\hat{x}_t - x_t\|^2 + 2\gamma\mathbb{E}\langle\hat{x}_t - x_t, g_t\rangle - 2\gamma e_t\,, \tag{A.1}$$

where $a_t := \mathbb{E}\|x_t - x^*\|^2$ and $e_t := \mathbb{E}f(\hat{x}_t) - f(x^*)$.

A.2 Proof of Lemma 11 (inequality in $g_t := g(\hat{x}_t, i_t)$)

To prove Lemma 11, we now bound both $\mathbb{E}\|\hat{x}_t - x_t\|^2$ and $\mathbb{E}\langle\hat{x}_t - x_t, g_t\rangle$ with respect to $(\mathbb{E}\|g_u\|^2)_{u\leq t}$.

Bounding $\mathbb{E}\langle\hat{x}_t - x_t, g_t\rangle$ in terms of g_u.

$$\begin{aligned}
\frac{1}{\gamma}\mathbb{E}\langle\hat{x}_t - x_t, g_t\rangle &= \sum_{u=(t-\tau)_+}^{t-1} \mathbb{E}\langle G_u^t g_u, g_t\rangle && \text{(by Equation (2.7))} \\
&\leq \sum_{u=(t-\tau)_+}^{t-1} \mathbb{E}|\langle g_u, g_t\rangle| && (G_u^t \text{ diagonal matrices with terms in } \{0,1\}) \\
&\leq \sum_{u=(t-\tau)_+}^{t-1} \frac{\sqrt{\Delta}}{2}(\mathbb{E}\|g_u\|^2 + \mathbb{E}\|g_t\|^2) && \text{(by Proposition 12)} \\
&\leq \frac{\sqrt{\Delta}}{2}\sum_{u=(t-\tau)_+}^{t-1} \mathbb{E}\|g_u\|^2 + \frac{\sqrt{\Delta}\tau}{2}\mathbb{E}\|g_t\|^2. && \text{(A.2)}
\end{aligned}$$

Bounding $\mathbb{E}\|\hat{x}_t - x_t\|^2$ with respect to g_u Thanks to the expansion for $\hat{x}_t - x_t$ (2.7), we get:

$$\begin{aligned}
\|\hat{x}_t - x_t\|^2 &\leq \gamma^2 \sum_{u,v=(t-\tau)_+}^{t-1} |\langle G_u^t g_u, G_v^t g_v\rangle| \\
&\leq \gamma^2 \sum_{u=(t-\tau)_+}^{t-1} \|g_u\|^2 + \gamma^2 \sum_{\substack{u,v=(t-\tau)_+ \\ u\neq v}}^{t-1} |\langle G_u^t g_u, G_v^t g_v\rangle|\,. && \text{(A.3)}
\end{aligned}$$

Using (2.14) from Proposition 12, we have that for $u \neq v$:

$$\mathbb{E}|\langle G_u^t g_u, G_v^t g_v\rangle| \leq \mathbb{E}|\langle g_u, g_v\rangle| \leq \frac{\sqrt{\Delta}}{2}(\mathbb{E}\|g_u\|^2 + \mathbb{E}\|g_v\|^2)\,. \tag{A.4}$$

By taking the expectation and using (A.4), we get:

$$\begin{aligned}\mathbb{E}\|\hat{x}_t - x_t\|^2 &\leq \gamma^2 \sum_{u=(t-\tau)_+}^{t-1} \mathbb{E}\|g_u\|^2 + \gamma^2\sqrt{\Delta}(\tau-1)_+ \sum_{u=(t-\tau)_+}^{t-1} \mathbb{E}\|g_u\|^2 \\ &= \gamma^2\big(1+\sqrt{\Delta}(\tau-1)_+\big) \sum_{u=(t-\tau)_+}^{t-1} \mathbb{E}\|g_u\|^2 \\ &\leq \gamma^2\big(1+\sqrt{\Delta}\tau\big) \sum_{u=(t-\tau)_+}^{t-1} \mathbb{E}\|g_u\|^2. \end{aligned} \tag{A.5}$$

We can now rewrite (2.11) in terms of $\mathbb{E}\|g_t\|^2$, which finishes the proof for Lemma 11 (by introducing C_1 and C_2 as specified by (2.13) in Lemma 11):

$$\begin{aligned}a_{t+1} &\leq (1-\frac{\gamma\mu}{2})a_t - 2\gamma e_t + \gamma^2\mathbb{E}\|g_t\|^2 + \gamma^3\mu(1+\sqrt{\Delta}\tau) \sum_{u=(t-\tau)_+}^{t-1} \mathbb{E}\|g_u\|^2 \\ &\quad + \gamma^2\sqrt{\Delta} \sum_{u=(t-\tau)_+}^{t-1} \mathbb{E}\|g_u\|^2 + \gamma^2\sqrt{\Delta}\tau\mathbb{E}\|g_t\|^2 \\ &\leq (1-\frac{\gamma\mu}{2})a_t - 2\gamma e_t + \gamma^2 C_1\mathbb{E}\|g_t\|^2 + \gamma^2 C_2 \sum_{u=(t-\tau)_+}^{t-1} \mathbb{E}\|g_u\|^2. \end{aligned} \tag{A.6}$$

■

A.3 Proof of Lemma 14 (suboptimality bound on $\mathbb{E}\|g_t\|^2$)

We simply introduce $f_i'(x^*)$ in g_t to derive our bound.

$$\begin{aligned}\mathbb{E}\|g_t\|^2 &= \mathbb{E}\|f_i'(\hat{x}_t)\|^2 \\ &= \mathbb{E}\|f_i'(\hat{x}_t) - f_i'(x^*) + f_i'(x^*)\|^2 \\ &\leq 2\mathbb{E}\|f_i'(\hat{x}_t) - f_i'(x^*)\|^2 + 2\mathbb{E}\|f_i'(x^*)\|^2 \qquad (\|a+b\|^2 \leq 2\|a\|^2 + 2\|b\|^2) \\ &\leq 4Le_t + 2\sigma^2\,. \qquad \text{(Hofmann et al. (2015), Eq (7) \& (8))}\end{aligned}$$

■

A.4 Proof of Theorem 9 (convergence guarantee and rate of HOGWILD)

Master inequality derivation. We plug Lemma 14 into Lemma 11 which gives us:

$$a_{t+1} \leq (1 - \frac{\gamma\mu}{2})a_t + \gamma^2 C_1(4Le_t + 2\sigma^2) + \gamma^2 C_2 \sum_{u=(t-\tau)_+}^{t-1} (4Le_u + 2\sigma^2) - 2\gamma e_t \,. \quad \text{(A.7)}$$

By grouping the e_t and the σ^2 terms we get our master inequality (2.22):

$$a_{t+1} \leq (1 - \frac{\gamma\mu}{2})a_t + (4L\gamma^2 C_1 - 2\gamma)e_t + 4L\gamma^2 C_2 \sum_{u=(t-\tau)_+}^{t-1} e_u + 2\gamma^2\sigma^2(C_1 + \tau C_2) \,.$$

Contraction inequality derivation (x_t). We now unroll (2.22) all the way to $t = 0$ to get:

$$\begin{aligned} a_{t+1} \leq (1 - \frac{\gamma\mu}{2})^{t+1} a_0 &+ \sum_{u=0}^{t} (1 - \frac{\gamma\mu}{2})^{t-u} (4L\gamma^2 C_1 - 2\gamma) e_u \\ &+ \sum_{u=0}^{t} (1 - \frac{\gamma\mu}{2})^{t-u} 4L\gamma^2 C_2 \sum_{v=(u-\tau)_+}^{u-1} e_v \\ &+ \sum_{u=0}^{t} (1 - \frac{\gamma\mu}{2})^{t-u} 2\gamma^2\sigma^2(C_1 + \tau C_2) \,. \end{aligned} \quad \text{(A.8)}$$

Now we can simplify these terms as follows:

$$\begin{aligned} \sum_{u=0}^{t} (1 - \frac{\gamma\mu}{2})^{t-u} \sum_{v=(u-\tau)_+}^{u-1} e_v &= \sum_{v=0}^{t-1} \sum_{u=v+1}^{\min(t, v+\tau)} (1 - \frac{\gamma\mu}{2})^{t-u} e_v \\ &= \sum_{v=0}^{t-1} (1 - \frac{\gamma\mu}{2})^{t-v} e_v \sum_{u=v+1}^{\min(t, v+\tau)} (1 - \frac{\gamma\mu}{2})^{v-u} \\ &\leq \sum_{v=0}^{t-1} (1 - \frac{\gamma\mu}{2})^{t-v} e_v \tau (1 - \frac{\gamma\mu}{2})^{-\tau} \\ &\leq \tau (1 - \frac{\gamma\mu}{2})^{-\tau} \sum_{v=0}^{t} (1 - \frac{\gamma\mu}{2})^{t-v} e_v \,. \end{aligned} \quad \text{(A.9)}$$

This $(1-\frac{\gamma\mu}{2})^{-\tau}$ term is easily bounded. Using Bernoulli's inequality (B.55), we get that if we assume $\tau \leq \frac{1}{\gamma\mu}$: [1]

$$(1-\frac{\gamma\mu}{2})^{-\tau} \leq 2\,. \tag{A.10}$$

We note that the last term in (A.8) is a geometric sum:

$$\sum_{u=0}^{t}(1-\frac{\gamma\mu}{2})^{t-u}\sigma^2 = \frac{2}{\gamma\mu}\sigma^2\,. \tag{A.11}$$

We plug (A.9)–(A.11) in (A.8) to obtain (2.23):

$$a_{t+1} \leq (1-\frac{\gamma\mu}{2})^{t+1}a_0 + (4L\gamma^2C_1 + 8L\gamma^2\tau C_2 - 2\gamma)\sum_{u=0}^{t}(1-\frac{\gamma\mu}{2})^{t-u}e_u + \frac{4\gamma\sigma^2}{\mu}(C_1+\tau C_2)\,.$$

Contraction inequality derivation ($\hat{x}_t$). We now have an contraction inequality for the convergence of x_t to x^*. However, since this quantity does not exist (except if we fix the number of iterations prior to running the algorithm and then wait for all iterations to be finished – an unwieldy solution), we rather want to prove that $\hat{x}_t$ converges to x^*. In order to do this, we use the simple following inequality:

$$\|\hat{x}_t - x^*\|^2 \leq 2a_t + 2\|\hat{x}_t - x_t\|^2\,. \tag{A.12}$$

We already have a contraction bound on the first term (2.23). For the second term, we combine (A.5) with Lemma 14 to get:

$$\mathbb{E}\|\hat{x}_t - x_t\|^2 \leq 4L\gamma^2C_1\sum_{u=(t-\tau)_+}^{t-1} e_u + 2\gamma^2\tau\sigma^2\,. \tag{A.13}$$

To make it easier to combine with (2.23), we rewrite (A.13) as:

$$\begin{aligned}
\mathbb{E}\|\hat{x}_t - x_t\|^2 &\leq 4L\gamma^2C_1(1-\frac{\gamma\mu}{2})^{-\tau}\sum_{u=(t-\tau)_+}^{t-1}(1-\frac{\gamma\mu}{2})^{t-1-u}e_u + 2\gamma^2\tau\sigma^2 \\
&\leq 8L\gamma^2C_1\sum_{u=(t-\tau)_+}^{t-1}(1-\frac{\gamma\mu}{2})^{t-1-u}e_u + 2\gamma^2\tau\sigma^2 \\
&\leq 8L\gamma^2C_1\sum_{u=0}^{t-1}(1-\frac{\gamma\mu}{2})^{t-1-u}e_u + 2\gamma^2\tau\sigma^2\,.
\end{aligned} \tag{A.14}$$

1. While this assumption on τ may appear restrictive, it is in fact weaker than the condition for a linear speed-up obtained by our analysis in Corollary 10.

Combining (2.23) and (A.14) gives us (2.24):

$$\mathbb{E}\|\hat{x}_t - x^*\|^2 \leq (1 - \frac{\gamma\mu}{2})^{t+1} 2a_0 + (\frac{8\gamma(C_1 + \tau C_2)}{\mu} + 4\gamma^2 C_1 \tau)\sigma^2 \\ + (24L\gamma^2 C_1 + 16L\gamma^2\tau C_2 - 4\gamma) \sum_{u=0}^{t} (1 - \frac{\gamma\mu}{2})^{t-u} e_u \,.$$

Maximum step size condition on γ. To prove Theorem 9, we need an inequality of the following type: $\mathbb{E}\|\hat{x}_t - x_t\|^2 \leq (1-\rho)^t a + b$. To give this form to Equation (2.24), we need to remove all the $(e_u, u < t)$ terms from its right-hand side. To safely do so, we need to enforce that all these terms are negative, hence that:

$$24L\gamma^2 C_1 + 16L\gamma^2\tau C_2 - 4\gamma \leq 0 \,. \tag{A.15}$$

Plugging the values of C_1 and C_2 we get:

$$4L\mu\tau(1 + \sqrt{\Delta}\tau)\gamma^2 + 6L(1 + 2\sqrt{\Delta}\tau)\gamma - 1 \leq 0 \,. \tag{A.16}$$

This reduces to a second-order polynomial sign condition. We remark that since $\gamma \geq 0$, we can upper bound our terms in γ and γ^2 in this polynomial, which will still give us sufficient conditions for convergence. This means if we define $C_3 := 1 + 2\sqrt{\Delta}\tau$, a sufficient condition is:

$$4L\mu\tau C_3\gamma^2 + 6LC_3\gamma - 1 \leq 0 \,. \tag{A.17}$$

The discriminant of this polynomial is always positive, so γ needs to be between its two roots. The smallest is negative, so the condition is not relevant to our case (where $\gamma > 0$). By solving analytically for the positive root ϕ, we get an upper bound condition on γ that can be used for any overlap τ and guarantee convergence. This positive root is:

$$\phi = \frac{3}{4} \frac{\sqrt{1 + \frac{\mu\tau}{2LC_3}} - 1)}{LC_3} \,. \tag{A.18}$$

We simplify it further by using the inequality: [2]

$$\sqrt{x} - 1 \geq \frac{x-1}{2\sqrt{x}} \qquad \forall x > 0 \,. \tag{A.19}$$

2. This inequality can be derived by using the concavity property $f(y) \leq f(x) + (y-x)f'(x)$ on the differentiable concave function $f(x) = \sqrt{x}$ with $y = 1$.

We get:

$$\phi \geq \frac{3}{16LC_3\sqrt{1+\frac{\tau}{2\kappa C_3}}}\,. \tag{A.20}$$

This finishes the proof for Theorem 9. ■

A.5 Proof of Theorem 8 (convergence result for serial SGD)

In order to analyze Corollary 10, we need to derive the maximum allowable step size for serial SGD. Note that SGD verifies a simpler contraction inequality than Lemma 11. For all $t \geq 0$:

$$a_{t+1} \leq (1-\gamma\mu)a_t + \gamma^2\mathbb{E}\|g_t\|^2 - 2\gamma e_t\,, \tag{A.21}$$

Here, the contraction factor is $(1-\gamma\mu)$ instead of $(1-\frac{\gamma\mu}{2})$ because $\hat{x}_t = x_t$ so there is no need for a triangle inequality to get back $\|x_t - x^*\|^2$ from $\|\hat{x}_t - x^*\|^2$ after we apply our strong convexity bound in our initial recursive inequality (see Section A.1). Lemma 14 also holds for serial SGD. By plugging it into (A.21), we get:

$$a_{t+1} \leq (1-\gamma\mu)a_t + (4L\gamma^2 - 2\gamma)e_t + 2\gamma^2\sigma^2\,. \tag{A.22}$$

We then unroll (A.22) until $t = 0$ to get:

$$a_{t+1} \leq (1-\gamma\mu)^{t+1}a_0 + (4L\gamma^2 - 2\gamma)\sum_{u=0}^{t}(1-\gamma\mu)^{t-u}e_u + 2\frac{\gamma\sigma^2}{\mu}\,. \tag{A.23}$$

To get linear convergence up to a ball around the optimum, we need to remove the $(e_u)_{0\leq u\leq t}$ terms from the right-hand side of the equation. To safely do this, we need these terms to be negative, i.e. $4L\gamma^2 - 2\gamma \leq 0$. We can then trivially derive the condition on γ to achieve linear convergence: $\gamma \leq \frac{1}{2L}$.

We see that if $\gamma = {}^{a}\!/_{L}$ with $a \leq {}^{1}\!/_{2}$, SGD converges at a geometric rate of at least: $\rho(a) = {}^{a}\!/_{\kappa}$, up to a ball of radius $2\frac{\gamma\sigma^2}{\mu}$ around the optimum. Now, to make sure we reach ϵ-accuracy, we need $\frac{2\gamma\sigma^2}{\mu} \leq \epsilon$, i.e. $\gamma \leq \frac{\epsilon\mu}{2\sigma^2}$. All told, in order to get linear convergence to ϵ-accuracy, serial SGD requires $\gamma \leq \min\left\{\frac{1}{2L}, \frac{\epsilon\mu}{2\sigma^2}\right\}$.

A.6 Proof of Corollary 10 (speedup regimes for HOGWILD)

The convergence rate of both SGD and HOGWILD is directly proportional to the step size. Thus, in order to make sure HOGWILD is linearly faster than SGD for any reasonable step

size, we need to show that the maximum allowable step size ensuring linear convergence for HOGWILD – given in Theorem 9 – is of the same order as the one for SGD, $\mathcal{O}(1/L)$. Recalling that $\gamma = \frac{a}{L}$, we get the following sufficient condition: $a^*(\tau) = \mathcal{O}(1)$.

Given (2.8), the definition of $a^*(\tau)$, we require both:

$$\tau\sqrt{\Delta} = \mathcal{O}(1)\,; \qquad \sqrt{1 + \frac{1}{2\kappa}\min\{\frac{1}{\sqrt{\Delta}}, \tau\}} = \mathcal{O}(1)\,. \tag{A.24}$$

This gives us the final condition on τ for a linear speedup: $\tau = \mathcal{O}(\min\{\frac{1}{\sqrt{\Delta}}, \kappa\})$.

To finish the proof of Corollary 10, we only have to show that under this condition, the size of the ball is of the same order regardless of the algorithm used.

Using $\gamma\mu\tau \leq 1$ and $\tau \leq \frac{1}{\sqrt{\Delta}}$, we get that $(\frac{8\gamma(C_1+\tau C_2)}{\mu} + 4\gamma^2 C_1\tau)\sigma^2 = \mathcal{O}(\frac{\gamma\sigma^2}{\mu})$, which finishes the proof of Corollary 10. Note that these two conditions are weaker than $\tau = \mathcal{O}(\min\{\frac{1}{\sqrt{\Delta}}, \kappa\})$, which allows us to get better bounds in case we want to reach ϵ-accuracy with $\frac{\epsilon\mu}{\sigma^2} \ll \frac{1}{2L}$. ■

Appendix B

Asynchronous variance reduction

Outline:

- In Appendix B.1.1, we adapt the proof from Hofmann et al. (2015) to prove Theorem 15, our convergence result for serial Sparse SAGA.
- In Appendix B.2, we give the complete details for the proof of convergence for ASAGA (Theorem 18) as well as its linear speedup regimes (Corollary 19).
- In Appendix B.3, we give the full details for the proof of convergence for KROMAGNON (Theorem 22) as well as a simpler convergence result for both SVRG (Corollary 23) and KROMAGNON (Corollary 24) and finally the latter's linear speedup regimes (Corollary 25)
- In Appendix B.4, we explain why adapting the lagged updates implementation of SAGA to the asynchronous setting is difficult.
- In Appendix B.5, we give additional details about the datasets and our implementation.

B.1 Sparse SAGA

B.1.1 Proof of Theorem 15

Proof sketch for Hofmann et al. (2015). As we will heavily reuse the proof technique from Hofmann et al. (2015), we start by giving its sketch.

First, the authors combine classical strong convexity and Lipschitz inequalities to derive the following inequality (Hofmann et al., 2015, Lemma 1):

$$\mathbf{E}\|x^+ - x^*\|^2 \leq (1-\gamma\mu)\|x-x^*\|^2 + 2\gamma^2\mathbf{E}\|\alpha_i - f_i'(x^*)\|^2 + (4\gamma^2 L - 2\gamma)\big(f(x) - f(x^*)\big). \tag{B.1}$$

This gives a contraction term, as well as two additional terms; $2\gamma^2\mathbf{E}\|\alpha_i - f_i'(x^*)\|^2$ is a positive variance term, but $(4\gamma^2 L - 2\gamma)\big(f(x) - f(x^*)\big)$ is a negative suboptimality term (provided γ is small enough). The suboptimality term can then be used to cancel the variance one.

Second, the authors use a classical smoothness upper bound to control the variance term and relate it to the suboptimality. However, since the α_i are partial gradients computed at previous time steps, the upper bounds of the variance involve suboptimality at previous time steps, which are not directly relatable to the current suboptimality.

Third, to circumvent this issue, a Lyapunov function is defined to encompass both current and past terms. To finish the proof, Hofmann et al. (2015) show that the Lyapunov function is a contraction.

Proof outline. Fortunately, we can reuse most of the proof from Hofmann et al. (2015) to show that Sparse SAGA converges at the same rate as regular SAGA. In fact, once we establish that Hofmann et al. (2015, Lemma 1) is still verified we are done.

To prove this, we show that the gradient estimator is unbiased, and then derive close variants of equations (6) and (9) in their paper, which we remind the reader of here:

$$\mathbf{E}\|f_i'(x) - \bar{\alpha}_i\|^2 \leq 2\mathbf{E}\|f_i'(x) - f_i'(x^*)\|^2 + 2\mathbf{E}\|\bar{\alpha}_i - f_i'(x^*)\|^2$$

Hofmann et al. (2015, Eq.(6))

$$\mathbf{E}\|\bar{\alpha}_i - f_i'(x^*)\|^2 \leq \mathbf{E}\|\alpha_i - f_i'(x^*)\|^2 \,. \qquad \text{Hofmann et al. (2015, Eq.(9))}$$

Unbiased gradient estimator. We first show that the update estimator is unbiased. The estimator is unbiased if:

$$\mathbf{E}D_i\bar{\alpha} = \mathbf{E}\alpha_i = \frac{1}{n}\sum_{i=1}^{n}\alpha_i \,. \tag{B.2}$$

We have:

$$\mathbf{E}D_i\bar{\alpha} = \frac{1}{n}\sum_{i=1}^{n} D_i\bar{\alpha} = \frac{1}{n}\sum_{i=1}^{n} P_{S_i}D\bar{\alpha} = \frac{1}{n}\sum_{i=1}^{n}\sum_{v\in S_i}\frac{[\bar{\alpha}]_v e_v}{p_v} = \sum_{v=1}^{d}\left(\sum_{i\,|\,v\in S_i} 1\right)\frac{[\bar{\alpha}]_v e_v}{np_v} \,,$$

where e_v is the vector whose only nonzero component is the v component which is equal to 1.

By definition, $\sum_{i|v\in S_i} 1 = np_v$, which gives us Equation (B.2).

Deriving Hofmann et al. (2015, Equation (6)). We define $\bar{\alpha}_i := \alpha_i - D_i\bar{\alpha}$ (contrary to Hofmann et al. (2015) where the authors define $\bar{\alpha}_i := \alpha_i - \bar{\alpha}$ since they do not concern themselves with sparsity). Using the inequality $\|a+b\|^2 \leq 2\|a\|^2 + 2\|b\|^2$, we get:

$$\mathbf{E}\|f_i'(x) - \bar{\alpha}_i\|^2 \leq 2\mathbf{E}\|f_i'(x) - f_i'(x^*)\|^2 + 2\mathbf{E}\|\bar{\alpha}_i - f_i'(x^*)\|^2, \tag{B.3}$$

which is our equivalent to Hofmann et al. (2015, Eq.(6)), where only our definition of $\bar{\alpha}_i$ differs.

Deriving Hofmann et al. (2015, Equation (9)). We want to prove Hofmann et al. (2015, Eq.(9)):

$$\mathbf{E}\|\bar{\alpha}_i - f_i'(x^*)\|^2 \leq \mathbf{E}\|\alpha_i - f_i'(x^*)\|^2. \tag{B.4}$$

We have:

$$\mathbf{E}\|\bar{\alpha}_i - f_i'(x^*)\|^2 = \mathbf{E}\|\alpha_i - f_i'(x^*)\|^2 - 2\mathbf{E}\langle \alpha_i - f_i'(x^*), D_i\bar{\alpha}\rangle + \mathbf{E}\|D_i\bar{\alpha}\|^2. \tag{B.5}$$

Let $D_{\neg i} := P_{S_i^c} D$; we then have the orthogonal decomposition $D\alpha = D_i\alpha + D_{\neg i}\alpha$ with $D_i\alpha \perp D_{\neg i}\alpha$, as they have disjoint support. We now use the orthogonality of $D_{\neg i}\alpha$ with any vector with support in S_i to simplify the expression (B.5) as follows:

$$\begin{aligned}
\mathbf{E}\langle \alpha_i - f_i'(x^*), D_i\bar{\alpha}\rangle &= \mathbf{E}\langle \alpha_i - f_i'(x^*), D_i\bar{\alpha} + D_{\neg i}\bar{\alpha}\rangle && (\alpha_i - f_i'(x^*) \perp D_{\neg i}\bar{\alpha}) \\
&= \mathbf{E}\langle \alpha_i - f_i'(x^*), D\bar{\alpha}\rangle \\
&= \langle \mathbf{E}\big(\alpha_i - f_i'(x^*)\big), D\bar{\alpha}\rangle \\
&= \langle \mathbf{E}\alpha_i, D\bar{\alpha}\rangle && (f'(x^*) = 0) \\
&= \bar{\alpha}^\intercal D\bar{\alpha}\,. && \text{(B.6)}
\end{aligned}$$

Similarly,

$$\begin{aligned}
\mathbf{E}\|D_i\bar{\alpha}\|^2 &= \mathbf{E}\langle D_i\bar{\alpha}, D_i\bar{\alpha}\rangle \\
&= \mathbf{E}\langle D_i\bar{\alpha}, D\bar{\alpha}\rangle && (D_i\bar{\alpha} \perp D_{\neg i}\bar{\alpha}) \\
&= \langle \mathbf{E}D_i\bar{\alpha}, D\bar{\alpha}\rangle \\
&= \bar{\alpha}^\intercal D\bar{\alpha}\,. && \text{(B.7)}
\end{aligned}$$

Putting it all together,

$$\mathbf{E}\|\bar{\alpha}_i - f_i'(x^*)\|^2 = \mathbf{E}\|\alpha_i - f_i'(x^*)\|^2 - \bar{\alpha}^\intercal D\bar{\alpha} \leq \mathbf{E}\|\alpha_i - f_i'(x^*)\|^2. \tag{B.8}$$

This is our version of Hofmann et al. (2015, Equation (9)), which finishes the proof of Hofmann et al. (2015, Lemma 1). The rest of the proof from Hofmann et al. (2015) can then be reused without modification to obtain Theorem 15. ■

B.1.2 Proof of Theorem 16

Proof outline Again, once we have proven an equivalent to Hofmann et al. (2015, Lemma 1) our proof is finished (since we have ensured that each α_j has the same probability to be updated in the gradient table at every step, we can then derive Hofmann et al. (2015, Lemma 2) and the convergence theorem). To do so, we need to adapt three equations this time: Hofmann et al. (2015, Equation (6)), Hofmann et al. (2015, Equation (8)) and Hofmann et al. (2015, Equation (9))

Unbiased gradient estimator. Again, we start by proving that the gradient estimator is unbiased.

$$\begin{aligned}\mathbf{E}\Big[\frac{1}{nq_i}\big(f_i'(x) - \alpha_i\big) + \tilde{D}_i\bar{\alpha}\Big] &= \frac{1}{n}\sum_{i=1}^{n}\big(f_i'(x) - \alpha_i\big) + \mathbf{E}\tilde{D}_i\bar{\alpha} \\ &= f'(x) - \bar{\alpha} + \mathbf{E}\tilde{D}_i\bar{\alpha}. \end{aligned}\tag{B.9}$$

All that remains to be proven is that:

$$\mathbf{E}\tilde{D}_i\bar{\alpha} = \bar{\alpha}. \tag{B.10}$$

We have:

$$\mathbf{E}\tilde{D}_i\bar{\alpha} = \sum_{i=1}^{n} q_i\tilde{D}_i\bar{\alpha} = \sum_{i=1}^{n} q_i P_{S_i}\tilde{D}\bar{\alpha} = \sum_{i=1}^{n} q_i \sum_{v\in S_i}\frac{[\bar{\alpha}]_v e_v}{\tilde{p}_v} = \sum_{v=1}^{d}\left(\sum_{i\,|\,v\in S_i} q_i\right)\frac{[\bar{\alpha}]_v e_v}{\tilde{p}_v}.$$

By definition, $\tilde{p}_v = \sum_{i|v\in S_i} q_i$, which gives us Equation (B.10), and so the gradient estimator is unbiased.

Deriving Hofmann et al. (2015, Equation (6)). As for Equation (B.3), we use the inequality $\|a+b\|^2 \leq 2\|a\|^2 + 2\|b\|^2$ to get:

$$\mathbf{E}\|\frac{f_i'(x)-\alpha_i}{nq_i}+\tilde{D}_i\bar{\alpha}\|^2 \leq 2\mathbf{E}\|\frac{f_i'(x)-f_i'(x^*)}{nq_i}\|^2+2\mathbf{E}\|\frac{\alpha_i-f_i'(x^*)}{nq_i}-\tilde{D}_i\bar{\alpha}\|^2, \quad \text{(B.11)}$$

which is our equivalent to Hofmann et al. (2015, Equation (6)).

Deriving Hofmann et al. (2015, Equation (8)). We now need to express $\mathbf{E}\|\frac{f_i'(x)-f_i'(x^*)}{nq_i}\|^2$ in terms of the suboptimality $f(x)-f(x^*)$. We recall Hofmann et al. (2015, Equation (7)):

$$\|f_i'(x)-f_i'(x^*)\|^2 \leq 2L_i\big(f_i(x)-f_i(x^*)-\langle x-x^*, f_i'(x)^*\rangle\big). \quad \text{(B.12)}$$

We can now divide by $(nq_i)^2$ and then take the expectation with respect to i.

$$\begin{aligned}
\mathbf{E}\|\frac{f_i'(x)-f_i'(x^*)}{nq_i}\|^2 &\leq 2\sum_{i=1}^n q_i\frac{L_i}{(nq_i)^2}\big(f_i(x)-f_i(x^*)-\langle x-x^*, f_i'(x^*)\rangle\big)\\
&\leq \frac{2}{n}\sum_{i=1}^n \frac{L_i\sum_{j=1}^n L_j}{nL_i}\big(f_i(x)-f_i(x^*)-\langle x-x^*, f_i'(x^*)\rangle\big)\\
&\leq \frac{2}{n}\sum_{i=1}^n \frac{n\bar{L}}{n}\big(f_i(x)-f_i(x^*)-\langle x-x^*, f_i'(x^*)\rangle\big)\\
&\leq 2\bar{L}\Big[\frac{1}{n}\sum_{i=1}^n\big(f_i(x)-f_i(x^*)\big)-\langle x-x^*, \frac{1}{n}\sum_{i=1}^n f_i'(x^*)\rangle\Big]\\
&\leq 2\bar{L}\big(f(x)-f(x^*)\big). \quad \text{(B.13)}
\end{aligned}$$

Note that this inequality is almost the same as in the uniform case. The only difference is that it involves $\bar{L}$ instead of L in the uniform sampling case. This is where the improved step size dependency comes from.

Deriving Hofmann et al. (2015, Equation (9)). We want to prove that:

$$\mathbf{E}\|\frac{\alpha_i-f_i'(x^*)}{nq_i}-\tilde{D}_i\bar{\alpha}\|^2 \leq \mathbf{E}\|\frac{\alpha_i-f_i'(x^*)}{nq_i}\|^2. \quad \text{(B.14)}$$

We have:

$$\begin{aligned}
\mathbf{E}\|\frac{\alpha_i-f_i'(x^*)}{nq_i}-\tilde{D}_i\bar{\alpha}\|^2 = \mathbf{E}\|\frac{\alpha_i-f_i'(x^*)}{nq_i}\|^2 - 2\mathbf{E}\langle\frac{\alpha_i-f_i'(x^*)}{nq_i}, \tilde{D}_i\bar{\alpha}\rangle\\
+\mathbf{E}\|\tilde{D}_i\bar{\alpha}\|^2. \quad \text{(B.15)}
\end{aligned}$$

Using the same orthogonality argument as in the uniform algorithm, we have:

$$\begin{aligned}
\mathbf{E}\langle \frac{\alpha_i - f_i'(x^*)}{nq_i}, \tilde{D}_i\bar{\alpha}\rangle &= \mathbf{E}\langle \frac{\alpha_i - f_i'(x^*)}{nq_i}, \tilde{D}\bar{\alpha}\rangle && (\alpha_i - f_i'(x^*) \perp \tilde{D}_{\neg i}\bar{\alpha}) \\
&= \langle \mathbf{E}\frac{\alpha_i - f_i'(x^*)}{nq_i}, \tilde{D}\bar{\alpha}\rangle \\
&= \Big\langle \sum_{i=1}^{n} \frac{q_i}{nq_i}\big(\alpha_i - f_i'(x^*)\big), \tilde{D}\bar{\alpha}\Big\rangle \\
&= \Big\langle \frac{1}{n}\sum_{i=1}^{n} \alpha_i, \tilde{D}\bar{\alpha}\Big\rangle && (f'(x^*) = 0) \\
&= \bar{\alpha}^\intercal \tilde{D}\bar{\alpha}\,. && \text{(B.16)}
\end{aligned}$$

Similarly,

$$\begin{aligned}
\mathbf{E}\|\tilde{D}_i\,\bar{\alpha}\|^2 &= \mathbf{E}\langle \tilde{D}_i\bar{\alpha}, \tilde{D}_i\bar{\alpha}\rangle \\
&= \mathbf{E}\langle \tilde{D}_i\bar{\alpha}, \tilde{D}\bar{\alpha}\rangle && (\tilde{D}_i\bar{\alpha} \perp \tilde{D}_{\neg i}\bar{\alpha}) \\
&= \langle \mathbf{E}\tilde{D}_i\bar{\alpha}, \tilde{D}\bar{\alpha}\rangle \\
&= \bar{\alpha}^\intercal \tilde{D}\bar{\alpha}\,. && \text{(B.17)}
\end{aligned}$$

We thus obtain our version of Hofmann et al. (2015, Equation (9)).

$$\mathbf{E}\|\frac{\alpha_i - f_i'(x^*)}{nq_i} - \tilde{D}_i\bar{\alpha}\|^2 = \mathbf{E}\|\frac{\alpha_i - f_i'(x^*)}{nq_i}\|^2 - \bar{\alpha}^\intercal\tilde{D}\bar{\alpha} \leq \mathbf{E}\|\frac{\alpha_i - f_i'(x^*)}{nq_i}\|^2. \qquad \text{(B.18)}$$

This allows us to finish the proof of our equivalent of Hofmann et al. (2015, Lemma 1):

$$\begin{aligned}
\mathbf{E}\|x^+ - x^*\|^2 \leq (1-\gamma\mu)\|x - x^*\|^2 + 2\gamma^2\mathbf{E}\|\frac{\alpha_i - f_i'(x^*)}{nq_i}\|^2 \\
- (2\gamma - 4\gamma^2\bar{L})\big(f(x) - f(x^*)\big). \qquad \text{(B.19)}
\end{aligned}$$

As the sampling for updating the historical gradients is uniform, the rest of the proof from Hofmann et al. (2015) can then be reused without modification to obtain Theorem 16.
■

B.2 ASAGA – Proof of Theorem 18 and Corollary 19

B.2.1 Proof of Lemma 20 (suboptimality bound on $\mathbb{E}\|g_t\|^2$)

As was the case for proving Lemma 14, we now have to derive a bound on g_t with respect to suboptimality. From Appendix B.1.1, we know that:

$$\mathbb{E}\|g_t\|^2 \leq 2\mathbb{E}\|f'_{i_t}(\hat{x}_t) - f'_{i_t}(x^*)\|^2 + 2\mathbb{E}\|\hat{\alpha}^t_{i_t} - f'_{i_t}(x^*)\|^2 \tag{B.20}$$

$$\mathbb{E}\|f'_{i_t}(\hat{x}_t) - f'_{i_t}(x^*)\|^2 \leq 2L\big(\mathbb{E}f(\hat{x}_t) - f(x^*)\big) = 2Le_t\,. \tag{B.21}$$

N. B.: In the following, i_t is a random variable picked uniformly at random in $\{1, ..., n\}$, whereas i is a fixed constant.

We still have to handle the $\mathbb{E}\|\hat{\alpha}^t_{i_t} - f'_{i_t}(x^*)\|^2$ term and express it in terms of past suboptimalities. We know from our definition of t that i_t and $\hat{x}_u$ are independent $\forall u < t$. Given the "after read" global ordering, $\mathbf{E}$ – the expectation on i_t conditioned on $\hat{x}_t$ and all "past" $\hat{x}_u$ and i_u – is well defined, and we can rewrite our quantity as:

$$\begin{aligned}\mathbb{E}\|\hat{\alpha}^t_{i_t} - f'_{i_t}(x^*)\|^2 = \mathbb{E}\big(\mathbf{E}\|\hat{\alpha}^t_{i_t} - f'_{i_t}(x^*)\|^2\big) &= \mathbb{E}\frac{1}{n}\sum_{i=1}^n \|\hat{\alpha}^t_i - f'_i(x^*)\|^2 \\ &= \frac{1}{n}\sum_{i=1}^n \mathbb{E}\|\hat{\alpha}^t_i - f'_i(x^*)\|^2. \end{aligned} \tag{B.22}$$

Now, with i fixed, let $u^t_{i,l}$ be the time of the iterate last used to write the $[\hat{\alpha}^t_i]_l$ quantity, i.e. $[\hat{\alpha}^t_i]_l = [f'_i(\hat{x}_{u^t_{i,l}})]_l$. We know [1] that $0 \leq u^t_{i,l} \leq t-1$. To use this information, we first need to split $\hat{\alpha}_i$ along its dimensions to handle the possible inconsistencies among them:

$$\mathbb{E}\|\hat{\alpha}^t_i - f'_i(x^*)\|^2 = \mathbb{E}\sum_{l=1}^d \big([\hat{\alpha}^t_i]_l - [f'_i(x^*)]_l\big)^2 = \sum_{l=1}^d \mathbb{E}\Big[\big([\hat{\alpha}^t_i]_l - [f'_i(x^*)]_l\big)^2\Big]. \tag{B.23}$$

1. In the case where $u = 0$, one would have to replace the partial gradient with α^0_i. We omit this special case here for clarity of exposition.

This gives us:

$$\begin{aligned}\mathbb{E}\|\hat{\alpha}_i^t - f_i'(x^*)\|^2 &= \sum_{l=1}^{d} \mathbb{E}\Big[\big(f_i'(\hat{x}_{u_{i,l}^t})_l - f_i'(x^*)_l\big)^2\Big] \\ &= \sum_{l=1}^{d} \mathbb{E}\Big[\sum_{u=0}^{t-1} \mathbb{1}_{\{u_{i,l}^t = u\}}\big(f_i'(\hat{x}_u)_l - f_i'(x^*)_l\big)^2\Big] \\ &= \sum_{u=0}^{t-1}\sum_{l=1}^{d} \mathbb{E}\Big[\mathbb{1}_{\{u_{i,l}^t = u\}}\big(f_i'(\hat{x}_u)_l - f_i'(x^*)_l\big)^2\Big]. \end{aligned} \tag{B.24}$$

We will now rewrite the indicator so as to obtain independent events from the rest of the equality. This will enable us to distribute the expectation. Suppose $u > 0$ ($u = 0$ is a special case which we will handle afterwards). $\{u_{i,l}^t = u\}$ requires two things:

1. at time u, i was picked uniformly at random,
2. (roughly) i was not picked again between u and t.

We need to refine both conditions because we have to account for possible collisions due to asynchrony. We know from our definition of τ that the t^{th} iteration finishes before at $t+\tau+1$, but it may still be unfinished by time $t+\tau$. This means that we can only be sure that an update selecting i at time v has been written to memory at time t if $v \leq t-\tau-1$. Later updates may not have been written yet at time t. Similarly, updates before $v = u+\tau+1$ may be overwritten by the u^{th} update so we cannot infer that they did not select i. From this discussion, we conclude that $u_{i,l}^t = u$ implies that $i_v \neq i$ for all v between $u+\tau+1$ and $t-\tau-1$, though it can still happen that $i_v = i$ for v outside this range.

Using the fact that i_u and i_v are independent for $v \neq u$, we can thus upper bound the indicator function appearing in (B.24) as follows:

$$\mathbb{1}_{\{u_{i,l}^t = u\}} \leq \mathbb{1}_{\{i_u = i\}}\mathbb{1}_{\{i_v \neq i \ \forall v \text{ s.t. } u+\tau+1 \leq v \leq t-\tau-1\}}. \tag{B.25}$$

This gives us:

$$\begin{aligned}&\mathbb{E}\Big[\mathbb{1}_{\{u_{i,l}^t = u\}}\big(f_i'(\hat{x}_u)_l - f_i'(x^*)_l\big)^2\Big] \\ &\quad\leq \mathbb{E}\Big[\mathbb{1}_{\{i_u = i\}}\mathbb{1}_{\{i_v \neq i \ \forall v \text{ s.t. } u+\tau+1 \leq v \leq t-\tau-1\}}\big(f_i'(\hat{x}_u)_l - f_i'(x^*)_l\big)^2\Big] \\ &\quad\leq P\{i_u = i\}P\{i_v \neq i \ \forall v \text{ s.t. } u+\tau+1 \leq v \leq t-\tau-1\}\mathbb{E}\big(f_i'(\hat{x}_u)_l - f_i'(x^*)_l\big)^2 \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (i_v \perp\!\!\!\perp \hat{x}_u, \forall v \geq u) \\ &\quad\leq \frac{1}{n}(1-\frac{1}{n})^{(t-2\tau-u-1)_+}\mathbb{E}\big(f_i'(\hat{x}_u)_l - f_i'(x^*)_l\big)^2. \end{aligned} \tag{B.26}$$

Note that the third line used the crucial independence assumption $i_v \perp\!\!\!\perp \hat{x}_u, \forall v \geq u$ arising from our "After Read" ordering. Summing over all dimensions l, we then get:

$$\mathbb{E}\Big[\mathbb{1}_{\{u_{i,l}^t=u\}}\|f_i'(\hat{x}_u) - f_i'(x^*)\|^2\Big] \leq \frac{1}{n}(1-\frac{1}{n})^{(t-2\tau-u-1)_+}\mathbb{E}\|f_i'(\hat{x}_u) - f_i'(x^*)\|^2. \quad \text{(B.27)}$$

So now:

$$\begin{aligned}
\mathbb{E}\|\hat{\alpha}_{i_t}^t - f_{i_t}'(x^*)\|^2 - \lambda\tilde{e}_0 &\leq \frac{1}{n}\sum_{i=1}^{n}\sum_{u=1}^{t-1}\frac{1}{n}(1-\frac{1}{n})^{(t-2\tau-u-1)_+}\mathbb{E}\|f_i'(\hat{x}_u) - f_i'(x^*)\|^2 \\
&= \sum_{u=1}^{t-1}\frac{1}{n}(1-\frac{1}{n})^{(t-2\tau-u-1)_+}\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\|f_i'(\hat{x}_u) - f_i'(x^*)\|^2 \\
&= \sum_{u=1}^{t-1}\frac{1}{n}(1-\frac{1}{n})^{(t-2\tau-u-1)_+}\mathbb{E}\Big(\mathbf{E}\|f_{i_u}'(\hat{x}_u) - f_{i_u}'(x^*)\|^2\Big) \\
&\qquad\qquad (i_u \perp\!\!\!\perp \hat{x}_u) \\
&\leq \frac{2L}{n}\sum_{u=1}^{t-1}(1-\frac{1}{n})^{(t-2\tau-u-1)_+}e_u \qquad \text{(by Equation (B.21))} \\
&= \frac{2L}{n}\sum_{u=1}^{(t-2\tau-1)_+}(1-\frac{1}{n})^{t-2\tau-u-1}e_u + \frac{2L}{n}\sum_{u=\max(1,t-2\tau)}^{t-1} e_u\,.
\end{aligned} \quad \text{(B.28)}$$

Note that we have excluded $\tilde{e}_0$ from our formula, using a generic λ multiplier. We need to treat the case $u=0$ differently to bound $\mathbb{1}_{\{u_{i,l}^t=u\}}$. Because all our initial α_i are initialized to a fixed α_i^0, $\{u_i^t = 0\}$ just means that i has not been picked between 0 and $t-\tau-1$, i.e. $\{i_v \neq i\ \forall\ v \text{ s.t. } 0 \leq v \leq t-\tau-1\}$. This means that the $\mathbb{1}_{\{i_u=i\}}$ term in (B.25) disappears and thus we lose a $\frac{1}{n}$ factor compared to the case where $u > 1$.

Let us now evaluate λ. We have:

$$\begin{aligned}
\mathbb{E}\Big[\mathbb{1}_{\{u_i^t=0\}}\|\alpha_i^0 - f_i'(x^*)\|^2\Big] &\leq \mathbb{E}\Big[\mathbb{1}_{\{i_v\neq i\ \forall\ v \text{ s.t. } 0\leq v\leq t-\tau-1\}}\|\alpha_i^0 - f_i'(x^*)\|^2\Big] \\
&\leq P\{i_v \neq i\ \forall\ v \text{ s.t. } 0 \leq v \leq t-\tau-1\}\mathbb{E}\|\alpha_i^0 - f_i'(x^*)\|^2 \\
&\leq (1-\frac{1}{n})^{(t-\tau)_+}\mathbb{E}\|\alpha_i^0 - f_i'(x^*)\|^2.
\end{aligned} \quad \text{(B.29)}$$

Plugging (B.28) and (B.29) into (B.20), we get Lemma 20:

$$\mathbb{E}\|g_t\|^2 \leq 4Le_t + \frac{4L}{n}\sum_{u=1}^{t-1}(1-\frac{1}{n})^{(t-2\tau-u-1)_+}e_u + 4L(1-\frac{1}{n})^{(t-\tau)_+}\tilde{e}_0\,, \quad \text{(B.30)}$$

where we have introduced $\tilde{e}_0 = \frac{1}{2L}\mathbb{E}\|\alpha_i^0 - f_i'(x^*)\|^2$. Note that in the original SAGA algorithm, a batch gradient is computed to set the $\alpha_i^0 = f_i'(x_0)$. In this setting, we can write Lemma 20 using only $\tilde{e}_0 \leq e_0$ thanks to (B.21). In the more general setting where we initialize all α_i^0 to a fixed quantity, we cannot use (B.21) to bound $\mathbb{E}\|\alpha_i^0 - f_i'(x^*)\|^2$ which means that we have to introduce $\tilde{e}_0$.

B.2.2 Lemma 20 for AHSVRG

In the simpler case of AHSVRG as described in 3.3.2, we have a slight variation of (B.24):

$$\mathbb{E}\|f_i'(x_0^t) - f_i'(x^*)\|^2 = \sum_{u=0}^{t-1} \mathbb{E}\|\mathbb{1}_{\{u_i^t=u\}}\big(f_i'(\hat{x}_u) - f_i'(x^*)\big)\|^2. \tag{B.31}$$

Note that there is no sum over dimensions in this case because the full gradient computations and writes are synchronized (so the reference gradient is consistent).

As in Section B.2.1, we can upper bound the indicator $\mathbb{1}_{\{u_i^t=u\}}$. Now, $\{u_{i,l}^t = u\}$ requires two things: first, the next B variable sampled after the u^{th} update, $\tilde{B}_u$, [2] was 1; second, B was 0 for every update between u and t (roughly). Since the batch step is fully synchronized, we do not have to worry about updates from the past overwriting the reference gradient (and the iterates x_u where we compute the gradient contains all past updates because we have waited for every core to finish its current update).

However, updating the state variable s to 1 once a $B = 1$ variable is sampled is not atomic. So it is possible to have iterations with time label bigger than u and that still use an older reference gradient for their update. [3] Fortunately, we can consider the state update as any regular update to shared parameters. As such, Assumption 5 applies to it. This means that we can be certain that the reference gradient has been updated for iterations with time label $v \geq u + \tau + 1$.

This gives us:

$$\begin{aligned}\mathbb{E}\|\mathbb{1}_{\{u_i^t=u\}}\big(f_i'(\hat{x}_u) - f_i'(x^*)\big)\|^2 &\leq \mathbb{E}\Big[\mathbb{1}_{\{\tilde{B}_u=1\}}\mathbb{1}_{\{B_v=0\ \forall v \text{ s.t. } u+1\leq v\leq t-\tau-1\}}\|f_i'(\hat{x}_u) - f_i'(x^*)\|^2\Big] \\ &\leq \frac{1}{n}\big(1-\frac{1}{n}\big)^{(t-\tau-u-1)_+}\mathbb{E}\|f_i'(\hat{x}_u) - f_i'(x^*)\|^2\,. \end{aligned} \tag{B.32}$$

This proves Lemma 20 for AHSVRG (while we actually have a slightly better exponent,

2. We introduce this quantity because the iterations where full gradients are computed do not receive a time label since they do not correspond to updates to the iterates.

3. Conceivably, another core could start a new iteration, draw $B = 1$ and try to update s to 1 themselves. This is not an issue since the operation of updating s to 1 is idempotent. Only one reference gradient would be computed in this case.

$(t-\tau-u-1)_+$, we can upperbound it by the term in Lemma 20). Armed with this result, we can finish the proof of Theorem 18 for AHSVRG in exactly the same manner as for ASAGA. By remarking that the cost to get to iteration t (including computing reference batch gradients) is the same in the sequential and parallel version, we see that our analysis for Corollary 19 for ASAGA also applies for AHSVRG, so both algorithms obey the same convergence and speedup results.

B.2.3 Master inequality derivation

Now, if we combine the bound on $\mathbb{E}\|g_t\|^2$ which we just derived (i.e. Lemma 20) with Lemma 11, we get:

$$\begin{aligned}
a_{t+1} \leq & (1-\frac{\gamma\mu}{2})a_t - 2\gamma e_t \\
& + 4L\gamma^2 C_1 e_t + \frac{4L\gamma^2 C_1}{n}\sum_{u=1}^{t-1}(1-\frac{1}{n})^{(t-2\tau-u-1)_+}e_u + 4L\gamma^2 C_1(1-\frac{1}{n})^{(t-\tau)_+}\tilde{e}_0 \\
& + 4L\gamma^2 C_2 \sum_{u=(t-\tau)_+}^{t-1} e_u + 4L\gamma^2 C_2 \sum_{u=(t-\tau)_+}^{t-1}(1-\frac{1}{n})^{(u-\tau)_+}\tilde{e}_0 \\
& + \frac{4L\gamma^2 C_2}{n}\sum_{u=(t-\tau)_+}^{t-1}\sum_{v=1}^{u-1}(1-\frac{1}{n})^{(u-2\tau-v-1)_+}e_v\,. \qquad \text{(B.33)}
\end{aligned}$$

If we define $H_t := \sum_{u=1}^{t-1}(1-\frac{1}{n})^{(t-2\tau-u-1)_+}e_u$, then we get:

$$\begin{aligned}
a_{t+1} \leq & (1-\frac{\gamma\mu}{2})a_t - 2\gamma e_t \\
& + 4L\gamma^2 C_1\big(e_t + (1-\frac{1}{n})^{(t-\tau)_+}\tilde{e}_0\big) + \frac{4L\gamma^2 C_1}{n}H_t \\
& + 4L\gamma^2 C_2 \sum_{u=(t-\tau)_+}^{t-1}\big(e_u + (1-\frac{1}{n})^{(u-\tau)_+}\tilde{e}_0\big) + \frac{4L\gamma^2 C_2}{n}\sum_{u=(t-\tau)_+}^{t-1} H_u\,, \qquad \text{(B.34)}
\end{aligned}$$

which is the master inequality (3.13).

B.2.4 Lyapunov function and associated recursive inequality

We define $\mathcal{L}_t := \sum_{u=0}^{t}(1-\rho)^{t-u}a_u$ for some target contraction rate $\rho < 1$ to be defined later. We have:

$$\begin{aligned}\mathcal{L}_{t+1} &= (1-\rho)^{t+1}a_0 + \sum_{u=1}^{t+1}(1-\rho)^{t+1-u}a_u \\ &= (1-\rho)^{t+1}a_0 + \sum_{u=0}^{t}(1-\rho)^{t-u}a_{u+1}\,. \end{aligned} \tag{B.35}$$

We now use our new bound on a_{t+1}, (B.34):

$$\begin{aligned}\mathcal{L}_{t+1} \leq (1-\rho)^{t+1}a_0 + \sum_{u=0}^{t}(1-\rho)^{t-u}\Big[&(1-\frac{\gamma\mu}{2})a_u - 2\gamma e_u \\ &+ 4L\gamma^2 C_1\big(e_u + (1-\frac{1}{n})^{(u-\tau)_+}\tilde{e}_0\big) \\ &+ \frac{4L\gamma^2 C_1}{n}H_u + \frac{4L\gamma^2 C_2}{n}\sum_{v=(u-\tau)_+}^{u-1}H_v \\ &+ 4L\gamma^2 C_2\sum_{v=(u-\tau)_+}^{u-1}\big(e_v + (1-\frac{1}{n})^{(v-\tau)_+}\tilde{e}_0\big)\Big] \\ \leq (1-\rho)^{t+1}a_0 + (1-\frac{\gamma\mu}{2})\mathcal{L}_t &\\ + \sum_{u=0}^{t}(1-\rho)^{t-u}\Big[&-2\gamma e_u + 4L\gamma^2 C_1\big(e_u + (1-\frac{1}{n})^{(u-\tau)_+}\tilde{e}_0\big) \\ &+ \frac{4L\gamma^2 C_1}{n}H_u + \frac{4L\gamma^2 C_2}{n}\sum_{v=(u-\tau)_+}^{u-1}H_v \\ &+ 4L\gamma^2 C_2\sum_{v=(u-\tau)_+}^{u-1}\big(e_v + (1-\frac{1}{n})^{(v-\tau)_+}\tilde{e}_0\big)\Big]. \end{aligned} \tag{B.36}$$

We can now rearrange the sums to expose a simple sum of e_u multiplied by factors r_u^t:

$$\mathcal{L}_{t+1} \leq (1-\rho)^{t+1}a_0 + (1-\frac{\gamma\mu}{2})\mathcal{L}_t + \sum_{u=1}^{t} r_u^t e_u + r_0^t\tilde{e}_0\,. \tag{B.37}$$

B.2.5 Proof of Lemma 21 (convergence condition for ASAGA)

We want to make explicit what conditions on ρ and γ are necessary to ensure that r_u^t is negative for all $u \geq 1$. Since each e_u is positive, we will then be able to safely drop

the sum term from the inequality. The r_0^t term is a bit trickier and is handled separately. Indeed, trying to enforce that r_0^t is negative results in a significantly worse condition on γ and eventually a convergence rate smaller by a factor of n than our final result. Instead, we handle this term directly in the Lyapunov function.

Computation of r_u^t. Let's now make the multiplying factor explicit. We assume $u \geq 1$. We split r_u^t into five parts coming from (B.36):

- r_1, the part coming from the $-2\gamma e_u$ terms;
- r_2, coming from $4L\gamma^2 C_1 e_u$;
- r_3, coming from $\frac{4L\gamma^2 C_1}{n} H_u$;
- r_4, coming from $4L\gamma^2 C_2 \sum_{v=(u-\tau)_+}^{u-1} e_v$;
- r_5, coming from $\frac{4L\gamma^2 C_2}{n} \sum_{v=(u-\tau)_+}^{u-1} H_v$.

r_1 is easy to derive. Each of these terms appears only in one inequality. So for u at time t, the term is:

$$r_1 = -2\gamma(1-\rho)^{t-u}. \tag{B.38}$$

For much the same reasons, r_2 is also easy to derive and is:

$$r_2 = 4L\gamma^2 C_1 (1-\rho)^{t-u}. \tag{B.39}$$

r_3 is a bit trickier, because for a given $v > 0$ there are several H_u which contain e_v. The key insight is that we can rewrite our double sum in the following manner:

$$\begin{aligned}
&\sum_{u=0}^{t}(1-\rho)^{t-u}\sum_{v=1}^{u-1}(1-\frac{1}{n})^{(u-2\tau-v-1)_+}e_v \\
&\qquad = \sum_{v=1}^{t-1} e_v \sum_{u=v+1}^{t}(1-\rho)^{t-u}(1-\frac{1}{n})^{(u-2\tau-v-1)_+} \\
&\qquad \leq \sum_{v=1}^{t-1} e_v \Big[\sum_{u=v+1}^{\min(t,v+2\tau)}(1-\rho)^{t-u} + \sum_{u=v+2\tau+1}^{t}(1-\rho)^{t-u}(1-\frac{1}{n})^{u-2\tau-v-1}\Big] \\
&\qquad \leq \sum_{v=1}^{t-1} e_v \Big[2\tau(1-\rho)^{t-v-2\tau} + (1-\rho)^{t-v-2\tau-1}\sum_{u=v+2\tau+1}^{t} q^{u-2\tau-v-1}\Big] \\
&\qquad \leq \sum_{v=1}^{t-1}(1-\rho)^{t-v} e_v (1-\rho)^{-2\tau-1}\big[2\tau + \frac{1}{1-q}\big],
\end{aligned} \tag{B.40}$$

where we have defined:

$$q := \frac{1-1/n}{1-\rho}, \quad \text{with the assumption } \rho < \frac{1}{n}\,. \tag{B.41}$$

Note that we have bounded the $\min(t, v+2\tau)$ term by $v+2\tau$ in the first sub-sum, effectively adding more positive terms.

This gives us that at time t, for u:

$$r_3 \leq \frac{4L\gamma^2 C_1}{n}(1-\rho)^{t-u}(1-\rho)^{-2\tau-1}\big[2\tau + \frac{1}{1-q}\big]. \tag{B.42}$$

For r_4 we use the same trick:

$$\begin{aligned}\sum_{u=0}^{t}(1-\rho)^{t-u}\sum_{v=(u-\tau)_+}^{u-1} e_v &= \sum_{v=0}^{t-1} e_v \sum_{u=v+1}^{\min(t,v+\tau)} (1-\rho)^{t-u} \\ &\leq \sum_{v=0}^{t-1} e_v \sum_{u=v+1}^{v+\tau}(1-\rho)^{t-u} \leq \sum_{v=0}^{t-1} e_v \tau (1-\rho)^{t-v-\tau}. \end{aligned} \tag{B.43}$$

This gives us that at time t, for u:

$$r_4 \leq 4L\gamma^2 C_2 (1-\rho)^{t-u}\tau(1-\rho)^{-\tau}\,. \tag{B.44}$$

Finally we compute r_5 which is the most complicated term. Indeed, to find the factor of e_w for a given $w > 0$, one has to compute a triple sum, $\sum_{u=0}^{t}(1-\rho)^{t-u}\sum_{v=(u-\tau)_+}^{u-1} H_v$. We start by computing the factor of e_w in the inner double sum, $\sum_{v=(u-\tau)_+}^{u-1} H_v$.

$$\sum_{v=(u-\tau)_+}^{u-1}\sum_{w=1}^{v-1}(1-\frac{1}{n})^{(v-2\tau-w-1)_+}e_w = \sum_{w=1}^{u-2} e_w \sum_{v=\max(w+1,u-\tau)}^{u-1}(1-\frac{1}{n})^{(v-2\tau-w-1)_+}\,. \tag{B.45}$$

Now there are at most τ terms for each e_w. If $w \leq u-3\tau-1$, then the exponent is positive in every term and it is always bigger than $u-3\tau-1-w$, which means we can bound the sum by $\tau(1-\frac{1}{n})^{u-3\tau-1-w}$. Otherwise we can simply bound the sum by τ. We get:

$$\sum_{v=(u-\tau)_+}^{u-1} H_v \leq \sum_{w=1}^{u-2}\big[\mathbb{1}_{\{u-3\tau\leq w\leq u-2\}}\tau + \mathbb{1}_{\{w\leq u-3\tau-1\}}\tau(1-\frac{1}{n})^{u-3\tau-1-w}\big]e_w\,. \tag{B.46}$$

This means that for w at time t:

$$
\begin{aligned}
r_5 &\leq \frac{4L\gamma^2 C_2}{n}\sum_{u=0}^{t}(1-\rho)^{t-u}\Big[\mathbb{1}_{\{u-3\tau\leq w\leq u-2\}}\tau + \mathbb{1}_{\{w\leq u-3\tau-1\}}\tau(1-\frac{1}{n})^{u-3\tau-1-w}\Big] \\
&\leq \frac{4L\gamma^2 C_2}{n}\Big[\sum_{u=w+2}^{\min(t,w+3\tau)}\tau(1-\rho)^{t-u} + \sum_{u=w+3\tau+1}^{t}\tau(1-\frac{1}{n})^{u-3\tau-1-w}(1-\rho)^{t-u}\Big] \\
&\leq \frac{4L\gamma^2 C_2}{n}\tau\Big[(1-\rho)^{t-w}(1-\rho)^{-3\tau}3\tau \\
&\qquad\qquad + (1-\rho)^{t-w}(1-\rho)^{-1-3\tau}\sum_{u=w+3\tau+1}^{t}(1-\frac{1}{n})^{u-3\tau-1-w}(1-\rho)^{-u+3\tau+1+w}\Big] \\
&\leq \frac{4L\gamma^2 C_2}{n}\tau(1-\rho)^{t-w}(1-\rho)^{-3\tau-1}\big(3\tau+\frac{1}{(1-q)}\big)\,. \qquad \text{(B.47)}
\end{aligned}
$$

By combining the five terms together ((B.38), (B.39), (B.49), (B.44) and (B.47)), we get that $\forall u$ s.t. $1\leq u\leq t$:

$$
\begin{aligned}
r_u^t \leq (1-\rho)^{t-u}\Big[&-2\gamma + 4L\gamma^2 C_1 + \frac{4L\gamma^2 C_1}{n}(1-\rho)^{-2\tau-1}\big(2\tau+\frac{1}{1-q}\big) \\
&+ 4L\gamma^2 C_2\tau(1-\rho)^{-\tau} + \frac{4L\gamma^2 C_2}{n}\tau(1-\rho)^{-3\tau-1}\big(3\tau+\frac{1}{1-q}\big)\Big]. \qquad \text{(B.48)}
\end{aligned}
$$

Computation of r_0^t. Recall that we treat the $\tilde{e}_0$ term separately in Section B.2.1. The initialization of SAGA creates an initial synchronization, which means that the contribution of $\tilde{e}_0$ in our bound on $\mathbb{E}\|g_t\|^2$ (B.30) is roughly n times bigger than the contribution of any e_u for $1<u<t$.[4] In order to safely handle this term in our Lyapunov inequality, we only need to prove that it is bounded by a reasonable constant. Here again, we split r_0^t in five contributions coming from (B.36):

- r_1, the part coming from the $-2\gamma e_u$ terms;
- r_2, coming from $4L\gamma^2 C_1 e_u$;
- r_3, coming from $4L\gamma^2 C_1(1-\frac{1}{n})^{(u-\tau)_+}\tilde{e}_0$;
- r_4, coming from $4L\gamma^2 C_2\sum_{v=(u-\tau)_+}^{u-1} e_v$;
- r_5, coming from $4L\gamma^2 C_2\sum_{v=(u-\tau)_+}^{u-1}(1-\frac{1}{n})^{(v-\tau)_+}\tilde{e}_0$.

Note that there is no $\tilde{e}_0$ in H_t, which is why we can safely ignore these terms here.

4. This is explained in details right before (B.29).

We have $r_1 = -2\gamma(1-\rho)^t$ and $r_2 = 4L\gamma^2 C_1(1-\rho)^t$. Let us compute r_3.

$$\begin{aligned}
&\sum_{u=0}^{t}(1-\rho)^{t-u}(1-\frac{1}{n})^{(u-\tau)_+} \\
&\qquad = \sum_{u=0}^{\min(t,\tau)}(1-\rho)^{t-u} + \sum_{u=\tau+1}^{t}(1-\rho)^{t-u}(1-\frac{1}{n})^{u-\tau} \\
&\qquad \leq (\tau+1)(1-\rho)^{t-\tau} + (1-\rho)^{t-\tau}\sum_{u=\tau+1}^{t}(1-\rho)^{\tau-u}(1-\frac{1}{n})^{u-\tau} \\
&\qquad \leq (1-\rho)^t(1-\rho)^{-\tau}\big(\tau+1+\frac{1}{1-q}\big)\,. \qquad \text{(B.49)}
\end{aligned}$$

This gives us:

$$r_3 \leq (1-\rho)^t 4L\gamma^2 C_1(1-\rho)^{-\tau}\big(\tau+1+\frac{1}{1-q}\big)\,. \qquad \text{(B.50)}$$

We have already computed r_4 for $u > 0$ and the computation is exactly the same for $u = 0$. $r_4 \leq (1-\rho)^t 4L\gamma^2 C_2 \frac{\tau}{1-\rho}$.

Finally we compute r_5.

$$\begin{aligned}
&\sum_{u=0}^{t}(1-\rho)^{t-u}\sum_{v=(u-\tau)_+}^{u-1}(1-\frac{1}{n})^{(v-\tau)_+} \\
&\qquad = \sum_{v=1}^{t-1}\sum_{u=v+1}^{\min(t,v+\tau)}(1-\rho)^{t-u}(1-\frac{1}{n})^{(v-\tau)_+} \\
&\qquad \leq \sum_{v=1}^{\min(t-1,\tau)}\sum_{u=v+1}^{v+\tau}(1-\rho)^{t-u} + \sum_{v=\tau+1}^{t-1}\sum_{u=v+1}^{\min(t,v+\tau)}(1-\rho)^{t-u}(1-\frac{1}{n})^{v-\tau} \\
&\qquad \leq \tau^2(1-\rho)^{t-2\tau} + \sum_{v=\tau+1}^{t-1}(1-\frac{1}{n})^{v-\tau}\tau(1-\rho)^{t-v-\tau} \\
&\qquad \leq \tau^2(1-\rho)^{t-2\tau} + \tau(1-\rho)^t(1-\rho)^{-2\tau}\sum_{v=\tau+1}^{t-1}(1-\frac{1}{n})^{v-\tau}\tau(1-\rho)^{-v+\tau} \\
&\qquad \leq (1-\rho)^t(1-\rho)^{-2\tau}\big(\tau^2+\tau\frac{1}{1-q}\big)\,. \qquad \text{(B.51)}
\end{aligned}$$

Which means:

$$r_5 \leq (1-\rho)^t 4L\gamma^2 C_2(1-\rho)^{-2\tau}\big(\tau^2+\tau\frac{1}{1-q}\big). \qquad \text{(B.52)}$$

Putting it all together, we get that: $\forall t \geq 0$,

$$r_0^t \leq (1-\rho)^t \Big[\Big(-2\gamma + 4L\gamma^2 C_1 + 4L\gamma^2 C_2 \frac{\tau}{1-\rho} \Big) \frac{e_0}{\tilde{e}_0} \\ + 4L\gamma^2 C_1 (1-\rho)^{-\tau} \big(\tau + 1 + \frac{1}{1-q}\big) + 4L\gamma^2 C_2 \tau (1-\rho)^{-2\tau} \big(\tau + \frac{1}{1-q}\big) \Big]. \tag{B.53}$$

Sufficient condition for convergence. We need all $r_u^t, u \geq 1$ to be negative so we can safely drop them from (B.37). Note that for every u, this is the same condition. We will reduce that condition to a second-order polynomial sign condition. We also remark that since $\gamma \geq 0$, we can upper bound our terms in γ and γ^2 in this upcoming polynomial, which will give us sufficient conditions for convergence.

Now, recall that $C_2(\gamma)$ (as defined in (2.13)) depends on γ. We thus need to expand it once more to find our conditions. We have:

$$C_1 = 1 + \sqrt{\Delta}\tau; \qquad C_2 = \sqrt{\Delta} + \gamma\mu C_1 \,.$$

Dividing the bracket in (B.48) by γ and rearranging as a second degree polynomial, we get the condition:

$$4L \Bigg(C_1 + \frac{C_1}{n}(1-\rho)^{-2\tau-1} \Big[2\tau + \frac{1}{1-q} \Big] \\ + \Big[\frac{\sqrt{\Delta}\tau}{(1-\rho)^{\tau}} + \frac{\sqrt{\Delta}\tau}{n}(1-\rho)^{-3\tau-1}(3\tau + \frac{1}{1-q}) \Big] \Bigg) \gamma \\ + 8\mu C_1 L\tau \Big[(1-\rho)^{-\tau} + \frac{1}{n}(1-\rho)^{-3\tau-1}(3\tau + \frac{1}{1-q}) \Big] \gamma^2 + 2 \leq 0 \,. \tag{B.54}$$

The discriminant of this polynomial is always positive, so γ needs to be between its two roots. The smallest is negative, so the condition is not relevant to our case (where $\gamma > 0$). By solving analytically for the positive root ϕ, we get an upper bound condition on γ that can be used for any overlap τ and guarantee convergence. Unfortunately, for large τ, the upper bound becomes exponentially small because of the presence of τ in the exponent in (B.54). More specifically, by using the bound $1/(1-\rho) \leq \exp(2\rho)$ [5] and thus $(1-\rho)^{-\tau} \leq \exp(2\tau\rho)$ in (B.54), we would obtain factors of the form $\exp(\tau/n)$ in the denominator for the root ϕ (recall that $\rho < 1/n$).

Our Lemma 21 is derived instead under the assumption that $\tau \leq \mathcal{O}(n)$, with the constants chosen in order to make the condition (B.54) more interpretable and to relate our

5. This bound can be derived from the inequality $(1 - x/2) \geq \exp(-x)$ which is valid for $0 \leq x \leq 1.59$.

convergence result with the standard SAGA convergence (see Theorem 15). As explained in Section 3.4.7, the assumption that $\tau \leq \mathcal{O}(n)$ appears reasonable in practice. First, by using Bernoulli's inequality, we have:

$$(1-\rho)^{k\tau} \geq 1 - k\tau\rho \qquad \text{for integers} \quad k\tau \geq 0\,. \tag{B.55}$$

To get manageable constants, we make the following slightly more restrictive assumptions on the target rate ρ [6] and overlap τ: [7]

$$\rho \leq \frac{1}{4n} \tag{B.56}$$

$$\tau \leq \frac{n}{10}\,. \tag{B.57}$$

We then have:

$$\frac{1}{1-q} \leq \frac{4n}{3} \tag{B.58}$$

$$\frac{1}{1-\rho} \leq \frac{4}{3} \tag{B.59}$$

$$k\tau\rho \leq \frac{3}{40} \qquad \text{for } 1 \leq k \leq 3 \tag{B.60}$$

$$(1-\rho)^{-k\tau} \leq \frac{1}{1-k\tau\rho} \leq \frac{40}{37} \qquad \text{for } 1 \leq k \leq 3 \text{ and by using (B.55).} \tag{B.61}$$

We can now upper bound loosely the three terms in brackets appearing in (B.54) as follows:

$$(1-\rho)^{-2\tau-1}\big[2\tau + \frac{1}{1-q}\big] \leq 3n \tag{B.62}$$

$$\sqrt{\Delta}\tau(1-\rho)^{-\tau} + \frac{\sqrt{\Delta}\tau}{n}(1-\rho)^{-3\tau-1}(3\tau + \frac{1}{1-q}) \leq 4\sqrt{\Delta}\tau \leq 4C_1 \tag{B.63}$$

$$(1-\rho)^{-\tau} + \frac{1}{n}(1-\rho)^{-3\tau-1}(3\tau + \frac{1}{1-q}) \leq 4\,. \tag{B.64}$$

By plugging (B.62)–(B.64) into (B.54), we get the simpler sufficient condition on γ:

$$-1 + 16LC_1\gamma + 16LC_1\mu\tau\gamma^2 \leq 0\,. \tag{B.65}$$

6. Note that we already expected $\rho < 1/n$.
7. This bound on τ is reasonable in practice, see Appendix 3.4.7.

The positive root ϕ is:

$$\phi = \frac{16LC_1(\sqrt{1+\frac{\mu\tau}{4LC_1}}-1)}{32LC_1\mu\tau} = \frac{\sqrt{1+\frac{\mu\tau}{4LC_1}}-1}{2\mu\tau}\,. \tag{B.66}$$

As in our HOGWILD analysis, we use (A.19) (from the concavity of the square root function) in (B.66), and recalling that $\kappa := L/\mu$, we get:

$$\phi \geq \frac{1}{16LC_1\sqrt{1+\frac{\tau}{4\kappa C_1}}}\,. \tag{B.67}$$

Since $\frac{\tau}{C_1} = \frac{\tau}{1+\sqrt{\Delta}\tau} \leq \min(\tau, \frac{1}{\sqrt{\Delta}})$, we get that a sufficient condition on our step size is:

$$\gamma \leq \frac{1}{16L(1+\sqrt{\Delta}\tau)\sqrt{1+\frac{1}{4\kappa}\min(\tau,\frac{1}{\sqrt{\Delta}})}}\,. \tag{B.68}$$

Subject to our conditions on γ, ρ and τ, we then have that: $r_u^t \leq 0$ for all u s.t. $1 \leq u \leq t$. This means we can rewrite (B.37) as:

$$\mathcal{L}_{t+1} \leq (1-\rho)^{t+1}a_0 + (1-\frac{\gamma\mu}{2})\mathcal{L}_t + r_0^t\tilde{e}_0\,. \tag{B.69}$$

Now, we could finish the proof from this inequality, but it would only give us a convergence result in terms of $a_t = \mathbb{E}\|x_t - x^*\|^2$. A better result would be in terms of the suboptimality at $\hat{x}_t$ (because $\hat{x}_t$ is a real quantity in the algorithm whereas x_t is virtual). Fortunately, to get such a result, we can easily adapt (B.69).

We make e_t appear on the left side of (B.69), by adding γ to r_t^t in (B.37): [8]

$$\gamma e_t + \mathcal{L}_{t+1} \leq (1-\rho)^{t+1}a_0 + (1-\frac{\gamma\mu}{2})\mathcal{L}_t + \sum_{u=1}^{t-1} r_u^t e_u + r_0^t\tilde{e}_0 + (r_t^t+\gamma)e_t. \tag{B.70}$$

We now require the stronger property that $\gamma + r_t^t \leq 0$, which translates to replacing -2γ with $-\gamma$ in (B.48):

$$\begin{aligned} 0 \geq \Big[&-\gamma + 4L\gamma^2C_1 + \frac{4L\gamma^2C_1}{n}(1-\rho)^{-2\tau-1}\big(2\tau + \frac{1}{1-q}\big) \\ &+ 4L\gamma^2C_2\tau(1-\rho)^{-\tau} + \frac{4L\gamma^2C_2}{n}\tau(1-\rho)^{-3\tau}\big(3\tau+\frac{1}{1-q}\big)\Big]. \end{aligned} \tag{B.71}$$

8. We could use any multiplier from 0 to 2γ, but choose γ for simplicity. For this reason and because our analysis of the r_t^t term was loose, we could derive a tighter bound, but it does not change the leading terms.

We can easily derive a new stronger condition on γ under which we can drop all the $e_u, u > 0$ terms in (B.70):

$$\gamma \leq \gamma^* = \frac{1}{32L(1+\sqrt{\Delta}\tau)\sqrt{1+\frac{1}{8\kappa}\min(\tau, \frac{1}{\sqrt{\Delta}})}}, \tag{B.72}$$

and thus under which we get:

$$\gamma e_t + \mathcal{L}_{t+1} \leq (1-\rho)^{t+1} a_0 + (1 - \frac{\gamma\mu}{2})\mathcal{L}_t + r_0^t \tilde{e}_0. \tag{B.73}$$

This finishes the proof of Lemma 21. ■

B.2.6 Proof of Theorem 18 (convergence guarantee and rate of ASAGA)

End of Lyapunov convergence. We continue with the assumptions of Lemma 21 which gave us (B.73). Thanks to (B.53), we can also rewrite $r_0^t \leq (1-\rho)^{t+1} A$ where A is a constant which depends on n, Δ, γ and L but is finite and crucially does not depend on t. In fact, by reusing similar arguments as in B.2.5, we can show the loose bound $A \leq \gamma n$ under the assumptions of Lemma 21 (including $\gamma \leq \gamma^*$). [9] We then have:

$$\begin{aligned}\mathcal{L}_{t+1} \leq \gamma e_t + \mathcal{L}_{t+1} &\leq (1 - \frac{\gamma\mu}{2})\mathcal{L}_t + (1-\rho)^{t+1}(a_0 + A\tilde{e}_0) \\ &\leq (1 - \frac{\gamma\mu}{2})^{t+1}\mathcal{L}_0 + (a_0 + A\tilde{e}_0)\sum_{k=0}^{t+1}(1-\rho)^{t+1-k}(1-\frac{\gamma\mu}{2})^k. \end{aligned} \tag{B.74}$$

We have two linearly contracting terms. The sum contracts linearly with the minimum geometric rate factor between $\gamma\mu/2$ and ρ. If we define $m := \min(\rho, \gamma\mu/2)$, $M :=$

9. In particular, note that e_0 does not appear in the definition of A because it turns out that the parenthesis group multiplying e_0 in (B.53) is negative. Indeed, it contains less positive terms than (B.48) which we showed to be negative under the assumptions from Lemma 21.

$\max(\rho, \gamma\mu/2)$ and $\rho^* := \nu m$ with $0 < \nu < 1$, [10] we then get: [11]

$$\begin{aligned}
\gamma e_t \leq \gamma e_t + \mathcal{L}_{t+1} &\leq (1 - \frac{\gamma\mu}{2})^{t+1}\mathcal{L}_0 + (a_0 + A\tilde{e}_0)\sum_{k=0}^{t+1}(1-m)^{t+1-k}(1-M)^k \\
&\leq (1 - \frac{\gamma\mu}{2})^{t+1}\mathcal{L}_0 + (a_0 + A\tilde{e}_0)\sum_{k=0}^{t+1}(1-\rho^*)^{t+1-k}(1-M)^k \\
&\leq (1 - \frac{\gamma\mu}{2})^{t+1}\mathcal{L}_0 + (a_0 + A\tilde{e}_0)(1-\rho^*)^{t+1}\sum_{k=0}^{t+1}(1-\rho^*)^{-k}(1-M)^k \\
&\leq (1 - \frac{\gamma\mu}{2})^{t+1}\mathcal{L}_0 + (1-\rho^*)^{t+1}\frac{1}{1-\eta}(a_0 + A\tilde{e}_0) \\
&\leq (1-\rho^*)^{t+1}\big(a_0 + \frac{1}{1-\eta}(a_0 + A\tilde{e}_0)\big), \qquad \text{(B.75)}
\end{aligned}$$

where $\eta := \frac{1-M}{1-\rho^*}$. We have $\frac{1}{1-\eta} = \frac{1-\rho^*}{M-\rho^*}$.

By taking $\nu = \frac{4}{5}$ and setting $\rho = \frac{1}{4n}$ – its maximal value allowed by the assumptions of Lemma 21 – we get $M \geq \frac{1}{4n}$ and $\rho^* \leq \frac{1}{5n}$, which means $\frac{1}{1-\eta} \leq 20n$. All told, using $A \leq \gamma n$, we get:

$$e_t \leq (1-\rho^*)^{t+1}\tilde{C}_0, \qquad \text{(B.76)}$$

where:

$$\tilde{C}_0 := \frac{21n}{\gamma}\left(\|x_0 - x^*\|^2 + \gamma\frac{n}{2L}\mathbb{E}\|\alpha_i^0 - f_i'(x^*)\|^2\right). \qquad \text{(B.77)}$$

Since we set $\rho = \frac{1}{4n}, \nu = \frac{4}{5}$, we have $\nu\rho = \frac{1}{5n}$. Using a step size $\gamma = \frac{a}{L}$ as in Theorem 18, we get $\nu\frac{\gamma\mu}{2} = \frac{2a}{5\kappa}$. We thus obtain a geometric rate of $\rho^* = \min\{\frac{1}{5n}, a\frac{2}{5\kappa}\}$, which we simplified to $\frac{1}{5}\min\{\frac{1}{n}, a\frac{1}{\kappa}\}$ in Theorem 18, finishing the proof. We also observe that $\tilde{C}_0 \leq \frac{60n}{\gamma}C_0$, with C_0 defined in Theorem 15. ■

B.2.7 Proof of Corollary 19 (speedup regimes for ASAGA)

Referring to Hofmann et al. (2015) and our own Theorem 15, the geometric rate factor of SAGA is $\frac{1}{5}\min\{\frac{1}{n}, \frac{a}{\kappa}\}$ for a step size of $\gamma = \frac{a}{5L}$. We start by proving the first part of the corollary which considers the step size $\gamma = \frac{a}{L}$ with $a = a^*(\tau)$. We distinguish between two regimes to study the parallel speedup our algorithm obtains and to derive a condition on τ for which we have a linear speedup.

10. ν is introduced to circumvent the problematic case where ρ and $\gamma\mu/2$ are too close together, which does not prevent the geometric convergence, but makes the constant $\frac{1}{1-\eta}$ potentially very big (in the case both terms are equal, the sum even becomes an annoying linear term in t).

11. Note that if $m \neq \rho$, we can perform the index change $t+1-k \to k$ to get the sum.

Well-conditioned regime. In this regime, $n > \kappa$ and the geometric rate factor of sequential SAGA is $\frac{1}{5n}$. To get a linear speedup (up to a constant factor), we need to enforce $\rho^* = \Omega(\frac{1}{n})$. We recall that $\rho^* = \min\{\frac{1}{5n}, a\frac{1}{5\kappa}\}$.

We already have $\frac{1}{5n} = \Omega(\frac{1}{n})$. This means that we need τ to verify $\frac{a^*(\tau)}{5\kappa} = \Omega(\frac{1}{n})$, where $a^*(\tau) = \frac{1}{32\left(1+\tau\sqrt{\Delta}\right)\xi(\kappa,\Delta,\tau)}$ according to Theorem 18. Recall that $\xi(\kappa,\Delta,\tau) := \sqrt{1+\frac{1}{8\kappa}\min\{\frac{1}{\sqrt{\Delta}},\tau\}}$. Up to a constant factor, this means we can give the following sufficient condition:

$$\frac{1}{\kappa\left(1+\tau\sqrt{\Delta}\right)\xi(\kappa,\Delta,\tau)} = \Omega\Big(\frac{1}{n}\Big) \tag{B.78}$$

i.e.

$$\left(1+\tau\sqrt{\Delta}\right)\xi(\kappa,\Delta,\tau) = \mathcal{O}\Big(\frac{n}{\kappa}\Big)\,. \tag{B.79}$$

We now consider two alternatives, depending on whether κ is bigger than $\frac{1}{\sqrt{\Delta}}$ or not. If $\kappa \geq \frac{1}{\sqrt{\Delta}}$, then $\xi(\kappa,\Delta,\tau) < 2$ and we can rewrite the sufficient condition (B.79) as:

$$\tau = \mathcal{O}(1)\frac{n}{\kappa\sqrt{\Delta}}. \tag{B.80}$$

In the alternative case, $\kappa \leq \frac{1}{\sqrt{\Delta}}$. Since $a^*(\tau)$ is decreasing in τ, we can suppose $\tau \geq \frac{1}{\sqrt{\Delta}}$ without loss of generality and thus $\xi(\kappa,\Delta,\tau) = \sqrt{1+\frac{1}{8\kappa\sqrt{\Delta}}}$. We can then rewrite the sufficient condition (B.79) as:

$$\begin{aligned}\frac{\tau\sqrt{\Delta}}{\sqrt{\kappa}\sqrt[4]{\Delta}} &= \mathcal{O}(\frac{n}{\kappa})\,;\\ \tau &= \mathcal{O}(1)\frac{n}{\sqrt{\kappa}\sqrt[4]{\Delta}}\,.\end{aligned} \tag{B.81}$$

We observe that since we have supposed that $\kappa \leq \frac{1}{\sqrt{\Delta}}$, we have $\sqrt{\kappa\sqrt{\Delta}} \leq \kappa\sqrt{\Delta} \leq 1$, which means that our initial assumption that $\tau < \frac{n}{10}$ is stronger than condition (B.81).

We can now combine both cases to get the following sufficient condition for the geometric rate factor of ASAGA to be the same order as sequential SAGA when $n > \kappa$:

$$\tau = \mathcal{O}(1)\frac{n}{\kappa\sqrt{\Delta}}; \quad \tau = \mathcal{O}(n)\,. \tag{B.82}$$

Ill-conditioned regime. In this regime, $\kappa > n$ and the geometric rate factor of sequential SAGA is $a\frac{1}{\kappa}$. Here, to obtain a linear speedup, we need $\rho^* = \mathcal{O}(\frac{1}{\kappa})$. Since $\frac{1}{n} > \frac{1}{\kappa}$, all we require is that $\frac{a^*(\tau)}{\kappa} = \Omega(\frac{1}{\kappa})$ where $a^*(\tau) = \frac{1}{32\left(1+\tau\sqrt{\Delta}\right)\xi(\kappa,\Delta,\tau)}$, which reduces to $a^*(\tau) = \Omega(1)$.

We can give the following sufficient condition:

$$\frac{1}{\left(1+\tau\sqrt{\Delta}\right)\xi(\kappa,\Delta,\tau)} = \Omega(1) \tag{B.83}$$

Using that $\frac{1}{n} \leq \Delta \leq 1$ and that $\kappa > n$, we get that $\xi(\kappa,\Delta,\tau) \leq 2$, which means our sufficient condition becomes:

$$\begin{aligned} \tau\sqrt{\Delta} &= \mathcal{O}(1) \\ \tau &= \frac{\mathcal{O}(1)}{\sqrt{\Delta}}. \end{aligned} \tag{B.84}$$

This finishes the proof for the first part of Corollary 19.

Universal step size. If $\tau = \mathcal{O}(\frac{1}{\sqrt{\Delta}})$, then $\xi(\kappa,\Delta,\tau) = \mathcal{O}(1)$ and $(1+\tau\sqrt{\Delta}) = \mathcal{O}(1)$, and thus $a^*(\tau) = \Omega(1)$ (for any n and κ). This means that the universal step size $\gamma = \Theta(1/L)$ satisfies $\gamma \leq a^*(\tau)$ for any κ, giving the same rate factor $\Omega(\min\{\frac{1}{n}, \frac{1}{\kappa}\})$ that sequential SAGA has, completing the proof for the second part of Corollary 19. ■

B.3 KROMAGNON – Proof of Theorem 22 and Corollary 25

B.3.1 Proof of Lemma 26 (suboptimality bound on $\mathbb{E}\|g_t\|^2$)

Mania et al. (2017, Lemma 9), tells us that for serial sparse SVRG we have for all $km \leq t \leq (k+1)m - 1$:

$$\mathbb{E}\|g_t\|^2 \leq 2\mathbb{E}\|f'_{i_t}(\hat{x}_t) - f'_{i_t}(x^*)\|^2 + 2\mathbb{E}\|f'_{i_t}(\hat{x}_k) - f'_{i_t}(x^*)\|^2. \tag{B.85}$$

This remains true in the case of KROMAGNON. We can use Hofmann et al. (2015, Equations (7) and (8)) to bound both terms in the following manner:

$$\mathbb{E}\|g_t\|^2 \leq 4L(\mathbb{E}f(\hat{x}_t) - f(x^*)) + 4L(\mathbb{E}f(\tilde{x}_k) - f(x^*)) \leq 4Le_t + 4L\tilde{e}_k\,. \tag{B.86}$$

■

B.3.2 Proof of Theorem 22 (convergence rate of KROMAGNON)

Master inequality derivation. As in our ASAGA analysis, we plug Lemma 26 in Lemma 11, which gives us that for all $k \geq 0, km \leq t \leq (k+1)m - 1$:

$$a_{t+1} \leq (1-\frac{\gamma\mu}{2})a_t+\gamma^2 C_1(4Le_t+4L\tilde{e}_k)+\gamma^2 C_2 \sum_{u=\max(km,t-\tau)}^{t-1}(4Le_t+4L\tilde{e}_k)-2\gamma e_t\,. \quad \text{(B.87)}$$

By grouping the $\tilde{e}_k$ and the e_t terms we get our master inequality (3.28):

$$a_{t+1} \leq (1-\frac{\gamma\mu}{2})a_t+(4L\gamma^2 C_1-2\gamma)e_t+4L\gamma^2 C_2 \sum_{u=\max(km,t-\tau)}^{t-1} e_u+(4L\gamma^2 C_1+4L\gamma^2\tau C_2)\tilde{e}_k\,.$$

Contraction inequality derivation. We now adopt the same method as in the original SVRG paper (Johnson and Zhang, 2013); we sum the master inequality over a whole epoch and then we use the randomization trick:

$$\tilde{e}_k = \mathbb{E}f(\tilde{x}_k) - f(x^*) = \frac{1}{m}\sum_{t=(k-1)m}^{km-1} e_t \quad \text{(B.88)}$$

This gives us:

$$\begin{aligned}\sum_{t=km+1}^{(k+1)m} a_t \leq &(1-\frac{\gamma\mu}{2})\sum_{t=km}^{(k+1)m-1} a_t + (4L\gamma^2 C_1 - 2\gamma)\sum_{t=km}^{(k+1)m-1} e_t \\ &+ 4L\gamma^2 C_2 \sum_{t=km}^{(k+1)m-1}\sum_{u=\max(km,t-\tau)}^{t-1} e_u + m(4L\gamma^2 C_1 + 4L\gamma^2\tau C_2)\tilde{e}_k\,. \quad \text{(B.89)}\end{aligned}$$

Since $1-\frac{\gamma\mu}{2} < 1$, we can upper bound it by 1, and then remove all the telescoping terms from (B.89). We also have:

$$\begin{aligned}\sum_{t=km}^{(k+1)m-1}\sum_{u=\max(km,t-\tau)}^{t-1} e_u = \sum_{u=km}^{(k+1)m-2}\sum_{t=u+1}^{\min((k+1)m-1,u+\tau)} e_u &\leq \tau\sum_{u=km}^{(k+1)m-2} e_u \\ &\leq \tau\sum_{u=km}^{(k+1)m-1} e_u\,. \quad \text{(B.90)}\end{aligned}$$

All told:

$$\begin{aligned}a_{(k+1)m} \leq a_{km} &+ (4L\gamma^2 C_1 + 4L\gamma^2\tau C_2 - 2\gamma)\sum_{t=km}^{(k+1)m-1} e_t \\ &+ m(4L\gamma^2 C_1 + 4L\gamma^2\tau C_2)\tilde{e}_k\,. \quad \text{(B.91)}\end{aligned}$$

Now we use the randomization trick (B.88):

$$a_{(k+1)m} \leq a_{km} + (4L\gamma^2 C_1 + 4L\gamma^2\tau C_2 - 2\gamma)m\tilde{e}_{k+1} + m(4L\gamma^2 C_1 + 4L\gamma^2\tau C_2)\tilde{e}_k\,. \quad \text{(B.92)}$$

Finally, in order to get a recursive inequality in $\tilde{e}_k$, we can remove the positive $a_{(k+1)m}$ term from the left-hand side of (B.92) and bound the a_{km} term on the right-hand side by $2e_{km}/\mu$ using a standard strong convexity inequality. We get our final contraction inequality (3.31):

$$(2\gamma m - 4mL\gamma^2 C_1 - 4mL\gamma^2\tau C_2)\tilde{e}_{k+1} \leq \big(\frac{2}{\mu} + 4mL\gamma^2 C_1 + 4mL\gamma^2\tau C_2\big)\tilde{e}_k\,.$$

■

B.3.3 Proof of Corollary 23, 24 and 25 (speedup regimes)

A simpler result for SVRG. The standard convergence rate for serial SVRG is given by:

$$\theta := \frac{\frac{1}{\mu\gamma m} + 2L\gamma}{1 - 2L\gamma}\,. \quad \text{(B.93)}$$

If we define a such that $\gamma = a/4L$, we obtain:

$$\theta = \frac{\frac{4\kappa}{am} + \frac{a}{2}}{1 - \frac{a}{2}}\,. \quad \text{(B.94)}$$

Now, since we need $\theta \leq 1$, we see that we require $a \leq 1$. The optimal value of the denominator is then 1 (when $a = 0$), whereas the worst case value is $1/2$ ($a = 1$). We can thus upper bound θ by replacing the denominator with $1/2$, while satisfied that we do not lose more than a factor of 2. This gives us:

$$\theta \leq \frac{8\kappa}{am} + a\,. \quad \text{(B.95)}$$

Enforcing $\theta \leq 1/2$ can be done easily by choosing $a \leq 1/4$ and $m = 32\kappa/a$. Now, to be able to compare algorithms easily, we want to frame our result in terms of rate factor per gradient computation ρ, such that (3.24) is verified:

$$\mathbb{E}f(\tilde{x}_k) - f(x^*) \leq (1-\rho)^{k(2m+n)}\,(\mathbb{E}f(x_0) - f(x^*)) \qquad \forall k \geq 0\,.$$

We define $\rho_b := 1 - \theta$. We want to estimate ρ such that $(1-\rho)^{2m+n} = 1 - \rho_b$. We get that $\rho = 1 - (1-\rho_b)^{\frac{1}{2m+n}}$. Using Bernoulli's inequality, we get:

$$\rho \geq \frac{\rho_b}{2m+n} \geq \frac{1}{2(2m+n)} \geq \frac{1}{4}\min\left\{\frac{1}{2m}, \frac{1}{n}\right\} \geq \frac{1}{4}\min\left\{\frac{a}{64\kappa}, \frac{1}{n}\right\}. \tag{B.96}$$

This finishes the proof for Corollary 23. ■

A simpler result for KROMAGNON. We also define a such that $\gamma = a/4L$. Theorem 22 tells us that:

$$\theta = \frac{\frac{4\kappa}{am} + \frac{a}{2}C_3(1+\frac{\tau}{16\kappa})}{1 - \frac{a}{2}C_3(1+\frac{\tau}{16\kappa})}. \tag{B.97}$$

We can once again upper bound θ by removing its denominator at a reasonable worst-case cost of a factor of 2:

$$\theta \leq \frac{8\kappa}{am} + aC_3(1+\frac{\tau}{16\kappa}). \tag{B.98}$$

Now, to enforce $\theta \leq 1/2$, we can choose $a \leq \frac{1}{4C_3(1+\frac{\tau}{16\kappa})}$ and $m = \frac{32\kappa}{a}$. We also obtain a rate factor per gradient computation of: $\rho \geq \frac{1}{4}\min\{\frac{a}{64\kappa}, \frac{1}{n}\}$. This finishes the proof of Corollary 24. ■

Speedup conditions. All we have to do now is to compare the rate factors of SVRG and KROMAGNON in different regimes. Note that while our convergence result hold for any $a \leq 1/4$ SVRG (or the slightly more complex expression in the case of KROMAGNON), the best step size (in terms of number of gradient computations) ensuring $\theta \leq \frac{1}{2}$ is the biggest allowable one – thus this is the one we use for our comparison.

Suppose we are in the "well-conditioned" regime where $n \geq \kappa$. The rate factor of SVRG is $\Omega(1/n)$. To make sure we have a linear speedup, we need the rate factor of KROMAGNON to also be $\Omega(1/n)$, which means that:

$$\frac{1}{256\kappa C_3 + 16\tau C_3} = \Omega(\frac{1}{n}) \tag{B.99}$$

Recalling that $C_3 = 1 + 2\tau\sqrt{\Delta}$, we can rewrite (B.99) as:

$$\kappa = \mathcal{O}(n); \quad \kappa\tau\sqrt{\Delta} = \mathcal{O}(n); \quad \tau = \mathcal{O}(n); \quad \tau^2\sqrt{\Delta} = \mathcal{O}(n). \tag{B.100}$$

We can condense these conditions down to:

$$\tau = \mathcal{O}(\frac{n}{\kappa\sqrt{\Delta}}); \qquad \tau = \mathcal{O}(\sqrt{n\Delta^{-1/2}}). \tag{B.101}$$

Suppose now we are in the "ill-conditioned" regime, where $\kappa \geq n$. The rate factor of SVRG is now $\Omega(1/\kappa)$. To make sure we have a linear speedup, we need the rate factor of KROMAGNON to also be $\Omega(1/\kappa)$, which means that:

$$\frac{1}{256\kappa C_3 + 16\tau C_3} = \Omega(\frac{1}{\kappa}) \tag{B.102}$$

We can derive the following sufficient conditions:

$$\tau = \mathcal{O}(\frac{1}{\sqrt{\Delta}})\,; \qquad \tau = \mathcal{O}(\kappa)\,. \tag{B.103}$$

Since $\kappa \geq n$, we obtain the conditions of Corollary 25 and thus finish its proof. ■

B.4 On the difficulty of parallel lagged updates

In the implementation presented in Schmidt et al. (2016), the dense part ($\bar{\alpha}$) of the updates is deferred. Instead of writing dense updates, counters c_d are kept for each coordinate of the parameter vector – which represent the last time these variables were updated – as well as the average gradient $\bar{\alpha}$ for each coordinate. Then, whenever a component $[\hat{x}]_d$ is needed (in order to compute a new gradient), we subtract $\gamma(t - c_d)[\bar{\alpha}]_d$ from it and c_d is set to t. It is possible to do this without modifying the algorithm because $[\bar{\alpha}]_d$ only changes when $[\hat{x}]_d$ also does.

In the sequential setting, this results in the same iterations as performing the updates in a dense fashion, since the coordinates are only stale when they are not used. Note that at the end of an execution all counters have to be subtracted at once to get the true final parameter vector (and to bring every c_d counter to the final t).

In the parallel setting, several issues arise:

- two cores might be attempting to correct the lag at the same time. In which case since updates are done as additions and not replacements (which is necessary to ensure that there are no overwrites), the lag might be corrected multiple times, i.e. overly corrected.
- we would have to read and write atomically to each $[\hat{x}_d], c_d, [\bar{\alpha}]_d$ triplet, which is highly impractical.
- we would need to have an explicit global counter, which we do not in ASAGA (our global counter t being used solely for the proof).
- in the dense setting, updates happen coordinate by coordinate. So at time t the

number of $\bar{\alpha}$ updates a coordinate has received from a fixed past time c_d is a random variable, which may differs from coordinate to coordinate. Whereas in the lagged implementation, the multiplier is always $(t - c_d)$ which is a constant (conditional to c_d), which means a potentially different $\hat{x}_t$.

- the trick used in Reddi et al. (2015) for asynchronous parallel SVRG does not apply here because it relies on the fact that the "reference" gradient term in SVRG is constant throughout a whole epoch, which is not the case for SAGA.

All these points mean both that the implementation of such a scheme in the parallel setting would be impractical, and that it would actually yields a different algorithm than the dense version, which would be even harder to analyze. This is confirmed by Pan et al. (2016), where the authors tried to implement a parallel version of the lagged updates scheme and had to alter the algorithm to succeed, obtaining an algorithm with suboptimal performance as a result.

B.5 Additional empirical details

B.5.1 Detailed description of datasets

We run our experiments on four datasets. In every case, we run logistic regression for the purpose of binary classification.

RCV1 ($n = 697,641$, $d = 47,236$). The first is the Reuters Corpus Volume I (RCV1) dataset (Lewis et al., 2004), an archive of over 800,000 manually categorized newswire stories made available by Reuters, Ltd. for research purposes. The associated task is a binary text categorization.

URL ($n = 2,396,130$, $d = 3,231,961$). Our second dataset was first introduced in Ma et al. (2009). Its associated task is a binary malicious URL detection. This dataset contains more than 2 million URLs obtained at random from Yahoo's directory listing (for the "benign" URLs) and from a large Web mail provider (for the "malicious" URLs). The benign to malicious ratio is 2. Features include lexical information as well as metadata. This dataset was obtained from the libsvmtools project. [12]

12. http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html

Covertype ($n = 581,012$**,** $d = 54$**).** On our third dataset, the associated task is a binary classification problem (down from 7 classes originally, following the pre-treatment of Collobert et al. (2002)). The features are cartographic variables. Contrarily to the first two, this is a dense dataset.

Realsim ($n = 73,218$**,** $d = 20,958$**).** We only use our fourth dataset for non-parallel experiments and a specific compare-and-swap test. It constitutes of UseNet articles taken from four discussion groups (simulated auto racing, simulated aviation, real autos, real aviation).

B.5.2 Implementation details

Hardware. All experiments were run on a Dell PowerEdge 920 machine with 4 Intel Xeon E7-4830v2 processors with 10 2.2GHz cores each and 384GB 1600 MHz RAM.

Software. All algorithms were implemented in the Scala language and the software stack consisted of a Linux operating system running Scala 2.11.7 and Java 1.6.

We chose this expressive, high level language for our experimentation despite its typical 20x slower performance compared to C because our primary concern was that the code may easily be reused and extended for research purposes (which is harder to achieve with low level, heavily optimized C code; especially for error prone parallel computing).

As a result our timed experiments exhibit sub-optimal running times, e.g. compared to Konečný and Richtárik (2013). This is as we expected. The observed slowdown is both consistent across datasets (roughly 20x) and with other papers that use Scala code (e.g. Mania et al. (2017), Ma et al. (2015, Fig. 2)).

Despite this slowdown, our experiments show state-of-the-art results in convergence per number of iterations. Furthermore, the speed-up patterns that we observe for our implementation of Hogwild and Kromagnon are similar to the ones given in Mania et al. (2017); Niu et al. (2011); Reddi et al. (2015) (in various languages).

The code we used to run all the experiments is available at `http://www.di.ens.fr/sierra/research/asaga/`.

Necessity of compare-and-swap operations. Interestingly, we have found necessary to use compare-and-swap instructions in the implementation of ASAGA. In Figure B-1, we display suboptimality plots using non-thread safe operations and compare-and-swap (CAS) operations. The non-thread safe version starts faster but then fails to converge beyond a

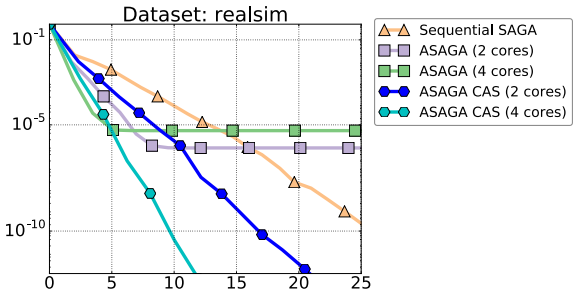


Figure B-1 – **Compare and swap in the implementation of ASAGA.** Suboptimality as a function of time for ASAGA, both using compare-and-swap (CAS) operations and using standard operations. The graph reveals that CAS is indeed needed in a practical implementation to ensure convergence to a high precision.

specific level of suboptimality, while the compare-and-swap version does converges linearly up to machine precision.

For *compare-and-swap* instructions we used the `AtomicDoubleArray` class from the Google library `Guava`. This class uses an `AtomicLongArray` under the hood (from package `java.util.concurrent.atomic` in the standard Java library), which does indeed benefit from lower-level CPU-optimized instructions.

Efficient storage of the α_i. Storing n gradient may seem like an expensive proposition, but for linear predictor models, one can actually store a single scalar per gradient (as proposed in Schmidt et al. (2016)), which is what we do in our implementation of ASAGA.

B.5.3 Biased update in the implementation

In the implementation detailed in Algorithm 4, $\bar{\alpha}$ is maintained in memory instead of being recomputed for every iteration. This saves both the cost of reading every data point for each iteration and of computing $\bar{\alpha}$ for each iteration.

However, this removes the unbiasedness guarantee. The problem here is the definition of the expectation of $\hat{\alpha}_i$. Since we are sampling uniformly at random, the average of the $\hat{\alpha}_i$ is taken at the precise moment when we read the α_i^t components. Without synchronization, between two reads to a single coordinate in α_i and in $\bar{\alpha}$, new updates might arrive in $\bar{\alpha}$ that are not yet taken into account in α_i. Conversely, writes to a component of α_i might precede the corresponding write in $\bar{\alpha}$ and induce another source of bias.

In order to alleviate this issue, we can use coordinate-level locks on α_i and $\bar{\alpha}$ to make sure they are always synchronized. Such low-level locks are quite inexpensive when d is large, especially when compared to vector-wide locks.

However, as previously noted, experimental results indicate that this fix is not necessary.

Appendix C

Extension to non-smooth objectives

Notations. Throughout the supplementary material we use the following extra notation. We denote by $\langle \cdot, \cdot \rangle_{(i)}$ (resp. $\| \cdot \|_{(i)}$) the scalar product (resp. norm) restricted to blocks in T_i, i.e., $\langle \boldsymbol{x}, \boldsymbol{y} \rangle_{(i)} := \sum_{B \in T_i} \langle [\boldsymbol{x}]_B, [\boldsymbol{y}]_B \rangle$ and $\|\boldsymbol{x}\|_{(i)} := \sqrt{\langle \boldsymbol{x}, \boldsymbol{x} \rangle_{(i)}}$. We will also use the following definitions: $\varphi := \sum_{B \in \mathcal{B}} d_B h_B(\boldsymbol{x})$ and $\boldsymbol{D}$ is the diagonal matrix defined block-wise as $[\boldsymbol{D}]_{B,B} = d_B \boldsymbol{I}_{|B|}$.

The **Bregman divergence** associated with a convex function f for points $\boldsymbol{x}, \boldsymbol{y}$ in its domain is defined as:

$$B_f(\boldsymbol{x}, \boldsymbol{y}) := f(\boldsymbol{x}) - f(\boldsymbol{y}) - \langle \nabla f(\boldsymbol{y}), \boldsymbol{x} - \boldsymbol{y} \rangle \,. \tag{C.1}$$

Note that this is always positive due to the convexity of f.

C.1 Basic properties

Lemma 31. *For any μ-strongly convex function f we have the following inequality:*

$$\langle \nabla f(\boldsymbol{y}) - \nabla f(\boldsymbol{x}), \boldsymbol{y} - \boldsymbol{x} \rangle \geq \frac{\mu}{2} \|\boldsymbol{y} - \boldsymbol{x}\|^2 + B_f(\boldsymbol{x}, \boldsymbol{y}) \,. \tag{C.2}$$

Proof. By strong convexity, f verifies the inequality:

$$f(\boldsymbol{y}) \leq f(\boldsymbol{x}) + \langle \nabla f(\boldsymbol{y}), \boldsymbol{y} - \boldsymbol{x} \rangle - \frac{\mu}{2} \|\boldsymbol{y} - \boldsymbol{x}\|^2 \,, \tag{C.3}$$

for any $\boldsymbol{x}, \boldsymbol{y}$ in the domain (see e.g. (Nesterov, 2004)). We then have the equivalences:

$$\begin{aligned}
& f(\boldsymbol{x}) \leq f(\boldsymbol{y}) + \langle \nabla f(\boldsymbol{x}), \boldsymbol{x} - \boldsymbol{y} \rangle - \frac{\mu}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2 \\
\iff & \frac{\mu}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2 + f(\boldsymbol{x}) - f(\boldsymbol{y}) \leq \langle \nabla f(\boldsymbol{x}), \boldsymbol{x} - \boldsymbol{y} \rangle \\
\iff & \frac{\mu}{2}\|\boldsymbol{x} - \boldsymbol{y}\|^2 + \underbrace{f(\boldsymbol{x}) - f(\boldsymbol{y}) - \langle \nabla f(\boldsymbol{y}), \boldsymbol{x} - \boldsymbol{y} \rangle}_{B_f(\boldsymbol{x}, \boldsymbol{y})} \leq \langle \nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y}), \boldsymbol{x} - \boldsymbol{y} \rangle \,,
\end{aligned}$$

where in the last line we have subtracted $\langle \nabla f(\boldsymbol{y}), \boldsymbol{x} - \boldsymbol{y} \rangle$ from both sides of the inequality. □

Lemma 32. *Let the f_i be L-smooth and convex functions. Then it is verified that:*

$$\frac{1}{n}\sum_{i=1}^{n} \|\nabla f_i(\boldsymbol{x}) - \nabla f_i(\boldsymbol{y})\|^2 \leq 2L B_f(\boldsymbol{x}, \boldsymbol{y}) \,. \tag{C.4}$$

Proof. Since each f_i is L-smooth, it is verified (see e.g. Nesterov (2004, Theorem 2.1.5)) that

$$\|\nabla f_i(\boldsymbol{x}) - \nabla f_i(\boldsymbol{y})\|^2 \leq 2L\big(f_i(\boldsymbol{x}) - f_i(\boldsymbol{y}) - \langle \nabla f_i(\boldsymbol{y}), \boldsymbol{x} - \boldsymbol{y} \rangle\big) \,. \tag{C.5}$$

The result is obtained by averaging over i. □

Lemma 33 (Characterization of the proximal operator)**.** *Let h be convex lower semicontinuous. Then we have the following characterization of the proximal operator:*

$$\boldsymbol{z} = \mathbf{prox}_{\gamma h}(\boldsymbol{x}) \iff \frac{1}{\gamma}(\boldsymbol{x} - \boldsymbol{z}) \in \partial h(\boldsymbol{z}) \,. \tag{C.6}$$

Proof. This is a direct consequence of the first order optimality conditions on the definition of proximal operator, see e.g. (Beck and Teboulle, 2009; Nesterov, 2013). □

Lemma 34 (Firm non-expansiveness)**.** *Let $\boldsymbol{x}, \tilde{\boldsymbol{x}}$ be two arbitrary elements in the domain of φ_i and $\boldsymbol{z}, \tilde{\boldsymbol{z}}$ be defined as $\boldsymbol{z} := \mathrm{prox}_{\varphi_i}(\boldsymbol{x})$, $\tilde{\boldsymbol{z}} := \mathrm{prox}_{\varphi_i}(\tilde{\boldsymbol{x}})$. Then it is verified that:*

$$\langle \boldsymbol{z} - \tilde{\boldsymbol{z}}, \boldsymbol{x} - \tilde{\boldsymbol{x}} \rangle_{(i)} \geq \|\boldsymbol{z} - \tilde{\boldsymbol{z}}\|_{(i)}^2 \,. \tag{C.7}$$

Proof. By the block-separability of φ_i, the proximal operator is the concatenation of the proximal operators of the blocks. In other words, for any block $B \in T_i$ we have:

$$[\boldsymbol{z}]_B = \mathbf{prox}_{\gamma\varphi_B}([\boldsymbol{x}]_B)\,, \quad [\tilde{\boldsymbol{z}}]_B = \mathbf{prox}_{\gamma\varphi_B}([\tilde{\boldsymbol{x}}]_B)\,, \tag{C.8}$$

where φ_B is the restriction of φ_i to B. By firm non-expansiveness of the proximal operator (see e.g. Bauschke and Combettes (2011, Proposition 4.2)) we have that:

$$\langle [\boldsymbol{z}]_B - [\tilde{\boldsymbol{z}}]_B, [\boldsymbol{x}]_B - [\tilde{\boldsymbol{x}}]_B \rangle \geq \|[\boldsymbol{z}]_B - [\tilde{\boldsymbol{z}}]_B\|^2\,.$$

Summing over the blocks in T_i yields the desired result. □

C.2 Sparse Proximal SAGA

This Appendix contains all proofs for Section 4.2. The main result of this section is Theorem 27, whose proof is structured as follows:

- We start by proving four auxiliary results that will be used later on in the proofs of both synchronous and asynchronous variants. The first is the unbiasedness of key quantities used in the algorithm. The second is a characterization of the solutions of (4.1) in terms of f and φ (defined below) in Lemma 36. The third is a key inequality in Lemma 37 that relates the gradient mapping to other terms that arise in the optimization. The fourth is an upper bound on the variance terms of the gradient estimator, relating it to the Bregman divergence of f and the past gradient estimator terms.
- In Lemma 39, we define an upper bound on the iterates $\|\boldsymbol{x}_t - \boldsymbol{x}^*\|^2$, called a Lyapunov function, and prove an inequality that relates this Lyapunov function value at the current iterate with its value at the previous iterate.
- Finally, in the proof of Theorem 27 we use the previous inequality in terms of the Lyapunov function to prove a geometric convergence of the iterates.

We start by proving the following unbiasedness result, mentioned in Section 4.2.

Lemma 35. *Let $\boldsymbol{D}_i$ and φ_i be defined as in Section 4.2. Then it is verified that $\mathbf{E}\boldsymbol{D}_i = \boldsymbol{I}_d$ and $\mathbf{E}\,\varphi_i = h$.*

Proof. Let $B \in \mathcal{B}$ an arbitrary block. We have the following sequence of equalities:

$$\mathbf{E}[\boldsymbol{D}_i]_{B,B} = \frac{1}{n}\sum_{i=1}^{n}[\boldsymbol{D}_i]_{B,B} = \frac{1}{n}\sum_{i=1}^{n} d_B \mathbb{1}\{B \in T_i\}\boldsymbol{I}_{|B|} \tag{C.9}$$

$$= \frac{1}{n}\sum_{i=1}^{n}\frac{n}{n_B}\mathbb{1}\{B \in T_i\}\boldsymbol{I}_{|B|} \tag{C.10}$$

$$= \left(\frac{1}{n_B}\sum_{i=1}^{n}\mathbb{1}\{B \in T_i\}\right)\boldsymbol{I}_{|B|} = \boldsymbol{I}_{|B|}\,, \tag{C.11}$$

where the last equality comes from the definition of n_B. $\mathbf{E}\boldsymbol{D}_i = \boldsymbol{I}_d$ then follows from the arbitrariness of B.

Similarly, for φ_i we have:

$$\mathbf{E}\varphi_i([\boldsymbol{x}]_B) = \frac{1}{n}\sum_{i=1}^{n} d_B \mathbb{1}\{B \in T_i\} h_B([\boldsymbol{x}]_B) \tag{C.12}$$

$$= \frac{1}{n}\sum_{i=1}^{n} \frac{n}{n_B} \mathbb{1}\{B \in T_i\} h_B([\boldsymbol{x}]_B) \tag{C.13}$$

$$= \left(\frac{1}{n_B}\sum_{i=1}^{n} \mathbb{1}\{B \in T_i\}\right) h_B([\boldsymbol{x}]_B) = h_B([\boldsymbol{x}]_B)\,, \tag{C.14}$$

Finally, the result $\mathbf{E}\,\varphi_i = h$ comes from adding over all blocks. □

Lemma 36. *$\boldsymbol{x}^*$ is a solution to (4.1) if and only if the following condition is verified:*

$$\boldsymbol{x}^* = \mathbf{prox}_{\gamma\varphi}\big(\boldsymbol{x}^* - \gamma \boldsymbol{D}\nabla f(\boldsymbol{x}^*)\big)\,. \tag{C.15}$$

Proof. By the first order optimality conditions, the solutions to (4.1) are characterized by the subdifferential inclusion $-\nabla f(\boldsymbol{x}^*) \in \partial h(\boldsymbol{x}^*)$. We can then write the following sequence of equivalences:

$$\begin{aligned}
-\nabla f(\boldsymbol{x}^*) \in \partial h(\boldsymbol{x}^*) \iff & -\boldsymbol{D}\nabla f(\boldsymbol{x}^*) \in \boldsymbol{D}\partial h(\boldsymbol{x}^*) \\
& \text{(multiplying by } \boldsymbol{D}\text{, equivalence since diagonals are nonzero)} \\
\iff & -\boldsymbol{D}\nabla f(\boldsymbol{x}^*) \in \partial\varphi(\boldsymbol{x}^*) \qquad \text{(by definition of } \varphi\text{)} \\
\iff & \tfrac{1}{\gamma}(\boldsymbol{x}^* - \gamma\boldsymbol{D}\nabla f(\boldsymbol{x}^*) - \boldsymbol{x}^*) \in \partial\varphi(\boldsymbol{x}^*) \\
& \text{(adding and subtracting } \boldsymbol{x}^*\text{)} \\
\iff & \boldsymbol{x}^* = \mathbf{prox}_{\gamma\varphi}(\boldsymbol{x}^* - \gamma\boldsymbol{D}\nabla f(\boldsymbol{x}^*))\,. \qquad \text{(by Lemma 33)}
\end{aligned}$$

Since all steps are equivalences, we have the desired result. □

The following lemma will be key in the proof of convergence for both the sequential and the parallel versions of the algorithm. With this result, we will be able to bound the product between the gradient mapping and the iterate suboptimality by:

- First, the negative norm of the gradient mapping, which will be key in the parallel setting to cancel out the terms arising from the asynchrony.
- Second, variance terms in $\|\boldsymbol{v}_i - \boldsymbol{D}_i\nabla f(\boldsymbol{x}^*)\|^2$ that we will be able to bound by the Bregman divergence using Lemma 32.

- Third and last, a product with terms in $\langle \boldsymbol{v}_i - \boldsymbol{D}_i \nabla f(\boldsymbol{x}^*), \boldsymbol{x} - \boldsymbol{x}^* \rangle$, which taken in expectation gives $\langle \nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{x}^*), \boldsymbol{x} - \boldsymbol{x}^* \rangle$ and will allow us to apply Lemma 31 to obtain the contraction terms needed to obtain a geometric rate of convergence.

Lemma 37 (Gradient mapping inequality)**.** *Let $\boldsymbol{x}$ be an arbitrary vector, $\boldsymbol{x}^*$ a solution to (4.1), $\boldsymbol{v}_i$ as defined in (4.3) and $\boldsymbol{g} := \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{v}_i, i)$ the gradient mapping defined in (4.6). Then the following inequality is verified for any $\beta > 0$:*

$$\langle \boldsymbol{g}, \boldsymbol{x} - \boldsymbol{x}^* \rangle \geq -\frac{\gamma}{2}(\beta - 2)\|\boldsymbol{g}\|^2 - \frac{\gamma}{2\beta}\|\boldsymbol{v}_i - \boldsymbol{D}_i \nabla f(\boldsymbol{x}^*)\|^2 + \langle \boldsymbol{v}_i - \boldsymbol{D}_i \nabla f(\boldsymbol{x}^*), \boldsymbol{x} - \boldsymbol{x}^* \rangle \,. \quad \text{(C.16)}$$

Proof. By firm non-expansiveness of the proximal operator (Lemma 34) applied to $\boldsymbol{z} = \mathbf{prox}_{\gamma\varphi_i}(\boldsymbol{x} - \gamma \boldsymbol{v}_i)$ and $\tilde{\boldsymbol{z}} = \mathbf{prox}_{\gamma\varphi_i}(\boldsymbol{x}^* - \gamma \boldsymbol{D} \nabla f(\boldsymbol{x}^*))$ we have:

$$\|\boldsymbol{z} - \tilde{\boldsymbol{z}}\|_{(i)}^2 - \langle \boldsymbol{z} - \tilde{\boldsymbol{z}}, \boldsymbol{x} - \gamma \boldsymbol{v}_i - \boldsymbol{x}^* + \gamma \boldsymbol{D} \nabla f(\boldsymbol{x}^*) \rangle_{(i)} \leq 0 \,. \quad \text{(C.17)}$$

By the (4.3) iteration we have $\boldsymbol{x}^+ = \boldsymbol{z}$ and by Lemma 33 we have that $[\boldsymbol{z}]_{T_i} = [\boldsymbol{x}^*]_{T_i}$, hence the above can be rewritten as

$$\|\boldsymbol{x}^+ - \boldsymbol{x}^*\|_{(i)}^2 - \langle \boldsymbol{x}^+ - \boldsymbol{x}^*, \boldsymbol{x} - \gamma \boldsymbol{v}_i - \boldsymbol{x}^* + \gamma \boldsymbol{D} \nabla f(\boldsymbol{x}^*) \rangle_{(i)} \leq 0 \,. \quad \text{(C.18)}$$

We can now write the following sequence of inequalities

$$
\begin{aligned}
\langle \gamma \boldsymbol{g}, \boldsymbol{x} - \boldsymbol{x}^* \rangle &= \langle \boldsymbol{x} - \boldsymbol{x}^+, \boldsymbol{x} - \boldsymbol{x}^* \rangle_{(i)} && \text{((by definition and sparsity of } g\text{))} \\
&= \langle \boldsymbol{x} - \boldsymbol{x}^+ + \boldsymbol{x}^* - \boldsymbol{x}^*, \boldsymbol{x} - \boldsymbol{x}^* \rangle_{(i)} \\
&= \|\boldsymbol{x} - \boldsymbol{x}^*\|_{(i)}^2 - \langle \boldsymbol{x}^+ - \boldsymbol{x}^*, \boldsymbol{x} - \boldsymbol{x}^* \rangle_{(i)} \\
&\geq \|\boldsymbol{x} - \boldsymbol{x}^*\|_{(i)}^2 - \langle \boldsymbol{x}^+ - \boldsymbol{x}^*, 2\boldsymbol{x} - \gamma \boldsymbol{v}_i - 2\boldsymbol{x}^* + \gamma \boldsymbol{D} \nabla f(\boldsymbol{x}^*) \rangle_{(i)} + \|\boldsymbol{x}^+ - \boldsymbol{x}^*\|_{(i)}^2 \\
& && \text{(adding Eq. (C.18))} \\
&= \|\boldsymbol{x} - \boldsymbol{x}^+\|_{(i)}^2 + \langle \boldsymbol{x}^+ - \boldsymbol{x}^*, \gamma \boldsymbol{v}_i - \gamma \boldsymbol{D} \nabla f(\boldsymbol{x}^*) \rangle_{(i)} && \text{(completing the square)} \\
&= \|\boldsymbol{x} - \boldsymbol{x}^+\|_{(i)}^2 + \langle \boldsymbol{x} - \boldsymbol{x}^*, \gamma \boldsymbol{v}_i - \gamma \boldsymbol{D} \nabla f(\boldsymbol{x}^*) \rangle_{(i)} - \langle \boldsymbol{x} - \boldsymbol{x}^+, \gamma \boldsymbol{v}_i - \gamma \boldsymbol{D} \nabla f(\boldsymbol{x}^*) \rangle_{(i)} \\
& && \text{(adding and substracting } \boldsymbol{x}\text{))} \\
&\geq \left(1 - \frac{\beta}{2}\right) \|\boldsymbol{x} - \boldsymbol{x}^+\|_{(i)}^2 - \frac{\gamma^2}{2\beta} \|\boldsymbol{v}_i - \boldsymbol{D} \nabla f(\boldsymbol{x}^*)\|_{(i)}^2 + \gamma \langle \boldsymbol{v}_i - \boldsymbol{D} \nabla f(\boldsymbol{x}^*), \boldsymbol{x} - \boldsymbol{x}^* \rangle_{(i)} \\
& && \text{(Young's inequality } 2\langle a, b \rangle \leq \tfrac{\|a\|^2}{\beta} + \beta \|b\|^2\text{, valid for arbitrary } \beta > 0\text{)} \\
&\geq \left(1 - \frac{\beta}{2}\right) \|\boldsymbol{x} - \boldsymbol{x}^+\|_{(i)}^2 - \frac{\gamma^2}{2\beta} \|\boldsymbol{v}_i - \boldsymbol{D}_i \nabla f(\boldsymbol{x}^*)\|^2 + \gamma \langle \boldsymbol{v}_i - \boldsymbol{D}_i \nabla f(\boldsymbol{x}^*), \boldsymbol{x} - \boldsymbol{x}^* \rangle \\
& && \text{(by definition of } \boldsymbol{D}_i \text{ and using the fact that } \boldsymbol{v}_i \text{ is } T_i\text{-sparse)} \\
&= \left(1 - \frac{\beta}{2}\right) \|\gamma \boldsymbol{g}\|^2 - \frac{\gamma^2}{2\beta} \|\boldsymbol{v}_i - \boldsymbol{D}_i \nabla f(\boldsymbol{x}^*)\|^2 + \gamma \langle \boldsymbol{v}_i - \boldsymbol{D} \nabla f(\boldsymbol{x}^*), \boldsymbol{x} - \boldsymbol{x}^* \rangle,
\end{aligned}
$$

where in the last inequality we have used the fact that $\boldsymbol{g}$ is T_i-sparse. Finally, dividing by γ both sides yields the desired result. □

Lemma 38 (Upper bound on the gradient estimator variance)**.** *For arbitrary vectors* $\boldsymbol{x}$*,* $(\boldsymbol{\alpha}_i)_{i=0}^n$*, and* $\boldsymbol{v}_i$ *as defined in* (4.3) *we have:*

$$
\mathbf{E} \|\boldsymbol{v}_i - \boldsymbol{D}_i \nabla f(\boldsymbol{x}^*)\|^2 \leq 4LB_f(\boldsymbol{x}, \boldsymbol{x}^*) + 2\mathbf{E} \|\boldsymbol{\alpha}_i - \nabla f_i(\boldsymbol{x}^*)\|^2 . \tag{C.19}
$$

Proof. We will now bound the variance terms. For this we have:

$$
\begin{aligned}
\mathbf{E} \|\boldsymbol{v}_i - \boldsymbol{D} \nabla f(\boldsymbol{x}^*)\|_{(i)}^2 &= \mathbf{E} \|\nabla f_i(\boldsymbol{x}) - \nabla f_i(\boldsymbol{x}^*) + \nabla f_i(\boldsymbol{x}^*) - \boldsymbol{\alpha}_i + \boldsymbol{D}_i \overline{\boldsymbol{\alpha}} - \boldsymbol{D} \nabla f(\boldsymbol{x}^*)\|_{(i)}^2 \\
&\leq 2\mathbf{E} \|\nabla f_i(\boldsymbol{x}) - \nabla f_i(\boldsymbol{x}^*)\|^2 + 2\mathbf{E} \|\nabla f_i(\boldsymbol{x}^*) - \boldsymbol{\alpha}_i - (\boldsymbol{D} \nabla f(\boldsymbol{x}^*) - \boldsymbol{D} \overline{\boldsymbol{\alpha}})\|_{(i)}^2 \\
& && \text{(by inequality } \|a + b\|^2 \leq 2\|a\|^2 + 2\|b\|^2\text{)} \\
&= 2\mathbf{E} \|\nabla f_i(\boldsymbol{x}) - \nabla f_i(\boldsymbol{x}^*)\|^2 + 2\mathbf{E} \|\nabla f_i(\boldsymbol{x}^*) - \boldsymbol{\alpha}_i\|^2 + 2\mathbf{E} \|\boldsymbol{D} \nabla f(\boldsymbol{x}^*) - \boldsymbol{D} \overline{\boldsymbol{\alpha}}\|_{(i)}^2 \\
&\quad - 4\mathbf{E} \langle \nabla f_i(\boldsymbol{x}^*) - \boldsymbol{\alpha}_i, \boldsymbol{D} \nabla f(\boldsymbol{x}^*) - \boldsymbol{D} \overline{\boldsymbol{\alpha}} \rangle_{(i)} . && \text{(developing the square)}
\end{aligned}
$$

We will now simplify the last two terms in the above expression. For the first of the two last

terms we have:

$$\begin{aligned}-4\mathbf{E}\langle \nabla f_i(\boldsymbol{x}^*) - \boldsymbol{\alpha}_i, \boldsymbol{D}\nabla f(\boldsymbol{x}^*) - \boldsymbol{D}\overline{\boldsymbol{\alpha}}\rangle_{(i)} &= -4\mathbf{E}\langle \nabla f_i(\boldsymbol{x}^*) - \boldsymbol{\alpha}_i, \boldsymbol{D}\nabla f(\boldsymbol{x}^*) - \boldsymbol{D}\overline{\boldsymbol{\alpha}}\rangle \\ &\qquad \text{(support of first term)} \\ &= -4\langle \nabla f(\boldsymbol{x}^*) - \overline{\boldsymbol{\alpha}}, \boldsymbol{D}\nabla f(\boldsymbol{x}^*) - \boldsymbol{D}\overline{\boldsymbol{\alpha}}\rangle \\ &= -4\|\nabla f(\boldsymbol{x}^*) - \overline{\boldsymbol{\alpha}}\|_{\boldsymbol{D}}^2 . \qquad \text{(C.20)}\end{aligned}$$

Similarly, for the last term we have:

$$\begin{aligned}2\mathbf{E}\|\boldsymbol{D}\nabla f(\boldsymbol{x}^*) - \boldsymbol{D}\overline{\boldsymbol{\alpha}}\|_{(i)}^2 &= 2\mathbf{E}\langle \boldsymbol{D}_i\nabla f(\boldsymbol{x}^*) - \boldsymbol{D}_i\overline{\boldsymbol{\alpha}}, \boldsymbol{D}\nabla f(\boldsymbol{x}^*) - \boldsymbol{D}\overline{\boldsymbol{\alpha}}\rangle \\ &= 2\langle \nabla f(\boldsymbol{x}^*) - \overline{\boldsymbol{\alpha}}, \boldsymbol{D}\nabla f(\boldsymbol{x}^*) - \boldsymbol{D}\overline{\boldsymbol{\alpha}}\rangle \qquad \text{(using Lemma 35)} \\ &= 2\|\nabla f(\boldsymbol{x}^*) - \overline{\boldsymbol{\alpha}}\|_{\boldsymbol{D}}^2 , \qquad \text{(C.21)}\end{aligned}$$

and so the addition of these terms is negative and can be dropped. In all, for the variance terms we have

$$\begin{aligned}\mathbf{E}\|\boldsymbol{v}_i - \boldsymbol{D}\nabla f(\boldsymbol{x}^*)\|_{(i)}^2 &\leq 2\mathbf{E}\|\nabla f_i(\boldsymbol{x}) - \nabla f_i(\boldsymbol{x}^*)\|^2 + 2\mathbf{E}\|\boldsymbol{\alpha}_i - \nabla f_i(\boldsymbol{x}^*)\|^2 \\ &\leq 4LB_f(\boldsymbol{x}, \boldsymbol{x}^*) + 2\mathbf{E}\|\boldsymbol{\alpha}_i - \nabla f_i(\boldsymbol{x}^*)\|^2 . \qquad \text{(by Lemma 32)}\end{aligned}$$

□

We now define an upper bound on the quantity that we would like to bound, often called a Lyapunov function, and establish a recursive inequality on this Lyapunov function.

Lemma 39 (Lyapunov inequality)**.** *Let $\mathcal{L}$ be the following c-parametrized function:*

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\alpha}) := \|\boldsymbol{x} - \boldsymbol{x}^*\|^2 + \frac{c}{n}\sum_{i=1}^n \|\boldsymbol{\alpha}_i - \nabla f_i(\boldsymbol{x}^*)\|^2 . \qquad \text{(C.22)}$$

Let $\boldsymbol{x}^+$ and $\boldsymbol{\alpha}^+$ be obtained from the Sparse Proximal SAGA *updates (4.3). Then we have:*

$$\begin{aligned}\mathbf{E}\mathcal{L}(\boldsymbol{x}^+, \boldsymbol{\alpha}^+) - \mathcal{L}(\boldsymbol{x}, \boldsymbol{\alpha}) \leq &-\gamma\mu\|\boldsymbol{x} - \boldsymbol{x}^*\|^2 + \left(4L\gamma^2 - 2\gamma + 2L\frac{c}{n}\right) B_f(\boldsymbol{x}, \boldsymbol{x}^*) \\ &+ \left(2\gamma^2 - \frac{c}{n}\right)\mathbf{E}\|\boldsymbol{\alpha}_i - \nabla f_i(\boldsymbol{x})\|^2 . \qquad \text{(C.23)}\end{aligned}$$

Proof. For the first term of $\mathcal{L}$ we have:

$$\begin{aligned}\|\boldsymbol{x}^+ - \boldsymbol{x}^*\|^2 &= \|\boldsymbol{x} - \gamma\boldsymbol{g} - \boldsymbol{x}^*\|^2 \qquad (\boldsymbol{g} := \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{v}_i, i)) \\ &= \|\boldsymbol{x} - \boldsymbol{x}^*\|^2 - 2\gamma\langle \boldsymbol{g}, \boldsymbol{x} - \boldsymbol{x}^*\rangle + \|\gamma\boldsymbol{g}\|^2 \\ &\leq \|\boldsymbol{x} - \boldsymbol{x}^*\|^2 + \gamma^2\|\boldsymbol{v}_i - \boldsymbol{D}_i\nabla f(\boldsymbol{x}^*)\|^2 - 2\gamma\langle \boldsymbol{v}_i - \boldsymbol{D}_i\nabla f(\boldsymbol{x}^*), \boldsymbol{x} - \boldsymbol{x}^*\rangle \\ &\qquad\qquad \text{(by Lemma 37 with } \beta = 1)\end{aligned}$$

Since $\boldsymbol{v}_i$ is an unbiased estimator of the gradient and $\mathbf{E}\boldsymbol{D}_i = \boldsymbol{I}_d$, taking expectations and using Lemma 31 we have:

$$\begin{aligned}\mathbf{E}\|\boldsymbol{x}^+ - \boldsymbol{x}^*\|^2 &\leq \|\boldsymbol{x} - \boldsymbol{x}^*\|^2 + \gamma^2\mathbf{E}\|\boldsymbol{v}_i - \boldsymbol{D}_i\nabla f(\boldsymbol{x}^*)\|^2 - 2\gamma\langle \nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{x}^*), \boldsymbol{x} - \boldsymbol{x}^*\rangle \\ &\leq (1 - \gamma\mu)\|\boldsymbol{x} - \boldsymbol{x}^*\|^2 + \gamma^2\mathbf{E}\|\boldsymbol{v}_i - \boldsymbol{D}_i\nabla f(\boldsymbol{x}^*)\|^2 \\ &\qquad - 2\gamma B_f(\boldsymbol{x}, \boldsymbol{x}^*)\,. \qquad \text{(C.24)}\end{aligned}$$

By using the variance terms bound (Lemma 38) in the previous equation we have:

$$\begin{aligned}\mathbf{E}\|\boldsymbol{x}^+ - \boldsymbol{x}^*\|^2 &\leq (1 - \gamma\mu)\|\boldsymbol{x} - \boldsymbol{x}^*\|^2 + (4L\gamma^2 - 2\gamma)B_f(\boldsymbol{x}, \boldsymbol{x}^*) \\ &\qquad + 2\gamma^2\mathbf{E}\|\boldsymbol{\alpha}_i - \nabla f_i(\boldsymbol{x}^*)\|^2\,. \qquad \text{(C.25)}\end{aligned}$$

We will now bound the second term of the Lyapunov function. Using the definition of $\boldsymbol{\alpha}^+$ and Lemma 32 we obtain:

$$\begin{aligned}\frac{1}{n}\sum_{i=1}^n \|\boldsymbol{\alpha}_i^+ - \nabla f_i(\boldsymbol{x}^*)\|^2 &= \left(1 - \frac{1}{n}\right)\mathbf{E}\|\boldsymbol{\alpha}_i - \nabla f_i(\boldsymbol{x}^*)\|^2 + \frac{1}{n}\mathbf{E}\|\nabla f_i(\boldsymbol{x}) - \nabla f_i(\boldsymbol{x}^*)\|^2 \\ &\leq \left(1 - \frac{1}{n}\right)\mathbf{E}\|\boldsymbol{\alpha}_i - \nabla f_i(\boldsymbol{x}^*)\|^2 + \frac{2}{n}LB_f(\boldsymbol{x}, \boldsymbol{x}^*)\,. \qquad \text{(C.26)}\end{aligned}$$

Combining Eq. (C.25) and (C.26) we have:

$$\begin{aligned}\mathbf{E}\mathcal{L}(\boldsymbol{x}^+, \boldsymbol{\alpha}^+) &\leq (1 - \gamma\mu)\|\boldsymbol{x} - \boldsymbol{x}^*\|^2 + (4L\gamma^2 - 2\gamma)B_f(\boldsymbol{x}, \boldsymbol{x}^*) + 2\gamma^2\mathbf{E}\|\boldsymbol{\alpha}_i - \nabla f_i(\boldsymbol{x}^*)\|^2 \\ &\qquad + c\left[\left(1 - \frac{1}{n}\right)\mathbf{E}\|\boldsymbol{\alpha}_i - \nabla f_i(\boldsymbol{x}^*)\|^2 + \frac{1}{n}2LB_f(\boldsymbol{x}, \boldsymbol{x}^*)\right] \\ &= (1 - \gamma\mu)\|\boldsymbol{x} - \boldsymbol{x}^*\|^2 + \left(4L\gamma^2 - 2\gamma + 2L\frac{c}{n}\right)B_f(\boldsymbol{x}, \boldsymbol{x}^*) \\ &\qquad + \left(2\gamma^2 - \frac{c}{n}\right)\mathbf{E}\|\boldsymbol{\alpha}_i - \nabla f_i(\boldsymbol{x}^*)\|^2 + c\mathbf{E}\|\boldsymbol{\alpha}_i - \nabla f_i(\boldsymbol{x}^*)\|^2 \\ &= \mathcal{L}(\boldsymbol{x}, \boldsymbol{\alpha}) - \gamma\mu\|\boldsymbol{x} - \boldsymbol{x}^*\|^2 + \left(4L\gamma^2 - 2\gamma + 2L\frac{c}{n}\right)B_f(\boldsymbol{x}, \boldsymbol{x}^*) \\ &\qquad + \left(2\gamma^2 - \frac{c}{n}\right)\mathbf{E}\|\boldsymbol{\alpha}_i - \nabla f_i(\boldsymbol{x}^*)\|^2\,. \qquad \text{(C.27)}\end{aligned}$$

Finally, subtracting $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\alpha})$ from both sides yields the desired result. $\square$

Theorem 27. *Let $\gamma = \frac{a}{5L}$ for any $a \leq 1$ and f be μ-strongly convex. Then Sparse Proximal* SAGA *converges geometrically in expectation with a rate factor of at least $\rho = \frac{1}{5}\min\{\frac{1}{n}, a\frac{1}{\kappa}\}$. That is, for $\boldsymbol{x}_t$ obtained after t updates and $\boldsymbol{x}^*$ the solution to (4.1), we have the bound:*

$$\mathbb{E}\|\boldsymbol{x}_t - \boldsymbol{x}^*\|^2 \leq (1-\rho)^t C_0\,, \qquad \text{with } C_0 := \|\boldsymbol{x}_0 - \boldsymbol{x}^*\|^2 + \tfrac{1}{5L^2}\textstyle\sum_{i=1}^n \|\boldsymbol{\alpha}_i^0 - \nabla f_i(\boldsymbol{x}^*)\|^2 \quad .$$

Proof. Let $\overline{H} := \frac{1}{n}\sum_i \|\boldsymbol{\alpha}_i - \nabla f_i(\boldsymbol{x}^*)\|^2$. By the Lyapunov inequality from Lemma 39, we have:

$$\begin{aligned}
\mathbf{E}\mathcal{L}_{t+1} - (1-\rho)\mathcal{L}_t &\leq \rho\mathcal{L}_t - \gamma\mu\|\boldsymbol{x}_t - \boldsymbol{x}^*\|^2 + \left(4L\gamma^2 - 2\gamma + 2L\frac{c}{n}\right) B_f(\boldsymbol{x}_t, \boldsymbol{x}^*) \\
&\qquad + \left(2\gamma^2 - \frac{c}{n}\right)\overline{H} \\
&= (\rho - \gamma\mu)\,\|\boldsymbol{x}_t - \boldsymbol{x}^*\|^2 + \left(4L\gamma^2 - 2\gamma + 2L\frac{c}{n}\right) B_f(\boldsymbol{x}_t, \boldsymbol{x}^*) \\
&\qquad + \left[2\gamma^2 + c\left(\rho - \frac{1}{n}\right)\right]\overline{H} \quad \text{(definition of } \mathcal{L}_t\text{)} \\
&\leq (\rho - \gamma\mu)\,\|\boldsymbol{x}_t - \boldsymbol{x}^*\|^2 + \left(4L\gamma^2 - 2\gamma + 2L\frac{c}{n}\right) B_f(\boldsymbol{x}_t, \boldsymbol{x}^*) \\
&\qquad + \left(2\gamma^2 - \frac{2c}{3n}\right)\overline{H} \qquad \text{(choosing } \rho \leq \tfrac{1}{3n}\text{)} \\
&= (\rho - \gamma\mu)\,\|\boldsymbol{x}_t - \boldsymbol{x}^*\|^2 + \left(10L\gamma^2 - 2\gamma\right) B_f(\boldsymbol{x}_t, \boldsymbol{x}^*) \\
&\qquad \text{(choosing } \tfrac{c}{n} = 3\gamma^2\text{)} \\
&\leq \left(\rho - \frac{a\mu}{5L}\right)\|\boldsymbol{x}_t - \boldsymbol{x}^*\|^2 \qquad \text{(for all } \gamma = \tfrac{a}{5L}, a \leq 1\text{)} \\
&\leq 0\,. \qquad \text{(for } \rho \leq \tfrac{a}{5}\cdot\tfrac{\mu}{L}\text{)}
\end{aligned}$$

And so we have the bound:

$$\mathbf{E}\mathcal{L}_{t+1} \leq \left(1 - \min\left\{\frac{1}{3n}, \frac{a}{5}\cdot\frac{1}{\kappa}\right\}\right)\mathcal{L}_t \leq \left(1 - \frac{1}{5}\min\left\{\frac{1}{n}, a\cdot\frac{1}{\kappa}\right\}\right)\mathcal{L}_t\,, \tag{C.28}$$

where in the last inequality we have used the trivial bound $\frac{1}{3n} \leq \frac{1}{5n}$ merely for clarity of

exposition. Chaining expectations from t to 0 we have:

$$\begin{aligned}
\mathbb{E}\mathcal{L}_{t+1} &\leq \left(1 - \frac{1}{5}\min\left\{\frac{1}{n}, a \cdot \frac{1}{\kappa}\right\}\right)^{t+1} \mathcal{L}_0 \\
&= \left(1 - \frac{1}{5}\min\left\{\frac{1}{n}, a \cdot \frac{1}{\kappa}\right\}\right)^{t+1} \left(\|\boldsymbol{x}_0 - \boldsymbol{x}^*\|^2 + \frac{3a^2}{5^2 L^2}\sum_{i=1}^{n}\|\boldsymbol{\alpha}_i^0 - \nabla f_i(\boldsymbol{x}^*)\|^2\right) \\
&\leq \left(1 - \frac{1}{5}\min\left\{\frac{1}{n}, a \cdot \frac{1}{\kappa}\right\}\right)^{t+1} \left(\|\boldsymbol{x}_0 - \boldsymbol{x}^*\|^2 + \frac{1}{5L^2}\sum_{i=1}^{n}\|\boldsymbol{\alpha}_i^0 - \nabla f_i(\boldsymbol{x}^*)\|^2\right) \\
&\qquad\qquad \text{(since } a \leq 1 \text{ and } {}^3\!/_5 \leq 1\text{)}
\end{aligned}$$

The fact that $\mathcal{L}_t$ is a majorizer of $\|\boldsymbol{x}_t - \boldsymbol{x}^*\|^2$ completes the proof. □

C.3 ProxASAGA

In this Appendix we provide the detailed proofs for results from Section 4.3, that is Theorem 29 (the convergence theorem for ProxASAGA) and Corollary 30 (its speedup result).

Notation. Through this section, we use the following shorthand for the gradient mapping: $\boldsymbol{g}_t := \boldsymbol{g}(\hat{\boldsymbol{x}}_t, \hat{\boldsymbol{v}}_{i_t}^t, i_t)$.

Theorem 29 (Convergence guarantee and rate of ProxASAGA)**.** *Suppose $\tau \leq \frac{1}{10\sqrt{\Delta}}$. For any step size $\gamma = \frac{a}{L}$ with $a \leq \frac{1}{36}\min\{1, \frac{6\kappa}{\tau}\}$, the inconsistent read iterates of Algorithm 7 converge in expectation at a geometric rate factor of at least: $\rho(a) = \frac{1}{5}\min\left\{\frac{1}{n}, a\frac{1}{\kappa}\right\}$, i.e. $\mathbb{E}\|\hat{\boldsymbol{x}}_t - \boldsymbol{x}^*\|^2 \leq (1-\rho)^t \tilde{C}_0$, where $\tilde{C}_0$ is a constant independent of t ($\approx \frac{n\kappa}{a} C_0$ with C_0 as defined in Theorem 27).*

Proof. In order to get an **initial recursive inequality**, we first unroll the (virtual) update:

$$\begin{aligned}\|\boldsymbol{x}_{t+1} - \boldsymbol{x}^*\|^2 &= \|\boldsymbol{x}_t - \gamma \boldsymbol{g}_t - \boldsymbol{x}^*\|^2 = \|\boldsymbol{x}_t - \boldsymbol{x}^*\|^2 + \|\gamma \boldsymbol{g}_t\|^2 - 2\gamma\langle \boldsymbol{g}_t, \boldsymbol{x}_t - \boldsymbol{x}^*\rangle \\ &= \|\boldsymbol{x}_t - \boldsymbol{x}^*\|^2 + \|\gamma \boldsymbol{g}_t\|^2 - 2\gamma\langle \boldsymbol{g}_t, \hat{\boldsymbol{x}}_t - \boldsymbol{x}^*\rangle + 2\gamma\langle \boldsymbol{g}_t, \hat{\boldsymbol{x}}_t - \boldsymbol{x}_t\rangle\,, \end{aligned} \tag{C.29}$$

and then apply Lemma 37 with $\boldsymbol{x} = \hat{\boldsymbol{x}}_t$ and $\boldsymbol{v} = \hat{\boldsymbol{v}}_{i_t}^t$. Note that in this case we have $\boldsymbol{g} = \boldsymbol{g}_t$, $P = P_{i_t}$ and $P_{i_t}\hat{\boldsymbol{v}}_{i_t}^t = \hat{\boldsymbol{v}}_{i_t}^t$.

$$\begin{aligned}\|\boldsymbol{x}_{t+1} - \boldsymbol{x}^*\|^2 &\leq \|\boldsymbol{x}_t - \boldsymbol{x}^*\|^2 + 2\gamma\langle \boldsymbol{g}_t, \hat{\boldsymbol{x}}_t - \boldsymbol{x}_t\rangle + \gamma^2\|\boldsymbol{g}_t\|^2 + \gamma^2(\beta - 2)\|\boldsymbol{g}_t\|^2 \\ &\qquad + \frac{\gamma^2}{\beta}\|\hat{\boldsymbol{v}}_{i_t}^t - \boldsymbol{D}\nabla f(\boldsymbol{x}^*)\|_{P_{i_t}}^2 - 2\gamma\langle \hat{\boldsymbol{v}}_{i_t}^t - \boldsymbol{D}\nabla f(\boldsymbol{x}^*), \hat{\boldsymbol{x}}_t - \boldsymbol{x}^*\rangle_{P_{i_t}} \\ &= \|\boldsymbol{x}_t - \boldsymbol{x}^*\|^2 + 2\gamma\langle \boldsymbol{g}_t, \hat{\boldsymbol{x}}_t - \boldsymbol{x}_t\rangle + \gamma^2(\beta - 1)\|\boldsymbol{g}_t\|^2 \\ &\qquad + \frac{\gamma^2}{\beta}\|\hat{\boldsymbol{v}}_{i_t}^t - \boldsymbol{D}_{i_t}\nabla f(\boldsymbol{x}^*)\|^2 - 2\gamma\langle \hat{\boldsymbol{v}}_{i_t}^t - \boldsymbol{D}_{i_t}\nabla f(\boldsymbol{x}^*), \hat{\boldsymbol{x}}_t - \boldsymbol{x}^*\rangle. \end{aligned} \tag{C.30}$$

We now use the property that i_t is independent of $\hat{\boldsymbol{x}}_t$ (which we enforce by reading $\hat{\boldsymbol{x}}_t$ before picking i_t, see Section 4.3), together with the unbiasedness of the gradient update $\hat{\boldsymbol{v}}_{i_t}^t$ ($\mathbf{E}\hat{\boldsymbol{v}}_{i_t}^t = \nabla f(\hat{\boldsymbol{x}}_t)$) and the definition of $\boldsymbol{D}$ to simplify the following expression as follows:

$$\begin{aligned}\mathbf{E}\langle \hat{\boldsymbol{v}}_{i_t}^t - \boldsymbol{D}_{i_t}\nabla f(\boldsymbol{x}^*), \hat{\boldsymbol{x}}_t - \boldsymbol{x}^*\rangle &= \langle \nabla f(\hat{\boldsymbol{x}}_t) - \nabla f(\boldsymbol{x}^*), \hat{\boldsymbol{x}}_t - \boldsymbol{x}^*\rangle \\ &\geq \frac{\mu}{2}\|\hat{\boldsymbol{x}}_t - \boldsymbol{x}^*\|^2 + B_f(\hat{\boldsymbol{x}}_t, \boldsymbol{x}^*)\,, \end{aligned} \tag{C.31}$$

where the last inequality comes from Lemma 31. Taking conditional expectations on (C.30) and using Lemma 38 on the variance terms we get:

$$\begin{aligned}
\mathbf{E}\|\boldsymbol{x}_{t+1}-\boldsymbol{x}^*\|^2 &\leq \|\boldsymbol{x}_t-\boldsymbol{x}^*\|^2+2\gamma\mathbf{E}\langle \boldsymbol{g}_t,\hat{\boldsymbol{x}}_t-\boldsymbol{x}_t\rangle+\gamma^2(\beta-1)\mathbf{E}\|\boldsymbol{g}_t\|^2 \qquad \text{(C.32)}\\
&\quad+\frac{\gamma^2}{\beta}\mathbf{E}\|\hat{\boldsymbol{v}}_{i_t}^t-\boldsymbol{D}_{i_t}\nabla f(\boldsymbol{x}^*)\|^2-\gamma\mu\|\hat{\boldsymbol{x}}_t-\boldsymbol{x}^*\|^2-2\gamma B_f(\hat{\boldsymbol{x}}_t,\boldsymbol{x}^*)\\
&\leq (1-\frac{\gamma\mu}{2})\|\boldsymbol{x}_t-\boldsymbol{x}^*\|^2+2\gamma\mathbf{E}\langle \boldsymbol{g}_t,\hat{\boldsymbol{x}}_t-\boldsymbol{x}_t\rangle+\gamma^2(\beta-1)\mathbf{E}\|\boldsymbol{g}_t\|^2\\
&\quad+\frac{\gamma^2}{\beta}\mathbf{E}\|\hat{\boldsymbol{v}}_{i_t}^t-\boldsymbol{D}_{i_t}\nabla f(\boldsymbol{x}^*)\|^2+\gamma\mu\|\hat{\boldsymbol{x}}_t-\boldsymbol{x}_t\|^2-2\gamma B_f(\hat{\boldsymbol{x}}_t,\boldsymbol{x}^*)\\
&\qquad\text{(using } \|a+b\|^2\leq 2\|a\|^2+2\|b\|^2 \text{ on } \|\boldsymbol{x}_t-\hat{\boldsymbol{x}}_t+\hat{\boldsymbol{x}}_t-\boldsymbol{x}^*\|^2)\\
&\leq (1-\frac{\gamma\mu}{2})\|\boldsymbol{x}_t-\boldsymbol{x}^*\|^2+\gamma^2(\beta-1)\mathbf{E}\|\boldsymbol{g}_t\|^2+\gamma\mu\|\hat{\boldsymbol{x}}_t-\boldsymbol{x}_t\|^2\\
&\quad+2\gamma\mathbf{E}\langle \boldsymbol{g}_t,\hat{\boldsymbol{x}}_t-\boldsymbol{x}_t\rangle-2\gamma B_f(\hat{\boldsymbol{x}}_t,\boldsymbol{x}^*)+\frac{4\gamma^2 L}{\beta}B_f(\hat{\boldsymbol{x}}_t,\boldsymbol{x}^*)\\
&\quad+\frac{2\gamma^2}{\beta}\mathbf{E}\|\hat{\boldsymbol{\alpha}}_{i_t}^t-\nabla f_{i_t}(\boldsymbol{x}^*)\|^2\,. \qquad \text{(C.33)}
\end{aligned}$$

Since we also have:

$$\hat{\boldsymbol{x}}_t-\boldsymbol{x}_t=\gamma\sum_{u=(t-\tau)_+}^{t-1}G_u^t\boldsymbol{g}(\hat{\boldsymbol{x}}_u,\hat{\boldsymbol{\alpha}}^u,i_u), \qquad \text{(C.34)}$$

the effect of asynchrony for the perturbed iterate updates was already derived in a very similar setup in Chapters 2 and 3. We re-use the following bounds: [1]

$$\mathbb{E}\|\hat{\boldsymbol{x}}_t-\boldsymbol{x}_t\|^2\leq\gamma^2(1+\sqrt{\Delta}\tau)\sum_{u=(t-\tau)_+}^{t-1}\mathbb{E}\|\boldsymbol{g}_u\|^2\,, \qquad \text{(A.5)} \qquad \text{(C.35)}$$

$$\mathbb{E}\langle \boldsymbol{g}_t,\hat{\boldsymbol{x}}_t-\boldsymbol{x}_t\rangle\leq\frac{\gamma\sqrt{\Delta}}{2}\sum_{u=(t-\tau)_+}^{t-1}\mathbb{E}\|\boldsymbol{g}_u\|^2+\frac{\gamma\sqrt{\Delta}\tau}{2}\mathbb{E}\|\boldsymbol{g}_t\|^2\,. \qquad \text{(A.2)} \qquad \text{(C.36)}$$

Because the updates on $\boldsymbol{\alpha}$ are the same for ProxASAGA as for ASAGA, we can re-use the same argument arising in the proof of Lemma 20 to get the following bound on $\mathbb{E}\|\hat{\boldsymbol{\alpha}}_{i_t}^t-$

1. The appearance of the sparsity constant Δ is coming from the crucial property that $\mathbf{E}\|\boldsymbol{x}\|_{(i)}^2\leq\Delta\|\boldsymbol{x}\|^2$ $\forall x\in\mathbb{R}^d$ (see (2.15), where we used the notation $\|\cdot\|_i$ for the current $\|\cdot\|_{(i)}$).

$\nabla f_{i_t}(\boldsymbol{x}^*)\|^2$:

$$\mathbb{E}\|\hat{\boldsymbol{\alpha}}_{i_t}^t - \nabla f_{i_t}(\boldsymbol{x}^*)\|^2 \leq \frac{2L}{n} \underbrace{\sum_{u=1}^{t-1}(1-\frac{1}{n})^{(t-2\tau-u-1)_+}\mathbb{E}B_f(\hat{\boldsymbol{x}}_u, \boldsymbol{x}^*)}_{\text{Henceforth denoted } H_t} + 2L(1-\frac{1}{n})^{(t-\tau)_+}\tilde{e}_0\,, \tag{C.37}$$

where $\tilde{e}_0 := \frac{1}{2L}\mathbb{E}\|\boldsymbol{\alpha}_i^0 - f_i'(\boldsymbol{x}^*)\|^2$. This bound is obtained by carefully analyzing which gradient could be the source of $\boldsymbol{\alpha}_{i_t}$ in the past (taking in consideration the inconsistent writes), and then applying Lemma 32 on the $\mathbf{E}\|\nabla f(\hat{\boldsymbol{x}}_u) - \nabla f(\boldsymbol{x}^*)\|^2$ terms, explaining the presence of $B_f(\hat{\boldsymbol{x}}_u, \boldsymbol{x}^*)$ terms. [2] The inequality (C.37) corresponds to (B.28) and (B.29) in Appendix B.

By taking the full expectation of (C.33) and plugging the above inequalities back, we obtain an inequality similar to the ASAGA Master inequality (3.13) which describes how the error terms $a_t := \mathbb{E}\|\boldsymbol{x}_t - \boldsymbol{x}^*\|^2$ of the virtual iterates are related:

$$\begin{aligned} a_{t+1} \leq & (1-\frac{\gamma\mu}{2})a_t + \frac{4\gamma^2 L}{\beta}(1-\frac{1}{n})^{(t-\tau)_+}\tilde{e}_0 \\ & + \gamma^2\left[\beta - 1 + \sqrt{\Delta}\tau\right]\mathbb{E}\|\boldsymbol{g}_t\|^2 + \left[\gamma^2\sqrt{\Delta} + \gamma^3\mu(1+\sqrt{\Delta}\tau)\right]\sum_{u=(t-\tau)_+}^{t}\mathbb{E}\|\boldsymbol{g}_u\|^2 \\ & - 2\gamma\mathbb{E}B_f(\hat{\boldsymbol{x}}_t, \boldsymbol{x}^*) + \frac{4\gamma^2 L}{\beta}\mathbb{E}B_f(\hat{\boldsymbol{x}}_t, \boldsymbol{x}^*) + \frac{4\gamma^2 L}{\beta n}H_t\,. \end{aligned} \tag{C.38}$$

We now have a promising inequality with a contractive term and several quantities that we need to bound. In order to achieve our final result, we introduce the same Lyapunov function as in the ASAGA convergence proof:

$$\mathcal{L}_t := \sum_{u=0}^{t}(1-\rho)^{t-u}a_u\,, \tag{C.39}$$

where ρ is a target rate factor for which we will provide a value later on. Proving that this

2. Note that in Chapter 3 we analyzed the unconstrained scenario, and so $B_f(\hat{\boldsymbol{x}}_u, \boldsymbol{x}^*)$ is replaced by the simpler $f(\hat{x}_u) - f(\boldsymbol{x}^*)$ in the ASAGA bound.

Lyapunov function is bounded by a contraction will finish our proof. We have:

$$\mathcal{L}_{t+1} = \sum_{u=0}^{t+1}(1-\rho)^{t+1-u}a_u = (1-\rho)^{t+1}a_0 + \sum_{u=1}^{t+1}(1-\rho)^{t+1-u}a_u$$
$$= (1-\rho)^{t+1}a_0 + \sum_{u=0}^{t}(1-\rho)^{t-u}a_{u+1}\,. \tag{C.40}$$

We now plug our new bound on a_{t+1}, (C.38):

$$\mathcal{L}_{t+1} \leq (1-\rho)^{t+1}a_0 + \sum_{u=0}^{t}(1-\rho)^{t-u}\Big[(1-\frac{\gamma\mu}{2})a_u + \frac{4\gamma^2 L}{\beta}(1-\frac{1}{n})^{(u-\tau)_+}\tilde{e}_0$$
$$+ \gamma^2\big(\beta - 1 + \sqrt{\Delta}\tau\big)\mathbb{E}\|\boldsymbol{g}_u\|^2$$
$$+ \big(\gamma^2\sqrt{\Delta} + \gamma^3\mu(1+\sqrt{\Delta}\tau)\big)\sum_{v=(u-\tau)_+}^{u}\mathbb{E}\|\boldsymbol{g}_v\|^2$$
$$- 2\gamma\mathbb{E}B_f(\hat{\boldsymbol{x}}_u, \boldsymbol{x}^*) + \frac{4\gamma^2 L}{\beta}\mathbb{E}B_f(\hat{\boldsymbol{x}}_u, \boldsymbol{x}^*) + \frac{4\gamma^2 L}{\beta n}H_u\Big]\,.$$

After regrouping similar terms, we get:

$$\mathcal{L}_{t+1} \leq (1-\rho)^{t+1}(a_0 + A\tilde{e}_0) + (1-\frac{\gamma\mu}{2})\mathcal{L}_t + \sum_{u=0}^{t}s_u^t\mathbb{E}\|\boldsymbol{g}_u\|^2$$
$$+ \sum_{u=1}^{t}r_u^t\mathbb{E}B_f(\hat{\boldsymbol{x}}_u, \boldsymbol{x}^*)\,. \tag{C.41}$$

Now, provided that we can prove that under certain conditions the s_u^t and r_u^t terms are all negative (and that the A term is not too big), we can drop them from the right-hand side of (C.41) which will allow us to finish the proof.

Let us compute these terms. Let $q := \frac{1-1/n}{1-\rho}$ and we assume in the rest that $\rho < 1/n$.

Computing A. We have from (B.49):

$$\frac{4\gamma^2 L}{\beta}\sum_{u=0}^{t}(1-\rho)^{t-u}(1-\frac{1}{n})^{(u-\tau)_+} \leq \frac{4\gamma^2 L}{\beta}(1-\rho)^t(1-\rho)^{-\tau}(\tau+1+\frac{1}{1-q})$$
$$= (1-\rho)^{t+1}\underbrace{\frac{4\gamma^2 L}{\beta}(1-\rho)^{-\tau-1}(\tau+1+\frac{1}{1-q})}_{:=A}\,.$$

Computing s_u^t. Since we have:

$$\sum_{u=0}^{t}(1-\rho)^{t-u}\sum_{v=(u-\tau)_+}^{u-1}\mathbb{E}\|\boldsymbol{g}_u\|^2 \le \tau(1-\rho)^{-\tau}\sum_{u=0}^{t}(1-\rho)^{t-u}\mathbb{E}\|\boldsymbol{g}_u\|^2\,, \tag{C.42}$$

we have for all $0 \le u \le t$:

$$s_u^t \le (1-\rho)^{t-u}\Big[\gamma^2\big(\beta - 1 + \sqrt{\Delta}\tau\big) + \tau(1-\rho)^{-\tau}\big(\gamma^2\sqrt{\Delta} + \gamma^3\mu(1+\sqrt{\Delta}\tau)\big)\Big]\,. \tag{C.43}$$

Computing r_u^t. To analyze these quantities, we need to compute: $\sum_{u=0}^{t}(1-\rho)^{t-u}\sum_{v=1}^{u-1}(1-\frac{1}{n})^{(u-2\tau-v-1)_+}$. Fortunately, we have already done this in (B.40), and thus we know that for all $1 \le u \le t$:

$$r_u^t \le (1-\rho)^{t-u}\left[-2\gamma + \frac{4\gamma^2 L}{\beta} + \frac{4L\gamma^2}{n\beta}(1-\rho)^{-2\tau-1}\Big(2\tau + \frac{1}{1-q}\Big)\right]\,, \tag{C.44}$$

recalling that $q := \frac{1-1/n}{1-\rho}$ and that we assumed $\rho < \frac{1}{n}$.

We now need some assumptions to further analyze these quantities. We make simple choices for simplicity, though a tighter analysis is possible. To get manageable (and simple) constants, we follow (B.56) and (B.57) and assume:

$$\rho \le \frac{1}{4n}; \quad \tau \le \frac{n}{10}\,. \tag{C.45}$$

This tells us:

$$\begin{aligned} \frac{1}{1-q} &\le \frac{4n}{3} \\ (1-\rho)^{-k\tau-1} &\le \frac{4}{3} \qquad \text{for } 0 \le k \le 2\,. \qquad \text{(using Bernoulli's inequality)} \end{aligned}$$

Additionally, we set $\beta = \frac{1}{2}$. Equation (C.43) thus becomes:

$$s_u^t \le \gamma^2(1-\rho)^{t-u}\left[-\frac{1}{2} + \sqrt{\Delta}\tau + \frac{4}{3}\big(\sqrt{\Delta}\tau + \gamma\mu\tau(1+\sqrt{\Delta}\tau)\big)\right]\,. \tag{C.46}$$

We see that for s_u^t to be negative, we need $\tau = \mathcal{O}(\frac{1}{\sqrt{\Delta}})$. Let us assume that $\tau \le \frac{1}{10\sqrt{\Delta}}$. We then get:

$$s_u^t \le \gamma^2(1-\rho)^{t-u}\left[-\frac{1}{2} + \frac{1}{10} + \frac{4}{30} + \gamma\mu\tau\frac{4}{3}\frac{11}{10}\right]\,. \tag{C.47}$$

Thus, the condition under which all s_u^t are negative boils down to:

$$\gamma\mu\tau \leq \frac{2}{11}\,. \tag{C.48}$$

Now looking at the r_u^t terms given our assumptions, the inequality (C.44) becomes:

$$\begin{aligned} r_u^t &\leq (1-\rho)^{t-u}\left[-2\gamma + 8\gamma^2 L + \frac{8\gamma^2 L}{n}\frac{4}{3}\big(\frac{n}{5} + \frac{4n}{3}\big)\right] \\ &\leq (1-\rho)^{t-u}\big(-2\gamma + 36\gamma^2 L\big)\,. \end{aligned} \tag{C.49}$$

The condition for all r_u^t to be negative then can be simplified down to:

$$\gamma \leq \frac{1}{18L}\,. \tag{C.50}$$

We now have a promising inequality for proving that our Lyapunov function is bounded by a contraction. However we have defined $\mathcal{L}_t$ in terms of the virtual iterate $\boldsymbol{x}_t$, which means that our result would only hold for a given T fixed in advance, as is the case in Mania et al. (2017). Fortunately, we can use the same trick as in (B.70): we simply add $\gamma B_f(\hat{\boldsymbol{x}}_t, \boldsymbol{x}^*)$ to both sides in (C.41). r_t^t is replaced by $r_t^t + \gamma$, which makes for a slightly worse bound on γ to ensure linear convergence:

$$\gamma \leq \frac{1}{36L}\,. \tag{C.51}$$

For this small cost, we get a contraction bound on $B_f(\hat{\boldsymbol{x}}_t, \boldsymbol{x}^*)$, and thus by the strong convexity of f (see (C.3)) we get a contraction bound for $\mathbb{E}\|\hat{\boldsymbol{x}}_t - \boldsymbol{x}^*\|^2$.

Recap. Let us use $\rho = \frac{1}{4n}$ and $\gamma := \frac{a}{L}$. Then the conditions (C.48) and (C.51) on the step size γ reduce to:

$$a \leq \frac{1}{36}\min\{1, \frac{72}{11}\frac{\kappa}{\tau}\}. \tag{C.52}$$

Moreover, the condition:

$$\tau \leq \frac{1}{10\sqrt{\Delta}} \tag{C.53}$$

is sufficient to also ensure that (C.45) is satisfied as $\Delta \in [\frac{1}{n}, 1]$, and thus $\frac{1}{\sqrt{\Delta}} \leq \sqrt{n} \leq n$.

Thus under the conditions (C.52) and (C.53), we have that all s_u^t and r_u^t terms are negative and we can rewrite the recurrent step of our Lyapunov function as:

$$\mathcal{L}_{t+1} \leq \gamma\mathbb{E}B_f(\hat{\boldsymbol{x}}_t) + \mathcal{L}_{t+1} \leq (1-\rho)^{t+1}(a_0 + A\tilde{e}_0) + (1 - \frac{\gamma\mu}{2})\mathcal{L}_t\,. \tag{C.54}$$

By unrolling the recursion (C.54), we can carefully combine the effect of the geometric

term $(1-\rho)$ with the one of $(1-\frac{\gamma\mu}{2})$. This was already done in (B.74) to (B.76), with a trick to handle various boundary cases, yielding the overall rate:

$$\mathbb{E}B_f(\hat{\boldsymbol{x}}_t, \boldsymbol{x}^*) \leq (1-\rho^*)^{t+1}\hat{C}_0, \tag{C.55}$$

where $\rho^* = \min\{\frac{1}{5n}, a\frac{2}{5\kappa}\}$ (that we simplified to $\rho^* = \frac{1}{5}\min\{\frac{1}{n}, a\frac{1}{\kappa}\}$ in the theorem statement). To get the final constant, we need to bound A. We have:

$$\begin{aligned} A &= \frac{4\gamma^2 L}{\beta}(1-\rho)^{-\tau-1}(\tau+1+\frac{1}{1-q}) \\ &\leq 8\gamma^2 L\frac{4}{3}(\frac{n}{10}+1+\frac{4n}{3}) \\ &\leq 26\gamma^2 Ln \\ &\leq \gamma n\,. \end{aligned} \tag{C.56}$$

This is the same bound on A that was used in Chapter 3 and so we obtain the same constant as in (B.77):

$$\hat{C}_0 := \frac{21n}{\gamma}(\|\boldsymbol{x}_0 - \boldsymbol{x}^*\|^2 + \gamma\frac{n}{2L}\mathbb{E}\|\boldsymbol{\alpha}_i^0 - \nabla f_i(\boldsymbol{x}^*)\|^2). \tag{C.57}$$

Note that $\hat{C}_0 = \mathcal{O}(\frac{n}{\gamma}C_0)$ with C_0 defined as in Theorem 27.

Now, using the strong convexity of f via (C.3), we get:

$$\mathbb{E}\|\hat{\boldsymbol{x}}_t - \boldsymbol{x}^*\|^2 \leq \frac{2}{\mu}\mathbb{E}B_f(\hat{\boldsymbol{x}}_t, \boldsymbol{x}^*) \leq (1-\rho^*)^{t+1}\tilde{C}_0, \tag{C.58}$$

where $\tilde{C}_0 = \mathcal{O}(\frac{n\kappa}{a}C_0)$.

This finishes the proof for Theorem 29. □

Corollary 30 (Speedup)**.** *Suppose $\tau \leq \frac{1}{10\sqrt{\Delta}}$. If $\kappa \geq n$, then using the step size $\gamma = 1/36L$,* ProxASAGA *converges geometrically with rate factor $\Omega(\frac{1}{\kappa})$. If $\kappa < n$, then using the step size $\gamma = 1/36n\mu$,* ProxASAGA *converges geometrically with rate factor $\Omega(\frac{1}{n})$. In both cases, the convergence rate is the same as Sparse Proximal* SAGA *and* ProxASAGA *is thus linearly faster than its sequential counterpart up to a constant factor. Note that in both cases* the step size does not depend on τ.

Furthermore, if $\tau \leq 6\kappa$, we can use a universal step size of $\Theta(1/L)$ to get a similar rate for ProxASAGA *than Sparse Proximal* SAGA*, thus making it adaptive to local strong convexity since the knowledge of κ is not required.*

Proof. If $\kappa \geq n$, the rate factor of Sparse Proximal SAGA is $1/\kappa$. To get the same rate factor, we need to choose $a = \Omega(1)$, which we can fortunately do since $\kappa \geq n \geq \sqrt{n} \geq 10\frac{1}{10\sqrt{\Delta}} \geq$

10τ.

If $\kappa < n$, then the rate factor of Sparse Proximal SAGA is $1/n$. Any choice of a bigger than $\Omega(\kappa/n)$ gives us the same rate factor for PROXASAGA. Since $\tau \leq \sqrt{n}/10$ we can pick such an a without violating the condition of Theorem 29. □

C.4 Comparison of bounds with Liu and Wright (2015)

Iteration costs. For both ProxASAGA and AsySPCD, the average cost of an iteration is $\mathcal{O}(n\overline{S})$ (where $\overline{S}$ is the average support size). In the case of ProxASAGA (see Algorithm 3), at each iteration the most costly operation is the computation of $\overline{\boldsymbol{\alpha}}$, while in the general case we need to compute a full gradient for AsySPCD.

In order to reduce these prohibitive computation costs, several tricks are introduced. Although they lead to much improved empirical performance, it should be noted that in both cases these tricks are not covered by the theory. In particular, the unbiasedness condition can be violated.

In the case of ProxASAGA, we store the average gradient term $\overline{\boldsymbol{\alpha}}$ in shared memory. The cost of each iteration then becomes the size of the extended support of the partial gradient selected at random at this iteration, hence it is in $\mathcal{O}(\Delta_l)$, where $\Delta_l := \max_{i=1..n} |T_i|$.

For AsySPCD, following Peng et al. (2016) we can store intermediary quantities for specific losses (e.g. ℓ_1-regularized logistic regression). The cost of an iteration then becomes the number of data points whose extended support includes the coordinate selected at random at this iteration, hence it is in $\mathcal{O}(n\Delta)$.

The relative difference in update cost of both algorithms then depends heavily on the data matrix: if the partial gradients usually have a extended support but coordinates belong to few of them (this can be the case if $n \ll d$ for example), then the iterations of AsySPCD can be cheaper than those of ProxASAGA. Conversely, if data points usually have small extended support but coordinates belong to many of them (which can happen when $d \ll n$ for example), then the updates of ProxASAGA are the cheaper ones.

Dependency of τ on the data matrix. In the case of ProxASAGA the sizes of the extended support of each data point are important – they are directly linked to the cost of each iteration. Identical iteration costs for each data point do not influence τ, whereas heterogeneous costs may cause τ to increase substantially. In contrast, in the case of AsySPCD, the relevant parts of the data matrix are the number of data points each dimension touches – for much the same reason. In the bipartite graph between data points and dimensions, either the left or the right degrees matter for τ, depending on which algorithm you choose.

In order to compare their respective bounds, we have to make the assumption that the iteration costs are homogeneous, which means that each data point has the same support size and each dimension is active in the same number of data points. This implies that τ is the same quantity for both algorithms.

Best case scenario bound for AsySPCD. The result obtained in Liu and Wright (2015) states that if $\tau^2\Lambda = \mathcal{O}(\sqrt{d})$, ASYSPCD can get a near-linear speedup (where Λ is a measure of the interactions between the components of the gradient, with $1 \leq \Lambda \leq \sqrt{d}$). In the best possible scenario where $\Lambda = 1$ (which means that the coordinates of the gradients are completely uncorrelated), τ can be as big as $\sqrt[4]{d}$.

Appendix D

SEARNN

D.1 Algorithms

SEARNN: reference roll-in with an RNN. As mentioned in Section 7.2, teacher forcing can be seen as the roll-in *reference policy* of the RNN. In this section, we detail this analogy further.

Let us consider the case where we perform the roll-in up until the t^{th} cell. In order to be able to perform roll-outs from that t^{th} cell, a hidden state is needed. If we used a *reference policy* roll-in, this state is obtained by running the RNN until the t^{th} cell by using the teacher forcing strategy, i.e. by conditioning the outputs on the ground truth. Finally, SEARNN also needs to know what the predictions for the full sequence were in order to compute the costs. When the *reference roll-in* is used, we obtain the predictions up until the t^{th} cell by simply copying the ground truth. Hence, we discard the outputs of the RNN that are before the t^{th} cell.

D.2 Design decisions

Choosing a classifier: to backpropagate or not to backpropagate? In standard L2S, the classifier and the feature extractor are clearly delineated. The latter is a fixed hand-crafted transformation applied on the input and the partial sequence that has already been predicted. One then has to pick a classifier and its convergence properties carry over to the initial problem.

In SEARNN, we choose the RNN itself as our classifier. The fixed feature extractor is reduced to the bare minimum (e.g. one-hot encoding) and the classifier performs feature learning afterwards. In this setting, the intermediate dataset is the initial state and all previous

decisions $(x, y_{1:t-1})$ combined with the cost vector. [1]

An alternative way to look at RNNs, is to consider the RNN cell as a shared classifier in its own right, and the beginning of the RNN (including the previous cells) as a feature extractor. One could then pick the RNN cell (instead of the full RNN) as the SEARNN classifier, in which case the intermediate dataset would be (h_{t-1}, y_{t-1}) [2] (the state at the previous step, combined with the previous decision) plus the cost vector.

While this last perspective – seeing the RNN cell as the shared classifier instead of the full RNN – is perhaps more intuitive, it actually fits the L2S framework less well. Indeed, there is no clear delineation between classifier and feature extractor as these functions are carried out by different instances of the same RNN cell (and as such share weights). This means that the feature extraction in this case is learned instead of being fixed.

This choice of classifier has a direct consequence on the optimization routine. In case we pick the RNN itself, then each loss gradient has to be fully backpropagated through the network. On the other hand, if the classifier is the cell itself, then one should not backpropagate the gradient updates.

Reference policy. The reference policy defined by Daumé et al. (2009) picks the action which "minimizes the (corresponding) cost, assuming all future decisions are made optimally", i.e. $\arg\min_{y_t} \min_{y_{t+1:T}} l(y_{1:T}, y)$.

For the roll-in phase, this policy corresponds to always picking the ground truth, since it leads to predicting the full ground truth sequence and hence the best possible loss.

For the roll-out phase, computing this policy explicitly is easy in a few select cases. However, in the general case it is not tractable. One then has to turn to heuristics, whose performance can be relatively poor. While Chang et al. (2015) tell us that overcoming a bad reference policy can be done through a careful choice of roll-in/roll-out policies, the fact remains that the better the reference policy is, the better performance will be. Choosing this heuristic well is then quite important.

The most basic heuristic is to simply use the ground truth. Of course, one can readily see that it is not always optimal. For example, when the model skips a token and outputs the next one, a, instead, it may be more beneficial to also skip a in the roll-out phase rather than to repeat it.

Although we mostly chose this basic heuristic in this paper, using tailored alternatives can yield better results for tasks where it is suboptimal, such as machine translation (see

1. In the encoder-decoder architecture, the decoder RNN does not receive x directly, but rather $\phi(x)$, the features extracted from the input by the encoder RNN. In this case, our SEARNN classifier includes both the encoder and the decoder RNNs.
2. One could also add $\psi(x)$, features learned from the input through e.g. an attention mechanism.

Appendix D.3).

D.3 Additional machine translation experimental details

Custom sampling. For this experiment, we decided to sample 15 tokens per cell according to the top-k policy (as the vocabulary size is quite big, sampling tokens with low probability is not very attractive), as well as 10 neighboring ground truth labels around the cell. The rationale for these neighboring tokens is that skipping or repeating words is quite a common mistake in NMT.

Custom reference policy. The very basic reference policy we have been using for the other experiments of the paper is too bad a heuristic for BLEU to perform well. Instead, we try adding every suffix in the ground truth sequence to the current predictions and we pick the one with the highest BLEU-1 score (using this strategy with BLEU-4 leads to unfortunate events when the best suffix to add is always the entire sequence, leading to uninformative costs).

Bibliography

Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. An Actor-Critic Algorithm for Sequence Prediction. In *Proceedings of the* 5^{th} *International Conference on Learning Representations (ICLR)*, 2017.
(Cited on pages 7, 113, 114, 117, 127, 128, 131, 132, 141, and 146.)

Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A Smith. Training with Exploration Improves a Greedy Stack-LSTM Parser. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2016.
(Cited on pages 104, 112, 116, 140, and 145.)

Heinz Bauschke and Patrick L. Combettes. *Convex analysis and monotone operator theory in Hilbert spaces*. Springer, 2011.
(Cited on page 190.)

Amir Beck and Marc Teboulle. Gradient-based algorithms with applications to signal recovery. *Convex Optimization in Signal Processing and Communications*, 2009.
(Cited on pages 24, 81, 97, and 189.)

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. In *Advances in Neural Information Processing Systems 28 (NIPS)*, 2015.
(Cited on pages 112, 132, and 140.)

Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.
(Cited on pages 20, 34, and 37.)

Alina Beygelzimer, Hal Daumé, III, John Langford, and Paul Mineiro. Learning reductions that really work. *Proceedings of the IEEE*, 2016.
(Cited on page 118.)

George Broyden. The convergence of a class of double-rank minimization algorithms. *IMA Journal of Applied Mathematics*, 1970.
(Cited on page 15.)

Rudy Bunel, Matthew Hausknecht, Jacob Devlin, Rishabh Singh, and Pushmeet Kohli. Leveraging grammar and reinforcement learning for neural program synthesis. In *Proceedings of the* 6^{th} *International Conference on Learning Representations (ICLR)*, 2018.
(Cited on pages 113 and 141.)

Augustin Cauchy. Méthode générale pour la résolution des systèmes d'équations simultanées. *Comptes-rendus hebdomadaires des séances de l'Académie des Sciences*, 1847.
(Cited on page 14.)

Mauro Cettolo, Jan Niehues, Sebastian Stuker, Luisa Bentivogli, and Marcello Federico. Report on the 11th IWSLT evaluation campaign. *Proceedings of the International Workshop on Spoken Language Translation (IWSLT)*, 2014.
(Cited on page 131.)

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daumé, III, and John Langford. Learning to Search Better than Your Teacher. In *Proceedings of the* 32^{nd} *International Conference on Machine Learning (ICML)*, 2015.
(Cited on pages 115, 117, 119, 126, 127, 128, 132, 139, 147, and 210.)

Kyunghyun Cho, Bart Van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.
(Cited on pages 107, 127, and 128.)

R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of SVMs for very large scale problems. *Neural Computation*, 2002.
(Cited on pages 69 and 185.)

Zihang Dai, Qizhe Xie, and Eduard Hovy. From credit assignment to entropy regularization: two new algorithms for neural sequence prediction. In *Proceedings of the* 56^{th} *annual meeting of the Association for Computational Linguistics (ACL)*, 2018.
(Cited on pages 114 and 142.)

Hal Daumé, III and Daniel Marcu. Learning as search optimization: approximate large margin methods for structured prediction. In *Proceedings of the* 22^{nd} *International*

Conference on Machine Learning (ICML), 2005.
(Cited on pages 112 and 140.)

Hal Daumé, III, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine Learning*, 2009.
(Cited on pages 7, 111, 112, 115, 117, 118, 119, 120, 132, 140, 146, and 210.)

Damek Davis, Brent Edmunds, and Madeleine Udell. The sound of APALM clapping: faster nonsmooth nonconvex optimization with stochastic asynchronous PALM. In *Advances in Neural Information Processing Systems 29*, 2016.
(Cited on pages 24 and 80.)

Christopher 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 (NIPS)*, 2015.
(Cited on pages 21, 23, 36, and 41.)

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 (NIPS)*, 2014a.
(Cited on pages 3, 17, 18, 22, 26, 30, 32, 47, 49, 58, 70, 78, 80, 82, 84, 92, and 95.)

Aaron Defazio, Tibério Caetano, and Justin Domke. Finito: A faster, permutable incremental gradient method for big data problems. In *Proceedings of the* 31^{st} *International Conference on Machine Learning (ICML)*, 2014b.
(Cited on page 18.)

Nan Ding and Radu Soricut. Cold-Start Reinforcement Learning with Softmax Policy Gradient. In *Advances in Neural Information Processing Systems 30 (NIPS)*, 2017.
(Cited on page 124.)

John C. Duchi, Sorathan Chaturapruek, and Christopher Ré. Asynchronous stochastic convex optimization: the noise is in the noise and SGD don't care. In *Advances in Neural Information Processing Systems 28 (NIPS)*, 2015.
(Cited on pages 21, 23, 33, and 72.)

Sergey Edunov, Myle Ott, Michael Auli, David Grangier, and Marc'Aurelio Ranzato. Classical structured prediction losses for sequence to sequence learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational*

Linguistics (NAACL), 2018.
(Cited on pages 114 and 143.)

Maha Elbayad, Laurent Besacier, and Jakob Verbeek. Token-level and sequence-level loss smoothing for RNN language models. In *Proceedings of the* 56^{th} *annual meeting of the Association for Computational Linguistics (ACL)*, 2018.
(Cited on pages 114 and 142.)

Roger Fletcher. A new approach to variable-metric algorithms. *The compute journal*, 1970.
(Cited on page 15.)

Kevin Gimpel and Noah A Smith. Softmax-margin CRFs: Training loglinear models with cost functions. In *Proceedings of the 2010 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2010.
(Cited on page 126.)

Yoav Golberg and Joakim Nivre. A Dynamic Oracle for Arc-Eager Dependency Parsing. *Proceedings of the* 24^{th} *International Conference on Computational Linguistics (COLING)*, 2012.
(Cited on page 125.)

Donald Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of computation*, 1970.
(Cited on page 15.)

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems 27 (NIPS)*, 2014.
(Cited on page 143.)

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
(Cited on pages 105, 106, 109, and 110.)

James Goodman, Andreas Vlachos, and Jason Naradowsky. Noise reduction and targeted exploration in imitation learning for Abstract Meaning Representation parsing. In *Proceedings of the* 54^{th} *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2016.
(Cited on pages 129, 138, and 147.)

Joshua Goodman. Classes for fast maximum entropy training. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2001.
(Cited on page 147.)

Alex Graves. *Supervised sequence labelling with recurrent neural networks*. Springer, 2012.
(Cited on page 109.)

Bin Gu, Zhouyuan Huo, and Heng Huang. Asynchronous stochastic block coordinate descent with variance reduction. *arXiv preprint arXiv:1610.09447v3*, 2016.
(Cited on pages 26, 81, and 92.)

Robert Hannah and Wotao Yin. More iterations per second, same quality – why asynchronous algorithms may drastically outperform traditional ones. *arXiv preprint arXiv:1708.05136*, 2017.
(Cited on page 19.)

Tamir Hazan and Raquel Urtasun. A Primal-Dual Message-Passing Algorithm for Approximated Large Scale Structured Prediction. In *Advances in Neural Information Processing Systems 23 (NIPS)*, 2010.
(Cited on page 126.)

Sepp Hochreiter and Jurgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 1997.
(Cited on page 127.)

Thomas Hofmann, Aurelien Lucchi, Simon Lacoste-Julien, and Brian McWilliams. Variance Reduced Stochastic Gradient Descent with Neighbors. In *Advances in Neural Information Processing Systems 28 (NIPS)*, 2015.
(Cited on pages x, 17, 23, 42, 47, 49, 50, 51, 53, 55, 56, 57, 58, 60, 61, 62, 151, 157, 158, 159, 160, 161, 162, 177, and 179.)

Cho-Jui Hsieh, Hsiang-Fu Yu, and Inderjit S Dhillon. PASSCoDe: parallel asynchronous stochastic dual coordinate descent. In *Proceedings of the* 32^{nd} *International Conference on Machine Learning (ICML)*, 2015.
(Cited on pages 22, 23, 26, 47, 48, and 80.)

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. On Using Very Large Target Vocabulary for Neural Machine Translation. In *Proceedings of the* 53^{rd}

Annual Meeting of the Association for Computational Linguistics (ACL), 2015.
(Cited on page 130.)

Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems 26 (NIPS)*, 2013.
(Cited on pages 3, 17, 18, 26, 30, 47, 60, 61, 64, 66, 67, 68, 78, 80, and 180.)

Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. Field-aware factorization machines for CTR prediction. In *Proceedings of the* 10^{th} *ACM Conference on Recommender Systems (ACM)*, 2016.
(Cited on page 93.)

Yaser Keneshloo, Tian Shi, Ramakrishnan Naren, and Chandan K. Reddy. Deep reinforcement learning for sequence to sequence models. *arXiv preprint, arXiv:1805.09461*, 2018.
(Cited on page 113.)

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the* 3^{rd} *International Conference on Learning Representations (ICLR)*, 2015.
(Cited on page 128.)

Jakub Konečný and Peter Richtárik. Semi-Stochastic Gradient Descent Methods. *arXiv preprint arXiv:1312.1666*, 2013.
(Cited on pages 66 and 185.)

Matti Kääriäinen. Lower bounds for reductions. *Talk at the Atomic Learning Workshop (TTI-C)*, 2006.
(Cited on page 111.)

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 (NIPS)*, 2012.
(Cited on pages 3, 17, 18, 25, 30, 32, 46, 69, and 80.)

Rémi Leblond, Fabian Pedregosa, and Simon Lacoste-Julien. ASAGA: asynchronous parallel SAGA. In *Proceedings of the* 20^{th} *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.
(Cited on pages 8, 9, 10, 22, 23, 47, and 91.)

Rémi Leblond, Jean-Baptiste Alayrac, Anton Osokin, and Simon Lacoste-Julien. SEARNN: training rnns with global-local losses. In *Proceedings of the* 6^{th} *International Conference on Learning Representations (ICLR)*, 2018a.
(Cited on pages 9, 10, and 115.)

Rémi Leblond, Fabian Pedregosa, and Simon Lacoste-Julien. Improved asynchronous parallel optimization analysis for stochastic incremental methods. *To appear in the Journal of Machine Learning Research (JMLR)*, 2018b.
(Cited on pages 8, 10, and 47.)

Yoonkyung Lee, Yi Lin, and Grace Wahba. Multicategory support vector machines: Theory and application to the classification of microarray data and satellite radiance data. *Journal of the American Statistical Association*, 2004.
(Cited on page 125.)

David D Lewis, Yiming Yang, Tony G Rose, and Fan Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research (JMLR)*, 2004.
(Cited on pages 69 and 184.)

Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. Asynchronous Parallel Stochastic Gradient for Nonconvex Optimization. In *Advances in Neural Information Processing Systems 28 (NIPS)*, 2015.
(Cited on page 23.)

Ji Liu and Stephen J Wright. Asynchronous stochastic coordinate descent: Parallelism and convergence properties. *SIAM Journal on Optimization*, 2015.
(Cited on pages xiii, 24, 80, 91, 97, 207, and 208.)

Ji Liu, Stephen J. Wright, Christopher Ré, Victor Bittorf, and Srikrishna Sridhar. An Asynchronous Parallel Stochastic Coordinate Descent Algorithm. *Journal of Machine Learning Research (JMLR)*, 2015.
(Cited on page 23.)

Chenxin Ma, Virginia Smith, Martin Jaggi, Michael I. Jordan, Peter Richtarik, and Martin Takac. Adding vs. averaging in distributed primal-dual optimization. In *Proceedings of the* 32^{nd} *International Conference on Machine Learning (ICML)*, 2015.
(Cited on page 185.)

Justin Ma, Lawrence K. Saul, Stefan Savage, and Geoffrey M. Voelker. Identifying suspicious URLs: an application of large-scale online learning. In *Proceedings of the* 26^{th}

International Conference on Machine Learning (ICML), 2009.
(Cited on pages 69 and 184.)

Julien Mairal. Incremental majorization-minimization optimization with application to large-scale machine learning. *SIAM Journal on Optimization*, 2015.
(Cited on page 18.)

Horia Mania, Xinghao Pan, Dimitris Papailiopoulos, Benjamin Recht, Kannan Ramchandran, and Michael I Jordan. Perturbed iterate analysis for asynchronous stochastic optimization. *SIAM Journal on Optimization*, 2017.
(Cited on pages 4, 5, 21, 22, 23, 27, 29, 30, 31, 32, 33, 34, 35, 36, 37, 40, 41, 42, 46, 47, 48, 50, 55, 56, 61, 62, 63, 66, 71, 179, 185, and 204.)

Qi Meng, Wei Chen, Jingcheng Yu, Taifeng Wang, Zhi-Ming Ma, and Tie-Yan Liu. Asynchronous stochastic proximal optimization algorithms with variance reduction. In *Proceedings of the* 31^{st} *AAAI Conference on Artificial Intelligence (AAAI)*, 2017.
(Cited on pages 26, 81, and 92.)

Eric Moulines and Francis R. Bach. Non-Asymptotic Analysis of Stochastic Approximation Algorithms for Machine Learning. In *Advances in Neural Information Processing Systems 24 (NIPS)*, 2011.
(Cited on page 40.)

Deanna Needell, Rachel Ward, and Nati Srebro. Stochastic Gradient Descent, Weighted Sampling, and the Randomized Kaczmarz algorithm. In *Advances in Neural Information Processing Systems 27 (NIPS)*, 2014.
(Cited on pages 41 and 51.)

Yurii Nesterov. *Introductory lectures on convex optimization*. Springer Science & Business Media, 2004.
(Cited on pages 14 and 189.)

Yurii Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 2012.
(Cited on page 51.)

Yurii Nesterov. Gradient methods for minimizing composite functions. *Mathematical Programming*, 2013.
(Cited on page 189.)

Lam M. Nguyen, Phuong Ha Nguyen, Marten van Dijk, Peter Richtárik, Katya Scheinberg, and Martin Takáč. SGD and Hogwild! Convergence without the bounded gradients assumption. In *Proceedings of the* 35^{th} *International Conference on Machine Learning (ICML)*, 2018.
(Cited on pages 21 and 23.)

Feng Niu, Benjamin Recht, Christopher Re, and Stephen Wright. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems 24 (NIPS)*, 2011.
(Cited on pages 4, 5, 20, 23, 25, 29, 30, 31, 33, 36, 38, 41, 47, 55, 71, 78, 85, 87, 91, and 185.)

Mohammad Norouzi, Sammy Bengio, Zhifeng Chen, Navdeep Jaitly, Mike Schuster, Yonghui Wu, and Dale Schuurmans. Reward Augmented Maximum Likelihood for Neural Structured Prediction. In *Advances in Neural Information Processing Systems 29 (NIPS)*, 2016.
(Cited on pages 114 and 142.)

Xinghao Pan, Maximilian Lam, Stephen Tu, Dimitris Papailiopoulos, Ce Zhang, Michael I Jordan, Kannan Ramchandran, and Christopher Ré. Cyclades: Conflict-free Asynchronous Machine Learning. In *Advances in Neural Information Processing Systems 29 (NIPS)*, 2016.
(Cited on pages 22, 48, and 184.)

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the* 40^{th} *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2002.
(Cited on page 131.)

Fabian Pedregosa, Rémi Leblond, and Simon Lacoste-Julien. Breaking the nonsmooth barrier: A scalable parallel method for composite optimization. In *Advances in Neural Information Processing Systems 30 (NIPS)*, 2017.
(Cited on pages 9 and 25.)

Zhimin Peng, Yangyang Xu, Ming Yan, and Wotao Yin. ARock: an algorithmic framework for asynchronous parallel coordinate updates. *SIAM Journal on Scientific Computing*, 2016.
(Cited on pages 24, 25, 80, 94, and 207.)

Gabriel Pereyra, George Tucker, Jan Chorowski, Lukasz Kaiser, and Geoffrey Hinton. Regularizing neural networks by penalizing confident output distributions. In *ICLR 2017 Workshop track*, 2017.
(Cited on pages 134 and 135.)

Patrick Pletscher, Cheng Soon Ong, and Joachim M. Buhmann. Entropy and Margin Maximization for Structured Output Learning. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2010.
(Cited on page 126.)

Sébastien Racanière, Theophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter Battaglia, Demis Hassabis, David Silver, and Daan Wierstra. Imagination-Augmented Agents for Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems 30 (NIPS)*, 2017.
(Cited on page 147.)

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence Level Training with Recurrent Neural Networks. In *Proceedings of the* 5^{th} *International Conference on Learning Representations (ICLR)*, 2016.
(Cited on pages 7, 110, 111, 113, 117, 131, 132, 138, 141, and 146.)

Sashank J Reddi, Ahmed Hefny, Suvrit Sra, Barnabas Poczos, and Alex Smola. On variance reduction in stochastic gradient descent and its asynchronous variants. In *Advances in Neural Information Processing Systems 28 (NIPS)*, 2015.
(Cited on pages 4, 22, 23, 26, 43, 47, 48, 56, 64, 66, 80, 91, 184, and 185.)

Sashank J. Reddi, Ahmed Hefny, Suvrit Sra, Barnabás Póczós, and Alex Smola. Stochastic Variance Reduction for Nonconvex Optimization. In *Proceedings of the* 33^{rd} *International Conference on Machine Learning (ICML)*, 2016.
(Cited on page 60.)

Steven Rennie, Etienne Marcheret, Youssef Mroueh, Jarret Ross, and Vaibhava Goel. self-critical sequence training for image captioning. In *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
(Cited on pages 113 and 141.)

Herbert Robbins and Sutton Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 1951.
(Cited on pages 3 and 16.)

Stephane Ross and J. Andrew Bagnell. Reinforcement and Imitation Learning via Interactive No-Regret Learning. *arXiv preprint arXiv:1406.5979*, 2014.
(Cited on pages 112, 115, 117, 141, and 147.)

Mark Schmidt. Convergence rate of stochastic gradient with constant step size. *UBC Technical Report*, 2014.
(Cited on pages 16 and 39.)

Mark Schmidt, Reza Babanezhad, Mohamed Osama Ahmed, Aaron Defazio, Ann Clifton, and Anoop Sarkar. Non-uniform stochastic average gradient method for training conditional random fields. In *Proceedings of the* 18^{th} *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2015.
(Cited on page 52.)

Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 2016.
(Cited on pages 22, 24, 47, 49, 50, 51, 69, 77, 81, 93, 94, 183, and 186.)

Shai Shalev-Shwartz and Tong Zhang. Proximal stochastic dual coordinate ascent. *arXiv preprint arXiv:1211.2717*, 2012.
(Cited on pages 26 and 80.)

Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research (JMLR)*, 2013.
(Cited on pages 17, 18, 26, 30, 46, and 80.)

David Shanno. Conditioning of quasi-Newton methods for function minimization. *Mathematics of computation*, 1970.
(Cited on page 15.)

Shiqi Shen, Yong Cheng, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. Minimum Risk Training for Neural Machine Translation. *Proceedings of the* 53^{rd} *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2016.
(Cited on pages 113 and 141.)

Silver David, Schrittwieser Julian, Simonyan Karen, Antonoglou Ioannis, Huang Aja, Guez Arthur, Hubert Thomas, Baker Lucas, Lai Matthew, Bolton Adrian, Chen Yutian, Lillicrap Timothy, Hui Fan, Sifre Laurent, van den Driessche George, Graepel Thore, and Hassabis Demis. Mastering the game of Go without human knowledge. *Nature*, 2017.
(Cited on page 143.)

Wen Sun, Arun Venkatraman, Geoffrey J. Gordon, Byron Boots, and J. Andrew Bagnell. Deeply AggreVaTeD: Differentiable Imitation Learning for Sequential Prediction. In *Proceedings of the* 34^{th} *International Conference on Machine Learning (ICML)*, 2017.
(Cited on pages 112, 140, and 141.)

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27 (NIPS)*, 2014.
(Cited on pages 104, 107, 116, and 145.)

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
(Cited on pages 114 and 142.)

B. Taskar, C. Guestrin, and D. Koller. Max-Margin Markov Networks. In *Advances in Neural Information Processing Systems 16 (NIPS)*, 2003.
(Cited on page 128.)

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large Margin Methods for Structured and Interdependent Output Variables. *Journal of Machine Learning Research (JMLR)*, 2005.
(Cited on page 124.)

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
(Cited on pages 104, 116, and 145.)

Ronald J Williams and Jing Peng. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 1991.
(Cited on page 135.)

Sam Wiseman and Alexander M Rush. Sequence-to-Sequence Learning as Beam-Search Optimization. In *Proceedings of the 2016 Conference on Empirical Methods in Natural*

Language Processing (EMNLP), 2016.
(Cited on pages 111, 112, 131, 132, and 140.)

Lin Xiao and Tong Zhang. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 2014.
(Cited on pages 26 and 80.)

Yang You, Xiangru Lian, Ji Liu, Hsiang-Fu Yu, Inderjit S Dhillon, James Demmel, and Cho-Jui Hsieh. Asynchronous parallel greedy coordinate descent. In *Advances In Neural Information Processing Systems 29 (NIPS)*, 2016.
(Cited on pages 24 and 80.)

Hsiang-Fu Yu, Hung-Yi Lo, Hsun-Ping Hsieh, Jing-Kai Lou, Todd G McKenzie, Jung-Wei Chou, Po-Han Chung, Chia-Hua Ho, Chun-Fu Chang, Yin-Hsuan Wei, et al. Feature engineering and classifier ensemble for KDD cup 2010. In *KDD Cup*, 2010.
(Cited on page 93.)

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. SeqGAN: Sequence generative adversarial nets with policy gradient. In *Proceedings of the* 31^{st} *AAAI Conference on Artificial Intelligence (AAAI)*, 2017.
(Cited on page 143.)

Shen-Yi Zhao and Wu-Jun Li. Fast Asynchronous parallel stochastic gradient descent. In *Proceedings of the* 30^{th} *AAAI Conference on Artificial Intelligence (AAAI)*, 2016.
(Cited on pages 22, 23, 47, and 48.)

Tuo Zhao, Mo Yu, Yiming Wang, Raman Arora, and Han Liu. Accelerated mini-batch randomized block coordinate descent method. In *Advances in neural information processing systems 27 (NIPS)*, 2014.
(Cited on pages 95 and 96.)

List of Figures

List of Tables

Résumé

Les explosions combinées de la puissance computationnelle et de la quantité de données disponibles ont fait des algorithmes les nouveaux facteurs limitants en machine learning. L'objectif de cette thèse est donc d'introduire de nouvelles méthodes capables de tirer profit de quantités de données et de ressources computationnelles importantes. Nous présentons deux contributions indépendantes.

Premièrement, nous développons des algorithmes d'optimisation rapides, adaptés aux avancées en architecture de calcul parallèle pour traiter des quantités massives de données. Nous introduisons un cadre d'analyse pour les algorithmes parallèles asynchrones, qui nous permet de faire des preuves correctes et simples. Nous démontrons son utilité en analysant les propriétés de convergence et d'accélération de deux nouveaux algorithmes.
ASAGA est une variante parallèle asynchrone et parcimonieuse de SAGA, un algorithme à variance réduite qui a un taux de convergence linéaire rapide dans le cas d'un objectif lisse et fortement convexe. Dans les conditions adéquates, ASAGA est linéairement plus rapide que SAGA, même en l'absence de parcimonie.
PROXASAGA est une extension d'ASAGA au cas plus général où le terme de régularisation n'est pas lisse. PROXASAGA obtient aussi une accélération linéaire.
Nous avons réalisé des expériences approfondies pour comparer nos algorithms à l'état de l'art.

Deuxièmement, nous présentons de nouvelles méthodes adaptées à la prédiction structurée. Nous nous concentrons sur les réseaux de neurones récurrents (RNNs), dont l'algorithme d'entraînement traditionnel – basé sur le principe du maximum de vraisemblance (MLE) – présente plusieurs limitations. La fonction de coût associée ignore l'information contenue dans les métriques structurées ; de plus, elle entraîne des divergences entre l'entraînement et la prédiction.
Nous proposons donc SEARNN, un nouvel algorithme d'entraînement des RNNs inspiré de l'approche dite "learning to search". SEARNN repose sur une exploration de l'espace d'états pour définir des fonctions de coût globales-locales, plus proches de la métrique d'évaluation que l'objectif MLE.
Les modèles entraînés avec SEARNN ont de meilleures performances que ceux appris via MLE pour trois tâches difficiles, dont la traduction automatique. Enfin, nous étudions le comportement de ces modèles et effectuons une comparaison détaillée de notre nouvelle approche aux travaux de recherche connexes.

Mots Clés

optimisation ; parallélisation ; réduction de variance ; prédiction structurée ; RNN.

Abstract

The impressive breakthroughs of the last two decades in the field of machine learning can be in large part attributed to the explosion of computing power and available data. These two limiting factors have been replaced by a new bottleneck: algorithms. The focus of this thesis is thus on introducing novel methods that can take advantage of high data quantity and computing power. We present two independent contributions.

First, we develop and analyze novel fast optimization algorithms which take advantage of the advances in parallel computing architecture and can handle vast amounts of data. We introduce a new framework of analysis for asynchronous parallel incremental algorithms, which enable correct and simple proofs. We then demonstrate its usefulness by performing the convergence analysis for several methods, including two novel algorithms.
ASAGA is a sparse asynchronous parallel variant of the variance-reduced algorithm SAGA which enjoys fast linear convergence rates on smooth and strongly convex objectives. We prove that it can be linearly faster than its sequential counterpart, even without sparsity assumptions.
PROXASAGA is an extension of ASAGA to the more general setting where the regularizer can be non-smooth. We prove that it can also achieve a linear speedup.
We provide extensive experiments comparing our new algorithms to the current state-of-art.

Second, we introduce new methods for complex structured prediction tasks. We focus on recurrent neural networks (RNNs), whose traditional training algorithm for RNNs – based on maximum likelihood estimation (MLE) – suffers from several issues. The associated surrogate training loss notably ignores the information contained in structured losses and introduces discrepancies between train and test times that may hurt performance.
To alleviate these problems, we propose SEARNN, a novel training algorithm for RNNs inspired by the "learning to search" approach to structured prediction. SEARNN leverages test-alike search space exploration to introduce global-local losses that are closer to the test error than the MLE objective.
We demonstrate improved performance over MLE on three challenging tasks, and provide several subsampling strategies to enable SEARNN to scale to large-scale tasks, such as machine translation. Finally, after contrasting the behavior of SEARNN models to MLE models, we conduct an in-depth comparison of our new approach to the related work.

Keywords

optimization; parallelization; variance reduction; structured prediction; RNN.