Practical Optimization Methods for Machine Learning Models

by

Reza Babanezhad Harikandeh

B.Sc., Sharif University of Technology, 2007

M.Sc., Sharif University of Technology, 2011

M.Sc., The University of British Columbia, 2013

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE STUDIES

(Computer Science)

The University of British Columbia

(Vancouver)

December 2019

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Practical Optimization Methods
for Machine Learning Models

submitted by **Reza Babanezhad Harikandeh** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy** in **Computer Science**.

Examining Committee:

Mark Schmidt, Computer Science
Supervisor

Uri Ascher, Computer Science
Supervisory Committee Member

Nick Harvey, Computer Science
Supervisory Committee Member

William Welch, Department of Statistics
University Examiner

Eldad Haber, Department of Earth, Ocean and Atmospheric Sciences
University Examiner

Abstract

This work considers optimization methods for large-scale machine learning (ML). Optimization in ML is a crucial ingredient in the training stage of ML models. Optimization methods in this setting need to have cheap iteration cost. First-order methods are known to have reasonably low iteration costs. A notable recent class of stochastic first-order methods leverage variance reduction techniques to improve their convergence speed. This group includes stochastic average gradient (SAG), stochastic variance reduced gradient (SVRG), and stochastic average gradient amélioré (SAGA). The SAG and SAGA approach to variance reduction use additional memory in their algorithm. SVRG, on the other hand, does not need additional memory but requires occasional full-gradient evaluation. We first introduce variants of SVRG that require fewer gradient evaluations. We then present the first linearly convergent stochastic gradient method to train conditional random fields (CRFs) using SAG. Our method addresses the memory issues required for SAG and proposes a better non-uniform sampling (NUS) technique. The third part of this work extends the applicability of SAGA to Riemannian manifolds. We modify SAGA with operations existing in the manifold to improve the convergence speed of SAGA in these new spaces. Finally, we consider the convergence of classic stochastic gradient methods, based on mirror descent (MD), in non-convex setting. We analyse the MD with more general divergence function and show its application for variational inference models.

Lay Summary

Machine learning algorithms have been applied to various problems related to data processing. These algorithms have a set of parameters which can be optimized using mathematical approaches for the task at hand. That parameter optimization problem is challenging when the data set is massive. The major challenge is the computational cost. This work focuses on improving the optimization methods in terms of their computational cost and their applicability. With our proposed methods, one can train a model in less time and achieve better results. Furthermore, we stretch the applicability of some existing methods to a new set of constraints and models.

Preface

The works presented in this thesis are results of four different collaborative research projects which are accepted in different conferences in computer science.

- The material in Chapter 2 was published in NeurIPS, 2015 (Harikandeh et al., 2015). My responsibility was developing the theory for the different variants of SVRG in this work. Mark Schmidt and Mohamed Ahmed helped in developing new variants and also the experimental section and Jakub Konečný worked on the generalization result. Alim Virani helped in developing the experiments. Scott Sallinen helped in developing idea for the partitioned method.

- The material in Chapter 3 was published in AISTATS, 2015 (Schmidt et al., 2015a). My role was developing the theory for the convergence analysis of SAG with non-uniform sampling, with Aaron Defazio. Adapting SAG for CRF has been done by Mark Schmidt. Mohammed Ahmed was responsible for implementing the experiments and testing various strategies to find a good step-size. Ann Clifton and Anoop Sarkar mainly contributed in developing the idea.

- The material in Chapter 4 was published in ECML, 2018 (Babanezhad et al., 2018). I was responsible for developing the algorithm and its theoretical analysis. Issam H. Laradji was responsible for the experimental section and its writing. Mark Schmidt and Alireza Shafaei helped in developing the idea and writing stages.

- The material in Chapter 5 was published in UAI, 2016 (Khan et al., 2015a). I

was responsible for the convergence proofs of the proposed algorithm. Mohammad E. Khan proposed the method and assessed its application for different variational inference models. Wu Lin worked on the experimental implementations. Mark Schmidt helped in developing the idea and considering different scenarios for the theoretical analysis.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to start by thanking my supervisor Mark Schmidt without whose help this work would not have been possible. Mark guided me all the way, from technical details to planning. I thoroughly enjoyed working with him during my Ph.D. I would like to thank my supervisory committee Nick Harvey and Uri Ascher for their time and feedback. I extend my appreciation to my examiner Jorge Nocedal for his time and valuable feedbacks.

I would also like to thank my co-authors Mohamed, Issam, Wu, Mohammed, and Alireza and my colleagues at UBC, specifically Zainab and Sharan. I appreciate the staff at UBC, especially Joyce Poon and Kath Imhiran for their patience and support.

Finally, I want to thank my friends and family; I thank Sohrab and Behrooz for supporting me through difficult times and last but not least, I express my deepest gratitude toward my family, my parents and sisters, in Iran for their continuous support and encouragement all through my journey.

To my parents.

Chapter 1

Introduction

Machine learning (ML) is a multidisciplinary area that comprises ideas from statistics, computer science, control theory, optimization, pattern recognition, information theory, and many other fields. In traditional computer science theory, different algorithms are developed to handle different computational problems independent of the input data. For example, the quicksort algorithm is oblivious of its input data. However, in ML, a model from a set of models gets picked via an appropriate learning algorithm that leverages the input data for model selection. ML models are practical now thanks to the presence of massive amounts of data and the development of accelerated hardware. ML has successfully performed in applications such as computer vision (Krizhevsky et al., 2012), speech recognition (Hinton et al., 2012), game playing (Silver et al., 2017), and robotics (Kober et al., 2013) with an acceptable accuracy.

1.1 Optimization in ML

One of the essential tasks in ML is "learning" or "training". In this task, given a set of data samples, the learning algorithm picks a model with a high score with respect to some given criteria. If the models belong to a parametric class of models, then the model selection task could be reduced to picking the best parameters that have the best performance concerning the required criteria. The parameter selection can be made using some optimization technique.

To be more formal, we can look at the supervised learning problem. In this problem, a sample data set $\{(a_i, b_i)\}_{i=1}^n \sim \mathscr{D}_{\mathscr{A},\mathscr{B}}$ is given, and $\mathscr{D}_{\mathscr{A},\mathscr{B}}$ is the actual data distribution and unknown. In each sample, a_i represents the observed feature vector, and b_i is the corresponding label. The goal of a learning algorithm is to find a model $h : \mathscr{A} \to \mathscr{B}$ that predicts the label of a new feature vector. In order to find the target predictor, a learning algorithm uses a loss function f to measure the success of a given predictor on a random data point. Finding the best predictor amounts to searching for the parameters that minimize the risk function. Let's assume that h_x is a predictor function and parametrized by x. Then the risk function for a given h_x is defined by

$$L_{(\mathscr{D}_{\mathscr{A},\mathscr{B}},f)}(h_x) = \mathbb{E}_{\mathscr{D}_{\mathscr{A},\mathscr{B}}}[f(h_x(a), b)] = \int_{\mathscr{A}\times\mathscr{B}} f(h_x(a), b) d\mathscr{D}(a, b).$$

Optimizing that risk functional with respect to the predictor parameter is the objective of a learning algorithm:

$$x^* = \underset{x\in\mathscr{X}}{\operatorname{argmin}} L_{(\mathscr{D}_{\mathscr{A},\mathscr{B}},f)}(h_x). \tag{1.1}$$

Finding the best parameters in (1.1) requires us to find the output of an integral that is not possible since the true distribution $\mathscr{D}_{\mathscr{A},\mathscr{B}}$ is unknown. Hence in practice, the numerical computation methods are adapted to approximate the exact risk function. In particular, we use independent and identical distribution (IID) samples as a Monte Carlo approximation. Therefore instead of minimizing the actual risk, the empirical risk gets minimized:

$$\hat{L}_{(\mathscr{D}_{\mathscr{A},\mathscr{B}},f)}(h_x) = \frac{1}{n}\sum_{i=1}^{n} f(h_x(a_i), b_i). \tag{1.2}$$

To make the notation more convenient, we shorten $f(h_x(a_i), b_i)$ to $f_i(x)$. It has been shown that adding appropriate regularizers can improve the generalizability of the model (Shalev-Shwartz and Ben-David, 2014). The general objective function thus comprises of two parts: a finite sum structure and a regularizer:

$$\min_{x\in\mathscr{X}} f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x) + \lambda g(x), \tag{1.3}$$

where f could be convex like the objective in a linear regression model or non-convex such as the objective of a deep neural network (DNN).

Empirical risk minimization (ERM) is not the only place that optimization techniques appear during the training of an ML model. Generally, minimizing any risk function over a class of parametrized models requires some optimization method. In the Bayesian framework, instead of finding one true parameter, a distribution over the valid parameter "posterior of the parameters" is inferred via employing the Bayes' rule. Formally we assume that data are sampled from a distribution $p(a \mid \theta)$ where a and θ are random variables. Assuming a prior distribution for θ, $p(\theta)$, the posterior distribution of θ can be formulated via applying Bayes rule:

$$p(\theta|a) \propto p(a|\theta)p(\theta).$$

However, working with $p(\theta|a)$ is required to do prediction or inference and for a big set of models is intractable. Therefore in the technique which is called variational inference(VI) (Murphy, 2012), the true posterior is approximated with another parametrized distribution, $q(\theta|x)$ that is tractable to use. To choose a suitable parameter, an appropriate divergence between these two distributions gets minimized with regards to the parameter x. For example, KL-divergence is one of the most popular divergence function which has been used in different applications (Kingma and Welling, 2013; Rezende and Mohamed, 2015):

$$x^* = \underset{x \in \mathscr{X}}{\operatorname{argmin}} \mathscr{D}_{KL}[q(\theta \mid x) \| p(\theta|a)] = \int q(\theta \mid x) \log \left(\frac{q(\theta \mid x)}{p(\theta \mid a)} \right) dx.$$

Similar to the risk minimization, this integral can be approximated using numerical methods.

1.1.1 Examples of Loss functions

In this section, we review two of the most common ML models and their loss functions, namely logistic regression and support vector machine (SVM).

Logistic Regression

This model can be applied for supervised binary classification problems. The dataset $\{(a_i, b_i)\}_{i=1}^n$ contains IID samples coming from $\mathscr{D}_{\mathscr{A},\mathscr{B}}$ where $\mathscr{A} \subseteq \mathbb{R}^d$ and $\mathscr{B} = \{-1, 1\}$. The logistic model is a probabilistic model that considers the logistic function, $\sigma(a) = \frac{1}{1+\exp(\langle -x,a\rangle)}$, to determine the probability of assigning a class to a data point. This function assign probability to a Bernoulli random variable. Formally,

$$P(b \mid a, x) = \frac{1}{1+\exp(\langle -x, ba\rangle)}.$$

The model minimizes the negative log-likelihood over all n examples to choose a value for the parameter x:

$$\min_x f(x) = \frac{1}{n}\sum_{i=1}^{n} \log(1+\exp(\langle -x, b_i a_i\rangle)).$$

Here $f(x)$ is called the logistic loss function and it is a smooth convex function. Depending on the dataset, $\{(a_i, b_i)\}_{i=1}^n$, f can be strongly convex which makes optimization algorithms converge faster than for general convex functions; however, a more convenient way to make it strongly convex is adding an $l2$-regularizer:

$$\min_x f(x) = \frac{1}{n}\sum_{i=1}^{n} \log(1+\exp(\langle -x, b_i a_i\rangle)) + \frac{\lambda}{2}\|x\|_2^2.$$

Support Vector Machine (SVM)

An SVM is a linear binary classifier. SVMs use a training set $\{(a_i, b_i)\}_{i=1}^n$ to find an optimal value for its parameter. This training set is called linearly separable if there is a half-space (x, y) such that $b_i = sign(\langle x, a_i\rangle + y)$ for all i or we have $\forall i,\ b_i(\langle x, a_i\rangle + y) > 0$. To make the notation simpler, we assume that y is given. There is an infinite number of separating hyperplanes x. The objective of the SVM model is to select a hyperplane that has the maximum margin (Shalev-Shwartz and Ben-David, 2014). That objective in the separable case can be formalized in the

following optimization problem:

$$\begin{aligned} &\min_{x} \|x\|_2^2 \\ &s.t.\ \forall i \quad b_i \langle x, a_i \rangle \geq 1. \end{aligned}$$

However, this objective only has a solution when the data points are separable. When they are not separable, the $b_i(\langle x, a_i \rangle) \geq 1$ condition does not hold for all data points. To adapt for this situation, we change the condition to $b_i(\langle x, a_i \rangle) \geq 1 - \zeta_i$ and the new objective would be:

$$\begin{aligned} &\min_{x,\zeta} \frac{\lambda}{2}\|x\|_2^2 + \frac{1}{n}\sum_{i=1}^{n} \zeta_i \\ &s.t.\ \forall i \quad b_i \langle x, a_i \rangle \geq 1 - \zeta_i. \end{aligned} \tag{1.4}$$

By writing the Lagrangian form of (1.4) and doing some algebraic manipulation, the following equivalent formulation can be achieved:

$$\min_{x} \frac{\lambda}{2}\|x\|_2^2 + \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - b_i \langle x, a_i \rangle\},$$

where $L^h(z) = \max\{0, 1-z\}$ is called hinge loss. This loss function is convex but not smooth. A smoothed version of this loss is called the Huberized hinge loss where:

$$L^{\text{Huber}}(z) = \begin{cases} 0 & \text{if } z \geq 1 \\ \frac{1}{2} - z & \text{if } z \leq 0 \\ \frac{1}{2}(1-z)^2 & \text{otherwise} \end{cases}$$

Figure 1.1 depicts the hinge loss and its smoothed version.

1.2 Big Data and Optimization

One of the main reason behind the current advancement in the development of ML models and their application in real life problems is the existence of an immense amount of data to train a model. Nowadays, every transaction which can be recorded in some storage medium provides a data point for an ML model. Due

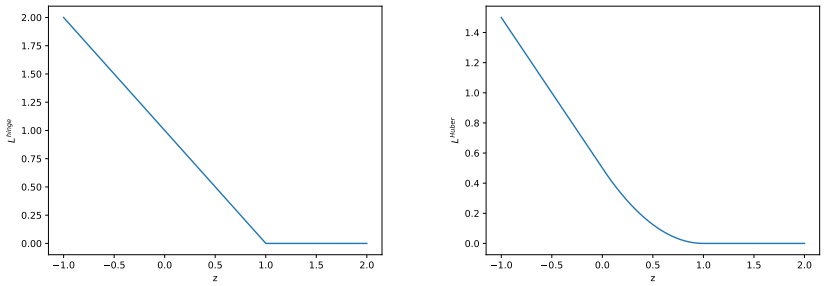

Figure 1.1: The hinge loss (left) is non-smooth at point $z = 1$, whereas the Huber loss (right) is smooth everywhere (Rennie and Srebro, 2005).

to sharing infrastructures provided by the web-giants such as Google, Facebook, and Amazon, these companies' customers provide various types of data for ML algorithms by uploading their information. Developing models that can handle the complexity of big data is a significant challenge in ML research.

Optimization is one of the most critical parts of most state of the art ML models such as DNNs. The optimization algorithms must be able to handle a massive dataset in order to train a practical ML model. The optimization algorithms could consider two issues: (1) Using parallel hardware and (2) having cost-efficient or scalable algorithms. The first issue is utilizing hardware parallelism to process the samples existing in a dataset concurrently so that the speed of processing of the whole dataset will be roughly equal to a single data point. Although this approach is an active area in the optimization domain, it is not the focus of this thesis. The second issue is developing optimization algorithms whose computational cost is growing linearly with regards to the size of a dataset.

In this thesis, we consider first-order optimization methods. In this class of algorithms, it is assumed that there exists a first-order oracle which gets a function f and a point x in its domain as input and returns the function value $f(x)$ and a sub-gradient $\partial f(x)$. When f is smooth at x, the sub-gradient is equal to the gradient $\nabla f(x)$. However, when f is non-smooth at x, $\partial f(x)$ could be any vector in the subdifferential of f at x. Formally, let $\mathscr{O}$ be the oracle, then for a smooth function

f we have:

$$\mathscr{O}(f,x) = (f(x), \nabla f(x)).$$

Therefore these methods get all their information from a first-order oracle. The cost of calling a first-order oracle depends on the input function's formula and its dependence on the dimensionality. For example for the quadratic function $f(x) = x^T Ax$ where $x \in \mathbb{R}^d$ and $A \in \mathbb{R}^{d \times d}$ is a dense matrix, the cost of calling $\mathscr{O}$ is of $O(d^2)$. For objectives like SVM and logistic regression this cost is of $O(d)$. The space cost to call the first-order oracle is of $O(d)$.

1.2.1 Iterative Methods

Almost all the optimization algorithms that are useful for current ML loss functions are iterative methods. This means that they start from some given point in the domain, and gradually progress toward a final optimal point. Therefore we can show their update rule at the iteration t by the following formulation:

$$x_{t+1} = \delta(x_t, g_t, \eta_t, K_t),$$

where x_t is the current value of the iterate and x_{t+1} is the updated variable. The update direction is specified by g_t, which is usually an estimator of the gradient of the objective function at x_t. The magnitude of the update is determined by the step size η_t. Any additional parameters of a specific method are denoted via K_t. For example in the gradient descent (GD) algorithm, $g_t = \nabla f(x_t)$, $K_t = \emptyset$, and $\delta = x_t - \eta_t g_t$. A variety of methods have been developed to set this update function and its parameters for various type of objective functions.

These methods can be compared based on time complexity, space complexity, and their practicality. To measure the time complexity of an iterative method, we consider two metrics:

- **Iteration cost**: the computational cost per iteration is mainly related to the evaluation of the function δ and its parameters. This cost includes the cost of calling the first order oracle that involves the evaluation of the gradient of the objective function at a given point. For example, in GD, this cost is $O(d)$ for the update rule plus the cost of computing the gradient.

- **Convergence rate**: the number of iterations required to reach a solution with accuracy less than some ε. Two types of convergence rates that we deal with in this manuscript are sublinear and linear. Let $\{a_t > 0\}$ be a sequence converging toward zero, in other words $\lim_{t\to\infty} a_t \to 0$. We say a_t has sublinear convergence rate if $a_t \leq O(t^{-\alpha})C$, where $\alpha > 0$ and C is a constant. Alternatively we say a_t has linear (or "exponential") convergence rate if $a_t \leq O(\rho^t)C$, where $0 < \rho < 1$ and C is a constant. For example, GD has a linear convergence rate for smooth and strongly-convex function and the sequences that converge to zero are $\left\{\|x_t - x^*\|_2^2\right\}$, $\{f(x_t) - f(x^*)\}$, and $\left\{\|\nabla f(x_t)\|_2^2\right\}$, where x^* is the minimizer of f.

The space cost is related to the amount of memory space the algorithm requires to save its parameters and variables. For GD, this cost is only $O(d)$ plus the memory needed in the gradient evaluation. We typically do not count the space used to store or load the dataset.

In terms of practicality, we look at how hard it is to tune the parameters of optimization algorithms. These algorithms usually have a couple of parameters, such as the step size, and tuning them to the optimal value can improve the convergence speed. There is not always a fixed optimal value for a parameter, but we need to find a schedule for that parameter's tuning throughout the iterates. For example, stochastic gradient descent (SGD) (Robbins and Monro, 1951a) cannot converge with a fixed step size; hence, there is not an optimal constant step size value. Instead, it needs a decreasing sequence of step sizes to achieve convergence. Needless to say that finding an optimal decreasing sequence of step sizes seems harder than finding a fixed optimal step size.

In the following sections, we first review some concepts and definitions related to convex optimization and then briefly describe some of the most common iterative optimization methods and their issues.

Background

The objective function we consider has the following general form:

$$\min_{x\in\mathscr{X}} f(x). \tag{1.5}$$

Depending on the problem at hand, we assume different structures for the objective function f. Here are the definitions of some structures we consider in the rest of this manuscript:

- **Convexity**. A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex iff for all $x, y \in \mathscr{X}$, and any $\alpha \in [0,1]$:
$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y).$$

- **Strong-convexity**. A convex function $f : \mathbb{R}^d \to \mathbb{R}$ is μ-strongly-convex iff for all $x, y \in \mathscr{X}$, there exists a constant $\mu > 0$ such that:
$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y) - \frac{\mu \alpha (1-\alpha)}{2} \|x-y\|_2^2.$$

- **Lipschitz smoothness**. A differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is L-Lipschitz smooth iff for all $x, y \in \mathscr{X}$, there exists a constant $L > 0$ such that
$$\|\nabla f(x) - \nabla f(y)\|_2 \leq L \|x-y\|_2.$$

- **Finite sum**. A function f is a sum of a finite number of other functions:
$$f(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x),$$
where $n < \infty$.

Gradient Descent

Gradient descent (GD) is one of the first and most significant iterative algorithms to solve (1.5). The update rule of GD is:

$$x_{t+1} = x_t - \eta_t \nabla f(x_t).$$

At each iteration t, it evaluates the gradient at the current iterate x_t and then moves in the opposite direction of that vector. The step size η_t can be set to a fixed value or can be optimized per iteration using a line search algorithm. When the objective function is strongly convex and smooth, this algorithm converges linearly with a

sufficiently small fixed step size (Nesterov, 2004). However, when the objective function has a finite-sum structure with a large n, then its iteration cost which is $O(nd)$ could be large as well. This cost makes GD unsuitable for the training of ML models with a massive amount of data.

Stochastic Gradient Descent

Compared to GD, stochastic gradient descent (SGD) has a low cost to evaluate an objective with finite sum structure. Generally, SGD can be utilized to minimize an objective function with the following structure:

$$\min_{x \in \mathscr{X}} f(x) = \mathbb{E}_{\zeta}\left[f(x,\zeta)\right],$$

where ζ is a random variable coming from a known distribution p. For an objective with finite-sum structure, we have $\zeta \in \{1,...n\}$, $P(\zeta = i) = 1/n$, and $f(x,\zeta) = f_{\zeta}(x)$. Note that for this objective we assume that a stochastic first order oracle is provided which means the oracle accepts an extra parameter ζ in addition to f and x. Here is the SGD's iteration at step t:

$$\begin{aligned} &\zeta_t \sim P \\ &x_{t+1} = x_t - \eta_t \nabla f(x_t, \zeta_t). \end{aligned}$$

In each iteration, first a sample ζ_t gets generated from p, then the gradient of $\nabla f(x_t, \zeta_t)$ is evaluated by the stochastic oracle. This gradient is a stochastic approximation for the full gradient $\nabla f(x_t)$. It is usually assumed that this estimation is unbiased, in other words $\mathbb{E}_{\zeta}\left[\nabla f(x_t, \zeta_t)\right] = \nabla f(x_t)$. Then similar to GD, a step is taken along the opposite direction of the stochastic gradient. For the finite-sum problem, the update rule would be:

$$\begin{aligned} &i_t \sim Unif\{1...n\} \\ &x_{t+1} = x_t - \eta_t \nabla f_{i_t}(x_t), \end{aligned}$$

where here $i_t = \zeta_t$ and p is a uniform distribution. Clearly $\nabla f_{i_t}(x_t)$ is an unbiased

estimator for $\nabla f(x_t)$ since:

$$\mathbb{E}_{i_t}\left[\nabla f_{i_t}(x_t)\right] = \frac{1}{n}\sum_{i=1}^{n}\nabla f_i(x_t) = \nabla f(x_t).$$

SGD has a cheaper iteration cost than GD for the finite-sum problem since it only needs one gradient computation per iteration. However, it converges slower than GD and only achives a sublinear convergence rate (Bach and Moulines, 2011). Furthermore, it does not converge with a fixed step size and requires a diminishing step size. In theory, its step size is needed to regress toward zero with the rate of $t^{-\alpha}$ for some $\alpha \in [1/2, 1]$ (Bach and Moulines, 2011), but in practice, we can pick bigger step size for $\alpha < 1/2$. Therefore, setting a good step size schedule is not an easy task for SGD which limits its practicality.

Variance Reduction based Methods

It has been shown that the main reason for the slow convergence rate in SGD is the variance of the stochastic gradient estimator. One way to measure this variance for the finite-sum problem at iteration t is:

$$\frac{1}{n}\sum_{i=1}^{n}\|\nabla f_{i_t}(x_t) - \nabla f(x_t)\|_2^2.$$

Variance reduction techniques have been introduced to battle this problem and also improve the convergence rate of SGD while keeping the iteration cost as low as possible. This stream of optimization methods was started with the introduction of the stochastic average gradient (SAG) (Le Roux et al., 2012) algorithm. The main building block of the SAG algorithm is an extra memory. Let M denote the memory of size $n \times d$. Corresponding to each function f_i, there is a cell in the memory, $M[i]$, which stores the gradient of f_i at some point. Incorporating the extra memory helps SAG to reduce the variance of the stochastic gradient estimator. Here is the update

rule of this algorithm:

$$
\begin{aligned}
&i_t \sim Unif\{1...n\} \\
&M[i_t] = \nabla f_{i_t}(x_t) \\
&x_{t+1} = x_t - \eta_t \left(\frac{1}{n} \sum_{i=1}^{n} M[i] \right).
\end{aligned}
$$

In each iteration, a function is selected randomly; then its gradient gets evaluated and stored in memory; finally a jump in the opposite direction of the average of stored value in the memory is taken.

Similar to SGD, SAG has a low iteration cost, which makes it feasible when n is large. It can be shown that for strongly-convex and smooth functions, SAG can converge linearly similar to GD. For many problems, we can find a range of fixed step sizes analytically, and by setting the SAG step size to any of those values, get convergence to the minimizer. However, SAG has some issues as well. First and foremost, it usually requires extra memory of the same size as the dataset. Also, its gradient estimator $\frac{1}{n}\sum_{i=1}^{n} M[i]$ is not unbiased due to stale gradients stored in the memory, which makes the convergence analysis of the method hard.

Following SAG, alternative variance reduction based methods have been introduced such as stochastic average gradient amélioré (SAGA) (Defazio et al., 2014), stochastic dual coordinate ascent (SDCA) (Shalev-Shwartz and Zhang, 2013a), minimization by incremental surrogate optimization (MISO) (Mairal, 2014) and stochastic variance reduced gradient (SVRG) (Johnson and Zhang, 2013). SVRG solves the additional memory requirement of the SAG method by computing the full gradient occasionally at the expense of increasing the iteration cost. SAGA combines the update rule of SAG and SVRG in a way that makes the gradient estimator an unbiased one.

Finally, we consider the usability of the variance reduction techniques beyond the convex minimization problems. For some non-convex problems when the objective function satisfies the so-called gradient dominance property or PL condition (Karimi et al., 2016), it has been shown that SVRG can converge linearly to the global optima (Allen-Zhu and Yuan, 2016; Reddi et al., 2016). However, for a complex non-convex loss function such as neural network models, these techniques

don't give us a faster convergence rate than SGD (Defazio and Bottou, 2019; Reddi et al., 2017). One possible explanation could be that the variance existing in the stochastic approximation of the gradient can help the optimization algorithm to escape from bad local minima.

SVRG

Stochastic variance reduced gradient (SVRG) (Johnson and Zhang, 2013) removes the extra memory requirement in SAG by adding extra gradient evaluations. Similar to other fast variance reduction based methods, it requires that the objective function have the finite-sum structure. The main idea of SVRG is to compute the full gradient of an objective function at some pivot point and leverage it for reducing the variance of gradient estimator. Here is the update rule for SVRG:

$$
\begin{aligned}
&i_t \sim Unif\{1...n\} \\
&x_{t+1} = x_t - \eta_t \left(\nabla f_{i_t}(x_t) - \nabla f_{i_t}(\tilde{x}) + \nabla f(\tilde{x})\right),
\end{aligned}
$$

where $\tilde{x}$ is the pivot point and gets updated occupationally. It has an unbiased gradient estimator such that when x_t converges toward x^*, that estimator converges toward full gradient $\nabla f(x_t)$. Therefore through the iterations, this estimator's variance gradually decreases toward zero. This idea is similar to the concept of control variates in statistical methods for reducing the variance of a random estimator (Owen, 2014).

SVRG converges linearly for smooth and strongly-convex function with a fixed step size. But its iteration cost is higher than SAG since it requires two gradient evaluation per iteration. Further, it occasionally needs to compute the full gradient at a pivot point.

Mirror Descent

The mirror descent algorithm (MDA) (Beck and Teboulle, 2003) is an optimization algorithm that considers the geometrical structure of the objective function. We can view MDA as a generic version of the projected (sub)gradient descent algorithm.

In the projected subgradient algorithm, the update rule is:

$$x_{t+1} \in \underset{x \in \mathscr{X}}{\operatorname{argmin}} \left\{ \langle x, \partial f(x_t) \rangle + \frac{1}{2\eta_t} \|x - x_t\|_2^2 \right\}. \tag{1.6}$$

MDA replaces the squared Euclidean distance in (1.6) with a more general Bregman divergence B_Φ. A Bregman divergence that is induced by a smooth and strongly-convex function Φ is defined as follows:

$$B_\Phi(x, y) = \Phi(x) - \Phi(y) - \langle x - y, \nabla\Phi(y) \rangle .$$

Therefore, the MDA's update is:

$$x_{t+1} \in \underset{x \in \mathscr{X}}{\operatorname{argmin}} \left\{ \langle x, \partial f(x_t) \rangle + \frac{1}{\eta_t} B_\Phi(x, x_t) \right\}. \tag{1.7}$$

The convergence of MDA for convex and non-convex objectives has been analyzed in (Beck and Teboulle, 2003; Ghadimi et al., 2014). Those analyses depend on the properties of the Bregman divergence and the order of its parameters in (1.7). But in some applications, such as VI for "exponential distribution families", the divergence function does not comply with the properties required by those analyses. Hence a new convergence analysis is required for these new situations.

1.3 Summary of Contributions

In this thesis, we focus on improving the existing stochastic methods for convex and non-convex optimization. We first discuss how we can reduce the iteration cost in SVRG and also improve the convergence speed in SAGA using non-uniform sampling. Then we adopt the SAGA algorithm for non-Euclidean spaces. Finally we generalize the non-convex SGD analysis beyond mirror descent to handle an important class of non-convex functions. Our list of contributions is as follows:

- Chapter 2: we consider the SVRG algorithm which does not require additional memory such as SAG. However, it needs more gradient evaluations than SAG in every iteration. We introduce variants of the SVRG algorithm that are more efficient in terms of gradient evaluation.

- Chapter 3: we consider the SAGA algorithm, which is the unbiased version of the SAG algorithm. We study that algorithm for the conditional random field problem and also introduce variants of it which have better convergence speed using a non-uniform sampling scheme.

- Chapter 4: we consider the optimization problem in non-Euclidean spaces. Specifically we consider Riemannian manifolds in our analysis and models such as principal component analysis (PCA) could be analysed in this framework. Using the structure of a space can help to improve the convergence speed and reduce the computational cost of an optimization algorithm. We adapt the SAGA algorithm for this new setting and show that the modified version still provides a linear convergence rate.

- Chapter 5: we consider the optimization problem of divergence minimization occurring in variational inference (VI). In this problem, the objective function is non-convex and does not follow the finite-sum structure. Further VI does not fall in the MD framework. We analyze the gradient descent and stochastic gradient descent update rules for this setting.

Chapter 2

Practical SVRG

In this chapter we consider the objective function with finite-sum structure:

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

As we have seen before, minimizing the empirical risk has the above structure. A huge proportion of the model-fitting procedures in machine learning can be mapped to this problem. This includes classic models like least squares and logistic regression but also includes more advanced methods like conditional random fields and deep neural network models. In the high dimensional setting (large d), the traditional approaches for solving (2.1) are: full gradient (FG) methods which have linear convergence rates for smooth and strongly-convex f, but need to evaluate the gradient f_i for all n examples on every iteration, and stochastic gradient (SG) methods which make rapid initial progress as they only use a single gradient on each iteration but ultimately have slower sublinear convergence rates.

Le Roux et al. (2012) proposed the first general method, *stochastic average gradient* (SAG), that only considers one training example on each iteration but still achieves a linear convergence rate. Other methods have subsequently been shown to have this property (Defazio et al., 2014; Mairal, 2013a; Shalev-Shwartz and Zhang, 2013b), but these all require storing a previous evaluation of the gradient ∇f_i or the dual variables for each i. For many objectives this only requires $O(n)$ space, but for general problems this requires $O(nd)$ space making them impractical.

Recently, several methods have been proposed with similar convergence rates to SAG but without the memory requirements (Johnson and Zhang, 2013; Konečnỳ and Richtárik, 2013; Mahdavi and Jin, 2013; Zhang et al., 2013). They are known as *mixed gradient*, *stochastic variance-reduced gradient* (SVRG), and *semi-stochastic gradient* methods (we will use SVRG). We give a canonical SVRG algorithm in the next section, but the salient features of these methods are that they evaluate two gradients on each iteration and occasionally must compute the gradient on all examples. SVRG methods often dramatically outperform classic FG and SG methods, but these extra evaluations mean that SVRG is slower than SG methods in the important early iterations. They also mean that SVRG methods are typically slower than memory-based methods like SAG.

In this chapter, we first show that SVRG is robust to inexact calculation of the full gradients it requires, provided the accuracy increases over time. We use this to explore growing-batch strategies that require fewer gradient evaluations when far from the solution, and we propose a mixed SGD/SVRG method that may improve performance in the early iterations. We next explore using support vectors to reduce the number of gradients required when close to the solution, give a justification for the regularized SVRG update that is commonly used in practice, consider alternative mini-batch strategies, and finally consider the generalization error of the method.

2.1 Notation and SVRG Algorithm

SVRG assumes f is μ-strongly convex, each f_i is convex, and each gradient ∇f_i is Lipschitz-continuous with constant L. The method begins with an initial estimate x^0, sets $x_0 = x^0$ and then generates a sequence of iterates x_t using

$$x_{t+1} = x_t - \eta(\nabla f_{i_t}(x_t) - \nabla f_{i_t}(x^s) + \nabla f(x^s)), \tag{2.2}$$

where η is the positive step size, and i_t is chosen uniformly from $\{1,2,\dots,n\}$. After every m steps, we set $x^{s+1} = x_t$ for a random $t \in \{1,\dots,m\}$, and we reset $t = 0$ with $x_0 = x^{s+1}$. The vanilla SVRG algorithm is presented in Algorithm 1.

To analyze the convergence rate of SVRG, we will find it convenient to define

Algorithm 1 SVRG

Input: initial vector x^0, update frequency m, learning rate η.
for $s = 0, 1, 2, \ldots$ **do**
  $g^s = \nabla f(x^s) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x^s)$
  $x_0 = x^s$
  for $t = 1, 2, \ldots, m$ **do**
    Randomly pick $i_t \in \{1, \ldots, n\}$
    $x_{t+1} = x_t - \eta(\nabla f_{i_t}(x_t) - \nabla f_{i_t}(x^s) + g^s)$ $(*)$
  end for
  option I: set $x^{s+1} = x_m$
  option II: set $x^{s+1} = x_t$ for random $t \in \{1, \ldots, m\}$
end for

the function

$$\rho(a,b) = \frac{1}{1-2\eta a}\left(\frac{1}{m\mu\eta} + 2b\eta\right),$$

as it appears repeatedly in our results. We will use $\rho(a)$ to indicate the value of $\rho(a,b)$ when $a = b$, and we will simply use ρ for the special case when $a = b = L$. Johnson and Zhang (2013) show that if η and m are chosen such that $0 < \rho < 1$, the algorithm achieves a linear convergence rate of the form

$$\mathbb{E}[f(x^{s+1}) - f(x^*)] \leq \rho\mathbb{E}[f(x^s) - f(x^*)],$$

where x^* is the optimal solution. This convergence rate is very fast for appropriate η and m. While this result relies on constants we may not know in general, practical choices with good empirical performance include setting $m = n$, $\eta = 1/L$, and using $x^{s+1} = x_m$ rather than a random iterate.

Unfortunately, the SVRG algorithm requires $2m + n$ gradient evaluations for every m iterations of (2.2), since updating x_t requires two gradient evaluations and computing g^s require n gradient evaluations. We can reduce this to $m+n$ if we store the gradients $\nabla f_{i_t}(x^s)$, but this is not practical in most applications. Thus, SVRG requires many more gradient evaluations than classic SGD iterations or memory-based methods like SAG.

2.2 SVRG with Error

We first give a result for the SVRG method where we assume that g^s is equal to $\nabla f(x^s)$ up to some error e^s. This is in the spirit of the analysis of Schmidt et al. (2011), who analyze GD methods under similar assumptions. We assume that $\|x_t - x^*\|_2 \leq Z$ for all t, which has been used in related work (Hu et al., 2009) and is reasonable because of the coerciveness implied by strong-convexity.

Theorem 2.1. *Assume f is μ-strongly convex and each f_i is L-Lipschitz smooth. Furthermore assume $\|x_t - x^*\|_2 \leq Z$ holds for every iteration. If $g^s = \nabla f(x^s) + e^s$ and we set η and m so that $\rho < 1$, then the SVRG algorithm* (2.2) *with x^{s+1} chosen randomly from $\{x_1, x_2, \ldots, x_m\}$ satisfies*

$$\mathbb{E}\left[f(x^{s+1}) - f(x^*)\right] \leq \rho \mathbb{E}\left[f(x^s) - f(x^*)\right] + \frac{Z\mathbb{E}\left[\|e^s\|_2\right] + \eta \mathbb{E}\left[\|e^s\|_2^2\right]}{1 - 2\eta L}.$$

We give the proof in Appendix A.1.1. This result implies that SVRG does not need a very accurate approximation of $\nabla f(x^s)$ in the crucial early iterations since the first term in the bound will dominate. Further, this result implies that we can maintain the *exact* convergence rate of SVRG as long as the errors e^s decrease at an appropriate rate. For example, we obtain the same convergence rate provided that $\max\{\mathbb{E}\left[\|e^s\|_2\right], \mathbb{E}\left[\|e^s\|_2^2\right]\} \leq \gamma\tilde{\rho}^s$ for any $\gamma \geq 0$ and some $\tilde{\rho} < \rho$. Further, we still obtain a linear convergence rate as long as $\|e^s\|_2$ converges to zero with a linear rate.

2.2.1 Non-Uniform Sampling

Xiao and Zhang (2014) show that non-uniform sampling (NUS) improves the performance of SVRG. They assume each ∇f_i is L_i-Lipschitz continuous, and sample $i_t = i$ with probability $L_i/n\bar{L}$ where $\bar{L} = \frac{1}{n}\sum_{i=1}^n L_i$. The iteration is then changed to

$$x_{t+1} = x_t - \eta\left(\frac{\bar{L}}{L_{i_t}}\left[\nabla f_{i_t}(x_t) - \nabla f_{i_t}(x^s)\right] + g^s\right),$$

which maintains that the search direction is unbiased. Here g^s can be an exact or approximate evaluation for $\nabla f(x^s)$. In Appendix A.1.2, we show that if g^s is computed with error for this algorithm and if we set η and m so that $0 < \rho(\bar{L}) < 1$,

Algorithm 2 Batching SVRG

Input: initial vector x^0, update frequency m, learning rate η.
for $s = 0, 1, 2, \ldots$ **do**
  Choose batch size $|\mathscr{B}^s|$
  $\mathscr{B}^s = |\mathscr{B}^s|$ elements sampled without replacement from $\{1, 2, \ldots, n\}$.
  $g^s = \frac{1}{|\mathscr{B}^s|} \sum_{i \in \mathscr{B}^s} \nabla f_i(x^s)$
  $x_0 = x^s$
  for $t = 1, 2, \ldots, m$ **do**
    Randomly pick $i_t \in \{1, \ldots, n\}$
    $x_{t+1} = x_t - \eta(\nabla f_{i_t}(x_t) - \nabla f_{i_t}(x^s) + g^s)$ $(*)$
  end for
  option I: set $x^{s+1} = x_m$
  option II: set $x^{s+1} = x_t$ for random $t \in \{1, \ldots, m\}$
end for

then we have a convergence rate of

$$\mathbb{E}\left[f(x^{s+1}) - f(x^*)\right] \leq \rho(\bar{L})\mathbb{E}\left[f(x^s) - f(x^*)\right] + \frac{Z\mathbb{E}\left[\|e^s\|_2\right] + \eta\mathbb{E}\left[\|e^s\|_2^2\right]}{1 - 2\eta\bar{L}},$$

which can be faster since the average $\bar{L}$ may be much smaller than the maximum value L.

2.2.2 SVRG with Batching

There are many ways we could allow an error in the calculation of g^s to speed up the algorithm. For example, if evaluating each ∇f_i involves solving an optimization problem, then we could solve this optimization problem inexactly. For example, if we are fitting a graphical model with an iterative approximate inference method, we can terminate the iterations early to save time.

When the f_i are simple but n is large, a natural way to approximate g^s is with a subset (or 'mini-batch') of training examples $\mathscr{B}^s$ (chosen *without* replacement),

$$g^s = \frac{1}{|\mathscr{B}^s|} \sum_{i \in \mathscr{B}^s} \nabla f_i(x^s).$$

This is justified because when we are far from the solution, most $\nabla f_i(x^s)$ likely

point in directions of progress. The batch size $|\mathscr{B}^s|$ controls the error in the approximation, and we can drive the error to zero by increasing it to n. Existing SVRG methods correspond to the special case where $|\mathscr{B}^s| = n$ for all s and we sample without replacement.

Algorithm 2 gives pseudo-code for an SVRG implementation that uses this sub-sampling strategy. If we assume that the sample variance of the norms of the gradients is bounded by S^2 for all x^s,

$$\frac{1}{n}\sum_{i=1}^{n}\left[\|\nabla f_i(x^s)\|_2^2 - \|\nabla f(x^s)\|_2^2\right] \leq S^2,$$

then we have that (Lohr, 2009, Chapter 2)

$$\mathbb{E}\left[\|e^s\|_2^2\right] \leq \frac{n-|\mathscr{B}^s|}{n|\mathscr{B}^s|}S^2.$$

So if we want $\mathbb{E}\left[\|e^s\|_2^2\right] \leq \gamma\tilde{\rho}^{2s}$, where $\gamma \geq 0$ is a constant for some $\tilde{\rho} < 1$, we need

$$|\mathscr{B}^s| \geq \frac{nS^2}{S^2 + n\gamma\tilde{\rho}^{2s}}. \tag{2.3}$$

If the batch size satisfies the above condition then

$$\begin{aligned} Z\mathbb{E}\left[\|e^{s-1}\|_2\right] + \eta\mathbb{E}\left[\|e^{s-1}\|_2^2\right] &\leq Z\sqrt{\gamma}\tilde{\rho}^s + \eta\gamma\tilde{\rho}^{2s} \\ &\leq 2\max\{Z\sqrt{\gamma}, \eta\gamma\tilde{\rho}\}\tilde{\rho}^s, \end{aligned}$$

and *the convergence rate of SVRG is unchanged compared to using the full batch* on all iterations.

The condition (2.3) guarantees a linear convergence rate under any exponentially-increasing sequence of batch sizes, the strategy suggested by Friedlander and Schmidt (2012) for classic SG methods. However, a tedious calculation shows that (2.3) has an inflection point at $s = \log(S^2/\gamma n)/2\log(1/\tilde{\rho})$, corresponding to $|\mathscr{B}^s| = \frac{n}{2}$. This was previously observed empirically (Aravkin et al., 2012, Figure 3), and occurs because we are sampling without replacement. This transition means we don't need to increase the batch size exponentially.

Algorithm 3 Mixed SVRG and SG Method

Replace (*) in Algorithm 1 with the following lines:
if $f_{i_t} \in \mathscr{B}^s$ **then**
$x_{t+1} = x_t - \eta(\nabla f_{i_t}(x_t) - \nabla f_{i_t}(x^s) + g^s)$
else
$x_{t+1} = x_t - \eta \nabla f_{i_t}(x_t)$
end if

2.3 Mixed SG and SVRG Method

An approximate g^s can drastically reduce the computational cost of the SVRG algorithm, but does not affect the 2 in the $2m+n$ gradients required for m SVRG iterations. This factor of 2 is significant in the early iterations, since this is when stochastic methods make the most progress and when we typically see the largest reduction in the *test* error.

To reduce this factor, we can consider a *mixed* strategy: if i_t is in the batch $\mathscr{B}^s$ then perform an SVRG iteration, but if i_t is not in the current batch then use a classic SG iteration. We illustrate this modification in Algorithm 3. This modification allows the algorithm to take advantage of the rapid initial progress of SG, since it predominantly uses SG iterations when far from the solution. Below we give a convergence rate for this mixed strategy.

Theorem 2.2. *Assume f is μ-strongly convex and each f_i is L-Lipschitz smooth. Furthermore assume $\|x_t - x^*\|_2 \leq Z$ holds for every iteration. Let $g^s = \nabla f(x^s) + e^s$ and we set η and m so that $0 < \rho(L, \alpha L) < 1$ with $\alpha = |\mathscr{B}^s|/n$. If we assume $\mathbb{E}\left[\|\nabla f_i(x)\|_2^2\right] \leq \sigma^2$ then Algorithm 3 has*

$$\mathbb{E}\left[f(x^{s+1}) - f(x^*)\right] \leq \rho(L, \alpha L)\mathbb{E}\left[f(x^s) - f(x^*)\right] + \frac{Z\mathbb{E}\left[\|e^s\|_2\right] + \eta\mathbb{E}\left[\|e^s\|_2^2\right] + \frac{\eta\sigma^2}{2}(1-\alpha)}{1-2\eta L}.$$

We give the proof in Appendix A.2.1. The extra term depending on the variance σ^2 is typically the bottleneck for SG methods. Classic SG methods require the step-size η to converge to zero because of this term. However, the mixed SG/SVRG method can keep the fast progress despite using a constant η since the

term depending on σ^2 converges to zero as α converges to one. Since $\alpha < 1$ implies that $\rho(L, \alpha L) < \rho$, this result implies that when $[f(x^s) - f(x^*)]$ is large compared to e^s and σ^2 that the mixed SG/SVRG method actually converges faster.

Sharing a single step size η between the SG and SVRG iterations in Theorem 2.2 is sub-optimal. For example, if x is close to x^* and $|\mathscr{B}^s| \approx n$, then the SG iteration might actually take us far away from the minimizer. Thus, we may want to use a decreasing sequence of step sizes for the SG iterations. In Appendix A.2.2, we show that using $\eta = O^*(\sqrt{(n-|\mathscr{B}|)/n|\mathscr{B}|})$ for the SG iterations can improve the dependence on the error e^s and variance σ^2.

2.4 Regularized SVRG

We are often interested in the special case where problem (2.1) has the decomposition

$$\min_{x \in \mathbb{R}^d} f(x) \equiv h(x) + \frac{1}{n} \sum_{i=1}^{n} f_i(x). \tag{2.4}$$

A common choice of h is a scaled 1-norm of the parameter vector, $h(x) = \lambda \|x\|_1$. This non-smooth regularizer encourages sparsity in the parameter vector, and can be addressed with the proximal-SVRG method of Xiao and Zhang (2014). Alternately, if we want an explicit Z (diameter of a space containing the x^*) we could set h to the indicator function for a 2-norm ball containing x^*. In Appendix A.3, we give a variant of Theorem 2.1 that allows errors in the proximal-SVRG method for non-smooth/constrained settings like this.

Another common choice is the ℓ_2-regularizer, $h(x) = \frac{\lambda}{2}\|x\|_2^2$. With this regularizer, the SVRG updates can be equivalently written in the form

$$x_{t+1} = x_t - \eta \left(\nabla h(x_t) + \nabla f_{i_t}(x_t) - \nabla f_{i_t}(x^s) + g^s \right), \tag{2.5}$$

where $g^s = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(x^s)$. That is, they take an exact gradient step with respect to the regularizer and an SVRG step with respect to the f_i functions. When the ∇f_i are sparse, this form of the update allows us to implement the iteration without needing full-vector operations. A related update is used by Le Roux et al. (2012, §4)to avoid full-vector operations in the SAG algorithm. In Appendix A.3.2, we prove the below convergence rate for this update.

Theorem 2.3. *Consider instances of problem* (2.1) *that can be written in the form* (2.4) *where* ∇h *is* L_h*-Lipschitz continuous and each* ∇f_i *is* L_f*-Lipschitz continuous,* f *is* μ*-strongly convex, and assume that we set* η *and* m *so that* $0 < \rho(L_m) < 1$ *with* $L_m = \max\{L_f, L_h\}$*. Then the regularized SVRG iteration* (2.5) *has*

$$\mathbb{E}\left[f(x^{s+1}) - f(x^*)\right] \leq \rho(L_m)\mathbb{E}\left[f(x^s) - f(x^*)\right].$$

Since $L_m \leq L$, and strictly so in the case of ℓ_2-regularization, this result shows that for ℓ_2-regularized problems SVRG actually converges faster than the standard analysis would indicate (a similar result appears in Konečnỳ et al. (2014)). Further, this result gives a theoretical justification for using the update (2.5) for other h functions where it is not equivalent to the original SVRG method.

2.5 Mini-Batching Strategies

Konečnỳ et al. (2014) have also recently considered using batches of data within SVRG. They consider using 'mini-batches' in the inner iteration (the update of x_t) to decrease the variance of the method, but still use full passes through the data to compute g^s. This prior work is thus complimentary to the current work (in practice, both strategies can be used to improve performance). In Appendix A.4.1 we show that sampling the inner mini-batch proportional to L_i achieves a convergence rate of

$$\mathbb{E}\left[f(x^{s+1}) - f(x^*)\right] \leq \rho_M\mathbb{E}\left[f(x^s) - f(x^*)\right],$$

where M is the size of the mini-batch while

$$\rho_M = \frac{1}{1 - 2\eta\bar{L}}\left(\frac{1}{m\mu\eta} + 2\bar{L}\eta\right),$$

and we assume $0 < \rho_M < 1$. This generalizes the standard rate of SVRG and improves on the result of Konečnỳ et al. (2014) in the smooth case. This rate can be faster than the rate of the standard SVRG method at the cost of a more expensive iteration, and may be clearly advantageous in settings where parallel computation allows us to compute several gradients simultaneously.

The regularized SVRG form (2.5) suggests an alternate mini-batch strategy for

problem (2.1): consider a mini-batch that contains a 'fixed' set $\mathscr{B}_f$ and a 'random' set $\mathscr{B}_r$. Without loss of generality, assume that we sort the f_i based on their L_i values so that $L_1 \geq L_2 \geq \cdots \geq L_n$. For the fixed $\mathscr{B}_f$ we will *always* choose the M_f values with the largest L_i, $\mathscr{B}_f = \{f_1, f_2, \ldots, f_{M_f}\}$. In contrast, we choose the members of the random set $\mathscr{B}_r$ by sampling from $B_t = \{f_{M_f+1}, \ldots, f_n\}$ proportional to their Lipschitz constants, $p_i = \frac{L_i}{(n-M_f)\bar{L}_r}$ with $\bar{L}_r = (1/(n-M_f))\sum_{i=M_f+1}^{n} L_i$. In Appendix A.4.2 we show the following convergence rate for this strategy:

Theorem 2.4. *Assume f is μ-strongly convex and each f_i is L-Lipschitz smooth. Let $g(x) = (1/n)\sum_{i\notin[\mathscr{B}_f]} f_i(x)$ and $h(x) = (1/n)\sum_{i\in[\mathscr{B}_f]} f_i(x)$. If we replace the SVRG update with*

$$x_{t+1} = x_t - \eta\left(\nabla h(x_t) + (1/M_r)\sum_{i\in\mathscr{B}_r}\frac{\bar{L}_r}{L_i}(\nabla f_i(x_t) - \nabla f_i(x^s)) + \nabla g(x^s)\right),$$

then the convergence rate is

$$\mathbb{E}\left[f(x^{s+1}) - f(x^*)\right] \leq \left(\frac{2}{m\mu(2-\eta\zeta)\eta} + \frac{4\bar{L}_r M_f \eta}{n(2-\zeta\eta)}\right)\mathbb{E}\left[f(x^s) - f(x^*)\right],$$

where $\zeta = \max\{\frac{4LM_f}{n}, \frac{4\bar{L}_r(n-M_f)}{n}\}$.

If $\zeta \leq 4\bar{L}_r$, then we get a faster convergence rate than SVRG with mini-batch of size M.

2.6 Using Support Vectors

Using a batch $\mathscr{B}^s$ decreases the number of gradient evaluations required when SVRG is far from the solution, but its benefit diminishes over time. However, for certain objectives we can further reduce the number of gradient evaluations by identifying *support vectors*. For example, consider minimizing the Huberized hinge loss (HSVM) with threshold ε (Rosset and Zhu, 2007),

$$\min_{x\in\mathbb{R}^d}\frac{1}{n}\sum_{i=1}^{n} f(b_i a_i^T x), \quad f(\tau) = \begin{cases} 0 & \text{if } \tau > 1+\varepsilon, \\ 1-\tau & \text{if } \tau < 1-\varepsilon, \\ \frac{(1+\varepsilon-\tau)^2}{4\varepsilon} & \text{if } |1-\tau| \leq \varepsilon. \end{cases}$$

In terms of (2.1), we have $f_i(x) = f(b_i a_i^T x)$. The performance of this loss function is similar to logistic regression and the hinge loss, but it has the appealing properties of both: it is *differentiable* like logistic regression meaning we can apply methods like SVRG, but it has *support vectors* like the hinge loss meaning that many examples will have $f_i(x^*) = 0$ and $\nabla f_i(x^*) = 0$. We can also construct Huberized variants of many non-smooth losses for regression and multi-class classification.

If we knew the support vectors where $f_i(x^*) > 0$, we could solve the problem faster by ignoring the non-support vectors. For example, if there are 100000 training examples but only 100 support vectors in the optimal solution, we could solve the problem 1000 times faster. While we typically don't know the support vectors, in this section we outline a heuristic that gives large practical improvements by trying to identify them as the algorithm runs.

Our heuristic has two components. The first component is maintaining the *list of non-support vectors at x^s*. Specifically, we maintain a list of examples i where $\nabla f_i(x^s) = 0$. When SVRG picks an example i_t that is part of this list, we know that $\nabla f_{i_t}(x^s) = 0$ and thus the iteration only needs one gradient evaluation. This modification is not a heuristic, in that it still applies the exact SVRG algorithm. However, at best it can only cut the runtime in half.

The heuristic part of our strategy is to skip $\nabla f_i(x^s)$ or $\nabla f_i(x_t)$ if our evaluation of ∇f_i has been zero more than two consecutive times (and skipping it an exponentially larger number of times each time it remains zero). Specifically, for each example i we maintain two variables, sk_i (for 'skip') and ps_i (for 'pass'). Whenever we need to evaluate ∇f_i for some x^s or x_t, we run Algorithm 4 which may skip the evaluation. This strategy can lead to huge computational savings in later iterations if there are few support vectors, since many iterations will require no gradient evaluations.

Identifying support vectors to speed up computation has long been an important part of SVM solvers, and is related to the classic shrinking heuristic (Joachims, 1999). While it has previously been explored in the context of dual coordinate ascent methods (Usunier et al., 2010), this is the first work exploring it for linearly-convergent stochastic gradient methods.

Algorithm 4 Heuristic for skipping evaluations of f_i at x

if $sk_i = 0$ **then**
  compute $\nabla f_i(x)$.
  if $f_i'(x) = 0$ **then**
    $ps_i = ps_i + 1$. {Update the number of consecutive times $f_i'(x)$ was zero.}
    $sk_i = 2^{\max\{0, ps_i - 2\}}$. {Skip exponential number of future evaluations if it remains zero.}
  else
    $ps_i = 0$. {This could be a support vector, do not skip it next time.}
  end if
  return $\nabla f_i(x)$.
else
  $sk_i = sk_i - 1$. {In this case, we skip the evaluation.}
  return 0.
end if

2.7 Learning efficiency

In this section we compare the performance of SVRG as a large-scale learning algorithm compared to FG and SG methods. Following Bottou and Bousquet (2007),we can formulate the generalization error $\mathscr{E}$ of a learning algorithm as the sum of three terms

$$\mathscr{E} = \mathscr{E}_{\text{app}} + \mathscr{E}_{\text{est}} + \mathscr{E}_{\text{opt}}$$

where the approximation error $\mathscr{E}_{\text{app}}$ measures the effect of using a limited class of models, the estimation error $\mathscr{E}_{\text{est}}$ measures the effect of using a finite training set, and the optimization error $\mathscr{E}_{\text{opt}}$ measures the effect of inexactly solving problem (2.1). Bottou and Bousquet (2007) study asymptotic performance of various algorithms for a fixed approximation error and under certain conditions on the distribution of the data depending on parameters α or ν. In Appendix A.5 we discuss how SVRG can be analyzed in their framework. The table below includes SVRG among their results.

In this table, the condition number is $\kappa = L/\mu$. In this setting, linearly-convergent stochastic gradient methods can obtain better bounds for ill-conditioned problems, with a better dependence on the dimension and without depending on the noise variance ν.

Algorithm	Time to reach $\mathscr{E}_{\text{opt}} \leq \varepsilon$	Time to reach $\mathscr{E} = O(\mathscr{E}_{app} + \varepsilon)$	Previous with $\kappa \sim n$
FG	$\mathscr{O}\left(n\kappa d \log\left(\frac{1}{\varepsilon}\right)\right)$	$\mathscr{O}\left(\frac{d^2\kappa}{\varepsilon^{1/\alpha}} \log^2\left(\frac{1}{\varepsilon}\right)\right)$	$\mathscr{O}\left(\frac{d^3}{\varepsilon^{2/\alpha}} \log^3\left(\frac{1}{\varepsilon}\right)\right)$
SG	$\mathscr{O}\left(\frac{d\nu\kappa^2}{\varepsilon}\right)$	$\mathscr{O}\left(\frac{d\nu\kappa^2}{\varepsilon}\right)$	$\mathscr{O}\left(\frac{d^3\nu}{\varepsilon} \log^2\left(\frac{1}{\varepsilon}\right)\right)$
SVRG	$\mathscr{O}\left((n+\kappa)d \log\left(\frac{1}{\varepsilon}\right)\right)$	$\mathscr{O}\left(\frac{d^2}{\varepsilon^{1/\alpha}} \log^2\left(\frac{1}{\varepsilon}\right) + \kappa d \log\left(\frac{1}{\varepsilon}\right)\right)$	$\mathscr{O}\left(\frac{d^2}{\varepsilon^{1/\alpha}} \log^2\left(\frac{1}{\varepsilon}\right)\right)$

Table 2.1: Learning efficiency for different optimization algorithms.

Data set	Data Points	Variables	Reference
quantum	50 000	78	Caruana et al. (2004)
protein	145 751	74	Caruana et al. (2004)
sido	12 678	4 932	Guyon (2008)
rcv1	20 242	47 236	Lewis et al. (2004)
covertype	581 012	54	Frank and Asuncion (2010)
news	19 996	1 355 191	Keerthi and DeCoste (2005)
spam	92 189	823 470	Carbonetto (2009); Cormack and Lynam (2005)
rcv1Full	697 641	47 236	Lewis et al. (2004)
alpha	500 000	500	Synthetic

Table 2.2: Binary data sets used in the experiments.

2.8 Experimental Results

In this section, we present experimental results that evaluate our proposed variations on the SVRG method. We focus on logistic regression classification: given a set of training data $(a_1, b_1) \ldots (a_n, b_n)$ where $a_i \in \mathbb{R}^d$ and $b_i \in \{-1, +1\}$, the goal is to find the $x \in \mathbb{R}^d$ solving

$$\underset{x \in \mathscr{X}}{\operatorname{argmin}} x \in \mathbb{R}^d \; \frac{\lambda}{2}\|x\|^2 + \frac{1}{n}\sum_{i=1}^{n} \log(1 + \exp(-b_i a_i^T x)),$$

where the objective is strongly convex and smooth. We consider the datasets used by Le Roux et al. (2012), whose properties are listed in table 2.2. As in their work we add a bias variable, normalize dense features, and set the regularization parameter λ to $1/n$. We used a step-size of $\eta = 1/L$ and we used $m = |\mathscr{B}^s|$ which gave good performance across methods and datasets. In our first experiment, we compared three variants of SVRG: the original strategy that uses all n examples to

form g^s (*Full*), a growing batch strategy that sets $|\mathcal{B}^s| = 2^s$ (*Grow*), and the mixed SG/SVRG described by Algorithm 3 under this same choice (*Mixed*). While a variety of practical batching methods have been proposed in the literature (Byrd et al., 2012; Friedlander and Schmidt, 2012; van den Doel and Ascher, 2012), we did not find that any of these strategies consistently outperformed the doubling used by the simple *Grow* strategy. Our second experiment focused on the ℓ_2-regularized HSVM on the same datasets, and we compared the original SVRG algorithm with variants that try to identify the support vectors (*SV*).

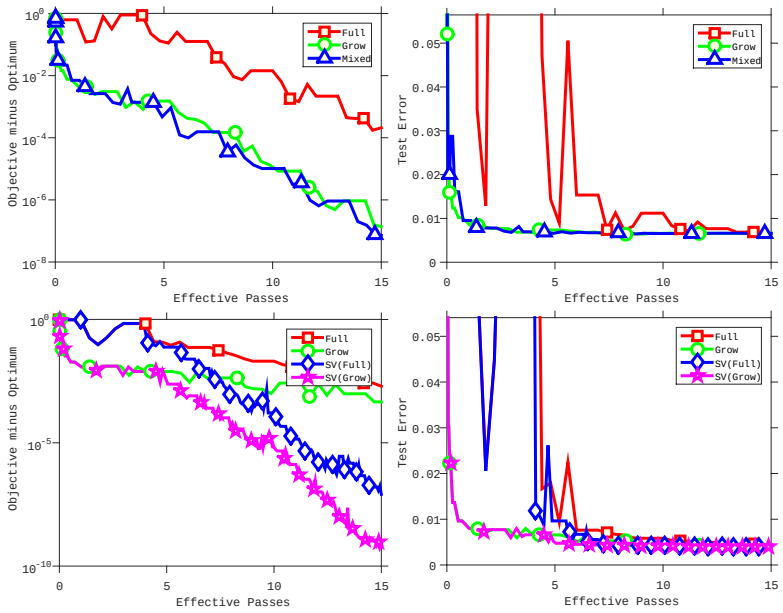

Figure 2.1: Comparison of training objective (left) and test error (right) on the *spam* dataset for the logistic regression (top) and the HSVM (bottom) losses under different batch strategies for choosing g^s (*Full*, *Grow*, and *Mixed*) and whether we attempt to identify support vectors (*SV*).

We plot the experimental results for one run of the algorithms on one dataset in Figure 2.1. In our results, the growing batch strategy (*Grow*) always had better test error performance than using the full batch, while for large datasets it also performed substantially better in terms of the training objective. In contrast, the *Mixed* strategy sometimes helped performance and sometimes hurt performance. Utilizing support vectors often improved the training objective, often by large margins,

but its effect on the test objective was smaller.

In Figures 2.2 to 2.5, we plot the performance on the various datasets in terms of both the training objective and test error, showing the maximum/mean/minimum performance across 10 random trials. In these plots, we see a clear advantage for the Grow strategy on the largest datasets (bottom row), but less of an advantage or no advantage on the smaller datasets. The advantage of using support vectors seemed less dependent on the data size, as it helped in some small datasets as well as some large datasets, while in some small/large datasets it did not make a big difference. We have made the code available at http://www.cs.ubc.ca/~schmidtm/Software/practicalSVRG.zip.

2.9 Discussion

As SVRG is the only memory-free method among the new stochastic linearly-convergent methods, it represents the natural method to use for a huge variety of machine learning problems. In this work we show that the convergence rate of the SVRG algorithm can be preserved even under an inexact approximation to the full gradient. We also showed that using mini-batches to approximate g^s gives a natural way to do this, explored the use of support vectors to further reduce the number of gradient evaluations, gave an analysis of the regularized SVRG update, and considered several new mini-batch strategies. Our theoretical and experimental results indicate that many of these simple modifications should be considered in any practical implementation of SVRG.

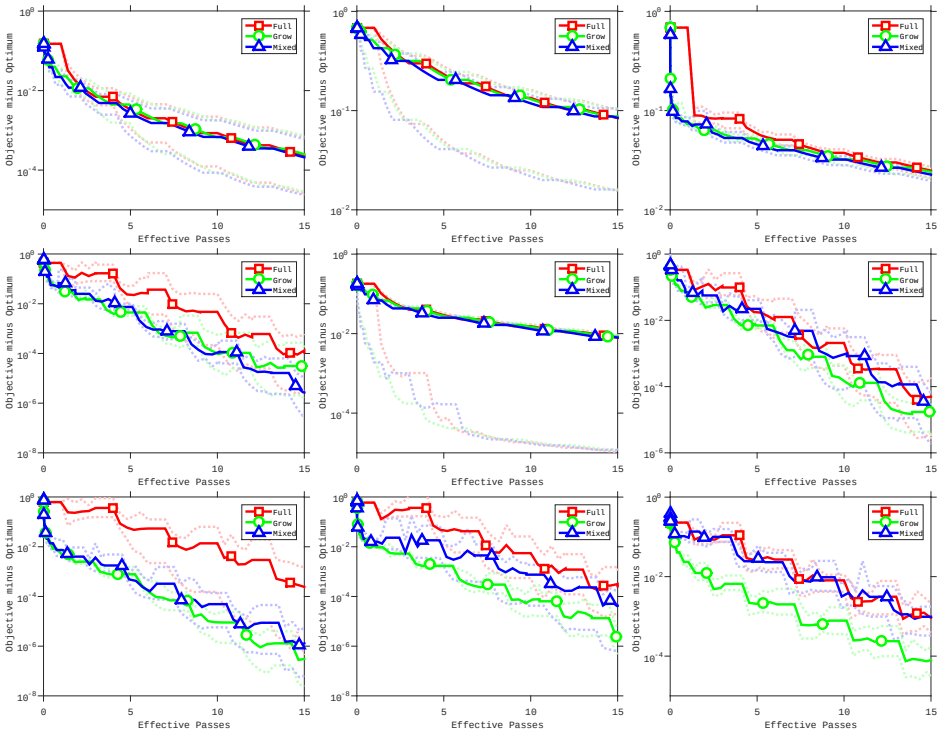

Figure 2.2: Comparison of training objective of logistic regression for different datasets. The top row gives results on the *quantum* (left), *protein* (center) and *sido* (right) datasets. The middle row gives results on the *rcv11* (left), *covertype* (center) and *news* (right) datasets. The bottom row gives results on the *spam* (left), *rcv1Full* (center), and *alpha* (right) datasets.

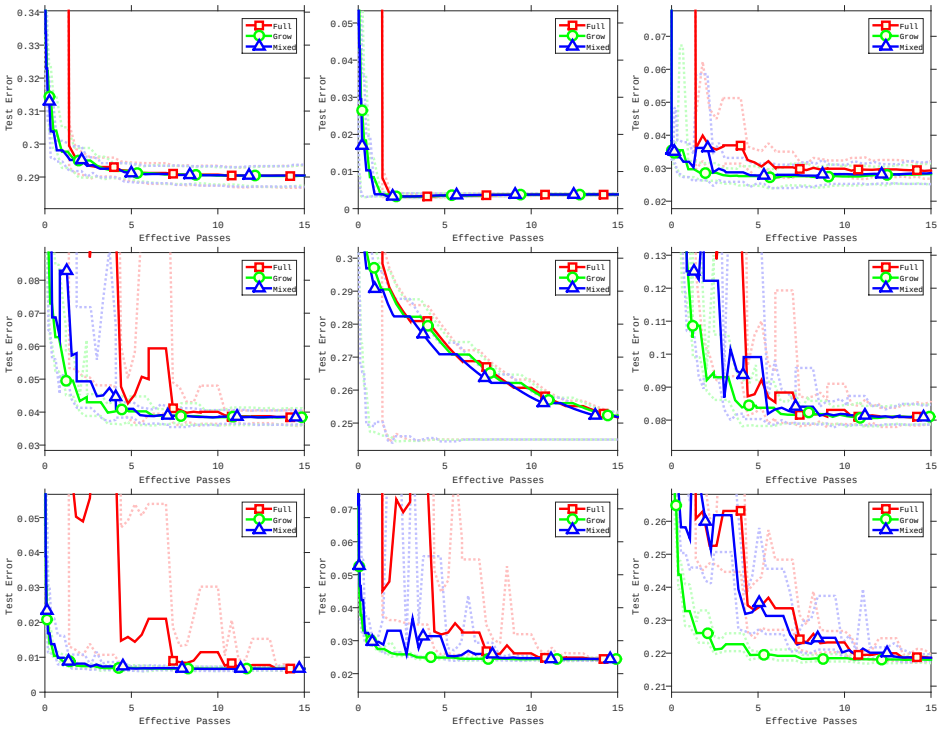

Figure 2.3: Comparison of test error of logistic regression for different datasets. The top row gives results on the *quantum* (left), *protein* (center) and *sido* (right) datasets. The middle row gives results on the *rcv11* (left), *covertype* (center) and *news* (right) datasets. The bottom row gives results on the *spam* (left), *rcv1Full* (center), and *alpha* (right) datasets.

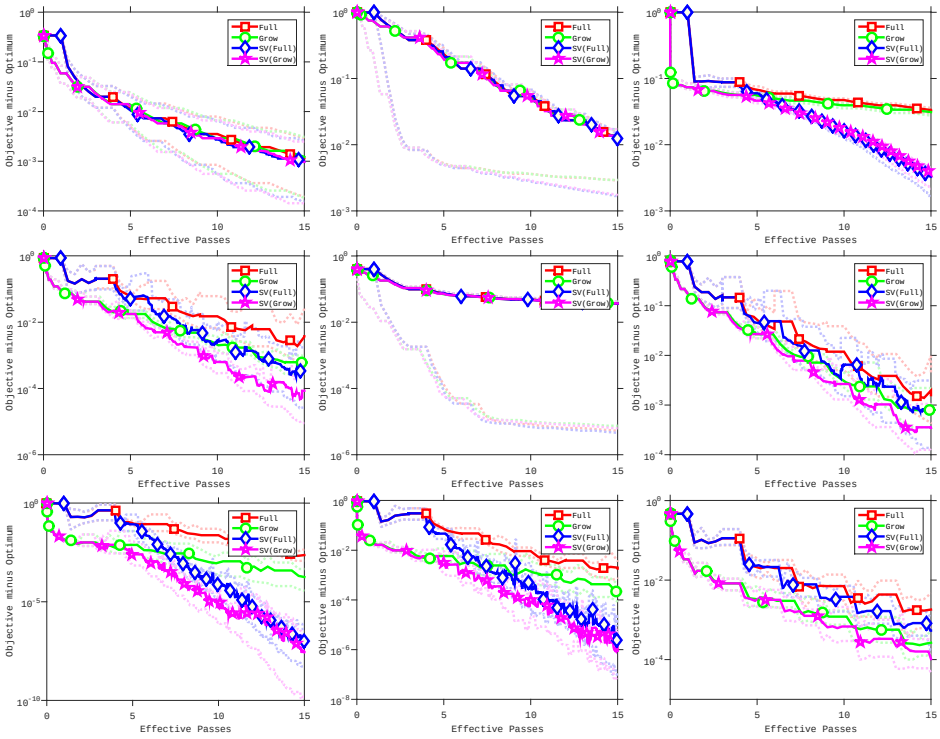

Figure 2.4: Comparison of training objective of SVM for different datasets. The top row gives results on the *quantum* (left), *protein* (center) and *sido* (right) datasets. The middle row gives results on the *rcv11* (left), *covertype* (center) and *news* (right) datasets. The bottom row gives results on the *spam* (left), *rcv1Full* (center), and *alpha* (right) datasets.

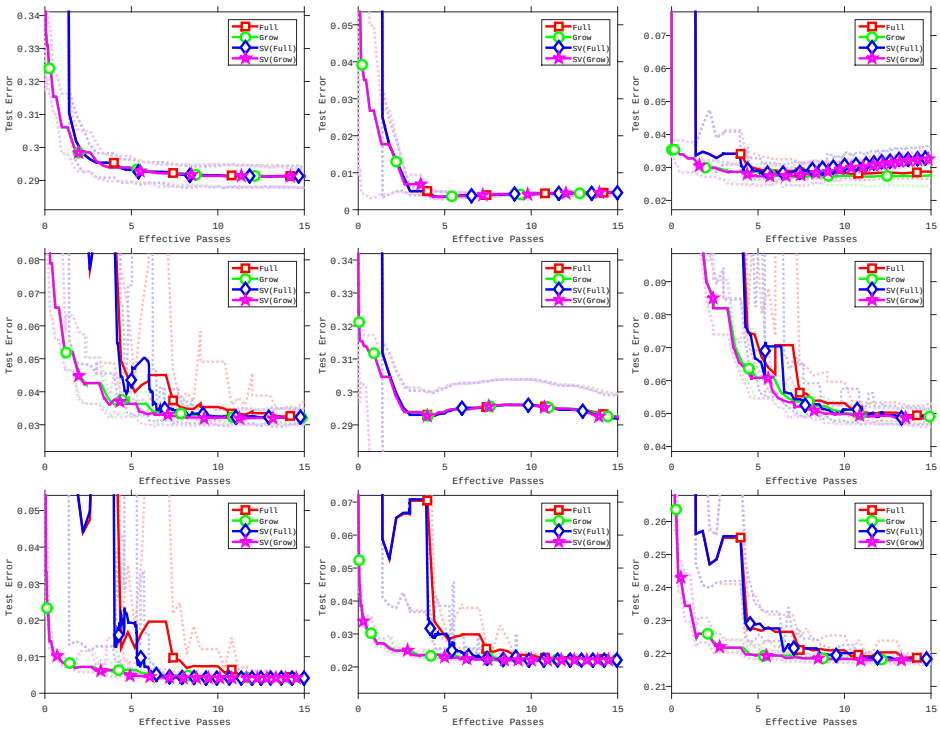

Figure 2.5: Comparison of test error of SVM for different datasets. The top row gives results on the *quantum* (left), *protein* (center) and *sido* (right) datasets. The middle row gives results on the *rcv11* (left), *covertype* (center) and *news* (right) datasets. The bottom row gives results on the *spam* (left), *rcv1Full* (center), and *alpha* (right) datasets.

Chapter 3

Practical Non-uniform SAG for Conditional Random Fields

Conditional random fields (CRFs) (Lafferty et al., 2001) are a ubiquitous tool in natural language processing. They are used for part-of-speech tagging (McCallum et al., 2003), semantic role labeling (Cohn and Blunsom, 2005), topic modeling (Zhu and Xing, 2010), information extraction (Peng and McCallum, 2006), shallow parsing (Sha and Pereira, 2003), named-entity recognition (Settles, 2004), as well as a host of other applications in natural language processing and in other fields such as computer vision (Nowozin and Lampert, 2011). Similar to generative Markov random field (MRF) models, CRFs allow us to model probabilistic dependencies between output variables. The key advantage of discriminative CRF models is the ability to use a very high-dimensional feature set, without explicitly building a model for these features (as required by MRF models). Despite the widespread use of CRFs, a major disadvantage of these models is that they can be very slow to train and the time needed for numerical optimization in CRF models remains a bottleneck in many applications.

Due to the high cost of evaluating the CRF objective function on even a single training example, it is now common to train CRFs using stochastic gradient methods (Vishwanathan et al., 2006). These methods are advantageous over deterministic methods because on each iteration they only require computing the gradient of a single example (and not *all* example as in deterministic methods). Thus, if

we have a data set with n training examples, the iterations of stochastic gradient methods are n times faster than deterministic methods. However, the number of stochastic gradient iterations required might be very high. This has been studied in the optimization community, which considers the problem of finding the minimum number of iterations t so that we can guarantee that we reach an accuracy of ε, meaning that

$$f(x_t) - f(x^*) \leq \varepsilon, \text{ and } \|x_t - x^*\|_2^2 \leq \varepsilon,$$

where f is our training objective function which is smooth and strongly-convex, x_t is our parameter estimate on iteration t, and x^* is the parameter vector minimizing the training objective function. For strongly-convex objectives like ℓ_2-regularized CRFs, stochastic gradient methods require $O(1/\varepsilon)$ iterations (Nemirovski et al., 2009).

This is in contrast to traditional deterministic methods which only require $O(\log(1/\varepsilon))$ iterations (Nesterov, 2004). However, this much lower number of iterations comes at the cost of requiring us to process the entire data set on each iteration. For problems with a finite number of training examples, Le Roux et al. (2012) proposed the stochastic average gradient (SAG) algorithm which combines the advantages of deterministic and stochastic methods: it only requires evaluating a single randomly-chosen training example on each iteration, and only requires $O(\log(1/\varepsilon))$ iterations to reach an accuracy of ε.

Beyond this faster convergence rate, the SAG method also allows us to address two issues that have traditionally frustrated users of stochastic gradient methods: *setting the step-size* and *deciding when to stop*. Implementations of the SAG method use both an adaptive step-size procedure and a cheaply-computable criterion for deciding when to stop. Le Roux et al. (2012) show impressive empirical performance of the SAG algorithm for binary classification.

This is the first work to apply a SAG algorithm to train CRFs. We show that tracking marginals in the CRF can drastically reduce the SAG method's huge memory requirement. We also give a non-uniform sampling (NUS) strategy that adaptively estimates how frequently we should sample each data point, and we show that the SAG-like algorithm of Defazio et al. (2014) converges under any NUS strategy while a particular NUS strategy achieves a faster rate. Our experiments

compare the SAG algorithm with a variety of competing deterministic, stochastic, and semi-stochastic methods on benchmark data sets for four common tasks: part-of-speech tagging, named entity recognition, shallow parsing, and optical character recognition. Our results indicate that the SAG algorithm with NUS often outperforms previous methods by an order of magnitude in terms of the training objective and, despite not requiring us to tune the step-size, performs as well or better than optimally-tuned stochastic gradient methods in terms of the test error.

3.1 Conditional Random Fields

CRFs model the conditional probability of a structured output $b \in \mathscr{B}$ (such as a sequence of labels) given an input $a \in \mathscr{A}$ (such as a sequence of words) based on features $F(a,b)$ and parameters x using

$$p(b|a,x) = \frac{\exp(x^T F(a,b))}{\sum_{b'} \exp(x^T F(a,b'))}. \tag{3.1}$$

Given n pairs $\{a_i, b_i\}$ comprising our training set, the standard approach to training the CRF is to minimize the ℓ_2-regularized negative log-likelihood,

$$\min_x f(x) = \frac{1}{n}\sum_{i=1}^{n} -\log p(b_i \mid a_i, x) + \frac{\lambda}{2}\|x\|_2^2, \tag{3.2}$$

where $\lambda > 0$ is the strength of the regularization parameter. Unfortunately, evaluating $\log p(b_i \mid a_i, x)$ is expensive due to the summation over all possible configurations y'. For example, in chain-structured models the forward-backward algorithm is used to compute $\log p(b_i \mid a_i, x)$ and its gradient. A second problem with solving (3.2) is that the number of training examples n in applications is constantly-growing, and thus we would like to use methods that only require a few passes through the data set.

3.2 Related Work

Lafferty et al. (2001) proposed an iterative scaling algorithm to solve problem (3.2), but this proved to be inferior to generic deterministic optimization strategies like the limited-memory quasi-Newton algorithm L-BFGS (Sha and Pereira, 2003; Wal-

lach, 2002). The bottleneck in these methods is that we must evaluate $\log p(b_i \mid a_i, x)$ and its gradient for all n training examples on every iteration. This is very expensive for problems where n is very large, so to deal with this problem stochastic gradient methods were examined (Finkel et al., 2008; Vishwanathan et al., 2006). However, traditional stochastic gradient methods require $O(1/\varepsilon)$ iterations rather than the much smaller $O(\log(1/\varepsilon))$ required by deterministic methods.

There have been several attempts at improving the cost of deterministic methods or the convergence rate of stochastic methods. For example, the exponentiated gradient method of Collins et al. (2008) processes the data online and only requires $O(\log(1/\varepsilon))$ iterations to reach an accuracy of ε in terms of the dual objective. However, this does not guarantee good performance in terms of the primal objective or the weight vector. Although this method is highly-effective if λ is very large, our experiments and the experiments of others show that the performance of online exponentiated gradient can degrade substantially if a small value of λ is used (which may be required to achieve the best test error), see Collins et al. (2008, Figures 5-6 and Table 3) and Lacoste-Julien et al. (2013, Figure 1). In contrast, SAG degrades more gracefully as λ becomes small, even achieving a convergence rate faster than classic SG methods when $\lambda = 0$ (Schmidt et al., 2013). Lavergne et al. (2010) consider using multiple processors and vectorized computation to reduce the high iteration cost of quasi-Newton methods, but when n is enormous these methods still have a high iteration cost. Friedlander and Schmidt (2012) explore a hybrid deterministic-stochastic method that slowly grows the number of examples that are considered in order to achieve an $O(\log(1/\varepsilon))$ convergence rate with a decreased cost compared to deterministic methods.

In Table 3.1 we state the convergence rates of different methods for training CRFs, including the fastest known rates for deterministic algorithms (like L-BFGS and accelerated gradient) (Nesterov, 2004), stochastic algorithms (like [averaged] stochastic gradient and AdaGrad) (Ghadimi and Lan, 2012), online exponentiated gradient, and SAG. Here L is the Lipschitz constant of the gradient of the objective, μ is the strong-convexity constant (and we have $\lambda \leq \mu \leq L$), and σ^2 bounds the variance of the gradients.

Method	Iterations to ε	Primal or Dual
Deterministic	$O(n\sqrt{\frac{L}{\mu}}\log(1/\varepsilon))$	(primal)
Online EG	$O((n+\frac{L}{\lambda})\log(1/\varepsilon))$	(dual)
Stochastic	$O(\frac{\sigma^2}{\mu\varepsilon}+\sqrt{\frac{L}{\mu\varepsilon}})$	(primal)
SAG	$O((n+\frac{L}{\mu})\log(1/\varepsilon))$	(primal)

Table 3.1: Convergence rates of different optimization methods

3.3 Stochastic Average Gradient

Le Roux et al. (2012) introduce the SAG algorithm, a simple method with the low iteration cost of stochastic gradient methods but that only requires $O(\log(1/\varepsilon))$ iterations. To motivate this new algorithm, we write the classic gradient descent iteration as

$$x_{t+1} = x_t - \frac{\eta}{n}\sum_{i=1}^{n} s_i^t, \tag{3.3}$$

where η is the step-size and at each iteration we set the 'slope' variables s_i^t to the gradient with respect to training example i at x_t, so that $s_i^t = -\nabla \log p(b_i \mid a_i, x_t) + \lambda x_t$. The SAG algorithm uses this same iteration, but instead of updating s_i^t for all n data points on every iterations, it simply sets $s_i^t = -\nabla \log p(b_i \mid a_i, x_t) + \lambda x_t$ for *one randomly chosen* data point and keeps the remaining s_i^t at their value from the previous iteration. Thus the SAG algorithm is a randomized version of the gradient algorithm where we use the gradient of each example from the last iteration where it was selected. The surprising aspect of the work of Le Roux et al. (2012) is that this simple *delayed* gradient algorithm achieves a similar convergence rate to the classic full gradient algorithm despite the iterations being n times faster.

3.3.1 Implementation for CRFs

Unfortunately, a major problem with applying (3.3) to CRFs is the requirement to store the s_i^t. While the CRF gradients $\nabla \log p(b_i \mid a_i, x_t)$ have a nice structure (see Section 3.3.2), s_i^t includes λx_t for some previous t, which is dense and unstructured. To get around this issue, instead of using (3.3) we use the following SAG-like

update (Le Roux et al., 2012, Section 4)

$$\begin{aligned} x_{t+1} &= x_t - \eta(\frac{1}{m}\sum_{i=1}^{n} g_i^t + \lambda x_t) \\ &= x_t - \eta(\frac{1}{m}d + \lambda x_t) \\ &= (1-\eta\lambda)x_t - \frac{\eta}{m}d, \end{aligned} \tag{3.4}$$

where g_i^t is the value of $-\nabla \log p(b_i \mid a_i, x_k)$ for the last iteration k where i was selected and d is the sum of the g_i^t over all i. Thus, this update uses the exact gradient of the regularizer and only uses an approximation for the (structured) CRF log-likelihood gradients. Since we don't yet have any information about these log-likelihoods at the start, we initialize the algorithm by setting $g_i^0 = 0$. But to compensate for this, we track the number of examples seen m, and normalize d by m in the update (instead of n). In Algorithm 5, we summarize this variant of the SAG algorithm for training CRFs.[1]

In many applications of CRFs the g_i^t are very sparse, and we would like to take advantage of this as in stochastic gradient methods. Fortunately, we can implement (3.4) without using dense vector operations by using the representation $x_t = \beta_t v_t$ for a scalar β_t and a vector v_t, and using 'lazy updates' that apply d repeatedly to an individual variable when it is needed (Le Roux et al., 2012).

Also following Le Roux et al. (2012), we set the step-size to $\eta = 1/L$, where L is an approximation to the maximum Lipschitz constant of the gradients. This is the smallest number L such that

$$\|\nabla f_i(x) - \nabla f_i(y)\|_2 \leq L\|x-y\|_2, \tag{3.5}$$

for all i, x, and y. This quantity is a bound on how fast the gradient can change as we change the weight vector. The Lipschitz constant with respect to the gradient of the regularizer is simply λ. This gives $L \leq L_g + \lambda$, where L_g is the Lipschitz constant of the gradient of the log-likelihood. Unfortunately, L_g depends on the covariance of the CRF and is typically too expensive to compute. To avoid this computation,

[1] If we solve the problem for a sequence of regularization parameters, we can obtain better performance by warm-starting g_i^0, d, and m.

Algorithm 5 SAG algorithm for training CRFs

Require: $\{a_i, b_i\}$, λ, x, δ

$m \leftarrow 0$, $g_i \leftarrow 0$ for $i = 1, 2, \ldots, n$
$d \leftarrow 0$, $L_g \leftarrow 1$
while $m < n$ and $\|\frac{1}{n}d + \lambda x\|_\infty \geq \delta$ **do**
  Sample i from $\{1, 2, \ldots, n\}$
  $f \leftarrow -\log p(b_i \mid a_i, x)$
  $g \leftarrow -\nabla \log p(b_i \mid a_i, x)$
  if this is the first time we sampled i **then**
    $m \leftarrow m + 1$
  end if
  Subtract old gradient g_i, add new gradient g:
  $d \leftarrow d - g_i + g$
  Replace old gradient of example i:
  $g_i \leftarrow g$
  if $\|g_i\|_2^2 > 10^{-8}$ **then**
    $L_g \leftarrow$lineSearch$(a_i, b_i, f, g_i, x, L_g)$
  end if
  $\eta \leftarrow 1/(L_g + \lambda)$
  $x \leftarrow (1 - \eta\lambda)x - \frac{\eta}{m}d$
  $L_g \leftarrow L_g \cdot 2^{-1/n}$
end while

as in Le Roux et al. (2012) we approximate L_g in an online fashion using the standard backtracking line-search given by Algorithm 6 (Beck and Teboulle, 2009). The test used in this algorithm is faster than testing (3.5), since it uses function values (which only require the forward algorithm for CRFs) rather than gradient values (which require the forward and backward steps). Algorithm 6 monotonically increases L_g, but we also slowly decrease it in Algorithm 5 in order to allow the possibility that we can use a more aggressive step-size as we approach the solution.

Algorithm 6 Lipschitz line-search algorithm

Require: a_i, b_i, f, g_i, x, L_g.

$f' = -\log p(b_i \mid a_i, x - \frac{1}{L_g} g_i)$

while $f' \geq f - \frac{1}{2L_g}\|g_i\|_2^2$ **do**

  $L_g = 2L_g$

  $f' = -\log p(b_i \mid a_i, x - \frac{1}{L_g} g_i)$

end while

return L_g.

Since the solution is the only stationary point, we must have $\nabla f(x_t) = 0$ at the solution. Further, the value $\frac{1}{n}d + \lambda x_t$ converges to $\nabla f(x_t)$ so we can use the size of this value to decide when to stop the algorithm (although we also require that $m = n$ to avoid premature stopping before we have seen the full data set). This is in contrast to classic stochastic gradient methods, where the step-size must go to zero and it is therefore difficult to decide if the algorithm is close to the optimal value or if we simply require a small step-size to continue making progress.

3.3.2 Reducing the Memory Requirements

Even if the gradients g_i^t are not sparse, we can often reduce the memory requirements of Algorithm 5 because it is known that the CRF gradients only depend on x through marginals of the features. Specifically, the gradient of the log-likelihood under model (3.1) with respect to feature j is given by

$$\begin{aligned}
\nabla_j \log p(b \mid a, x) &= F_j(a,b) - \frac{\sum_{b'} \exp(F(a,b'))F_j(a,b')}{\sum_{b'} \exp(F(a,b'))} \\
&= F_j(a,b) - \sum_{b'} p(b' \mid a, x) F_j(a,b') \\
&= F_j(a,b) - \mathbb{E}_{(b' \mid a,x)}[F_j(a,b')].
\end{aligned}$$

Typically, each feature j only depends on a small "part" of b. For example, we typically include features of the form $F_j(a,b) = F(a)\mathbb{I}[b_k = s]$ for some function F, where k is an element of b and s is a discrete state that b_k can take. In this case, the gradient can be written in terms of the marginal probability of element b_k

taking state s,

$$\begin{aligned}\nabla_j \log p(b \mid a,x) &= F(a)\mathbb{I}[b_k = s] - \mathbb{E}_{(b' \mid a,x)}[F(a)\mathbb{I}[b_k = s]] \\ &= F(a)(\mathbb{I}[b_k = s] - \mathbb{E}_{(b' \mid a,x)}[\mathbb{I}[b_k = s]) \\ &= F(a)(\mathbb{I}[b_k = s] - p(b_k = s \mid a,x)).\end{aligned}$$

Notice that Algorithm 5 only depends on the old gradient through its difference with the new gradient (line 10), which in this example gives

$$\begin{aligned}\nabla_j \log p(b \mid a,x) - \nabla_j \log p(b \mid a,x_{\text{old}}) =& F(a)(p(b_k = s \mid a,x_{\text{old}}) \\ & - p(b_k = s \mid a,x)),\end{aligned}$$

where x is the current parameter vector and x_{old} is the old parameter vector. Thus, to perform this calculation the only thing we need to know about x_{old} is the unary marginal $p(b_k = s \mid a,x_{\text{old}})$, which will be *shared* across features that only depend on the event that $b_k = s$. Similarly, features that depend on pairs of values in b will need to store pairwise marginals, $p(b_k = s, b'_k = s' \mid a,x_{\text{old}})$. For general pairwise graphical model structures, the memory requirements to store these marginals will thus be $O(VK + EK^2)$, where V is the number of vertices and E is the number of edges. This can be an enormous reduction since *it does not depend on the number of features*. Further, since computing these marginals is a by-product of computing the gradient, this potentially-enormous reduction in the memory requirements comes at no extra computational cost.

3.4 Non-Uniform Sampling

Several works show that we can improve the convergence rates of randomized optimization algorithms by using non-uniform sampling (NUS) schemes. This includes randomized Kaczmarz (Strohmer and Vershynin, 2009), randomized coordinate descent (Nesterov, 2012), and stochastic gradient methods (Needell et al., 2014a). The key idea behind all of these NUS strategies is to *bias the sampling towards the Lipschitz constants of the gradients*, so that gradients that change quickly get sampled more often and gradients that change slowly get sampled less often.

Specifically, we maintain a Lipschitz constant L_i for each training example i and, instead of the usual sampling strategy $p_i = 1/n$, we bias towards the distribution $p_i = L_i/\sum_j L_j$. In these various contexts, NUS allows us to improve the dependence on the values L_i in the convergence rate, since the NUS methods depend on $\bar{L} = (1/n)\sum_j L_j$, which may be substantially smaller than the usual dependence on $L = \max_j\{L_j\}$. Schmidt et al. (2013) argue that faster convergence rates might be achieved with NUS for SAG since it allows a larger step size η that depends on $\bar{L}$ instead of L.[2]

The scheme for SAG proposed by Schmidt et al. (2013, Section 5.5) uses a fairly complicated adaptive NUS scheme and step-size, but the key ingredient is estimating each constant L_i using Algorithm 6. Our experiments show this method often already improves on state of the art methods for training CRFs by a substantial margin, but we found we could obtain improved performance for training CRFs using the following simple NUS scheme for SAG: as in Needell et al. (2014a), with probability 0.5 choose i uniformly and with probability 0.5 sample i with probability $L_i/(\sum_j L_j)$ (restricted to the examples we have previously seen).[3] We also use a step-size of $\eta = \frac{1}{2}(1/L + 1/\bar{L})$, since the faster convergence rate with NUS is due to the ability to use a larger step-size than $1/L$. This simple step-size and sampling scheme contrasts with the more complicated choices described by Schmidt et al. (2013, Section 5.5), that make the degree of non-uniformity grow with the number of examples seen m. This prior work initializes each L_i to 1, and updates L_i to $0.5L_i$ each subsequent time an example is chosen. In the context of CRFs, this leads to a large number of expensive backtracking iterations. To avoid this, we initialize L_i with $0.5\bar{L}$ the first time an example is chosen, and decrease L_i to $0.9L_i$ each time it is subsequently chosen. Allowing the L_i to decrase seems crucial to obtaining the best practical performance of the method, as it allows the algorithm to take bigger step sizes if the values of L_i are small near the solution.

[2] An interesting difference between the SAG update with NUS and NUS for stochastic gradient methods is that the SAG update does not seem to need to decrease the step-size for frequently-sampled examples (since the SAG update does not rely on using an unbiased gradient estimate).

[3] Needell et al. (2014a) analyze the basic stochastic gradient method and thus require $O(1/\varepsilon)$ iterations.

3.4.1 Convergence Analysis under NUS

Schmidt et al. (2013) give an intuitive but non-rigorous motivation for using NUS in SAG. More recently, Xiao and Zhang (2014) analyse SVRG and show that NUS gives a dependence on $\bar{L}$. Below, we analyze a NUS extension of the SAGA algorithm of Defazio et al. (2014), which does not require full passes through the data and has similar performance to SAG in practice but is much easier to analyze.

Theorem 3.1. *Assume the objective function f is μ-strongly convex and each f_i is L-Lipschitz smooth. Let the sequences $\{x_t\}$ and $\{s_j^t\}$ be defined by*

$$x_{t+1} = x_t - \eta \left[\frac{1}{n p_{j_t}} (\nabla f_{j_t}(x_t) - s_{j_t}^t) + \frac{1}{n} \sum_{i=1}^{n} s_i^t \right],$$

$$s_j^{t+1} = \begin{cases} \nabla f_{r_t}(x_t) & \text{if } j = r_t, \\ s_j^t & \text{otherwise.} \end{cases}$$

where j_t is chosen with probability p_j.

(a) If r_t is set to j_t, then with $\eta = \frac{n p_{min}}{4L + n\mu}$ we have

$$\mathbb{E}[\|x_t - x^*\|_2^2] \leq (1 - \mu\eta)^t \left[\|x_0 - x^*\|_2^2 + C_a \right],$$

where $p_{min} = \min_i\{p_i\}$ and

$$C_a = \frac{2 p_{min}}{(4L + n\mu)^2} \sum_{i=1}^{n} \frac{1}{p_i} \|\nabla f_i(x_0) - \nabla f_i(x^*)\|_2^2.$$

(b) If $p_j = \frac{L_j}{\sum_{i=1}^n L_i}$ and r_t is chosen uniformly at random, then with $\eta = \frac{1}{4\bar{L}}$ we have

$$\mathbb{E}[\|x_t - x^*\|_2^2] \leq \left(1 - \min\left\{ \frac{1}{3n}, \frac{\mu}{8\bar{L}} \right\} \right)^t \left[\|x_0 - x^*\|_2^2 + C_b \right],$$

where:

$$C_b = \frac{n}{2\bar{L}} \left[f(x_0) - f(x^*) \right]$$

This result (which we prove in Appendix B.1 and B.2) shows that SAGA has (a) a linear convergence rate for any NUS scheme where $p_i > 0$ for all i, and (b) a

rate depending on $\bar{L}$ by sampling proportional to the Lipschitz constants and also generating a uniform sample. However, (a) achieves the fastest rate when $p_i = 1/n$ while (b) requires two samples on each iteration. We were not able to show a faster rate using only one sample on each iteration as used in our implementation.

3.4.2 Line-Search Skipping

To reduce the number of function evaluations required by the NUS strategy, we also explored a line-search *skipping* strategy. The general idea is to consider skipping the line-search for example i if the line-search criterion was previously satisfied for example i without backtracking. Specifically, if the line-search criterion was satisfied ξ consecutive times for example i (without backtracking), then we do not do the line-search on the next $2^{\xi-1}$ times example i is selected (we also do not multiply L_i by 0.9 on these iterations). This drastically reduces the number of function evaluations required in the later iterations.

3.5 Experiments

We compared a wide variety of approaches on four CRF training tasks: the optical character recognition (OCR) dataset of Taskar et al. (2003), the CoNLL-2000 shallow parse chunking dataset,[4] the CoNLL-2002 Dutch named-entity recognition dataset,[5] and a part-of-speech (POS) tagging task using the Penn Treebank Wall Street Journal data (POS-WSJ). The optimal character recognition dataset labels the letters in images of words. Chunking segments a sentence into syntactic chunks by tagging each sentence token with a chunk tag corresponding to its constituent type (e.g., 'NP', 'VP', etc.) and location (e.g., beginning, inside, ending, or outside any constituent). We use standard n-gram and POS tag features (Sha and Pereira, 2003). For the named-entity recognition task, the goal is to identify named entities and correctly classify them as persons, organizations, locations, times, or quantities. We again use standard n-gram and POS tag features, as well as word shape features over the case of the characters in the token. The POS-tagging task assigns one of 45 syntactic tags to each token in each of the sentences in the data.

[4] http://www.cnts.ua.ac.be/conll2000/chunking
[5] http://www.cnts.ua.ac.be/conll2002/ner

For this data, we follow the standard division of the WSJ data given by Collins (2002), using sections 0-18 for training, 19-21 for development, and 22-24 for testing. We use the standard set of features following Ratnaparkhi (1996) and Collins (2002): n-gram, suffix, and shape features. As is common on these tasks, our pairwise features do not depend on x.

On these datasets we compared the performance of a set of competitive methods, including five variants on classic stochastic gradient methods: *Pegasos* which is a standard stochastic gradient method with a step-size of $\eta = \alpha/\lambda t$ on iteration t (Shalev-Shwartz et al., 2011),[6] a basic stochastic gradient (*SG*) method where we use a constant $\eta = \alpha$, an averaged stochastic gradient (*ASG*) method where we use a constant step-size $\eta = \alpha$ and average the iterations,[7] *AdaGrad* where we use the per-variable $\eta_j = \alpha/(\delta + \sqrt{\sum_{i=1}^{t} \nabla_j \log p(b_i \mid a_i, x_t)^2})$ and the proximal-step with respect to the ℓ_2-regularizer (Duchi et al., 2011), and stochastic meta-descent (*SMD*) where we initialize with $\eta_j = \alpha$ and dynamically update the step-size (Vishwanathan et al., 2006). Since setting the step-size is a notoriously hard problem when applying stochastic gradient methods, we let these classic stochastic gradient methods cheat by choosing the η which gives the best performance among powers of 10 on the training data (for SMD we additionally tested the four choices among the paper and associated code of Vishwanathan et al. (2006), and we found $\delta = 1$ worked well for *AdaGrad*).[8]

Our comparisons also included a deterministic *L-BFGS* algorithm (Schmidt, 2005) and the *Hybrid* L-BFGS/stochastic algorithm of Friedlander and Schmidt (2012). We also included the online exponentiated gradient (*OEG*) method (Collins et al., 2008), and we followed the heuristics in the author's code.[9] Finally, we

[6] We also tested *Pegasos* with averaging but it always performed worse than the non-averaged version.

[7] We also tested *SG* and *ASG* with decreasing step-sizes of either $\eta_t = \alpha/\sqrt{t}$ or $\eta_t = \alpha/(\delta + t)$, but these gave worse performance than using a constant step size.

[8] Because of the extra implementation effort required to implement it efficiently, we did not test SMD on the POS dataset, but we do not expect it to be among the best performers on this data set.

[9] Specifcially, for OEG we proceed through a random permutation of the dataset on the first pass through the data, we perform a maximum of 2 backtracking iterations per example on this first pass (and 5 on subsequent passes), we initialize the per-sample step-sizes to 0.5 and divide them by 2 if the dual objective does not increase (and multiply them by 1.05 after processing the example), and to initialize the dual variables we set parts with the correct label from the training set to 3 and parts with the incorrect label to 0.

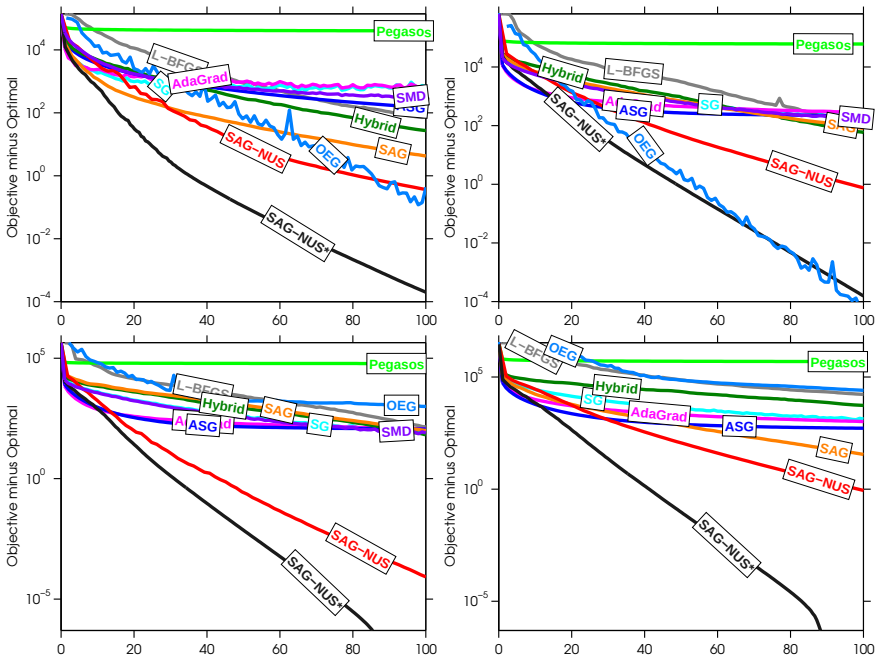

Figure 3.1: Objective minus optimal objective value against effective number of passes for different deterministic, stochastic, and semi-stochastic optimization strategies. Top-left: OCR, Top-right: CoNLL-2000, bottom-left: CoNLL-2002, bottom-right: POS-WSJ.

included the *SAG* algorithm as described in Section 3.3, the *SAG-NUS* variant of Schmidt et al. (2013), and our proposed *SAG-NUS** strategy from Section 3.4.[10] We also tested SAGA variants of each of the SAG algorithms, and found that they gave very similar performance. All methods (except OEG) were initialized at zero.

Figure 3.1 shows the result of our experiments on the training objective and Figure 3.2 shows the result of tracking the test error. Here we measure the number of "effective passes", meaning $(1/n)$ times the number of times we performed the bottleneck operation of computing $\log p(b_i \mid a_i, x)$ and its gradient. This is an implementation-independent way to compare the convergence of the different al-

[10] We also tested *SG* with the proposed NUS scheme, but the performance was similar to the regular SG method. This is consistent with the analysis of Needell et al. (2014a, Corollary 3.1) showing that NUS for regular *SG* only improves the non-dominant term.

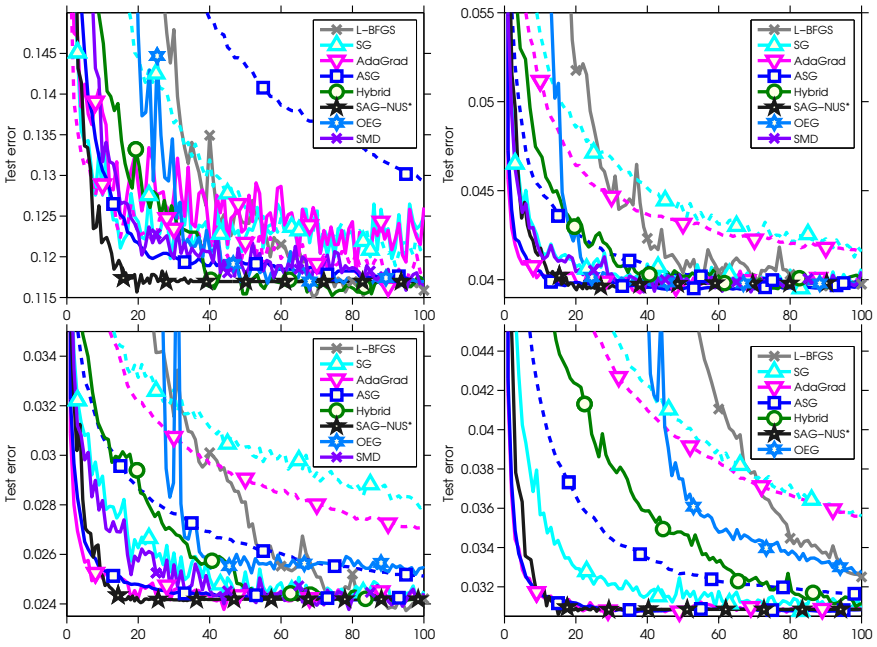

Figure 3.2: Test error against effective number of passes for different deterministic, stochastic, and semi-stochastic optimization strategies (this figure is best viewed in colour). Top-left: OCR, Top-right: CoNLL-2000, bottom-left: CoNLL-2002, bottom-right: POS-WSJ. The dotted lines show the performance of the classic stochastic gradient methods when the optimal step-size is not used. *Note that the performance of all classic stochastic gradient methods is much worse when the optimal step-size is not used*, whereas the SAG methods have an adaptive step-size so are not sensitive to this choice.

gorithms (most of whose runtimes differ only by a small constant), but we have included the performance in terms of runtime in Appendix B.5. For the different *SAG* methods that use a line-search we count the extra 'forward' operations used by the line-search as full evaluations of $\log p(b_i \mid a_i, x)$ and its gradient, even though these operations are cheaper because they do not require the backward pass nor computing the gradient. In these experiments we used $\lambda = 1/n$, which yields a value close to the optimal test error across all data sets. The objective is strongly-convex and thus has a unique minimum value. We approximated this value by

running L-BFGS for up to 1000 iterations, which always gave a value of x satisfying $\|\nabla f(x)\|_\infty \leq 1.4 \times 10^{-7}$, indicating that this is a very accurate approximation of the true solution. In the test error plots, we have excluded the *SAG* and *SAG-NUS* methods to keep the plots interpretable (while Pegasos does not appear because it performs very poorly), but Appendix B.3 includes these plots with all methods added. In the test error plots, we have also plotted as dotted lines the performance of the classic stochastic gradient methods when the second-best step-size is used.

We make several observations based on these experiments:

- *SG* outperformed *Pegasos*. Pegasos is known to move exponentially away from the solution in the early iterations (Bach and Moulines, 2011), meaning that $\|x_t - x^*\|_2 \geq \rho^t \|x_0 - x^*\|_2$ for some $\rho > 1$, while SG moves exponentially towards the solution ($\rho < 1$) in the early iterations (Nedic and Bertsekas, 2000).
- *ASG* outperformed *AdaGrad* and *SMD* (in addition to *SG*). ASG methods are known to achieve the same asymptotic efficiency as an optimal stochastic Newton method (Polyak and Juditsky, 1992), while AdaGrad and SMD can be viewed as approximations to a stochastic Newton method. Vishwanathan et al. (2006) did not compare to ASG, because applying ASG to large/sparse data requires the recursion of Xu (2010).
- *Hybrid* outperformed *L-BFGS*. The hybrid algorithm processes fewer data points in the early iterations, leading to cheaper iterations.
- None of the three algorithms *ASG*/*Hybrid*/*SAG* dominated the others: the relative ranks of these methods changed based on the data set and whether we could choose the optimal step-size.
- *OEG* performed very well on the first two datasets, but was less effective on the second two. By experimenting with various initializations, we found that we could obtain much better performance with OEG on these two datasets. We report these results in Appendix B.4, although Appendix B.5 shows that OEG was less competitive in terms of runtime.
- Both *SAG-NUS* methods outperform all other methods (except OEG) by a substantial margin based on the training objective, and are always among the

Dataset	Sparse	Marginals	Mixed
OCR	7.8×10^{-1}	1.1×10^{0}	2.1×10^{-1}
CoNLL-2000	4.8×10^{-3}	7.0×10^{-3}	6.1×10^{-4}
CoNLL-2002	6.4×10^{-4}	3.8×10^{-4}	7.0×10^{-5}
POS-WJ	1.3×10^{-3}	5.5×10^{-3}	3.6×10^{-4}

Table 3.2: Memory required by the datasets used in the experiments.

best methods in terms of the test error. Further, our proposed *SAG-NUS** always outperforms *SAG-NUS*.

On three of the four data sets, the best classic stochastic gradient methods (*AdaGrad* and *ASG*) seem to reach the optimal test error with a similar speed to the *SAG-NUS** method, although they require many passes to reach the optimal test error on the OCR data. Further, we see that the good test error performance of the *AdaGrad* and *ASG* methods *is very sensitive to choosing the optimal step-size*, as the methods perform much worse if we don't use the optimal step-size (dashed lines in Figure 3.2). In contrast, SAG uses an adaptive step-size and has virtually identical performance even if the initial value of L_g is too small by several orders of magnitude (the line-search quickly increases L_g to a reasonable value on the first training example, so the dashed black line in Figure 3.2 would be on top of the solid line).

To quantify the memory savings given by the choices in Section 3.3, in Table 3.2 we report the size of the memory required for these datasets under different memory-saving strategies divided by the memory required by the naive SAG algorithm. *Sparse* refers to only storing non-zero gradient values, *Marginals* refers to storing all unary and pairwise marginals, and *Mixed* refers to storing node marginals and the gradient with respect to pairwise features (recall that the pairwise features do not depend on a in our models). We have made the code available at https://www.cs.ubc.ca/~schmidtm/Software/SAG4CRF.html.

3.6 Discussion

Due to its memory requirements, it may be difficult to apply the SAG algorithm for natural language applications involving complex features that depend on a large number of labels.

However, grouping training examples into mini-batches can also reduce the memory requirement (since only the gradients with respect to the mini-batches would be needed). An alternative strategy for reducing the memory is to use the algorithm of Johnson and Zhang (2013) or Zhang et al. (2013). These require evaluating the chosen training example twice on each iteration, and occasionally require full passes through the data, but do not have the memory requirements of SAG (in our experiments, these performed similar to or slightly worse than running SAG at half speed).

We believe linearly-convergent stochastic gradient algorithms with non-uniform sampling could give a substantial performance improvement in a large variety of CRF training problems, and we emphasize that the method likely has extensions beyond what we have examined. For example, we have focused on the case of ℓ_2-regularization but for large-scale problems there is substantial interest in using ℓ_1-regularization CRFs (Lavergne et al., 2010; Tsuruoka et al., 2009; Zhou et al., 2011). Fortunately, such non-smooth regularizers can be handled with a proximal-gradient variant of the method, see Defazio et al. (2014). While we have considered chain-structured data the algorithm applies to general graph structures, and any method for computing/approximating the marginals could be adopted. Finally, the SAG algorithm could be modified to use multi-threaded computation as in the algorithm of Lavergne et al. (2010), and indeed might be well-suited to massively distributed parallel implementations.

Chapter 4

MASAGA: A Method for Optimization on Manifolds

The most common supervised learning methods in machine learning use empirical risk minimization during the training. The minimization problem can be expressed as minimizing a finite sum of loss functions that are evaluated at a single data sample. We consider the problem of minimizing a finite sum over a Riemannian manifold,

$$\min_{x \in \mathscr{X} \subseteq \mathscr{M}} f(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x),$$

where $\mathscr{X}$ is a geodesically convex set in the Riemannian manifold $\mathscr{M}$. Each function f_i is geodesically Lipschitz-smooth and the sum is geodesically strongly-convex over the set $\mathscr{X}$. The learning phase of several machine learning models can be written as an optimization problem of this form. This includes principal component analysis (PCA) (Zhang and Yang, 2017), dictionary learning (Sun et al., 2015), Gaussian mixture models (GMM) (Hosseini and Sra, 2015), covariance estimation (Wiesel, 2012), computing the Riemannian centroid (Jeuris et al., 2012), and the PageRank problem (Sra and Hosseini, 2016).

When $\mathscr{M} \equiv \mathbb{R}^d$, the problem reduces to convex optimization over a standard Euclidean space. An extensive body of literature studies this problem in deterministic and stochastic settings (Cauchy, 1847; Nemirovski et al., 2009; Nesterov, 1983; Polyak and Juditsky, 1992; Robbins and Monro, 1951b). It is possible to

convert the optimization over a manifold into an optimization in a Euclidean space by adding $x \in \mathscr{X}$ as an optimization constraint. The problem can then be solved using projected-gradient methods. However, the problem with this approach is that we are not explicitly exploiting the geometrical structure of the manifold. Further, a function could be non-convex in the Euclidean space but geodesically convex over an appropriate manifold. These factors can lead to poor performance for algorithms that operate with the Euclidean geometry, but algorithms that use the Riemannian geometry may converge as fast as algorithms for convex optimization in Euclidean spaces.

Stochastic optimization over manifolds and related convergence properties have received significant interest in the recent literature (Bonnabel, 2013; Kasai et al., 2016; Sato et al., 2017; Zhang et al., 2016; Zhang and Sra, 2016). Bonnabel (2013) and Zhang and Sra (2016) analyze the application of stochastic gradient descent (SGD) for optimization over manifolds. Similar to optimization over Euclidean spaces with SGD, these methods suffer from the aggregating variance problem (Zhao and Zhang, 2015) which leads to sublinear convergence rates.

When optimizing finite sums over Euclidean spaces, variance-reduction techniques have been introduced to reduce the variance in SGD in order to achieve faster convergence rates. The variance-reduction techniques can be categorized into two groups. The first group is memory-based approaches (Defazio et al., 2014; Le Roux et al., 2012; Mairal, 2013b; Shalev-Shwartz and Zhang, 2013b) such as the stochastic average gradient (SAG) method and its variant SAGA. Memory-based methods use the memory to store a stale gradient of each f_i, and in each iteration they update this "memory" of the gradient of a random f_i. The averaged stored value is used as an approximation of the gradient of f.

The second group of variance-reduction methods explored for Euclidean spaces require full gradient calculations and include the stochastic variance-reduced gradient (SVRG) method (Johnson and Zhang, 2013) and its variants (Konečnỳ and Richtárik, 2013; Mahdavi and Jin, 2013; Nguyen et al., 2017). These methods only store the gradient of f, and not the gradient of the individual f_i functions. But, these methods occasionally require evaluating the full gradient of f as part of their gradient approximation and require two gradient evaluations per iteration. Although SVRG often dramatically outperforms the classical gradient descent (GD)

and SGD, the extra gradient evaluation typically lead to a slower convergence than memory-based methods. Furthermore, the extra gradient calculations of SVRG can lead to worse performance than the classical SGD during the early iterations where SGD has the most advantage (Harikandeh et al., 2015). Thus, when the bottleneck of the process is the gradient computation itself, using memory-based methods like SAGA can improve performance (Bietti and Mairal, 2017; Dubey et al., 2016). Furthermore, for several applications it has been shown that the memory requirements can be alleviated by exploiting special structures in the gradients of the f_i (Le Roux et al., 2012; Shalev-Shwartz and Zhang, 2013b), as we did in the previous chapter for CRFs.

Several recent methods have extended SVRG to optimize the finite sum problem over a Riemannian manifold (Kasai et al., 2016; Sato et al., 2017; Zhang et al., 2016), which we refer to as RSVRG methods. Similar to the case of Euclidean spaces, RSVRG converges linearly for geodesically Lipschitz-smooth and strongly-convex functions. However, these methods also require the extra gradient evaluations associated with the original SVRG method. Thus, they may not perform as well as potential generalizations of memory-based methods like SAGA.

In this chapter we present *MASAGA*, a variant of SAGA to optimize finite sums over Riemannian manifolds. Similar to RSVRG, we show that it converges linearly for geodesically strongly-convex functions. We also show that both MASAGA and RSVRG with a non-uniform sampling strategy can converge faster than the uniform sampling scheme used in prior work. Finally, we consider the problem of finding the leading eigenvector, which minimizes a quadratic function over a sphere. We show that MASAGA converges linearly with uniform and non-uniform sampling schemes on this problem. For evaluation, we consider one synthetic and two real datasets. The real datasets are MNIST (LeCun et al., 1998) and the Ocean data (Mahadevan and Vasconcelos, 2010). We find the leading eigenvector of each class and visualize the results. On MNIST, the leading eigenvectors resemble the images of each digit class, while for the Ocean dataset we observe that the leading eigenvector represents the background image in the dataset.

In Section 4.1 we present an overview of essential concepts in Riemannian geometry, defining the geodesically convex and smooth function classes following Zhang and Sra (2016). We also briefly review the original SAGA algorithm. In

Section 4.2, we introduce the MASAGA algorithm and analyze its convergence under both uniform and non-uniform sampling. Finally, in Section 4.3 we empirically verify the theoretical linear convergence results.

4.1 Preliminaries

In this section we first present a review of Riemannian manifold concepts. However, for a more detailed review we refer the interested reader to the literature (Absil et al., 2009; Petersen et al., 2006; Udriste, 1994). Then, we introduce the class of functions that we optimize over such manifolds. Finally, we briefly review the original SAGA algorithm.

4.1.1 Riemannian Manifold

A Riemannian manifold is denoted by the pair $(\mathscr{M}, G)$, that consists of a smooth manifold $\mathscr{M}$ over $\mathbb{R}^d$ and a metric G. At any point x in the manifold $\mathscr{M}$, we define $\mathscr{T}_{\mathscr{M}}(x)$ to be the tangent plane of that point, and G defines an inner product in this plane. Formally, if p and q are two vectors in $\mathscr{T}_{\mathscr{M}}(x)$, then $\langle p, q \rangle_x = G(p,q)$. Similar to Euclidean space, we can define the norm of a vector and the angle between two vectors using G.

To measure the distance between two points on the manifold, we use the geodesic distance. Geodesics on the manifold generalize the concept of straight lines in Euclidean space. Let us denote a geodesic with $\gamma(t)$ which maps $[0,1] \to \mathscr{M}$ and is a function with constant gradient,

$$\frac{d^2}{dt^2}\gamma(t) = 0.$$

To map a point in $\mathscr{T}_{\mathscr{M}}(x)$ to $\mathscr{M}$, we use the exponential function $\mathrm{Exp}_x : \mathscr{T}_{\mathscr{M}}(x) \to \mathscr{M}$. Specifically, $\mathrm{Exp}_x(p) = z$ means that there is a geodesic curve $\gamma_x^z(t)$ on the manifold that starts from x (so $\gamma_x^z(0) = x$) and ends at z (so $\gamma_x^z(1) = z = \mathrm{Exp}_x(p)$) with a velocity of p ($\frac{d}{dt}\gamma_x^z(0) = p$). When the Exp function is defined for every point in the manifold, we call the manifold geodesically-complete. For example, the unit sphere in $\mathscr{R}^n$ is geodesically complete. If there is a unique geodesic curve between any two points in $\mathscr{M}' \in \mathscr{M}$, then the Exp_x function has an inverse defined

by the Log_x function. Formally the $\text{Log}_x \equiv \text{Exp}_x^{-1} : \mathscr{M}' \to \mathscr{T}_{\mathscr{M}}(x)$ function maps a point from $\mathscr{M}'$ back into the tangent plane at x. Moreover, the geodesic distance between x and z is the length of the unique shortest path between z and x, which is equal to $\|\text{Log}_x(z)\|_2 = \|\text{Log}_z(x)\|_2$.

Let u and $v \in \mathscr{T}_{\mathscr{M}}(x)$ be linearly independent so they specify a two dimensional subspace $\mathscr{S}_x \in \mathscr{T}_{\mathscr{M}}(x)$. The exponential map of this subspace, $\text{Exp}_x(\mathscr{S}_x) = \mathscr{S}_{\mathscr{M}}$, is a two dimensional submanifold in $\mathscr{M}$. The sectional curvature of S_M denoted by $\text{K}(\mathscr{S}_{\mathscr{M}}, x)$ is defined as a Gauss curvature of S_M at x (Ziller, 2014). The sectional curvature measures the curvature of surface of the manifold around a given point. This sectional curvature helps us in the convergence analysis of the optimization method. We use the following lemma in our analysis to give a trigonometric distance bound.

Lemma 4.1. *(Lemma 5 in Zhang and Sra (2016)) Let a, b, and c be the side lengths of a geodesic triangle in a manifold with sectional curvature lower-bounded by* $K_{\min}$. *Then*

$$a^2 \leq \frac{c\sqrt{|K_{\min}|}}{\tanh(\sqrt{|K_{\min}|}c)} b^2 + c^2 - 2bc\cos(\measuredangle(b,c)).$$

Another important map used in our algorithm is the parallel transport. It transfers a vector from a tangent plane to another tangent plane along a geodesic. This map is denoted by $\Gamma_x^z : \mathscr{T}_{\mathscr{M}}(x) \to \mathscr{T}_{\mathscr{M}}(z)$, and maps a vector from the tangent plane $\mathscr{T}_{\mathscr{M}}(x)$ to a vector in the tangent plane $\mathscr{T}_{\mathscr{M}}(z)$ while preserving the norm and inner product values.

$$\langle p, q \rangle_x = \langle \Gamma_x^z(p), \Gamma_x^z(q) \rangle_z$$

Grassmann manifold. Here we review the Grassmann manifold, denoted $\text{Grass}(p,n)$, as a practical Riemannian manifold used in machine learning. Let p and n be positive integers with $p \leq n$. $\text{Grass}(p,n)$ contains all matrices in $\mathbb{R}^{n \times p}$ with orthonormal columns (the class of orthogonal matrices). By the definition of an orthogonal matrix, if $M \in \text{Grass}(p,n)$ then we have $M^\top M = I$, where $I \in \mathbb{R}^{p \times p}$ is the identity matrix. Let $q \in \mathscr{T}_{\text{Grass}(p,n)}(x)$, and $q = U\Sigma V^\top$ be its p-rank singular value decom-

position. Then we have:

$$\mathrm{Exp}_x(tq) = xV\cos(t\Sigma)V^\top + U\sin(t\Sigma)V^\top.$$

The parallel transport along a geodesic curve $\gamma(t)$ such that $\gamma(0) = x$ and $\gamma(1) = z$ is defined as:

$$\Gamma_x^z(tq) = (-xV\sin(t\Sigma)U^\top + U\cos(t\Sigma)U^\top + I - UU^\top)q.$$

4.1.2 Smoothness and Convexity on Manifold

In this section, we define convexity and smoothness of a function over a manifold following Zhang and Sra (2016). We call $\mathscr{X} \in \mathscr{M}$ geodesically convex if for any two points y and z in $\mathscr{X}$, there is a geodesic $\gamma(t)$ starting from y and ending in z with a curve inside of $\mathscr{X}$. For simplicity, we drop the subscript in the inner product notation.

Formally, a function $f : \mathscr{X} \to \mathbb{R}$ is called geodesically convex if for any y and z in $\mathscr{X}$ and the corresponding geodesic γ, for any $t \in [0,1]$ we have:

$$f(\gamma(t)) \leq (1-t)f(y) + tf(z).$$

Similar to the Euclidean space, if the Log function is well defined we have the following for convex functions:

$$f(z) + \left\langle g_z, \mathrm{Log}_z(y) \right\rangle \leq f(y),$$

where g_z is a subgradient of f at x. If f is a differentiable function, the Riemannian gradient of f at z is a vector g_z which satisfies $\frac{d}{dt}|_{t=0} f(\mathrm{Exp}_z(tg_z)) = \langle g_z, \nabla f(z) \rangle_z$, with ∇f being the gradient of f in $\mathbb{R}^n$. Furthermore, we say that f is geodesically μ-strongly convex if there is a $\mu > 0$ such that:

$$f(z) + \left\langle g_z, \mathrm{Log}_z(y) \right\rangle + \frac{\mu}{2}\|\mathrm{Log}_z(y)\|_2^2 \leq f(y).$$

Let $x^* \in \mathscr{X}$ be the optimum of f. This implies that there exists a subgradient at x^*

Algorithm 7 The Original SAGA Algorithm

1: **Input:** Learning rate η.
2: Initialize $x_0 = 0$ and memory $M^{(0)}$ with gradient of x_0.
3: **for** $t = 1, 2, 3, \ldots$ **do**
4: $\quad \hat{\mu} = \frac{1}{n}\sum_{j=1}^{n} M^t[j]$
5: $\quad$ Pick i_t uniformly at random from $\{1 \ldots n\}$.
6: $\quad v_t = \nabla f_{i_t}(x_t) - M^t[i_t] + \frac{1}{n}\sum_{j=1}^{n} M^t[j]$
7: $\quad x_{t+1} = x_t - \eta(v_t)$
8: $\quad$ Set $M^{t+1}[i_t] = \nabla f_{i_t}(x_t)$ and $M^{t+1}[j] = M^t[j]$ for all $j \neq i_t$.
9: **end for**

with $g_{x^*} = 0$ which implies that the following inequalities hold:

$$\|\mathrm{Log}_z(x^*)\|_2^2 \leq \frac{2}{\mu}(f(z) - f(x^*))$$
$$\left\langle g_z, \mathrm{Log}_z(x^*)\right\rangle + \frac{\mu}{2}\|\mathrm{Log}_z(x^*)\|_2^2 \leq 0.$$

Finally, an f that is differentiable over $\mathscr{M}$ is said to be a Lipschitz-smooth function with the parameter $L > 0$ if its gradient satisfies the following inequality:

$$\|g_z - \Gamma_y^z[g_y]\|_2 \leq L\|\mathrm{Log}_z(y)\|_2 = L\,\mathrm{d}(z,y),$$

where $\mathrm{d}(z,y)$ is the distance between z and y. For a geodesically smooth f the following inequality also holds:

$$f(y) \leq f(z) + \left\langle g_z, \mathrm{Log}_z(y)\right\rangle + \frac{L}{2}\|\mathrm{Log}_z(y)\|_2^2.$$

4.1.3 SAGA Algorithm

In this section we briefly review the SAGA method (Defazio et al., 2014) and the assumptions associated with it. SAGA assumes f is μ-strongly convex, each f_i is convex, and each gradient ∇f_i is Lipschitz-continuous with constant L. The method generates a sequence of iterates x_t using the SAGA Algorithm 7 (line 7). In the algorithm, M is the memory used to store stale gradients. During each iteration, SAGA picks one f_{i_t} randomly and evaluates its gradient at the current iterate value,

$\nabla f_{i_t}(x_t)$. Next, it computes v_t as the difference between the current $\nabla f_{i_t}(x_t)$ and the corresponding stale gradient of f_{i_t} stored in the memory plus the average of all stale gradients (line 6). Then it uses this vector v_t as an approximation of the full gradient and updates the current iterate similar to the gradient descent update rule. Finally, SAGA updates the stored gradient of f_{i_t} in the memory with the new value of $\nabla f_{i_t}(x_t)$.

Let $\rho_{\text{saga}} = \frac{\mu}{2(n\mu+L)}$. Defazio et al. (2014) show that the iterate value x_t converges to the optimum x^* linearly with a contraction rate $1-\rho_{\text{saga}}$,

$$\mathbb{E}\left[\|x_t - x^*\|_2^2\right] \leq (1-\rho_{\text{saga}})^t C,$$

where C is a positive scalar.

4.2 Optimization on Manifolds with SAGA

In this section we introduce the MASAGA algorithm (see Alg. 8). We make the following assumptions:

1. Each f_i is geodesically L-Lipschitz continuous.
2. f is geodesically μ-strongly convex.
3. f has an optimum in $\mathcal{X}$, in other words $x^* \in \mathcal{X}$.
4. The diameter of $\mathcal{X}$ is bounded above, in other words $\max_{u,v\in\mathcal{X}} \mathrm{d}(u,v) \leq D$.
5. Log_x is defined when $x \in \mathcal{X}$.
6. The sectional curvature of $\mathcal{X}$ is bounded, in other words, $K_{\min} \leq K_{\mathcal{X}} \leq K_{\max}$.

These assumptions also commonly appear in the previous work (Kasai et al., 2016; Sato et al., 2017; Zhang et al., 2016; Zhang and Sra, 2016). Similar to the previous work (Kasai et al., 2016; Zhang et al., 2016; Zhang and Sra, 2016), we

Algorithm 8 MASAGA Algorithm

1: **Input:** Learning rate η and $x_0 \in \mathscr{M}$.
2: Initialize memory $M^{(0)}$ with gradient of x_0.
3: **for** $t = 1, 2, 3, \ldots$ **do**
4: $\quad \hat{\mu} = \frac{1}{n} \sum_{j=1}^{n} M^t[j]$
5: $\quad$ Pick i_t uniformly at random from $\{1 \ldots n\}$.
6: $\quad v_t = \nabla f_{i_t}(x_t) - \Gamma_{x_0}^{x_t} [M^t[i_t] - \hat{\mu}]$
7: $\quad x_{t+1} = \mathrm{Exp}_{x_t}(-\eta(v_t))$
8: $\quad$ Set $M^{t+1}[i_t] = \Gamma_{x_t}^{x_0} [\nabla f_{i_t}(x_t)]$ and $M^{t+1}[j] = M^t[j]$ for all $j \neq i_t$.
9: **end for**

also define the constant ζ which is essential in our analysis:

$$\zeta = \begin{cases} \frac{\sqrt{|K_{\min}|}D}{\tanh(\sqrt{|K_{\min}|}D)} & \text{if } K_{\min} < 0 \\ 1 & \text{if } K_{\min} \geq 0 \end{cases}$$

In MASAGA we modify two parts of the original SAGA: (i) since gradients are in different tangent planes, we use parallel transport to map them into the same tangent plane and then do the variance reduction step (line 6 of Alg. 8), and (ii) we use the Exp function to map the update step back into the manifold (line 7 of Alg. 8).

4.2.1 Convergence Analysis

We analyze the convergence of MASAGA considering the above assumptions and show that it converges linearly. In our analysis, we use the fact that MASAGA's estimation of the full gradient v_t is unbiased (like SAGA), in other words $\mathbb{E}[v_t] = \nabla f(x_t)$. For simplicity, we use ∇f to denote the Riemannian gradient instead of g_x. We assume that there exists an incremental first-order oracle (IFO) (Agarwal and Bottou, 2014) that gets an $i \in \{1, ..., n\}$, and an $x \in \mathscr{X}$, and returns $(f_i(x), \nabla f_i(x)) \in (\mathbb{R} \times \mathscr{T}_{\mathscr{M}}(x))$.

Theorem 4.2. *If each f_i is geodesically L-smooth and f is geodesically μ-strongly convex over the Riemannian manifold $\mathscr{M}$, the MASAGA algorithm with the constant step size $\eta = \frac{2\mu + \sqrt{\mu^2 - 8\rho(1+\alpha)\zeta L^2}}{4(1+\alpha)\zeta L^2}$ converges linearly while satisfying the fol-*

lowing:

$$\mathbb{E}\left[\mathrm{d}^2(x_t, x^*)\right] \leq (1-\rho)^t \Upsilon^0,$$

where $\rho = \min\{\frac{\mu^2}{8(1+\alpha)\zeta L^2}, \frac{1}{n} - \frac{1}{\alpha n}\}$, $\Upsilon^0 = 2\alpha\zeta\eta^2 \sum_{i=1}^n \|M^0[i] - \Gamma_{x^*}^{x_0}[\nabla f_i(x^*)]\|_2^2 + \mathrm{d}^2(x_0, x^*)$ *is a positive scalar, and* $\alpha > 1$ *is a constant.*

The proof of this theorem can be found in Appendix C.1.

Corollary 4.3. *Let* $\beta = \frac{n\mu^2}{8\zeta L^2}$, *and* $\bar{\alpha} = \beta + \sqrt{\frac{\beta^2}{4} + 1} > 1$. *If we set* $\alpha = \bar{\alpha}$ *then we will have* $\rho = \frac{\mu^2}{8(1+\bar{\alpha})\zeta L^2} = \frac{1}{n} - \frac{1}{\bar{\alpha} n}$. *Furthermore, to reach an* ε *accuracy, in other words* $\mathbb{E}\left[\mathrm{d}^2(x_T, x^*)\right] < \varepsilon$, *we require that the total number of MASAGA (Alg. 8) iterations* T *satisfy the following inequality:*

$$T \geq \left(\frac{8(1+\bar{\alpha})\zeta L^2}{\mu^2}\right) \log\left(\frac{1}{\varepsilon}\right). \tag{4.1}$$

Note that this bound is similar to the bound of Zhang et al. (2016). To make it clear, notice that $\bar{\alpha} \leq 2\beta + 1$. Therefore, if we plug this upper-bound into Inequality 4.1 we get

$$T = \mathscr{O}\left(\frac{(2\beta+2)\zeta L^2}{\mu^2}\right) \log\left(\frac{1}{\varepsilon}\right) = \mathscr{O}\left(\frac{n\mu^2}{8\zeta L^2}\frac{\zeta L^2}{\mu^2} + \frac{\zeta L^2}{\mu^2}\right) \log\left(\frac{1}{\varepsilon}\right) = \mathscr{O}\left(n + \frac{\zeta L^2}{\mu^2}\right) \log\left(\frac{1}{\varepsilon}\right).$$

The $\frac{L^2}{\mu^2}$ term in the above bound is the squared condition number that could be prohibitively large in machine learning applications. RSVRG also has this dependency. In contrast, the original SAGA and SVRG algorithms only depend on $\frac{L}{\mu}$ on convex function within linear spaces. In the next section, we improve upon this bound through non-uniform sampling techniques.

4.2.2 MASAGA with Non-uniform Sampling

Using non-uniform sampling for stochastic optimization in Euclidean spaces can help stochastic optimization methods achieve a faster convergence rate (Harikandeh et al., 2015; Needell et al., 2014b; Schmidt et al., 2015b). In this section, we assume that each f_i has its own geodesically L_i-Lipschitz smoothness as opposed to a single geodesic Lipschitz smoothness $L = \max\{L_i\}$. Now, instead of uniformly sampling f_i, we sample f_i with probability $\frac{L_i}{n\bar{L}}$, where $\bar{L} = \frac{1}{n}\sum_{i=1}^n L_i$. In machine

learning applications, we often have $\bar{L} \ll L$. Using this non-uniform sampling scheme, the iteration update is set to

$$x_{t+1} = \mathrm{Exp}_{x_t}\left(-\eta(\frac{\bar{L}}{L_{i_t}}v_t)\right),$$

which keeps the search direction unbiased, in other words $\mathbb{E}\left[\frac{\bar{L}}{L_{i_t}}v_t\right] = \nabla f(x_t)$. The following theorem shows the convergence of the new method.

Theorem 4.4. *If f_i is geodesically L_i-smooth and f is geodesically μ-strongly convex over the manifold $\mathscr{M}$, the MASAGA algorithm with the defined non-uniform sampling scheme and the constant step size $\eta = \frac{2\mu+\sqrt{\mu^2-8\rho(\bar{L}+\alpha L)\frac{\zeta}{\gamma}\bar{L}}}{4(\bar{L}+\alpha L)\frac{\zeta}{\gamma}\bar{L}}$ converges linearly as follows:*

$$\mathbb{E}\left[\mathrm{d}^2(x_t,x^*)\right] \leq (1-\rho)^t\Upsilon^0,$$

where $\rho = \min\{\frac{\gamma\mu^2}{8(1+\alpha)\zeta L\bar{L}}, \frac{\gamma}{n} - \frac{\gamma}{\alpha n}\}$, $\gamma = \frac{\min\{L_i\}}{\bar{L}}$, $L = \max\{L_i\}$, $\bar{L} = \frac{1}{n}\sum_{i=1}^n L_i$, and $\alpha > 1$ is a constant, and $\Upsilon^0 = \frac{2\alpha\zeta\eta^2}{\gamma}\sum_{i=1}^n \frac{\bar{L}}{L_i}\|M^0[i] - \Gamma_{x^*}^{x_0}[\nabla f_i(x^*)]\|_2^2 + \mathrm{d}^2(x_0,x^*)$ are positive scalars.

Proof of the above theorem can be found in Appendix C.2.

Corollary 4.5. *Let $\beta = \frac{n\mu^2}{8\zeta L\bar{L}}$, and $\bar{\alpha} = \beta + \sqrt{\frac{\beta^2}{4}+1} > 1$. If we set $\alpha = \bar{\alpha}$ then we have $\rho = \frac{\gamma\mu^2}{8(1+\bar{\alpha})\zeta L\bar{L}} = \frac{\gamma}{n} - \frac{\gamma}{\bar{\alpha}n}$. Now, to reach an ε accuracy, in other words $\mathbb{E}\left[\mathrm{d}^2(x_T,x^*)\right] < \varepsilon$, we require:*

$$T = \mathscr{O}(n + \frac{\zeta L\bar{L}}{\gamma\mu^2})\log(\frac{1}{\varepsilon}), \tag{4.2}$$

where T is the number of the necessary iterations.

Observe that the number of iterations T in Equality 4.2 depends on $\bar{L}L$ instead of L^2. When $\bar{L} \ll L$, the difference could be significant. Thus, MASAGA with non-uniform sampling could achieve an ε accuracy faster than MASAGA with uniform sampling. Unlike our previous non-uniform SAGA analysis, this analysis doesn't require two gradients per iteration. Note that computing the L_i for general f_i is not a tractable problem. However we can approximate it locally by using the definition of g-smoothness.

Similarly we can use the same sampling scheme for the RSVRG algorithm (Zhang et al., 2016) and improve its convergence. Specifically, if we change the update rule of Algorithm 1 of Zhang et al. (2016) to

$$x_{t+1}^{s+1} = \mathrm{Exp}_{x_t^{s+1}} - \eta(\frac{\bar{L}}{L_{i_t}} v_t^{s+1}),$$

then Theorem 1 and Corollary 1 of Zhang et al. (2016) will change to the following ones.

Theorem 4.6. ***[Theorem 1 of Zhang et al. (2016) with non-uniform sampling]*** *If we use non-uniform sampling in Algorithm 1 of RSVRG (Zhang et al., 2016) and run it with the option I as described in the work, and let*

$$\alpha = \frac{3\zeta\eta\bar{L}^2}{\mu - 2\zeta\eta\bar{L}^2} + \frac{(1 + 4\zeta\eta^2 - 2\eta\mu)^m(\mu - 5\zeta\eta\bar{L}^2)}{\mu - 2\zeta\eta\bar{L}^2} < 1,$$

where m is the number of the inner loop iterations, then through S iterations of the outer loop, we have

$$\mathbb{E}\left[\mathrm{d}^2(\tilde{x}^S, x^*)\right] \leq (\alpha)^S \mathrm{d}^2(\tilde{x}^0, x^*).$$

The above theorem can be proved through a simple modification to the proof of Theorem 1 in RSVRG (Zhang et al., 2016).

Corollary 4.7. ***[Corollary 1 of Zhang et al. (2016) with non-uniform sampling]*** *With non-uniform sampling in Algorithm 1 of RSVRG, after* $\mathcal{O}(n + \frac{\zeta\bar{L}^2}{\gamma\mu^2})\log(\frac{1}{\varepsilon})$ *IFO calls, the output* x_a *satisfies*

$$\mathbb{E}\left[f(x_a) - f(x^*)\right] \leq \varepsilon.$$

Note that through the non-uniform sampling scheme we improved the RSVRG (Zhang et al., 2016) convergence by replacing the L^2 term with a smaller $\bar{L}^2$ term.

4.3 Experiments: Computing the leading eigenvector

Computing the parallel transport and exponential maps, which are the essential parts of MASAGA are hard for different manifolds. But for some problems such

as finding the largest eigenvalue and the PCA problem, we can find the closed-form formula for the parallel transport and exponential map. Since these maps are structurally similar for PCA and the largest eigenvector problems, we consider one of these problems. Computing the leading eigenvector is important in many real-world applications. It is widely used in social networks, computer networks, and metabolic networks for community detection and characterization (Newman, 2006). It can be used to extract a feature that "best" represents the dataset (Guyon et al., 2012) to aid in tasks such as classification, regression, and background subtraction. Furthermore, it is used in the PageRank algorithms which requires computing the principal eigenvector of the matrix describing the hyperlinks in the web (Page et al., 1999). These datasets can be huge (the web has more than three billion pages (Kamvar et al., 2004)). Therefore, speeding up the leading eigenvector computation will have a significant impact on many applications.

We evaluate the convergence of MASAGA on the problem of computing the leading eigenvalue on several datasets. The problem is written as follows:

$$\min_{\{x \,|\, x^\top x = 1\}} f(x) = -\frac{1}{n} x^\top \left(\sum_{i=1}^{n} z_i z_i^\top \right) x, \tag{4.3}$$

which is a non-convex objective in the Euclidean space $\mathbb{R}^d$, but a (strongly-)convex objective over the Riemannian manifold. Therefore, MASAGA can achieve a linear convergence rate on this problem. We apply our algorithm on the following datasets:

- **Synthetic.** We generate Z as a 1000×100 matrix where each entry is sampled uniformly from $(0, 1)$. To diversify the Lipschitz constants of the individual z_i's, we multiply each z_i with an integer obtained uniformly between 1 and 100.

- **MNIST (LeCun et al., 1998).** We randomly pick $10,000$ examples corresponding to digits 0-9 resulting in a matrix $Z \in \mathbb{R}^{10,000 \times 784}$.

- **Ocean.** We use the ocean video sequence data found in the UCSD background subtraction dataset (Mahadevan and Vasconcelos, 2010). It consists of 176 frames, each resized to a 94×58 image.

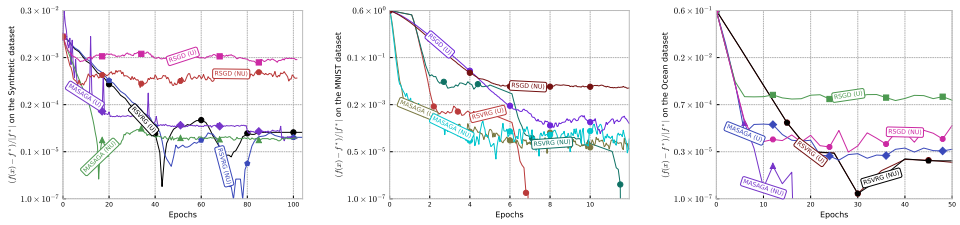

Figure 4.1: Comparison of MASAGA (ours), RSVRG, and RSGD for computing the leading eigenvector. The suffix (U) represents uniform sampling and (NU) the non-uniform sampling variant.

(a) Example MNIST images (b) Exact solution (c) MASAGA

Figure 4.2: The obtained leading eigenvectors of all MNIST digits.

In all experiments, we compare MASAGA against RSGD (Zhang et al., 2016) and RSVRG (Bonnabel, 2013). For solving geodesically smooth convex functions on the Riemannian manifold, RSGD and RSVRG achieve sublinear and linear convergence rates respectively. Since the manifold for Eq. (4.3) is that of a sphere, we have the following functions:

$$\begin{aligned} P_X(H) &= H - \text{trace}(X^\top H)X, & \nabla_r f(X) &= P_X(\nabla f(X)), \\ \text{Exp}_X(U) &= \cos(||U||)X + \frac{\sin(||U||)}{||U||}U, & \Gamma_y^x(U) &= P_y(U), \end{aligned} \tag{4.4}$$

where P corresponds to the tangent space projection function, $\nabla_r f$ the Riemannian gradient function, Exp the exponential map function, and Γ the transport function. We evaluate the progress of our algorithms at each epoch t by computing the relative error between the objective value and the optimum as $\frac{f(x^t)-f^*}{|f^*|}$. We have made the code available at https://github.com/IssamLaradji/MASAGA.

For each algorithm, a grid-search over the learning rates $\{10^{-1}, 10^{-2}, \ldots, 10^{-9}\}$ is performed and plot the results of the algorithm with the best performance in Figure 4.1. This plot shows that MASAGA is consistently faster than RSGD

(a) MNIST digits (b) MASAGA for digits 1-6

Figure 4.3: The obtained leading eigenvectors of the MNIST digits 1-6.

(a) Example Ocean frames

(b) Exact solution (c) RSGD (d) MASAGA (e) RSVRG

Figure 4.4: The obtained leading eigenvectors of the ocean dataset after 20 iterations.

and RSVRG in the first few epochs. While it is expected that MASAGA beats RSGD since it has a better convergence rate, the reason MASAGA can outperform RSVRG is that RSVRG needs to occasionally re-compute the full gradient. Further, at each iteration MASAGA requires a single gradient evaluation instead of the two evaluations required by RSVRG. We see in Figure 4.1 that non-uniform (NU) sampling often leads to faster progress than uniform (U) sampling, which is consistent with the theoretical analysis. In the NU sampling case, we sample a vector z_i based on its Lipschitz constant $L_i = ||z_i||^2$. Note that for problems where L_i is not known or costly to compute, we can estimate it by using Algorithm 2 of Schmidt et al. (2015b).

Figures 4.2 and 4.3 show the leading eigenvectors obtained for the MNIST

dataset. We run MASAGA on 1.0000×10^4 images of the MNIST dataset and plot its solution in Figure 4.2. We see that the exact solution is similar to the solution obtained by MASAGA, which represent the most common strokes among the MNIST digits. Furthermore, we ran MASAGA on 500 images for digits 1-6 independently and plot its solution for each class in Figure 4.3. Since most digits of the same class have similar shapes and are fairly centered, it is expected that the leading eigenvector would be similar to one of the digits in the dataset.

Figure 4.4 shows qualitative results comparing MASAGA, RSVRG, and RSGD. We run each algorithm for 20 iterations and plot the results. MASAGA's and RSVRG's results are visually similar to the exact solution. However, the RSGD result is visually different than the exact solution (the difference is in the center-left of the two images).

4.4 Conclusion

We introduced MASAGA which is a stochastic variance-reduced optimization algorithm for Riemannian manifolds. We analyzed the algorithm and showed that it converges linearly when the objective function is geodesically Lipschitz-smooth and strongly convex. We also showed that using non-uniform sampling improves the convergence speed of both MASAGA and RSVRG algorithms (RSVRG could be further speed up using the idea from Chapter 2). Finally, we evaluated our method on a synthetic dataset and two real datasets where we empirically observed linear convergence. The empirical results show that MASAGA outperforms RSGD and is faster than RSVRG in the early iterations. For future research, one can extend MASAGA by deriving convergence rates for the non-convex case of geodesic objective functions. One also can explore accelerated variance-reduction methods and block coordinate descent based methods (Nutini et al., 2017) for Riemannian optimization. Another potential future work of interest is a study of relationships between the condition number of a function within the Euclidean space and its corresponding condition number within a Riemannian manifold, and the effects of sectional curvature on it.

Chapter 5

Stochastic Variational Inference with General Divergence Functions

Variational inference methods are one of the most widely-used computational tools to deal with the intractability of Bayesian inference, while stochastic gradient (SG) methods are one of the most widely-used tools for solving optimization problems on huge datasets. There is a large number of works exploring SG methods for variational inference (Hoffman et al., 2013; Kucukelbir et al., 2014; Mnih and Gregor, 2014; Ranganath et al., 2013; Salimans et al., 2013; Titsias and Lázaro-Gredilla, 2014). In many settings, these methods can yield simple updates and scale to huge datasets.

A challenge that has been addressed in many of those works on this topic is that the "black-box" SG method ignores the geometry of the variational-parameter space. This has lead to methods like the stochastic variational inference (SVI) method of Hoffman et al. (2013), that uses *natural gradients* to exploit the geometry. This leads to better performance in practice, but this approach only applies to conditionally-conjugate models. In addition, it is not clear how using natural gradients for variational inference affects the theoretical convergence rate of SG methods.

In this chapter we consider a general framework that (i) can be stochastic to al-

low huge datasets, (ii) can exploit the geometry of the variational-parameter space to improve performance, and (iii) can yield a closed-form update even for non-conjugate models. The new framework can be viewed as a stochastic generalization of the proximal-gradient method of Khan et al. (2015b), which splits the objective into conjugate and non-conjugate terms. By linearizing the non-conjugate terms, this previous method as well as our new method yield simple closed-form proximal-gradient updates even for non-conjugate models.

While proximal-gradient methods have been well-studied in the optimization community (Beck and Teboulle, 2009), like SVI there is nothing known about the convergence rate of the method of Khan et al. (2015b) because it uses "divergence" functions which do not satisfy standard assumptions. Our second contribution is to *analyze the convergence rate* of the proposed method. In particular, we generalize an existing result on the convergence rate of stochastic mirror descent in non-convex settings (Ghadimi et al., 2014) to allow a general class of divergence functions that includes the cases above (in both deterministic and stochastic settings). While it has been observed empirically that including an appropriate divergence function enables larger steps than basic SG methods, this work gives the first theoretical result justifying the use of these more-general divergence functions. It in particular reveals how different factors affect the convergence rate such as the Lipschitz-continuity of the lower bound, the information geometry of the divergence functions, and the variance of the stochastic approximation. Our results also suggest conditions under which the proximal-gradient steps of Khan et al. (2015b) can make more progress than (non-split) gradient steps, and sheds light on the choice of step-size for these methods.

5.1 Variational Inference

Consider a general latent variable model where we have a data vector $\mathbf{y}$ of length N and a latent vector $\mathbf{z}$ of length D. In Bayesian inference, we are interested in computing the marginal likelihood $p(\mathbf{y})$, which can be written as the integral of the joint distribution $p(\mathbf{y},\mathbf{z})$ over all values of $\mathbf{z}$. This integral is often intractable, and in variational inference we typically approximate it with the evidence lower-bound optimization (ELBO) approximation $\underline{\mathscr{L}}$. This approximation introduces

a distribution $q(\mathbf{z}|\lambda)$ and chooses the variational parameters λ to maximize the following lower bound on the marginal likelihood:

$$\begin{aligned} \log p(\mathbf{y}) &= \log \int q(\mathbf{z}|\lambda) \frac{p(\mathbf{y},\mathbf{z})}{q(\mathbf{z}|\lambda)} \, d\mathbf{z} \\ &\geq \max_{\lambda \in \mathscr{S}} \underline{\mathscr{L}}(\lambda) := \mathbb{E}_{q(\mathbf{z}|\lambda)} \left[\log \frac{p(\mathbf{y},\mathbf{z})}{q(\mathbf{z}|\lambda)} \right]. \end{aligned} \tag{5.1}$$

The inequality follows from concavity of the logarithm function. The set $\mathscr{S}$ is the set of valid parameters λ.

To optimize λ, one of the seemingly-simplest approaches is gradient descent: $\lambda_{k+1} = \lambda_k + \beta_k \nabla \underline{\mathscr{L}}(\lambda_k)$, which can be viewed as optimizing a quadratic approximation of $\underline{\mathscr{L}}$,

$$\lambda_{k+1} = \operatorname*{argmin}_{\lambda \in \mathscr{S}} \left[-\lambda^T \nabla \underline{\mathscr{L}}(\lambda_k) + \frac{1}{2\beta_k} \|\lambda - \lambda_k\|_2^2 \right]. \tag{5.2}$$

While we can often choose the family q so that it has convenient computational properties, it might be impractical to apply gradient descent in this context when we have a very large dataset or when some terms in the lower bound are intractable.

Recently, SG methods have been proposed to deal with these issues (Ranganath et al., 2013; Titsias and Lázaro-Gredilla, 2014): they allow large datasets by using random subsets (mini-batches) and can approximate intractable integrals using Monte Carlo methods that draw samples from $q(\mathbf{z}|\lambda)$.

A second drawback of applying gradient descent to variational inference is that it uses the Euclidean distance and thus ignores the *geometry of the variational-parameter space*, which often results in slow convergence. Intuitively, (5.2) implies that we should move in the direction of the gradient, but not move λ_{k+1} too far away from λ_k in terms of the Euclidean distance. However, the Euclidean distance is not appropriate for variational inference because λ is the parameter vector of a distribution; the Euclidean distance is often a poor measure of dissimilarity between distributions. The following example from Hoffman et al. (2013) illustrates this point: the two normal distributions $\mathscr{N}(0, 10000)$ and $\mathscr{N}(10, 10000)$ are almost indistinguishable, yet the Euclidean distance between their parameter vectors is 10, whereas the distributions $\mathscr{N}(0, 0.01)$ and $\mathscr{N}(0.1, 0.01)$ barely overlap,

but their Euclidean distance between parameters is only 0.1.

Natural-Gradient Methods: The canonical way to address the problem above is by replacing the Euclidean distance in (5.2) with another divergence function. For example, the *natural gradient* method defines the iteration by using the symmetric Kullback-Leibler (KL) divergence (Amari, 1998; Hoffman et al., 2013; Pascanu and Bengio, 2013),

$$\lambda_{k+1} = \underset{\lambda \in \mathscr{S}}{\operatorname{argmin}} \left[-\lambda^T \nabla \underline{\mathscr{L}}(\lambda_k) + \frac{1}{\beta_k} \mathbb{D}_{KL}^{sym}[q(\mathbf{z}|\lambda) \,||\, q(\mathbf{z}|\lambda_k)] \right]. \tag{5.3}$$

This leads to the update

$$\lambda_{k+1} = \lambda_k + \beta_k \left[\nabla^2 \mathbf{G}(\lambda_k) \right]^{-1} \nabla \underline{\mathscr{L}}(\lambda_k), \tag{5.4}$$

where $\mathbf{G}(\lambda)$ is the Fisher information-matrix,

$$\mathbf{G}(\lambda) := \mathbb{E}_{q(\mathbf{z}|\lambda)} \left\{ \left[\nabla_\lambda \log q(\mathbf{z}|\lambda) \right] \left[\nabla_\lambda \log q(\mathbf{z}|\lambda) \right]^T \right\}.$$

Hoffman et al. (2013) show that the natural-gradient update can be computationally simpler than gradient descent for conditionally-conjugate exponential family models. In this family, we assume that the distribution of $\mathbf{z}$ factorizes as $\prod_i p(\mathbf{z}^i|\mathrm{pa}^i)$ where $\mathbf{z}^i$ are disjoint subsets of $\mathbf{z}$ and pa^i are the parents of the $\mathbf{z}^i$ in a directed acyclic graph. This family also assumes that each conditional distribution is in the exponential family,

$$p(\mathbf{z}^i|\mathrm{pa}^i) := h^i(\mathbf{z}^i) \exp\left[[\eta^i(\mathrm{pa}^i)]^T \mathbf{T}^i(\mathbf{z}^i) - A^i(\eta^i) \right],$$

where η^i are the natural parameters, $\mathbf{T}^i(\mathbf{z}^i)$ are the sufficient statistics, $A^i(\eta^i)$ is the partition function, and $h^i(\mathbf{z}^i)$ is the base measure. Hoffman et al. (2013) consider a mean-field approximation $q(\mathbf{z}|\lambda) = \prod_i q^i(\mathbf{z}^i|\lambda^i)$ where each q^i belongs to the same exponential-family distribution as the joint distribution,

$$q^i(\mathbf{z}^i) := h^i(\mathbf{z}^i) \exp\left[(\lambda^i)^T \mathbf{T}^i(\mathbf{z}^i) - A^i(\lambda^i) \right].$$

The parameters of this distribution are denoted by λ^i to differentiate them from the joint-distribution parameters η^i.

As shown by Hoffman et al. (2013), the Fisher matrix for this problem is equal to $\nabla^2 A^i(\lambda^i)$ and the gradient of the lower bound with respect to λ^i is equal to $\nabla^2 A^i(\lambda^i)(\lambda^i - \lambda^i_*)$ where λ^i_* are the mean-field parameters (see Paquet, 2014). Therefore, when computing the natural-gradient, the $\nabla^2 A^i(\lambda^i)$ terms cancel out and the natural-gradient is simply $\lambda^i - \lambda^i_*$ which is much easier to compute than the actual gradient. Unfortunately, for non-conjugate models this cancellation does not happen and the simplicity of the update is lost. The Riemannian conjugate-gradient method of Honkela et al. (2011) has similar issues, in that computing $\nabla^2 A(\lambda)$ is typically very costly.

KL-Divergence Based Methods: Rather than using the symmetric-KL, Theis and Hoffman (2015) consider using the KL divergence $\mathbb{D}_{KL}[q(\mathbf{z}|\lambda) \,\|\, q(\mathbf{z}|\lambda_k)]$ within a stochastic proximal-point method:

$$\lambda_{k+1} = \underset{\lambda \in \mathscr{S}}{\operatorname{argmin}} \left[-\underline{\mathscr{L}}(\lambda) + \frac{1}{\beta_k} \mathbb{D}_{KL}[q(\mathbf{z}|\lambda) \,\|\, q(\mathbf{z}|\lambda_k)] \right]. \tag{5.5}$$

This method yields better convergence properties, but requires numerical optimization to implement the update even for conditionally-conjugate models. Khan et al. (2015b) considers a deterministic proximal-gradient variant of this method by splitting the lower bound into $-\underline{\mathscr{L}} := f + h$, where f contains all the "difficult" terms and h contains all the "easy" terms. By linearizing the "difficult" terms, this leads to a closed-form update even for non-conjugate models. The update is given by:

$$\begin{aligned} \lambda_{k+1} = \underset{\lambda \in \mathscr{S}}{\operatorname{argmin}} \Big[& \lambda^T [\nabla f(\lambda_k)] + h(\lambda) \\ & + \frac{1}{\beta_k} \mathbb{D}_{KL}[q(\mathbf{z}|\lambda) \,\|\, q(\mathbf{z}|\lambda_k)] \Big]. \end{aligned} \tag{5.6}$$

However, this method requires the exact gradients which is usually not feasible for large dataset and/or complex models.

Mirror Descent Methods: In the optimization literature, *mirror descent* (and stochastic mirror descent) algorithms are a generalization of (5.2) where the squared-Euclidean distance can be replaced by any Bregman divergence $\mathbb{D}_F(\lambda \| \lambda_k)$ gener-

ated from a strongly-convex function $F(\lambda)$ (Beck and Teboulle, 2003),

$$\lambda_{k+1} = \underset{\lambda \in \mathscr{S}}{\operatorname{argmin}} \left\{ -\lambda^T \nabla \underline{\mathscr{L}}(\lambda_k) + \frac{1}{\beta_k} \mathbb{D}_F(\lambda \| \lambda_k) \right\}. \tag{5.7}$$

The convergence rate of mirror descent algorithm has been analyzed in convex (Duchi et al., 2010) and more recently in non-convex (Ghadimi et al., 2014) settings. However, mirror descent does not cover the cases described above in (5.5) and (5.6) when a KL divergence between two exponential-family distributions is used with λ as the natural-parameter. For such cases, the Bregman divergence corresponds to a KL divergence with swapped parameters (see Nielsen and Garcia, 2009, Equation 29),

$$\begin{aligned} \mathbb{D}_A(\lambda \| \lambda_k) &:= A(\lambda) - A(\lambda_k) - [\bigtriangledown A(\lambda_k)]^T (\lambda - \lambda_k) \\ &= \mathbb{D}_{KL}[q(\mathbf{z}|\lambda_k) \| q(\mathbf{z}|\lambda)], \end{aligned} \tag{5.8}$$

where $A(\lambda)$ is the partition function of q. Because (5.5) and (5.6) both use a KL divergence where the second argument is fixed to λ_k, instead of the first argument, they are not covered under the mirror-descent framework. In addition, even though mirror-descent has been used for variational inference (Ravikumar et al., 2010), Bregman divergences do not yield an efficient update in many scenarios.

5.2 Proximal-gradient SVI

Our proximal-gradient stochastic variational inference (PG-SVI) method extends (5.6) to allow stochastic gradients $\widehat{\nabla} f(\lambda_k)$ and general divergence functions $\mathbb{D}(\lambda \| \lambda_k)$ by using the iteration

$$\lambda_{k+1} = \underset{\lambda \in \mathscr{S}}{\operatorname{argmin}} \left\{ \lambda^T \left[\widehat{\bigtriangledown} f(\lambda_k) \right] + h(\lambda) + \frac{1}{\beta_k} \mathbb{D}(\lambda \,\|\, \lambda_k) \right\}. \tag{5.9}$$

This unifies a variety of existing approaches since it allows:

1. Splitting of $\underline{\mathscr{L}}$ into a difficult term f and a simple term h, similar to the method of Khan et al. (2015b).
2. A stochastic approximation $\widehat{\nabla} f$ of the gradient of the difficult term, similar

to SG methods.

3. Divergence functions $\mathbb{D}$ that incorporate the geometry of the parameter space, similar to methods discussed in Section 5.1 (see (5.3), (5.5), (5.6), and (5.7)).

Below, we describe each feature in detail, along with the precise assumptions used in our analysis.

5.2.1 Splitting

Following Khan et al. (2015b), we split the lower bound into a sum of a "difficult" term f and an "easy" term h, enabling a closed-form solution for (5.9). Specifically, we split using $p(\mathbf{y},\mathbf{z})/q(\mathbf{z}|\boldsymbol{\lambda}) = c\,\tilde{p}_d(\mathbf{z}|\boldsymbol{\lambda})\tilde{p}_e(\mathbf{z}|\boldsymbol{\lambda})$, where $\tilde{p}_d$ contains all factors that make the optimization difficult, and $\tilde{p}_e$ contains the rest (while c is a constant). By substituting in (5.1), we get the following split of the lower bound:

$$\underline{\mathscr{L}}(\boldsymbol{\lambda}) = \underbrace{\mathbb{E}_q[\log \tilde{p}_d(\mathbf{z}|\boldsymbol{\lambda})]}_{-f(\boldsymbol{\lambda})} + \underbrace{\mathbb{E}_q[\log \tilde{p}_e(\mathbf{z}|\boldsymbol{\lambda})]}_{-h(\boldsymbol{\lambda})} + \log c.$$

Note that $\tilde{p}_d$ and $\tilde{p}_e$ need not be probability distributions.

We make the following assumptions about f and h:

(A1) The function f is differentiable and its gradient is $L-$Lipschitz-continuous, i.e. $\forall\boldsymbol{\lambda}$ and $\boldsymbol{\lambda}' \in \mathscr{S}$ we have

$$\|\nabla f(\boldsymbol{\lambda}) - \nabla f(\boldsymbol{\lambda}')\|_2 \leq L\|\boldsymbol{\lambda} - \boldsymbol{\lambda}'\|_2.$$

(A2) The function h can be a general convex function.

These assumptions are very weak. The function f can be non-convex and the Lipschitz-continuity assumption is typically satisfied in practice (and indeed the analysis can be generalized to only require this assumption on a smaller set containing the iterations). The assumption that h is convex seems strong, but note that we can always take $h = 0$ in the split if the function has no "nice" convex part. Below, we give several illustrative examples of such splits for variational-Gaussian inference with $q(\mathbf{z}|\boldsymbol{\lambda}) := \mathscr{N}(\mathbf{z}|\mathbf{m},\mathbf{V})$, so that $\boldsymbol{\lambda} = \{\mathbf{m},\mathbf{V}\}$ with $\mathbf{m}$ being the mean and $\mathbf{V}$ being the covariance matrix.

Gaussian Process (GP) Models: Consider GP models (Kuss and Rasmussen, 2005) for N input-output pairs $\{y_n, \mathbf{x}_n\}$ indexed by n. Let $z_n := f(\mathbf{x}_n)$ be the latent function drawn from a GP with mean 0 and covariance $\mathbf{K}$. We use a non-Gaussian likelihood $p(y_n|z_n)$ to model the output. We can then use the following split, where the non-Gaussian terms are in $\tilde{p}_d$ and the Gaussian terms are in $\tilde{p}_e$:

$$\frac{p(\mathbf{y},\mathbf{z})}{q(\mathbf{z}|\lambda)} = \underbrace{\prod_{n=1}^{N} p(y_n|z_n)}_{\tilde{p}_d(\mathbf{z}|\lambda)} \underbrace{\frac{\mathcal{N}(\mathbf{z}|0,\mathbf{K})}{\mathcal{N}(\mathbf{z}|\mathbf{m},\mathbf{V})}}_{\tilde{p}_e(\mathbf{z}|\lambda)}. \tag{5.10}$$

The detailed derivation is in Appendix D.1.1. By substituting in (5.1), we obtain the lower bound $\underline{\mathscr{L}}(\lambda)$ shown below along with its split:

$$\underbrace{\sum_n \mathbb{E}_q[\log p(y_n|z_n)]}_{-f(\lambda)} - \underbrace{\mathbb{D}_{KL}[\mathcal{N}(\mathbf{z}|\mathbf{m},\mathbf{V}) \,\|\, \mathcal{N}(\mathbf{z}|0,\mathbf{K})]}_{h(\lambda)}. \tag{5.11}$$

Assumption A1 is satisfied for common likelihoods, while it is easy to establish that h is convex. We show in Section 5.5 that this split leads to a closed-form update for iteration (5.9).

Generalized Linear Models (GLMs): A similar split can be obtained for GLMs (Nelder and Wedderburn, 1972), where the non-conjugate terms are in $\tilde{p}_d$ and the rest are in $\tilde{p}_e$. Denoting the weights by $\mathbf{z}$ and assuming a standard Gaussian prior over it, we can use the following split:

$$\frac{p(\mathbf{y},\mathbf{z})}{q(\mathbf{z}|\lambda)} = \underbrace{\prod_{n=1}^{N} p(y_n|\mathbf{x}_n^T\mathbf{z})}_{\tilde{p}_d(\mathbf{z}|\lambda)} \underbrace{\frac{\mathcal{N}(\mathbf{z}|0,\mathbf{I})}{\mathcal{N}(\mathbf{z}|\mathbf{m},\mathbf{V})}}_{\tilde{p}_e(\mathbf{z}|\lambda)}.$$

We give further details about the bound for this case in Appendix D.1.2.

Correlated Topic Model (CTM): Given a text document with a vocabulary of N words, denote its word-count vector by $\mathbf{y}$. Let K be the number of topics and $\mathbf{z}$

be the vector of topic-proportions. We can then use the following split:

$$\frac{p(\mathbf{y},\mathbf{z})}{q(\mathbf{z}|\lambda)} = \underbrace{\prod_{n=1}^{N}\left[\sum_{k=1}^{K}\beta_{n,k}\frac{e^{z_k}}{\sum_j e^{z_j}}\right]^{y_n}}_{\tilde{p}_d(\mathbf{z}|\lambda)}\underbrace{\frac{\mathcal{N}(\mathbf{z}|\mu,\Sigma)}{\mathcal{N}(\mathbf{z}|\mathbf{m},\mathbf{V})}}_{\tilde{p}_e(\mathbf{z}|\lambda)},$$

where μ,Σ are parameters of the Gaussian prior and $\beta_{n,k}$ are parameters of K multinomials. We give further details about the bound in Appendix D.1.3.

5.2.2 Stochastic Approximation

The approach of Khan et al. (2015b) considers (5.9) in the special case of (5.6) where we use the exact gradient $\nabla f(\lambda_k)$ in the first term. But in practice this gradient is often difficult to compute. In our framework, we allow a stochastic approximation of $\nabla f(\lambda)$ which we denote by $\widehat{\nabla} f(\lambda_k)$.

As shown in the previous section, f might take a finite-sum form $f(\lambda) := \Sigma_{n=1}^{N}\mathbb{E}_q[\tilde{f}_n(\mathbf{z})]$ for a set of functions $\tilde{f}_n$ as in the GP model (5.11). In some situations, $\mathbb{E}_q[\tilde{f}_n(\mathbf{z})]$ is computationally expensive or intractable. For example, in GP models the expectation is equal to $\mathbb{E}_q[\log p(y_n|z_n)]$, which is intractable for most non-Gaussian likelihoods. In such cases, we can form a stochastic approximation by using a few samples $\mathbf{z}^{(s)}$ from $q(\mathbf{z}|\lambda)$, as shown below:

$$\nabla\mathbb{E}_q[\tilde{f}_n(\mathbf{z})] \approx \widehat{\mathbf{g}}(\lambda,\xi_n) := \frac{1}{S}\sum_{s=1}^{S}\tilde{f}_n(\mathbf{z}^{(s)})\nabla[\log q(\mathbf{z}^{(s)}|\lambda)],$$

where ξ_n represents the noise in the stochastic approximation $\widehat{\mathbf{g}}$ and we use the identity $\nabla q(\mathbf{z}|\lambda) = q(\mathbf{z}|\lambda)\nabla[\log q(\mathbf{z}|\lambda)]$ to derive the expression (Ranganath et al., 2013). We can then form a stochastic-gradient by randomly selecting a mini-batch of M functions $\tilde{f}_{n_i}(\mathbf{z})$ and employing the estimate

$$\widehat{\nabla} f(\lambda) = \frac{N}{M}\sum_{i=1}^{M}\widehat{\mathbf{g}}(\lambda,\xi_{n_i}). \tag{5.12}$$

In our analysis we make the following two assumptions regarding the stochastic approximation of the gradient:

(A3) The estimate is unbiased: $\mathbb{E}[\widehat{\mathbf{g}}(\lambda, \xi_n)] = \bigtriangledown f(\lambda)$.

(A4) Its variance is upper bounded: $\text{Var}[\widehat{\mathbf{g}}(\lambda, \xi_n)] \leq \sigma^2$.

In both the assumptions, the expectation is taken with respect to the noise ξ_n. The first assumption is true for the stochastic approximations of (5.12). The second assumption is stronger, but only needs to hold for all λ_k so is almost always satisfied in practice.

5.2.3 Divergence Functions

To incorporate the geometry of q we incorporate a divergence function $\mathbb{D}$ between λ and λ_k. The set of divergence functions need to satisfy two assumptions:

(A5) $\mathbb{D}(\lambda \,\|\, \lambda') > 0$, for all $\lambda \neq \lambda'$.

(A6) There exist an $\alpha > 0$ such that for all λ, λ' generated by (5.9) we have:

$$(\lambda - \lambda')^T \nabla_\lambda \mathbb{D}(\lambda \,\|\, \lambda') \geq \alpha \|\lambda - \lambda'\|^2. \tag{5.13}$$

The first assumption is reasonable and is satisfied by typical divergence functions like the squared Euclidean distance and variants of the KL divergence. In the next section we show that, whenever the iteration (5.9) is defined and all λ_k stay within a compact set, the second assumption is satisfied for all divergence functions considered in Section 5.1.

5.3 Special Cases

Most methods discussed in Section 5.1 are special cases of the proposed iteration (5.9). We obtain gradient descent if $h = 0$, $f = -\underline{\mathscr{L}}$, $\widehat{\nabla} f = \nabla f$, and $\mathbb{D}(\lambda \| \lambda_k) = (1/2)\|\lambda - \lambda_k\|^2$ (in this case A6 is satisfied with $\alpha = 1$). From here, there are three standard generalizations in the optimization literature: SG methods do not require that $\widehat{\nabla} f = \nabla f$, proximal-gradient methods do not require that $h = 0$, and mirror descent allows $\mathbb{D}$ to be a different Bregman divergence generated by a strongly-convex function. Our analysis applies to all these variations on existing optimization algorithms because A1 to A5 are standard assumptions (Ghadimi et al., 2014)

and, as we now show, A6 is satisfied for this class of Bregman divergences. In particular, consider the generic Bregman divergence shown in the left side of (5.8) for some strongly-convex function $A(\lambda)$. By taking the gradient with respect to λ and substituting in (5.13), we obtain that A6 is equivalent to

$$(\lambda - \lambda_k)^T [\bigtriangledown A(\lambda) - \bigtriangledown A(\lambda_k)] \geq \alpha \|\lambda - \lambda_k\|^2,$$

which is equivalent to strong-convexity of the function $A(\lambda)$ (Nesterov, 2004, Theorem 2.1.9).

The method of Theis and Hoffman (2015) corresponds to choosing $h = -\underline{\mathscr{L}}$, $f = 0$, and $\mathbb{D}(\lambda||\lambda_k) := \mathbb{D}_{KL}[q(\mathbf{z}|\lambda) \,\|\, q(\mathbf{z}|\lambda_k)]$ where q is an exponential family distribution with natural parameters λ. Since we assume h to be convex, only limited cases of their approach are covered under our framework. The method of Khan et al. (2015b) also uses the KL divergence and focuses on the deterministic case where $\widehat{\nabla} f(\lambda) = \nabla f(\lambda)$, but uses the split $-\underline{\mathscr{L}} = f + h$ to allow for non-conjugate models. In both of these models, A6 is satisfied when the Fisher matrix $\bigtriangledown^2 A(\lambda)$ is positive-definite. This can be shown by using the definition of the KL divergence for exponential families (Nielsen and Garcia, 2009):

$$\begin{aligned} &\mathbb{D}_{KL}[q(\mathbf{z}|\lambda) \,\|\, q(\mathbf{z}|\lambda_k)] \\ &\quad := A(\lambda_k) - A(\lambda) - [\bigtriangledown A(\lambda)]^T (\lambda_k - \lambda). \end{aligned} \tag{5.14}$$

Taking the derivative with respect to λ and substituting in (5.13) with $\lambda' = \lambda_k$, we get the condition

$$(\lambda - \lambda_k)^T [\bigtriangledown^2 A(\lambda)](\lambda - \lambda_k) \geq \alpha \|\lambda - \lambda_k\|^2,$$

which is satisfied when $\bigtriangledown^2 A(\lambda)$ is positive-definite over a compact set for α equal to its lowest eigenvalue on the set.

Methods based on natural-gradient using iteration (5.3) (like SVI) correspond to using $h = 0$, $f = -\underline{\mathscr{L}}$, and the symmetric KL divergence. Assumption A1 to A5 are usually assumed for these methods and, as we show next, A6 is also satisfied. In particular, when q is an exponential family distribution the symmetric KL divergence can be written as the sum of the Bregman divergence shown in (5.8)

and the KL divergence shown in (5.14),

$$\begin{aligned}
&\mathbb{D}_{KL}^{sym}[q(\mathbf{z}|\lambda) \,\|\, q(\mathbf{z}|\lambda_k)] \\
&:= \mathbb{D}_{KL}[q(\mathbf{z}|\lambda_k) \,\|\, q(\mathbf{z}|\lambda)] + \mathbb{D}_{KL}[q(\mathbf{z}|\lambda) \,\|\, q(\mathbf{z}|\lambda_k)] \\
&= \mathbb{D}_A(\lambda \,\|\, \lambda_k) + \mathbb{D}_{KL}[q(\mathbf{z}|\lambda) \,\|\, q(\mathbf{z}|\lambda_k)],
\end{aligned}$$

where the first equality follows from the definition of the symmetric KL divergence and the second one follows from (5.8). Since the two divergences in the sum satisfy A6, the symmetric KL divergence also satisfies the assumption.

5.4 Convergence of PG-SVI

We first analyze the convergence rate of deterministic methods where the gradient is exact, $\widehat{\nabla} f(\lambda) = \nabla f(\lambda)$. This yields a simplified result that applies to a wide variety of existing variational methods. Subsequently, we consider the more general case where a stochastic approximation of the gradient is used.

5.4.1 Deterministic Methods

We first establish the convergence under a fixed step-size when using the exact gradient. We use $C_0 = \underline{\mathscr{L}}^* - \underline{\mathscr{L}}(\lambda_0)$ as the initial (constant) sub-optimality, and express our result in terms of the quantity

$$G_k := \frac{1}{\beta}(\lambda_k - \lambda_{k+1}),$$

where λ_{k+1} is computed using (5.9).

Theorem 5.1. *Let A1, A2, A5, and A6 be satisfied. If we run t iterations of* (5.9) *with a fixed step-size $\beta_k = \beta = \alpha/L$ for all k and an exact gradient $\nabla f(\lambda)$, then we have*

$$\min_{k \in \{0,1,\ldots,t-1\}} \|G_k\|_2^2 \leq \frac{2LC_0}{\alpha^2 t}. \tag{5.15}$$

We give a proof in Appendix D.2. Stating the result in terms of G_k may appear to be unconventional, but this quantity is a natural measure of first-order optimal-

ity. For example, consider the special case of gradient descent where $h = 0$ and $\mathbb{D}(\lambda, \lambda_k) = \frac{1}{2}\|\lambda - \lambda_k\|_2^2$. In this case, $\alpha = 1$ and $\beta_k = 1/L$, therefore we have $\|G_k\|_2 = \|\nabla f(\lambda_k)\|_2$ and Proposition 5.1 implies that $\min_k \|\nabla f(\lambda_k)\|_2^2$ has a convergence rate of $O(1/t)$. This means that the method converges at a sublinear rate to an approximate stationary point, which would be a global minimum in the special case where f is convex.

In more general settings, the quantity G_k provides a generalized notion of first-order optimality for problems that may be non-smooth or use a non-Euclidean geometry. Further, if the objective is bounded below (C_0 is finite), this result implies that the algorithm converges to such a stationary point and also gives a rate of convergence of $O(1/t)$.

If we use a divergence with $\alpha > 1$ then we can use a step-size larger than $1/L$ and the error will decrease faster than gradient-descent. To our knowledge, this is the first result that formally shows that natural-gradient methods can achieve faster convergence rates. The splitting of the objective into f and h functions is also likely to improve the step-size. Since L only depends on f, sometimes it might be possible to reduce the Lipschitz constant by choosing an appropriate split.

We next give a more general result that allows a per-iteration step size.

Theorem 5.2. *If we choose the step-sizes β_k to be such that $0 < \beta_k \leq 2\alpha/L$ with $\beta_k < 2\alpha/L$ for at least one k, then,*

$$\min_{k \in \{0,1\ldots t-1\}} \|G_k\|_2^2 \leq \frac{C_0}{\sum_{k=0}^{t-1} \left(\alpha\beta_k - L\beta_k^2/2\right)}. \tag{5.16}$$

We give a proof in Appendix D.2. For gradient-descent, the above result implies that we can use any step-size less than $2/L$, which agrees with the classical step-size choices for gradient and proximal-gradient methods.

5.4.2 Stochastic Methods

We now give a bound for the more general case where we use a stochastic approximation of the gradient.

Theorem 5.3. *Let A1-A6 be satisfied. If we run t iterations of* (5.9) *for a fixed step-size $\beta_k = \alpha_*/L$ (where $0 < \gamma < 2$ is a scalar) and fixed batch-size $M_k = M$ for*

all k with a stochastic gradient $\widehat{\nabla} f(\lambda)$, then we have

$$\mathbb{E}_{R,\xi}(\|G_R\|_2^2) \leq \left[\frac{2LC_0}{\alpha_*^2 t} + \frac{c\sigma^2}{M\alpha^*}\right],$$

where c is a constant such that $c > 1/(2\alpha)$ and $\alpha_* := \alpha - 1/(2c)$. The expectation is taken with respect to the noise $\xi := \{\xi_0, \xi_1, \ldots, \xi_{t-1}\}$, and a random variable R which follows the uniform distribution $Prob(R = k) = 1/t, \forall k \in \{0, 1, 2, \ldots, t-1\}$.

Unlike the bound of Theorem 5.1, this bound depends on the noise variance σ^2 as well the mini-batch size M. In particular, as we would expect, the bound gets tighter as the variance gets smaller and as the size of our mini-batch grows. Notice that the dependence on the variance σ^2 is also improved if we have a favourable geometry that increases α^*. Thus, we can achieve a higher accuracy by either increasing the mini-batch size or improving the geometry.

In Appendix D.3 we give a more general result that allows non-constant sequences of step sizes, although we found that constant step-sizes work better empirically. Note that while stating the result in terms of a randomized iteration might seem strange, in practice we typically just take the last iteration as the approximate minimizer.

5.5 Closed-Form Updates for Non-conjugate Models

We now give an example where iteration (5.9) attains a closed-form solution. We expect such closed-form solution to exist for a large class of problems, including models where q is an exponential-family distribution, but here we focus on the GP model discussed in Section 5.2.1. For the GP model, we rewrite the lower bound (5.11) as

$$-\mathscr{L}(\mathbf{m}, \mathbf{V}) := \underbrace{\sum_{n=1}^{N} f_n(m_n, v_n)}_{f(m,V)} + \underbrace{\mathbb{D}_{KL}[q \,\|\, p]}_{h(m,V)}, \tag{5.17}$$

where we've used $q := \mathcal{N}(\mathbf{z}|\mathbf{m}, \mathbf{V})$, $p := \mathcal{N}(\mathbf{z}|0, \mathbf{K})$, and $f_n(m_n, v_n) := -\mathbb{E}_q[\log p(y_n|z_n)]$ with m_n being the entry n of $\mathbf{m}$ and v_n being the diagonal entry n of $\mathbf{V}$. We can compute a stochastic approximation of f using (5.12) by randomly selecting an

example n_k (choosing $M = 1$) and using a Monte Carlo gradient approximation of f_{n_k}. Using this approximation, the linearized term in (5.9) can be simplified to the following:

$$\begin{aligned}\boldsymbol{\lambda}^T \left[\widehat{\bigtriangledown} f(\boldsymbol{\lambda}_k)\right] &= m_n \underbrace{N[\nabla_{m_n} f_{n_k}(m_{n_k,k}, v_{n_k,k})]}_{:=\alpha_{n_k,k}} \\ &\quad + v_n \underbrace{N[\nabla_{v_n} f_{n_k}(m_{n_k,k}, v_{n_k,k})]}_{:=2\,\gamma_{n_k,k}} \\ &= m_n \alpha_{n_k,k} + \tfrac{1}{2} v_n \gamma_{n_k,k}, \end{aligned} \tag{5.18}$$

where $m_{n_k,k}$ and $v_{n_k,k}$ denote the value of m_n and v_n in the k'th iteration for $n = n_k$. By using the KL divergence as our divergence function in iteration (5.9), and by denoting $\mathcal{N}(\mathbf{z}|\mathbf{m}_k, \mathbf{V}_k)$ by q_k, we can express the two last two terms in (5.9) as a single KL divergence function as shown below:

$$\begin{aligned}&\boldsymbol{\lambda}^T \left[\widehat{\bigtriangledown} f(\boldsymbol{\lambda}_k)\right] + h(\boldsymbol{\lambda}) + \frac{1}{\beta_k} \mathbb{D}(\boldsymbol{\lambda} \| \boldsymbol{\lambda}_k) \\ &= (m_n \alpha_{n,k} + \tfrac{1}{2} v_n \gamma_{n,k}) + \mathbb{D}_{KL}[q \,\|\, p] + \frac{1}{\beta_k} \mathbb{D}_{KL}[q \,\|\, q_k] \\ &= (m_n \alpha_{n,k} + \tfrac{1}{2} v_n \gamma_{n,k}) + \frac{1}{1 - r_k} \mathbb{D}_{KL}[q \,\|\, p^{1-r_k} q_k^{r_k}], \end{aligned}$$

where $r_k := 1/(1+\beta_k)$. Comparing this to (5.17), we see that this objective is similar to that of a GP model with a Gaussian prior[1] $p^{1-r_k} q_k^{r_k}$ and a linear Gaussian-like log-likelihood. Therefore, we can obtain closed-form updates for its minimization.

The updates are shown below and a detailed derivation is given in Appendix D.4.

$$\begin{aligned}&\widetilde{\gamma}_k = r_k \widetilde{\gamma}_{k-1} + (1 - r_k) \gamma_{n_k,k} 1_{n_k}, \\ &\mathbf{m}_{k+1} = \mathbf{m}_k - (1 - r_k)(\mathbf{I} - \mathbf{K}\mathbf{A}_k^{-1})(\mathbf{m}_k + \alpha_{n_k,k} \boldsymbol{\kappa}_{n_k}), \\ &v_{n_{k+1},k+1} = \kappa_{n_{k+1},n_{k+1}} - \boldsymbol{\kappa}_{n_{k+1}}^T \mathbf{A}_k^{-1} \boldsymbol{\kappa}_{n_{k+1}}, \end{aligned} \tag{5.19}$$

where $\widetilde{\gamma}_0$ is initialized to a small positive constant to avoid numerical issues, 1_{n_k} is a vector with all zero entries except n_k'th entry which is equal to 1, $\boldsymbol{\kappa}_k$ is n_k'th

[1] Since p and q are Gaussian, the product is a Gaussian.

column of $\mathbf{K}$, and $\mathbf{A}_k := \mathbf{K} + [\text{diag}(\widetilde{\gamma}_k)]^{-1}$. For iteration $k+1$, we use $m_{n_{k+1},k+1}$ and $v_{n_{k+1},k+1}$ to compute the gradients $\alpha_{n_{k+1},k+1}$ and $\gamma_{n_{k+1},k+1}$, and run the above updates again. We continue until a convergence criteria is reached.

There are numerous advantages of these updates. First, We do not need to store the full covariance matrix $\mathbf{V}$. The updates avoid forming the matrix and only update $\mathbf{m}$. This works because we only need one diagonal element in each iteration to compute the stochastic gradient $\gamma_{n_k,k}$. For large N this is a clear advantage since the memory cost is $O(N)$ rather than $O(N^2)$. Second, computation of the mean vector $\mathbf{m}$ and a diagonal entry of $\mathbf{V}$ only require solving two linear equations, as shown in the second and third line of (5.19). In general, for a mini-batch of size M, we need a total of $2M$ linear equations, which is a lot cheaper than an explicit inversion. Finally, the linear equations at iteration $k+1$ are very similar to those at iteration k, since $\mathbf{A}_k$ differs only at one entry from $\mathbf{A}_{k+1}$. Therefore, we can reuse computations from the previous iteration to improve the computational efficiency of the updates.

5.6 Discussion

This chapter has made two contributions. First, we proposed a new variational inference method that combines variable splitting, stochastic gradients, and general divergence functions. This method is well-suited for a huge variety of the variational inference problems that arise in practice, and we anticipate that it may improve over state of the art methods in a variety of settings. Our second contribution is a theoretical analysis of the convergence rate of this general method. Our analysis generalizes existing results for the mirror descent algorithm in optimization, and establishes convergences rates of a variety of existing variational inference methods. Due to its generality we expect that this analysis could be useful to establish convergence rates of other algorithms that we have not thought of, perhaps beyond the variational inference settings we consider in this work. However, an open problem that is also discussed by Ghadimi et al. (2014) it to establish convergence to an arbitrary accuracy with a fixed batch size.

One issue that we have not satisfactorily resolved is giving a theoretically-justified way to set the step-size in practice; our analysis only indicates that it

must be sufficiently small. However, this problem is common in many methods in the literature and our analysis at least suggests the factors that should be taken into account. Another open issue is the applicability our method to many other latent variable models; in this chapter we have shown applications to variational-Gaussian inference, but we expect that our method should result in simple updates for a larger class of latent variable models such as non-conjugate exponential family distribution models. Additional work on these issues will improve usability of our method.

Chapter 6

Conclusion and Future Work

This thesis presents our works on practical optimization methods for structured machine learning models. The primary purpose of these methods is making the training stage of ML models more efficient in terms of computational cost.

6.1 Practical SVRG

The main advantage of SVRG among the variance reduction based methods is its independence to the extra memory. However, this freedom is acquired via sacrificing in computational cost. The SVRG algorithm requires to evaluate full-gradient occasionally to reduce the variance of its stochastic gradient estimator. In this work, we propose and analyse some variants which can reduce the number of gradient evaluation of SVRG while preserving its convergence speed. Further, we also present variants that utilize the nested structure of the SVRG algorithm in order to improve the convergence speed of the algorithm. For the SVM loss, we use the gradient evaluation for each data sample in the outer and inner loop of its algorithm to discern the non-support vectors as soon as possible. By doing so we can ignore those non-support vector points for gradient evaluation in the next iterations of the algorithm and improve the convergence speed. Applying SVRG and its variants in the training of a deep neural network model is an interesting direction for future research. However, using the SVRG algorithm to optimize the whole parameter set of a deep neural network may not outperform SGD and its variants due to overpa-

rameterization (Vaswani et al., 2018). But combining SVRG to train the last layer of a deep neural network with SGD to optimize the rest of parameters may be an appealing direction for future research.

6.2 SAG for CRF

This work presents the first work that applies SAG to CRFs. SAG requires memory in its algorithm to store the stale gradients. This requirement hinders its applicability for natural language models where there are a vast number of labels and complex features. In addition to our approach of storing marginals grouping training examples into fixed mini-batches can further help to reduce the memory requirement (since only the average marginals with respect to the mini-batches would be needed).

Utilizing non-uniform sampling in stochastic gradient estimation has been shown to improve the convergence speed of stochastic optimization methods. We propose two variant of SAG with non-uniform sampling and analyse the convergence of them. We examined those non-uniform sampling methods in the training of CRF models and observed their superiority over uniform sampling approaches. Furthermore, we considered smooth objective functions with $\ell 2$-regularizer, but our analysis could be extended to non-smooth objective with $\ell 1$-regularizer. We also just considered chain-structured data in our experiments, but the algorithm can be applied for general graph structure. Another alternative can be adapting the SAG algorithm to run in multi-threaded computation.

6.3 MASAGA: SAGA for Manifolds

The loss function of a significant group of problems in ML such PCA, dictionary learning, Gaussian mixture models, and the Page-rank algorithm are defined on a Riemannian manifold. Considering that structure in the optimization method can improve the convergence speed of it. In this work, we modify the SAGA algorithm for manifold optimization. This modification extends the linear convergence of the SAGA algorithm for a strongly-convex function in Euclidean spaces to Riemannian manifolds for a function submitting to a notion of strong-convexity over the manifold. We applied non-uniform sampling for SAGA for manifolds algorithm and

observed analytically and experimentally its speed-up. For future research, one can extend MASAGA by deriving convergence rates for the non-convex case of geodesic objective functions. One also can explore accelerated variance-reduction methods and block coordinate descent based methods for Riemannian optimization.

6.4 Proximal Gradient SVI

In this work, we proposed a new variational inference method that combines variable splitting, stochastic gradients, and general divergence functions. The proposed method could be applied for a considerable group of variational inference problems and may improve over the state of the art results. Furthermore, we theoretically analysed the convergence of this general method. This analysis generalizes existing results for the mirror descent algorithm in optimization, and give convergence rates for a variety of existing variational inference methods. However, an open problem that is also discussed by Ghadimi et al. (2014) it to establish convergence to an arbitrary accuracy with a fixed batch size.

One problem which is not covered in our analysis is giving a theoretically-justified way to set the step-size in practice; our analysis only indicates that it must be sufficiently small. However, this problem is common in many methods in the literature, and our analysis at least suggests the factors that should be taken into account.

Bibliography

Absil, P.-A., Mahony, R., and Sepulchre, R. (2009). *Optimization algorithms on matrix manifolds*. Princeton University Press. → page 56

Agarwal, A. and Bottou, L. (2014). A lower bound for the optimization of finite sums. *arXiv preprint*. → page 61

Allen-Zhu, Z. and Yuan, Y. (2016). Improved svrg for non-strongly-convex or sum-of-non-convex objectives. In *International conference on machine learning*, pages 1080–1089. → page 12

Amari, S.-I. (1998). Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276. → page 72

Aravkin, A., Friedlander, M. P., Herrmann, F. J., and Van Leeuwen, T. (2012). Robust inversion, dimensionality reduction, and randomized sampling. *Mathematical Programming*, 134(1):101–125. → page 21

Babanezhad, R., Laradji, I. H., Shafaei, A., and Schmidt, M. (2018). Masaga: A linearly-convergent stochastic first-order method for optimization on manifolds. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 344–359. Springer. → page v

Bach, F. and Moulines, E. (2011). Non-asymptotic analysis of stochastic approximation algorithms for machine learning. *Advances in Neural Information Processing Systems*. → pages 11, 50

Beck, A. and Teboulle, M. (2003). Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167–175. → pages 13, 14, 74

Beck, A. and Teboulle, M. (2009). A fast iterative shrinkage-thresholding algorithm with application to wavelet-based image deblurring. In *Acoustics,*

Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on, pages 693–696. IEEE. → pages 41, 70

Bietti, A. and Mairal, J. (2017). Stochastic optimization with variance reduction for infinite datasets with finite sum structure. In *Advances in Neural Information Processing Systems*, pages 1622–1632. → page 55

Bonnabel, S. (2013). Stochastic gradient descent on riemannian manifolds. *IEEE Transactions on Automatic Control*, 58(9):2217–2229. → pages 54, 66

Bottou, L. and Bousquet, O. (2007). The tradeoffs of large scale learning. *Advances in Neural Information Processing Systems*. → pages 27, 116

Byrd, R. H., Chin, G. M., Nocedal, J., and Wu, Y. (2012). Sample size selection in optimization methods for machine learning. *Mathematical programming*, 134(1):127–155. → page 29

Carbonetto, P. (2009). *New probabilistic inference algorithms that harness the strengths of variational and Monte Carlo methods*. PhD thesis, Univ. of British Columbia. → page 28

Caruana, R., Joachims, T., and Backstrom, L. (2004). KDD-cup 2004: results and analysis. *ACM SIGKDD Newsletter*, 6(2):95–108. → page 28

Cauchy, M. A. (1847). Méthode générale pour la résolution des systèmes d'équations simultanées. *Comptes rendus des séances de l'Académie des sciences de Paris*, 25:536–538. → page 53

Cohn, T. and Blunsom, P. (2005). Semantic role labelling with tree conditional random fields. *Conference on Computational Natural Language Learning*. → page 35

Collins, M. (2002). Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. *Conference on Empirical Methods in Natural Language Processing*. → page 47

Collins, M., Globerson, A., Koo, T., Carreras, X., and Bartlett, P. (2008). Exponentiated gradient algorithms for conditional random fields and max-margin Markov networks. *The Journal of Machine Learning Research*, 9:1775–1822. → pages 38, 47

Cormack, G. V. and Lynam, T. R. (2005). Spam corpus creation for TREC. In *Proc. 2nd Conference on Email and Anti-Spam*. http://plg.uwaterloo.ca/~gvcormac/treccorpus/. → page 28

Defazio, A., Bach, F., and Lacoste-Julien, S. (2014). SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. *Advances in Neural Information Processing Systems*. → pages 12, 16, 36, 45, 52, 54, 59, 60, 122

Defazio, A. and Bottou, L. (2019). On the ineffectiveness of variance reduced optimization for deep learning. In *Advances in Neural Information Processing Systems*, pages 1753–1763. → page 13

Dubey, K. A., Reddi, S. J., Williamson, S. A., Poczos, B., Smola, A. J., and Xing, E. P. (2016). Variance reduction in stochastic gradient langevin dynamics. In *Advances in neural information processing systems*, pages 1154–1162. → page 55

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159. → page 47

Duchi, J. C., Shalev-Shwartz, S., Singer, Y., and Tewari, A. (2010). Composite objective mirror descent. In *COLT*, pages 14–26. → page 74

Finkel, J. R., Kleeman, A., and Manning, C. D. (2008). Efficient, feature-based, conditional random field parsing. *Annual Meeting of the Association for Comptuational Linguistics: Human Language Technologies*. → page 38

Frank, A. and Asuncion, A. (2010). UCI machine learning repository. → page 28

Friedlander, M. P. and Schmidt, M. (2012). Hybrid deterministic-stochastic methods for data fitting. *SIAM Journal of Scientific Computing*, 34(3):A1351–A1379. → pages 21, 29, 38, 47

Ghadimi, S. and Lan, G. (2012). Optimal stochastic approximation algorithms for strongly convex stochastic composite optimization i: A generic algorithmic framework. *SIAM Journal on Optimization*, 22(4):1469–1492. → page 38

Ghadimi, S., Lan, G., and Zhang, H. (2014). Mini-batch stochastic approximation methods for nonconvex stochastic composite optimization. *Mathematical Programming*, pages 1–39. → pages 14, 70, 74, 78, 84, 88, 150

Guyon, C., Bouwmans, T., and Zahzah, E.-h. (2012). Robust principal component analysis for background subtraction: Systematic evaluation and comparative analysis. In *Principal component analysis*. InTech. → page 65

Guyon, I. (2008). Sido: A phamacology dataset. → page 28

Harikandeh, R., Ahmed, M. O., Virani, A., Schmidt, M., Konečnỳ, J., and Sallinen, S. (2015). Stopwasting my gradients: Practical svrg. In *Advances in Neural Information Processing Systems*, pages 2251–2259. → pages v, 55, 62

Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Kingsbury, B., et al. (2012). Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29. → page 1

Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. (2013). Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347. → pages 69, 71, 72, 73

Honkela, A., Raiko, T., Kuusela, M., Tornio, M., and Karhunen, J. (2011). Approximate Riemannian conjugate gradient learning for fixed-form variational Bayes. *JMLR*, 11:3235–3268. → page 73

Hosseini, R. and Sra, S. (2015). Matrix manifold optimization for gaussian mixtures. In *Advances in Neural Information Processing Systems*, pages 910–918. → page 53

Hu, C., Kwok, J., and Pan, W. (2009). Accelerated gradient methosd for stochastic optimization and online learning. *Advances in Neural Information Processing Systems*. → page 19

Jeuris, B., Vandebril, R., and Vandereycken, B. (2012). A survey and comparison of contemporary algorithms for computing the matrix geometric mean. *Electronic Transactions on Numerical Analysis*, 39(EPFL-ARTICLE-197637):379–402. → page 53

Joachims, T. (1999). Making large-scale SVM learning practical. In Schölkopf, B., Burges, C., and Smola, A., editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 11, pages 169–184. MIT Press, Cambridge, MA. → page 26

Johnson, R. and Zhang, T. (2013). Accelerating stochastic gradient descent using predictive variance reduction. *Advances in Neural Information Processing Systems*. → pages 12, 13, 17, 18, 52, 54, 101

Kamvar, S., Haveliwala, T., and Golub, G. (2004). Adaptive methods for the computation of pagerank. *Linear Algebra and its Applications*, 386:51–65. → page 65

Karimi, H., Nutini, J., and Schmidt, M. (2016). Linear convergence of gradient and proximal-gradient methods under the polyak-łojasiewicz condition. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 795–811. Springer. → page 12

Kasai, H., Sato, H., and Mishra, B. (2016). Riemannian stochastic variance reduced gradient on grassmann manifold. *arXiv preprint arXiv:1605.07367*. → pages 54, 55, 60

Keerthi, S. and DeCoste, D. (2005). A modified finite newton method for fast solution of large scale linear svms. *Journal of Machine Learning Research*, 6:341–361. → page 28

Khan, M. E., Babanezhad, R., Lin, W., Schmidt, M., and Sugiyama, M. (2015a). Faster stochastic variational inference using proximal-gradient methods with general divergence functions. *arXiv preprint arXiv:1511.00146*. → page v

Khan, M. E., Baque, P., Flueret, F., and Fua, P. (2015b). Kullback-Leibler Proximal Variational Inference. In *Advances in Neural Information Processing Systems*. → pages 70, 73, 74, 75, 77, 79

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*. → page 3

Kober, J., Bagnell, J. A., and Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274. → page 1

Konečnỳ, J., Liu, J., Richtárik, P., and Takáč, M. (2014). ms2gd: Mini-batch semi-stochastic gradient descent in the proximal setting. *arXiv preprint*. → page 24

Konečnỳ, J. and Richtárik, P. (2013). Semi-stochastic gradient descent methods. *arXiv preprint*. → pages 17, 54

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105. → page 1

Kucukelbir, A., Ranganath, R., Gelman, A., and Blei, D. (2014). Fully automatic variational inference of differentiable probability models. In *NIPS Workshop on Probabilistic Programming*. → page 69

Kuss, M. and Rasmussen, C. (2005). Assessing approximate inference for binary Gaussian process classification. *Journal of Machine Learning Research*, 6:1679–1704. → page 76

Lacoste-Julien, S., Jaggi, M., Schmidt, M., and Pletscher, P. (2013). Block-coordinate frank-wolfe optimization for structural svms. *International Conference on Machine Learning*. → page 38

Lafferty, J., McCallum, A., and Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *International Conference on Machine Learning*. → pages 35, 37

Lavergne, T., Cappé, O., and Yvon, F. (2010). Practical very large scale CRFs. In *Proceedings the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 504–513. → pages 38, 52

Le Roux, N., Schmidt, M., and Bach, F. (2012). A stochastic gradient method with an exponential convergence rate for strongly-convex optimization with finite training sets. *Advances in Neural Information Processing Systems*. → pages 11, 16, 23, 28, 36, 39, 40, 41, 54, 55

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. → pages 55, 65

Lewis, D., Yang, Y., Rose, T., and Li, F. (2004). RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397. → page 28

Lohr, S. (2009). *Sampling: design and analysis*. Cengage Learning. → page 21

Mahadevan, V. and Vasconcelos, N. (2010). Spatiotemporal saliency in dynamic scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1):171–177. → pages 55, 65

Mahdavi, M. and Jin, R. (2013). Mixedgrad: An $o(1/t)$ convergence rate algorithm for stochastic smooth optimization. *Advances in Neural Information Processing Systems*. → pages 17, 54

Mairal, J. (2013a). Optimization with first-order surrogate functions. *International Conference on Machine Learning*. → page 16

Mairal, J. (2013b). Optimization with first-order surrogate functions. *arXiv preprint arXiv:1305.3120*. → page 54

Mairal, J. (2014). Incremental majorization-minimization optimization with application to large-scale machine learning. *arXiv preprint arXiv:1402.4419*. → page 12

McCallum, A., Rohanimanesh, K., and Sutton, C. (2003). Dynamic conditional random fields for jointly labeling multiple sequences. In *NIPS Workshop on Syntax, Semantics, Statistics*. → page 35

Mnih, A. and Gregor, K. (2014). Neural variational inference and learning in belief networks. *arXiv preprint arXiv:1402.0030*. → page 69

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press. → page 3

Nedic, A. and Bertsekas, D. (2000). Convergence rate of incremental subgradient algorithms. In *Stochastic Optimization: Algorithms and Applications*, pages 263–304. Kluwer Academic. → page 50

Needell, D., Srebro, N., and Ward, R. (2014a). Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm. *Advances in Neural Information Processing Systems*. → pages 43, 44, 48

Needell, D., Ward, R., and Srebro, N. (2014b). Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. In *Advances in Neural Information Processing Systems*, pages 1017–1025. → page 62

Nelder, J. A. and Wedderburn, R. W. (1972). Generalized linear models. *Journal of the Royal Statistical Society: Series A (General)*, 135(3):370–384. → page 76

Nemirovski, A., Juditsky, A., Lan, G., and Shapiro, A. (2009). Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19(4):1574–1609. → pages 36, 53

Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence $O(1/k^2)$. *Doklady AN SSSR*, 269(3):543–547. → page 53

Nesterov, Y. (2004). *Introductory lectures on convex optimization: A basic course*. Springer. → pages 10, 36, 38, 79, 101, 104, 128

Nesterov, Y. (2012). Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM J. Optim.*, 22(2):341–362. → page 43

Newman, M. E. (2006). Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23):8577–8582. → page 65

Nguyen, L., Liu, J., Scheinberg, K., and Takáč, M. (2017). Sarah: A novel method for machine learning problems using stochastic recursive gradient. *arXiv preprint arXiv:1703.00102*. → page 54

Nielsen, F. and Garcia, V. (2009). Statistical exponential families: A digest with flash cards. *arXiv preprint arXiv:0911.4863*. → pages 74, 79

Nowozin, S. and Lampert, C. H. (2011). Structured learning and prediction in computer vision. *Foundation and Trends in Computer Vision*, 6. → page 35

Nutini, J., Laradji, I., and Schmidt, M. (2017). Let's Make Block Coordinate Descent Go Fast: Faster Greedy Rules, Message-Passing, Active-Set Complexity, and Superlinear Convergence. *ArXiv e-prints*. → page 68

Owen, A. B. (2014). Monte carlo theory, methods and examples (book draft). → page 13

Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab. → page 65

Paquet, U. (2014). On the convergence of stochastic variational inference in bayesian networks. *NIPS Workshop on variational inference*. → page 73

Pascanu, R. and Bengio, Y. (2013). Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*. → page 72

Peng, F. and McCallum, A. (2006). Information extraction from research papers using conditional random fields. *Information Processing & Management*, 42(4):963–979. → page 35

Petersen, P., Axler, S., and Ribet, K. (2006). *Riemannian geometry*, volume 171. Springer. → page 56

Polyak, B. T. and Juditsky, A. B. (1992). Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855. → pages 50, 53

Ranganath, R., Gerrish, S., and Blei, D. M. (2013). Black box variational inference. *arXiv preprint arXiv:1401.0118*. → pages 69, 71, 77

Ratnaparkhi, A. (1996). A maximum entropy model for part-of-speech tagging. *Conference on Empirical Methods in Natural Language Processing*. → page 47

Ravikumar, P., Agarwal, A., and Wainwright, M. J. (2010). Message-passing for graph-structured linear programs: Proximal methods and rounding schemes. *The Journal of Machine Learning Research*, 11:1043–1080. → page 74

Reddi, S. J., Hefny, A., Sra, S., Poczos, B., and Smola, A. (2016). Stochastic variance reduction for nonconvex optimization. In *International conference on machine learning*, pages 314–323. → page 12

Reddi, S. J., Zaheer, M., Sra, S., Poczos, B., Bach, F., Salakhutdinov, R., and Smola, A. J. (2017). A generic approach for escaping saddle points. *arXiv preprint arXiv:1709.01434*. → page 13

Rennie, J. D. and Srebro, N. (2005). Loss functions for preference levels: Regression with discrete ordered labels. In *Proceedings of the IJCAI multidisciplinary workshop on advances in preference handling*, pages 180–186. Kluwer Norwell, MA. → pages xii, 6

Rezende, D. J. and Mohamed, S. (2015). Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*. → page 3

Robbins, H. and Monro, S. (1951a). A stochastic approximation method. *Annals of Mathematical Statistics*, 22(3):400–407. → page 8

Robbins, H. and Monro, S. (1951b). A stochastic approximation method. *Ann. Math. Statist.*, 22(3):400–407. → page 53

Rosset, S. and Zhu, J. (2007). Piecewise linear regularized solution paths. *The Annals of Statistics*, 35(3):1012–1030. → page 25

Salimans, T., Knowles, D. A., et al. (2013). Fixed-form variational posterior approximation through stochastic linear regression. *Bayesian Analysis*, 8(4):837–882. → page 69

Sato, H., Kasai, H., and Mishra, B. (2017). Riemannian stochastic variance reduced gradient. *arXiv preprint arXiv:1702.05594*. → pages 54, 55, 60

Schmidt, M. (2005). minFunc: unconstrained multivariate differentiable optimization in Matlab. → page 47

Schmidt, M., Babanezhad, R., Ahmed, M., Clifton, A., and Sarkar, A. (2015a). Non-uniform stochastic average gradient method for training conditional

random fields. *International Conference on Artificial Intelligence and Statistics*. → page v

Schmidt, M., Babanezhad, R., Ahmed, M., Defazio, A., Clifton, A., and Sarkar, A. (2015b). Non-uniform stochastic average gradient method for training conditional random fields. In *Artificial Intelligence and Statistics*, pages 819–828. → pages 62, 67

Schmidt, M., Le Roux, N., and Bach, F. (2011). Convergence rates of inexact proximal-gradient methods for convex optimization. *Advances in Neural Information Processing Systems*. → page 19

Schmidt, M., Le Roux, N., and Bach, F. (2013). Minimizing finite sums with the stochastic average gradient. *arXiv preprint*. → pages 38, 44, 45, 48

Settles, B. (2004). Biomedical named entity recognition using conditional random fields and rich feature sets. In *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications*. → page 35

Sha, F. and Pereira, F. (2003). Shallow parsing with conditional random fields. *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technology*. → pages 35, 37, 46

Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge university press. → pages 2, 4

Shalev-Shwartz, S., Singer, Y., Srebro, N., and Cotter, A. (2011). Pegasos: primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30. → page 47

Shalev-Shwartz, S. and Zhang, T. (2013a). Accelerated mini-batch stochastic dual coordinate ascent. *Advances in Neural Information Processing Systems*. → page 12

Shalev-Shwartz, S. and Zhang, T. (2013b). Stochastic dual coordinate ascent methods for regularized loss minimization. *Journal of Machine Learning Research*, 14:567–599. → pages 16, 54, 55

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *Nature*, 550(7676):354. → page 1

Sra, S. and Hosseini, R. (2016). Geometric optimization in machine learning. In *Algorithmic Advances in Riemannian Geometry and Applications*, pages 73–91. Springer. → page 53

Strohmer, T. and Vershynin, R. (2009). A randomized Kaczmarz algorithm with exponential convergence. *Journal of Fourier Analysis and Applications*, 15(2):262–278. → page 43

Sun, J., Qu, Q., and Wright, J. (2015). Complete dictionary recovery over the sphere. In *Sampling Theory and Applications (SampTA), 2015 International Conference on*, pages 407–410. IEEE. → page 53

Taskar, B., Guestrin, C., and Koller, D. (2003). Max-margin Markov networks. *Advances in Neural Information Processing Systems*. → page 46

Theis, L. and Hoffman, M. D. (2015). A trust-region method for stochastic variational inference with applications to streaming data. *arXiv preprint arXiv:1505.07649*. → pages 73, 79

Titsias, M. and Lázaro-Gredilla, M. (2014). Doubly stochastic variational bayes for non-conjugate inference. In *ICML*. → pages 69, 71

Tsuruoka, Y., Tsujii, J., and Ananiadou, S. (2009). Stochastic gradient descent training for L1-regularized log-linear models with cumulative penalty. *Annual Meeting of the Association for Computational Linguisitics*, pages 477–485. → page 52

Udriste, C. (1994). *Convex functions and optimization methods on Riemannian manifolds*, volume 297. Springer Science & Business Media. → page 56

Usunier, N., Bordes, A., and Bottou, L. (2010). Guarantees for approximate incremental svms. *International Conference on Artificial Intelligence and Statistics*. → page 26

van den Doel, K. and Ascher, U. (2012). Adaptive and stochastic algorithms for eit and dc resistivity problems with piecewise constant solutions and many measurements. *SIAM J. Scient. Comput*, 34. → page 29

Vaswani, S., Bach, F., and Schmidt, M. (2018). Fast and faster convergence of sgd for over-parameterized models and an accelerated perceptron. *arXiv preprint arXiv:1810.07288*. → page 87

Vishwanathan, S., Schraudolph, N. N., Schmidt, M. W., and Murphy, K. P. (2006). Accelerated training of conditional random fields with stochastic gradient

methods. *Iinternational conference on Machine learning*. → pages 35, 38, 47, 50

Wallach, H. (2002). Efficient training of conditional random fields. Master's thesis, University of Edinburgh. → page 37

Wiesel, A. (2012). Geodesic convexity and covariance estimation. *IEEE Transactions on Signal Processing*, 60(12):6182–6189. → page 53

Xiao, L. and Zhang, T. (2014). A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(2):2057–2075. → pages 19, 23, 45, 108, 109, 110

Xu, W. (2010). Towards optimal one pass large scale learning with averaged stochastic gradient descent. *arXiv preprint*. → page 50

Zhang, H., Reddi, S. J., and Sra, S. (2016). Riemannian svrg: fast stochastic optimization on riemannian manifolds. In *Advances in Neural Information Processing Systems*, pages 4592–4600. → pages 54, 55, 60, 62, 64, 66

Zhang, H. and Sra, S. (2016). First-order methods for geodesically convex optimization. In *Conference on Learning Theory*, pages 1617–1638. → pages 54, 55, 57, 58, 60

Zhang, L., Mahdavi, M., and Jin, R. (2013). Linear convergence with condition number independent access of full gradients. *Advances in Neural Information Processing Systems*. → pages 17, 52

Zhang, T. and Yang, Y. (2017). Robust principal component analysis by manifold optimization. *arXiv preprint arXiv:1708.00257*. → page 53

Zhao, P. and Zhang, T. (2015). Stochastic optimization with importance sampling for regularized loss minimization. In *international conference on machine learning*, pages 1–9. → page 54

Zhou, J., Qiu, X., and Huang, X. (2011). A fast accurate two-stage training algorithm for L1-regularized CRFs with heuristic line search strategy. *International Joint Conference on Natural Language Processing*. → page 52

Zhu, J. and Xing, E. (2010). Conditional Topic Random Fields. In *International Conference on Machine Learning*. → page 35

Ziller, W. (2014). Riemannian manifolds with positive sectional curvature. In *Geometry of manifolds with non-negative sectional curvature*, pages 1–19. Springer. → page 57

Appendix A

Chapter 2 Supplementary Material

A.1 Convergence Rate of SVRG with Error

We first give the proof of Theorem 2.1, which gives a convergence rate for SVRG with an error and uniform sampling. We then turn to the case of non-uniform sampling.

A.1.1 Proof of Theorem 1

We follow a similar argument to Johnson and Zhang (2013), but propagating the error e^s through the analysis. We begin by deriving a simple bound on the variance of the sub-optimality of the gradients.

Lemma A.1. *For any x,*

$$\frac{1}{n}\sum_{i=1}^{n}\|\nabla f_i(x)-\nabla f_i(x^*)\|^2 \leq 2L[f(x)-f(x^*)].$$

Proof. Because each ∇f_i is L-Lipschitz continuous, we have (Nesterov, 2004, Theorem 2.1.5)

$$f_i(x) \geq f_i(y)+\langle \nabla f_i(x), x-y\rangle + \frac{1}{2L}\|\nabla f_i(x)-\nabla f_i(y)\|^2.$$

Setting $y = x^*$ and summing this inequality times $(1/n)$ over all i we obtain the result. □

In this section we'll use $\tilde{x}$ to denote x^s, e to denote e^s, and we'll use v_t to denote the search direction at iteration t,

$$v_t = \nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x}) + \tilde{g} + e.$$

Note that $\mathbb{E}[v_t] = \nabla f(x_{t-1}) + e$ and the next lemma bounds the variance of this value.

Lemma A.2. *In each iteration t of the inner loop,*

$$\mathbb{E}\|v_t\|^2 \leq 4L[f(x_{t-1}) - f(x^*)] + 4L[f(\tilde{x}) - f(x^*)] + 2\|e\|^2.$$

Proof. By using the inequality $\|x+y\|^2 \leq 2\|x\|^2 + 2\|y\|^2$ and the property $\mathbb{E}[\nabla f_{i_t}(\tilde{x}) - \nabla f_{i_t}(x^*)] = \nabla f(\tilde{x})$, we have

$$\begin{aligned}
\mathbb{E}\|v_t\|^2 &= \mathbb{E}\|\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(\tilde{x}) + \tilde{g} + e\|^2 \\
&\leq 2\mathbb{E}\|\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(x^*)\|^2 \\
&\quad + 2\mathbb{E}\|[\nabla f_{i_t}(\tilde{x}) - \nabla f_{i_t}(x^*)] - \nabla f(\tilde{x}) - e\|^2 \\
&= 2\mathbb{E}\|\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(x^*)\|^2 + 2\mathbb{E}\|[\nabla f_{i_t}(\tilde{x}) - \nabla f_{i_t}(x^*)] \\
&\quad - \mathbb{E}[\nabla f_{i_t}(\tilde{x}) - \nabla f_{i_t}(x^*)] - e\|^2 \\
&= 2\mathbb{E}\|\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(x^*)\|^2 + 2\mathbb{E}\|[\nabla f_{i_t}(\tilde{x}) - \nabla f_{i_t}(x^*)] \\
&\quad - \mathbb{E}[\nabla f_{i_t}(\tilde{x}) - \nabla f_{i_t}(x^*)]\|^2 + 2\|e\|^2 \\
&\quad - 4\mathbb{E}\left\langle [\nabla f_{i_t}(\tilde{x}) - \nabla f_{i_t}(x^*)] - \mathbb{E}[\nabla f_{i_t}(\tilde{x}) - \nabla f_{i_t}(x^*)], e \right\rangle \\
&= 2\mathbb{E}\|\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(x^*)\|^2 + 2\mathbb{E}\|[\nabla f_{i_t}(\tilde{x}) - \nabla f_{i_t}(x^*)] \\
&\quad - \mathbb{E}[\nabla f_{i_t}(\tilde{x}) - \nabla f_{i_t}(x^*)]\|^2 + 2\|e\|^2.
\end{aligned}$$

If we now use that $\mathbb{E}[\|X - \mathbb{E}[X]\|^2] \leq \mathbb{E}\|X\|^2$ for any random variable X, we obtain the result by applying Lemma A.1 to bound $\mathbb{E}\|\nabla f_{i_t}(x_{t-1}) - \nabla f_{i_t}(x^*)\|^2$ and $\mathbb{E}\|[\nabla f_{i_t}(\tilde{x}) - \nabla f_{i_t}(x^*)]\|^2$. □

The following Lemma gives a bound on the distance to the optimal solution.

Lemma A.3. *In every iteration t of the inner loop,*

$$\mathbb{E}\|x_t - x^*\|^2 \leq \|x_{t-1} - x^*\|^2 - 2\eta(1-2\eta L)[f(x_{t-1}) - f(x^*)] \\ + 4L\eta^2[f(\tilde{x}) - f(x^*)] + 2\eta(Z\|e\| + \eta\|e\|^2).$$

Proof. We expand the expectation and bound $\mathbb{E}\|v_t\|^2$ using Lemma A.2 to obtain

$$\begin{aligned}
&\mathbb{E}\|x_t - x^*\|^2 \\
&\quad = \|x_{t-1} - x^*\|^2 - 2\eta\,\langle x_{t-1} - x^*, \mathbb{E}[v_t]\rangle + \eta^2\mathbb{E}\|v_t\|^2 \\
&\quad = \|x_{t-1} - x^*\|^2 - 2\eta\,\langle x_{t-1} - x^*, \nabla f(x_{t-1}) + e\rangle + \eta^2\mathbb{E}\|v_t\|^2 \\
&\quad = \|x_{t-1} - x^*\|^2 - 2\eta\,\langle x_{t-1} - x^*, \nabla f(x_{t-1})\rangle - 2\eta\,\langle x_{t-1} - x^*, e\rangle + \eta^2\mathbb{E}\|v_t\|^2 \\
&\quad \leq \|x_{t-1} - x^*\|^2 - 2\eta\,\langle x_{t-1} - x^*, e\rangle - 2\eta[f(x_{t-1}) - f(x^*)] + 2\eta^2\|e\|^2 \\
&\qquad + 4L\eta^2[f(x_{t-1}) - f(x^*)] + 4L\eta^2[f(\tilde{x}) - f(x^*)].
\end{aligned}$$

The inequality above follows from convexity of f. The result follows from applying Cauchy-Schwartz to the linear term in e and that $\|x_{t-1} - x^*\|_2 \leq Z$. □

To prove Theorem 2.1 from Chapter 2, we first sum the inequality in Lemma A.3 for all $t = 1, ..., m$ and take the expectation with respect to the choice of x^s to get

$$\mathbb{E}\|x_m - x^*\|^2 \leq \mathbb{E}\|x_0 - x^*\|^2 - 2\eta(1-2L\eta)m\mathbb{E}[f(x_{t-1}) - f(x^*)] \\ + 4L\eta^2 m\mathbb{E}[f(\tilde{x}) - f(x^*)] + 2m\eta(Z\mathbb{E}\|e\| + \eta\mathbb{E}\|e\|^2).$$

Re-arranging, and noting that $x_0 = \tilde{x}_{s-1}$ and $\mathbb{E}\,[f(x_{t-1})] = \mathbb{E}\,[f(x^s)]$, we have that

$$\begin{aligned}
&2\eta(1-2L\eta)m\mathbb{E}[f(x_s) - f(x^*)] \\
&\qquad \leq \mathbb{E}\|\tilde{x}_{s-1} - x^*\|^2 + 4L\eta^2 m\mathbb{E}[f(\tilde{x}_{s-1}) - f(x^*)] \\
&\qquad\quad + 2m\eta(Z\mathbb{E}\|e^{s-1}\| + \eta\mathbb{E}\|e^{s-1}\|^2) \\
&\qquad \leq \frac{2}{\mu}\mathbb{E}[f(\tilde{x}_{s-1}) - f(x^*)] + 4L\eta^2 m\mathbb{E}[f(\tilde{x}_{s-1}) - f(x^*)] \\
&\qquad\quad + 2m\eta(Z\mathbb{E}\|e\| + \eta\mathbb{E}\|e\|^2),
\end{aligned}$$

where the last inequality uses strong-convexity and that $\nabla f(x^*) = 0$. By dividing

both sides by $2\eta(1-2L\eta)m$ (which is positive due to the constraint $\eta \leq 1/2L$ implied by $0 < \rho < 1$ and $\eta > 0$), we get

$$\begin{aligned}
&\mathbb{E}[f(x_s) - f(x^*)] \\
&\leq \left(\frac{1}{m\mu(1-2\eta L)\eta} + \frac{2L\eta}{1-2\eta L}\right)\mathbb{E}[f(\tilde{x}_{s-1}) - f(x^*)] \\
&\quad + \frac{1}{1-2\eta L}\left(Z\mathbb{E}\|e^{s-1}\| + \eta\mathbb{E}\|e^{s-1}\|^2\right).
\end{aligned}$$

A.1.2 Non-Uniform Sampling

If we sample i_t proportional to the individual Lipschitz constants L_i, then we have the following analogue of Lemma A.1.

Lemma A.4. *For any x,*

$$\mathbb{E}\left\|\frac{\bar{L}}{L_i}[\nabla f_i(x) - \nabla f_i(x^*)]\right\|^2 \leq 2\bar{L}[f(x) - f(x^*)].$$

Proof. Because each ∇f_i is L_i-Lipschitz continuous, we have (Nesterov, 2004, Theorem 2.1.5)

$$f_i(x) \geq f_i(y) + \langle \nabla f_i(x), x - y\rangle + \frac{1}{2L_i}\|\nabla f_i(x) - \nabla f_i(y)\|^2.$$

Setting $y = x^*$ and summing this inequality times $(1/n)$ over all i we have

$$\begin{aligned}
\mathbb{E}\left\|\frac{\bar{L}}{L_i}[\nabla f_i(x) - \nabla f_i(x^*)]\right\|^2 &= \sum_{i=1}^{n}\frac{L_i}{n\bar{L}}\frac{\bar{L}^2}{L_i^2}\|\nabla f_i(x) - \nabla f_i(y)\|^2 \\
&= \frac{\bar{L}}{n}\sum_{i=1}^{n}\frac{1}{L_i}\|\nabla f_i(x) - \nabla f_i(y)\|^2 \\
&\leq \frac{\bar{L}}{n}\sum_{i=1}^{n}\frac{1}{L_i}2L_i[f_i(x) - f_i(x^*) - \langle \nabla f_i(x), x - x^*\rangle] \\
&= 2\bar{L}[f(x) - f(x^*)]
\end{aligned}$$

□

With this modified lemma, we can derive the convergence rate under this non-uniform sampling scheme by following an identical sequence of steps but where each instance of L is replaced by $\bar{L}$.

A.2 Mixed SVRG and SG Method

We first give the proof of Theorem 2.2 in Chapter 2, which analyzes a method that mixes SG and SVRG updates using a constant step size. We then consider a variant where the SG and SVRG updates use different step sizes.

A.2.1 Proof of Theorem 2

Recall that the SG update is

$$x_t = x_{t-1} - \eta \nabla f_{i_t}(x_{t-1}).$$

Using this in Lemma A.3 and following a similar argument we have

$$\begin{aligned}
\mathbb{E}\|x_t - x^*\|^2 \leq \alpha\{&\|x_{t-1} - x^*\|^2 - 2\eta(1-2\eta L)[f(x_{t-1}) - f(x^*)] \\
&+ 4L\eta^2[f(\tilde{x}) - f(x^*)] + 2\eta(Z\|e\| + \eta\|e\|^2)\} \\
&+ \beta\{\|x_{t-1} - x^*\|^2 + \eta^2\mathbb{E}\|\nabla f_{i_t}(x_{t-1})\|^2 \\
&- 2\eta\,\langle x_{t-1} - x^*, \mathbb{E}[\nabla f_{i_t}(x_{t-1})]\rangle\} \\
\leq \|&x_{t-1} - x^*\|^2 - 2\eta(1-2\eta L)[f(x_{t-1}) - f(x^*)] \\
&+ \alpha 4L\eta^2[f(\tilde{x}) - f(x^*)] + \alpha 2\eta(Z\|e\| + \eta\|e\|^2) + \beta\eta^2\sigma^2,
\end{aligned}$$

where the second inequality uses convexity of f and we have defined $\beta = (1-\alpha)$.

We now sum up both sides and take the expectation with respect to the history,

$$\begin{aligned}
\mathbb{E}\|x_m - x^*\|^2 \leq &\mathbb{E}\|x_0 - x^*\|^2 - 2\eta(1-2L\eta)m\mathbb{E}[f(x_{t-1}) - f(x^*)] \\
&+ 4\alpha L\eta^2 m\mathbb{E}[f(\tilde{x}) - f(x^*)] + 2m\alpha\eta(Z\mathbb{E}\|e\| + \eta\mathbb{E}\|e\|^2) \\
&+ m\beta\eta^2\sigma^2.
\end{aligned}$$

By re-arranging the terms we get

$$\begin{aligned}2\eta(1-2L\eta)m\mathbb{E}[f(x_s)-f(x^*)] \leq \frac{2}{\mu}\mathbb{E}[f(\tilde{x}_{s-1})-f(x^*)] \\ +4\alpha L\eta^2 m\mathbb{E}[f(\tilde{x}_{s-1})-f(x^*)] \\ +2m\alpha\eta(Z\mathbb{E}\|e\|+\eta\mathbb{E}\|e\|^2)+m\beta\eta^2\sigma^2,\end{aligned}$$

and by dividing both sides by $2\eta(1-2L\eta)m$ we get the result.

A.2.2 Mixed SVRG and SG with Different Step Sizes

Consider a variant where we use a step size of η in the SVRG update and a step-size η_s in the SG update (which will decrease as the iterations proceed). Analyzing the mixed algorithm in this setting gives

$$\begin{aligned}\mathbb{E}\|x_t-x^*\|^2 &\leq \alpha\{\|x_{t-1}-x^*\|^2-2\eta(1-2\eta L)[f(x_{t-1})-f(x^*)] \\ &\quad +4L\eta^2[f(x^s)-f(x^*)]+2\eta(Z\|e^s\|+\eta\|e^s\|^2)\} \\ &\quad +\beta\{\|x_{t-1}-x^*\|^2+\eta_s^2\mathbb{E}\|\nabla f_{i_t}(x_{t-1})\|^2 \\ &\quad -2\eta_s\langle x_{t-1}-x^*,\mathbb{E}[\nabla f_{i_t}(x_{t-1})]\rangle\} \\ &= \mathbb{E}\left[\|x_{t-1}-x^*\|_2^2\right]-2\alpha\eta(1-2\eta L)[f(x_{t-1})-f(x^*)] \\ &\quad +4\alpha L\eta^2[f(x^s)-f(x^*)]+2\alpha\eta(Z\|e^s\|+\eta\|e^s\|^2) \\ &\quad +\beta\eta_s^2\mathbb{E}\|\nabla f_{i_t}(x_{t-1})\|^2-2\beta\eta_s\langle x_{t-1}-x^*,f(x_{t-1})]\rangle \\ &\leq \mathbb{E}\left[\|x_{t-1}-x^*\|_2^2\right]-\{2\alpha\eta(1-2\eta L)+2\beta\eta_s\}[f(x_{t-1})-f(x^*)] \\ &\quad +4\alpha L\eta^2[f(x^s)-f(x^*)]+2\alpha\eta(Z\|e^s\|+\eta\|e^s\|^2)+\beta\eta_s^2\sigma^2.\end{aligned}$$

As before, we take the expectation for all t and sum up these values, then rearranage and use strong-convexity of f to get

$$\begin{aligned}&2m\{\alpha\eta(1-2\eta L)+\beta\eta_s\}[f(x_s)-f(x^*)] \\ &\quad\leq\left\{\frac{2}{\mu}+4m\alpha L\eta^2\right\}[f(x^s)-f(x^*)]+2m\alpha\eta(Z\|e^s\|+\eta\|e^s\|^2)+m\beta\eta_s^2\sigma^2.\end{aligned}$$

If we now divide both side by $2m(\alpha\eta(1-2\eta L)+\beta\eta_s)$, we get

$$\begin{aligned}
&\mathbb{E}\left[f(x_s)-f(x^*)\right] \\
&\leq \left\{\frac{1}{\mu m(\alpha\eta(1-2\eta L)+\beta\eta_s)}+\frac{2\alpha L\eta^2}{\alpha\eta(1-2\eta L)+\beta\eta_s}\right\}[f(x^s)-f(x^*)] \\
&\quad+\frac{\alpha\eta}{\alpha\eta(1-2\eta L)+\beta\eta_s}(Z\mathbb{E}\left[\|e^s\|\right]+\eta\mathbb{E}\left[\|e^s\|^2\right]) \\
&\quad+\frac{1}{2(\alpha\eta(1-2\eta L)+\beta\eta_s)}\beta\eta_s^2\sigma^2.
\end{aligned}$$

To improve the dependence on the error e^s and variance σ^2 compared to the basic SVRG algorithm with error e^s (Theorem 2.1), we require that the terms depending on these values are smaller,

$$\begin{aligned}
&\frac{\alpha\eta}{\alpha\eta(1-2\eta L)+\beta\eta_s}\left(Z\mathbb{E}\left[\|e^s\|\right]+\eta\mathbb{E}\left[\|e^s\|^2\right]\right) \\
&+\frac{1}{2(\alpha\eta(1-2\eta L)+\beta\eta_s)}\beta\eta_s^2\sigma^2 \leq \frac{1}{1-2\eta L}\left(Z\mathbb{E}\left[\|e^s\|\right]+\eta\mathbb{E}\left[\|e^s\|^2\right]\right).
\end{aligned}$$

Let $\kappa=(1-2\eta L)$ and $\zeta=Z\mathbb{E}\left[\|e^s\|\right]+\eta\mathbb{E}\left[\|e^s\|^2\right]$, this requires

$$\frac{\alpha\eta}{\alpha\eta\kappa+\beta\eta_s}\zeta+\frac{\beta\eta_s^2}{2(\alpha\eta\kappa+\beta\eta_s)}\sigma^2 \leq \frac{\zeta}{\kappa}.$$

Thus, it is sufficient that η_s satisfies

$$\eta_s \leq \frac{2\zeta}{\kappa\sigma^2}.$$

Using the relationship between expected error and S^2, while noting that $S^2 \leq \sigma^2$ and $\frac{(n-|\mathscr{B}|)}{n|\mathscr{B}|} \leq 1$, a step size of the form $\eta_s = O^*(\sqrt{(n-|\mathscr{B}|)/n|\mathscr{B}|})$ will improve the dependence on e^s and σ^2 compared to the dependence on e^s in the pure SVRG method.

A.3 Proximal and Regularized SVRG

In this section we consider objectives of the form

$$f(x) = h(x) + g(x),$$

where $g(x) = \frac{1}{n}\sum_{i=1}^{n} g_i(x)$. We first consider the case where h is non-smooth and consider a proximal-gradient variant of SVRG where there is an error in the calculation of the gradient (Algorithm 9). We then consider smooth functions h where we use a modified SVRG iteration,

$$x_{t+1} = x_t - \eta\left(\nabla h(x_t) + \nabla g_{i_t}(x_t) - \nabla g_{i_t}(x^s) + \mu^s\right),$$

where $\mu^s = \nabla g(x^s)$.

A.3.1 Composite Case

Similar to the work of Xiao and Zhang (2014), in this section we assume that f, g and h are μ-, μ_g-, μ_h-strongly convex (respectively). As before, we assume each g_i is convex and has an L-Lipschitz continuous gradient, but h can potentially be non-smooth. The algorithm we propose here extends the algorithm of Xiao and Zhang (2014), but adding an error term. In the algorithm we use the proximal operator which is defined by

$$prox_h(y) = \underset{x\in\mathbb{R}^d}{\operatorname{argmin}}\left\{\frac{1}{2}\|x-y\|^2 + h(x)\right\}.$$

Below, we give a convergence rate for this algorithm with an error e^s.

Theorem A.5. *If we have $\tilde{\mu}_s = \nabla g(x^s) + e^s$ and set the step-size η and number of inner iterations m so that*

$$\rho \equiv \frac{1}{m\mu(1-4\eta L)\eta} + \frac{4L\eta(m+1)}{(1-4\eta L)m} < 1,$$

then Algorithm 9 has

$$\mathbb{E}[f(x_{s+1}) - f(x^*)] \leq \rho E[f(\tilde{x}_s) - f(x^*)] + \frac{1}{1-4\eta L}(Z\mathbb{E}\|e^s\| + \eta\mathbb{E}\|e^s\|^2),$$

Algorithm 9 Batching Prox SVRG

Input: update frequency m and learning rate η and sample size increasing rate α
Initialize $\tilde{x}$
for $s = 1, 2, 3, \ldots$ **do**
  Choose batch size $|\mathscr{B}|$
  $\mathscr{B}$ = randomly choose $|\mathscr{B}|$ elements of $\{1, 2, \ldots, n\}$.
  $\tilde{\mu} = \frac{1}{|\mathscr{B}|} \sum_{i \in \mathscr{B}} \nabla g_i(\tilde{x})$
  $x_0 = \tilde{x}$
  for $t = 1, 2, \ldots, m$ **do**
    Randomly pick $i_t \in 1, \ldots, n$
    $v_t = \nabla g_{i_t}(x_{t-1}) - \nabla g_{i_t}(\tilde{x}) + \tilde{\mu}$
    $x_t = prox_{\eta h}(x_{t-1} - \eta v_t)$ $(*)$
  end for
  set $\tilde{x} = \frac{1}{m} \sum_{t=1}^{m} x_t$
end for

where $\|x_t - x^*\| < Z$.

To prove Theorem A.5, we use Lemma 1,2 and 3 from Xiao and Zhang (2014), which are unchanged when we allow an error. Below we modify their Corollary 3 and then the proof of their main theorem.

Lemma A.6. *Consider* $v_t = \nabla g_{i_t}(x_{t-1}) - \nabla g_{i_t}(\tilde{x}) + \nabla g(\tilde{x}) + e$. *Then,*

$$\mathbb{E}\|v_t - \nabla g(x_{t-1})\|^2 \leq \|e\|^2 + 4L[f(x_{t-1}) - f(x^*) + f(\tilde{x}) - f(x^*)].$$

Proof.

$$\begin{aligned}
&\mathbb{E}\|v_t - \nabla g(x_{t-1})\|^2 \\
&= \mathbb{E}\|\nabla g_{i_t}(x_{t-1}) - \nabla g_{i_t}(\tilde{x}) + \nabla g(\tilde{x}) + e - \nabla g(x_{t-1})\|^2 \\
&= \|e\|^2 + E\|\nabla g_{i_t}(x_{t-1}) - \nabla g_{i_t}(\tilde{x}) + \nabla g(\tilde{x}) - \nabla g(x_{t-1})\|^2 \\
&\leq \|e\|^2 + E\|\nabla g_{i_t}(x_{t-1}) - \nabla g_{i_t}(\tilde{x})\|^2 \\
&\leq \|e\|^2 + 2E\|\nabla g_{i_t}(x_{t-1}) - \nabla g_{i_t}(x^*)\|^2 + 2E\|\nabla g_{i_t}(\tilde{x}) - \nabla g_{i_t}(x^*)\|^2.
\end{aligned}$$

Using Lemma 1 from Xiao and Zhang (2014) and bounding the two expectations gives the result. □

Now we turn to proving Theorem A.5.

Proof. Following the proof of Theroem 1 in Xiao and Zhang (2014), we have

$$\|x_t - x^*\|^2 \leq \|x_{t-1} - x^*\|^2 - 2\eta[f(x_t) - f(x^*)] - 2\eta \langle \Delta_t, x_t - x^* \rangle,$$

where $\Delta_t = v_t - \nabla g(x_{t-1})$ and $\mathbb{E}[\Delta_t] = e$. Now to bound $\langle \Delta_t, x_t - x^* \rangle$, we define

$$\bar{x_t} = prox_h(x_{t-1} - \eta \nabla g(x_{t-1})),$$

and subsequently that

$$-2\eta \langle \Delta_t, x_t - x^* \rangle \leq 2\eta^2 \|\Delta_t\|^2 - 2\eta \langle \Delta_t, \bar{x}_t - x^* \rangle.$$

Combining with the two previous inequalities we get

$$\|x_t - x^*\|^2 \leq \|x_{t-1} - x^*\|^2 - 2\eta[f(x_t) - f(x^*)] + 2\eta^2 \|\Delta_t\|^2 - 2\eta \langle \Delta_t, \bar{x}_t - x^* \rangle.$$

If we take the expectation with respect to i_t we have

$$\mathbb{E}\|x_t - x^*\|^2 \leq \|x_{t-1} - x^*\|^2 - 2\eta \mathbb{E}[f(x_t) - f(x^*)] + 2\eta^2 \mathbb{E}\|\Delta_t\|^2 - 2\eta \langle \mathbb{E}\Delta_t, \bar{x}_t - x^* \rangle.$$

Now by using the Lemma A.6 and $\|\bar{x}_t - x^*\| < Z$ we have

$$\begin{aligned}
&\mathbb{E}\|x_t - x^*\|^2 \\
&\leq \|x_{t-1} - x^*\|^2 - 2\eta \mathbb{E}[f(x_t) - f(x^*)] + 8\eta^2 L[f(x_{t-1}) - f(x^*) + f(\tilde{x}) - f(x^*)] \\
&\quad + 2\eta^2 \|e\|^2 + 2\eta \|e\| Z.
\end{aligned}$$

The rest of the proof follows the argument of Xiao and Zhang (2014), and is simlar to the previous proofs in this appendix. We take the expectation and sum up values,

using convexity to give

$$2\eta(1-4L\eta)m[\mathbb{E}f(x^s)-f(x^*)] \leq (\frac{2}{\mu}+8L\eta^2(m+1))[f(\tilde{x}_{s-1}-f(x^*)] + 2\eta^2\|e\|^2+2\eta\|e\|Z.$$

By dividing both sides to $2\eta(1-4L\eta)m$, we get the result. □

A.3.2 Proof of Theorem 3

We now turn to the case where h is differentiable, and we use an iteration that incorporates the gradient $\nabla h(x_t)$. Let $G(x)=\frac{1}{n}\sum_{i=1}^n f_i(x)$. Recall that for this result we assume that ∇G is L_G-Lipschitz continuous, ∇h is L_h-Lipschitz continuous, and we defined $L_m=\max\{L_G,L_h\}$. If we let $v_t=\nabla h(x_t)+\nabla f_{i_t}(x_t)-\nabla f_{i_t}(x^s)+g^s$, then note that we have $\mathbb{E}[v_t]=\nabla f(x_t)$. Now as before we want to bound the expected second moment of v_t,

$$\begin{aligned}
\mathbb{E}\left[\|v_t\|_2^2\right] &= \mathbb{E}\left[\|\nabla h(x_t)+\nabla f_{i_t}(x_t)-\nabla f_{i_t}(x^s)+g^s\|_2^2\right] \\
&= \mathbb{E}[\|\nabla h(x_t)+\nabla f_{i_t}(x_t)-\nabla f_{i_t}(x^s)+g^s+\nabla h(x^*) \\
&\quad +\nabla f_{i_t}(x^*)-\nabla h(x^*)-\nabla f_{i_t}(x^*)+\nabla h(x^s)-\nabla h(x^s)\|^2] \\
&\leq 2\|\nabla h(x_t)-\nabla h(x^*)\|_2^2+2\mathbb{E}\left[\|\nabla f_{i_t}(x_t)-\nabla f_{i_t}(x^*)\|_2^2\right] \\
&\quad +2\mathbb{E}[\|-\nabla f_{i_t}(x^s)+g^s+\nabla h(x^*)+\nabla f_{i_t}(x^*) \\
&\quad +\nabla h(x^s)-\nabla h(x^s)\|^2] \\
&= 2\|\nabla h(x_t)-\nabla h(x^*)\|_2^2+2\mathbb{E}\left[\|\nabla f_{i_t}(x_t)-\nabla f_{i_t}(x^*)\|_2^2\right] \\
&\quad +2\mathbb{E}[\|\nabla f_{i_t}(x^s)-g^s-\nabla h(x^*)-\nabla f_{i_t}(x^*)-\nabla h(x^s) \\
&\quad +\nabla h(x^s)+\nabla h(x^*)+\nabla G(x^*)\|^2] \\
&= 2\|\nabla h(x_t)-\nabla h(x^*)\|_2^2+2\mathbb{E}\left[\|\nabla f_{i_t}(x_t)-\nabla f_{i_t}(x^*)\|_2^2\right] \\
&\quad +2\mathbb{E}\left[\|\nabla f_{i_t}(x^s)-g^s-\nabla f_{i_t}(x^*)+\nabla G(x^*)\|_2^2\right] \\
&\leq 2\|\nabla h(x_t)-\nabla h(x^*)\|_2^2+2\mathbb{E}\left[\|\nabla f_{i_t}(x_t)-\nabla f_{i_t}(x^*)\|_2^2\right] \\
&\quad +2\mathbb{E}\left[\|\nabla f_{i_t}(x^s)-\nabla f_{i_t}(x^*)\|_2^2\right].
\end{aligned}$$

Now using that $\|\nabla h(x^s) - \nabla h(x^*)\|_2^2 \geq 0$ and $\|f(x) - f(y)\|_2^2 \leq 2L[f(x) - f(y) - \langle \nabla f(y), x - y\rangle]$,

$$\begin{aligned}
\mathbb{E}\left[\|v_t\|_2^2\right] &\leq 2\|\nabla h(x_t) - \nabla h(x^*)\|_2^2 + 2\mathbb{E}\left[\|\nabla f_{i_t}(x_t) - \nabla f_{i_t}(x^*)\|_2^2\right] \\
&\quad + 2\mathbb{E}\left[\|\nabla f_{i_t}(x^s) - \nabla f_{i_t}(x^*)\|_2^2\right] + 2\|\nabla h(x^s) - \nabla h(x^*)\|_2^2 \\
&\leq 4L_h[h(x_t) - h(x^*) - \langle \nabla h(x^*), x_t - x^*\rangle] \\
&\quad + 4L_G[G(x_t) - G(x^*) - \langle \nabla G(x^*), x_t - x^*\rangle] \\
&\quad + 4L_h[h(x^s) - h(x^*) - \langle \nabla h(x^*), x^s - x^*\rangle] \\
&\quad + 4L_G[G(x^s) - G(x^*) - \langle \nabla G(x^*), x^s - x^*\rangle] \\
&\leq 4L_m[f(x_t) - f(x^*)] + 4L_m[f(x^s) - f(x^*)].
\end{aligned}$$

From this point, we follow the standard SVRG argument to obtain

$$\mathbb{E}\left[f(x_{s+1}) - f(x^*)\right] \leq \left(\frac{1}{m\mu(1 - 2\eta L_m)} + \frac{2L_m\eta}{1 - 2\eta L_m}\right)[f(x_s - f(x^*)].$$

A.4 Mini-Batch

We first give an analysis of SVRG where mini-batches are selected by sampling propotional to the Lipschitz constants of the gradients. We then consider the mixed deterministic/random sampling scheme described in Chapter 2.

A.4.1 SVRG with Mini-batch

Here we consider using a 'mini-batch' of examples in the *inner* SVRG loop. We use M to denote the batch size, and we assume that the elements of the mini-batch are sampled with a probability of $p_i = L_i/n\bar{L}$. This gives a search direction and inner iteration of:

$$\begin{aligned}
v_t &= g^s + \frac{1}{M}\left[\sum_{i \in M} \frac{1}{np_i}\left(\nabla f_i(x_t) - \nabla f_i(x^s)\right)\right], \\
x_{t+1} &= x_t - \eta v_t.
\end{aligned}$$

Observe that $\mathbb{E}[v_t] = \nabla f(x_t)$, and since each f_i is L_i-smooth we still have that

$$\|\nabla f_i(x) - \nabla f_i(y)\|_2^2 \leq L_i \left(f_i(x) - f_i(y) - \langle \nabla f_i(y), x - y\rangle\right).$$

It follows from the definition of p_i that

$$\begin{aligned}\mathbb{E}\left[\left\|\frac{1}{np_i}(\nabla f_i(x) - \nabla f_i(y))\right\|^2\right] &= \frac{1}{n}\sum_i \frac{1}{np_i}\|\nabla f_i(x) - \nabla f_i(y)\|_2^2 \\ &\leq 2\bar{L}\left(f(x) - f(y) - \langle \nabla f(y), x - y\rangle\right),\end{aligned}$$

which we use to bound $\mathbb{E}\left[\|v_t\|_2^2\right]$ as before,

$$\begin{aligned}\mathbb{E}\left[\|v_t\|_2^2\right] &= \mathbb{E}\left[\left\|\frac{1}{M}\sum_i\left(\frac{1}{np_i}(\nabla f_i(x_t) - \nabla f_i(x^s) + g^s\right)\right\|^2\right] \\ &= \mathbb{E}[\|\frac{1}{M}\sum_i(\frac{1}{np_i}(\nabla f_i(x_t) - \nabla f_i(x^*) + \\ &\qquad \nabla f_i(x^*) - \nabla f_i(x^s) + g^s)\|^2] \\ &\leq \frac{2}{M}\sum_i \mathbb{E}\left[\left\|\left(\frac{1}{np_i}(\nabla f_i(x_t) - \nabla f_i(x^*))\right)\right\|^2\right] \\ &\qquad + \frac{2}{M}\sum_i \mathbb{E}\left[\left\|\left(\frac{1}{np_i}(\nabla f_i(x^s) - \nabla f_i(x^*)) - g^s\right)\right\|^2\right] \\ &\leq \frac{2}{M}\sum_i \mathbb{E}\left[\left\|\left(\frac{1}{np_i}(\nabla f_i(x_t) - \nabla f_i(x^*))\right)\right\|^2\right] \\ &\qquad + \frac{2}{M}\sum_i \mathbb{E}\left[\left\|\left(\frac{1}{np_i}(\nabla f_i(x^s) - \nabla f_i(x^*))\right)\right\|^2\right] \\ &\leq 4\bar{L}\left[f(x_t) - f(x^*)\right] + 4\bar{L}\left[f(x^s) - f(x^*)\right].\end{aligned}$$

It subsequently follows that

$$\mathbb{E}\left[f(x^{s+1}) - f(x^*)\right] \leq \left(\frac{1}{m\mu(1 - 2\eta\bar{L})\eta} + \frac{2\bar{L}\eta}{1 - 2\eta\bar{L}}\right)\mathbb{E}\left[f(x^s) - f(x^*)\right].$$

A.4.2 Proof of Theorem 4

We now consider the case where we have $g(x) = (1/n)\sum_{i\notin[\mathscr{B}_f]} f_i(x)$ and $h(x) = (1/n)\sum_{i\in[\mathscr{B}_f]} f_i(x)$ for some batch $\mathscr{B}_f$ with $|\mathscr{B}_f| = M_f$. We assume that we sample M_r elements of g with probability of $p_i = \frac{L_i}{(n-M_f)\bar{L}_r}$ and that we use:

$$\begin{aligned}
v_t &= \nabla g(x^s) + \nabla h(x_t) + \frac{1}{M_r}\left[\sum_{i\in M_r}\frac{1}{np_i}\left(\nabla f_i(x_t) - \nabla f_i(x^s)\right)\right] \\
&= g^s + \nabla h(x_t) + \frac{1}{M_r}\left[\sum_{i\in M_r}\frac{1}{np_i}\left(\nabla f_i(x_t) - \nabla f_i(x^s)\right)\right] - \nabla h(x^s) \\
x_{t+1} &= x_t - \eta v_t,
\end{aligned}$$

where as usual $\mu^s = \frac{1}{n}\sum_{i=1}^n f_i'(x^s) = \nabla g(x^s) + \nabla h(x^s)$. Note that $\mathbb{E}[v_t] = \nabla f(x_t)$. We first bound $\mathbb{E}\left[\|v_t\|_2^2\right]$,

$$\begin{aligned}
\mathbb{E}\left[\|v_t\|^2\right] &= \mathbb{E}[\|g^s + \nabla h(x_t) + 1/M_r\left[\sum_{i\in M_r}\frac{1}{np_i}\left(\nabla f_i(x_t) - \nabla f_i(x^s)\right)\right] \\
&\qquad - \nabla h(x^s)\|^2] \\
&= \mathbb{E}[\|g^s + \nabla h(x_t) - \nabla h(x^*) + 1/M_r\left[\sum_{i\in M_r}\frac{1}{np_i}\left(\nabla f_i(x_t) - \nabla f_i(x^*)\right)\right] \\
&- 1/M_r\left[\sum_{i\in M_r}\frac{1}{np_i}\left(\nabla f_i(x^s) - \nabla f_i(x^*)\right)\right] - \nabla h(x^s) + \nabla h(x^*)\|_2^2] \\
&\leq \frac{2M_f}{n^2}\underbrace{\sum_{i\in\mathscr{B}_f}\|\nabla f_i(x_t) - \nabla f_i(x^*)\|_2^2}_{\text{Fixed part}} + \\
&\qquad \underbrace{2/M_r\sum_{i\in\mathscr{B}_r}\mathbb{E}\left[\left\|\frac{1}{np_i}(\nabla f_i(x_t) - \nabla f_i(x^*))\right\|^2\right]}_{\text{Random part}} \\
&+ 2\mathbb{E}\left[\left\|1/M_r\left[\sum_{i\in M_r}\frac{1}{np_i}\left(\nabla f_i(x^s) - \nabla f_i(x^*)\right)\right] + \nabla h(x^s) - \nabla h(x^*) - g^s\right\|^2\right],
\end{aligned}$$

where the inequality uses $\|a+b\|^2 \leq 2\|a\|^2 + 2\|b\|^2$. Now we bound each of the above terms separately,

$$\begin{aligned}
&2M_f/n^2 \sum_{i\in\mathscr{B}_f} \|\nabla f_i(x_t) - \nabla f_i(x^*)\|_2^2 \\
&\qquad \leq 2M_f/n^2 \sum_{i\in\mathscr{B}_f} 2L_i(f_i(x_t) - f_i(x^*) - \langle \nabla f_i(x^*), x_t - x^*\rangle) \\
&\qquad \leq \frac{4LM_f}{n}\left(h(x_t) - h(x^*) - \langle \nabla h(x^*), x_t - x^*\rangle\right),
\end{aligned}$$

$$\begin{aligned}
2/M_r \sum_{i\in\mathscr{B}_r} \mathbb{E}\left[\|\frac{1}{np_i}(\nabla f_i(x_t) - \nabla f_i(x^*))\|_2^2\right] &= 2\mathbb{E}\left[\|\frac{1}{np_i}(\nabla f_i(x_t) - \nabla f_i(x^*))\|_2^2\right] \\
&\leq 1/n^2 \sum_{j\notin\mathscr{B}_f} 1/p_i \|\nabla f_i(x_t) - \nabla f_i(x^*)\|_2^2 \\
&\leq 2/n^2 \sum_{j\notin\mathscr{B}_f} (n-M_f)\bar{L}_r\left(f_i(x_t) - f_i(x^*) - \langle \nabla f_i(x^*), x_t - x^*\rangle\right) \\
&= \frac{4(n-M_f)\bar{L}_r}{n}(g(x_t) - g(x^*) - \langle \nabla g(x^*), x_t - x^*\rangle).
\end{aligned}$$

Finally for the last term we have,

$$\begin{aligned}
&2\mathbb{E}\left[\left\|\frac{1}{M_r}\left[\sum_{i\in M_r}\frac{1}{np_i}(\nabla f_i(x^s) - \nabla f_i(x^*))\right] + \underbrace{\nabla h(x^s) - \nabla h(x^*) - g^s}_{=\nabla g(x^*) - \nabla g(x^s)}\right\|^2\right] \\
&\leq 2\mathbb{E}\left[\left\|\frac{1}{M_r}\sum_{i\in M_r}\frac{1}{np_i}(\nabla f_i(x^s) - \nabla f_i(x^*))\right\|^2\right] \\
&\leq \frac{4(n-M_f)\bar{L}_r}{n}\left(g(x^s) - g(x^*) - \langle \nabla g(x^*), x^s - x^*\rangle\right)
\end{aligned}$$

where the first inequality uses variance inequality ($\mathbb{E}\|X - \mathbb{E}X\|^2 \leq \mathbb{E}\|X\|^2$) and the second one comes from Lemma A.1. Since h is convex we can add $\frac{4(n-M_f)\bar{L}_r}{n}(h(x^s) -$

$h(x^*) - \langle \nabla h(x^*), x^s - x^* \rangle)$ to the right side of the above term, giving

$$2\mathbb{E}\left[\left\|\frac{1}{M_r}\left[\sum_{i\in M}\frac{1}{np_i}\left(\nabla f_i(x^s) - \nabla f_i(x^*)\right)\right] + \nabla h(x^s) - \nabla h(x^*) - g^s\right\|^2\right]$$
$$\leq \frac{4(n-M_f)\bar{L}_r}{n}\left(f(x^s) - f(x^*)\right).$$

Now following the proof technique we used several times, we can show that:

$$\mathbb{E}\left[f(x^{s+1}) - f(x^*)\right] \leq \left(\frac{2}{m\mu(2-\eta\zeta)\eta} + \frac{4\bar{L}_r M_f \eta}{n(2-\zeta\eta)}\right)\mathbb{E}\left[f(x^s) - f(x^*)\right],$$

where $\zeta = \max\{\frac{4LM_f}{n}, \frac{4\bar{L}_r(n-M_f)}{n}\}$.

A.5 Learning efficiency

In this section we closely follow Bottou and Bousquet (2007) to discuss the performance of SVRG, and other linearly-convergent stochastic methods, as learning algorithms. In the typical supervised learning setting, we are giving n independently drawn input-output pairs (x_i, y_i) from some distribution $P(x,y)$ and we seek to minimize the empirical risk,

$$E_n(f) = \frac{1}{n}\sum_{i=1}^{n}\ell(f(x_i), y_i) = \mathbb{E}_n[\ell(f(x), y)],$$

where ℓ is our loss function. However, in machine learning this is typically just a surrogate for the objective we are ultimately interested in. In particular, we typically want to minimize the *expected* risk,

$$E(f) = \int \ell(f(x), y) dP(x,y) = \mathbb{E}[\ell(f(x), y)],$$

which tells us how well we do on test data from the same distribution. We use f^* to denote the minimizer of the expected risk,

$$f^*(x) = \underset{\hat{y}\in\mathcal{X}}{\operatorname{argmin}}\,\mathbb{E}[\ell(\hat{y}, y)\,|\,x],$$

which is the best that a learner can hope to achieve.

Consider a family $\mathscr{F}$ of possible functions that we use to predict y_i from x_i. We write the minimizer of the expected risk over this restricted set as $f^*_{\mathscr{F}}$,

$$f^*_{\mathscr{F}} = \operatorname*{argmin}_{x \in \mathscr{X}} f \in \mathscr{F} E(f),$$

while we denote the empirical risk minimizer within this family as f_n,

$$f_n = \operatorname*{argmin}_{x \in \mathscr{X}} f \in \mathscr{F} E_n(f).$$

But, since we are applying a numerical optimizer we only assume that we find a ρ-optimal minimizer of the empirical risk $\tilde{f}_n$,

$$E_n(\tilde{f}_n) < E_n(f_n) + \rho,$$

In this setting, Bottou & Bousquet consider writing the sub-optimality of the approximate empirical risk minimizer $\tilde{f}_n$ compared to the minimizer of the expected risk f^* as

$$\begin{aligned} \mathscr{E} &= \mathbb{E}[E(\tilde{f}_n) - E(f^*)] \\ &= \underbrace{\mathbb{E}[E(f^*_{\mathscr{F}}) - E(f^*)]}_{\mathscr{E}_{\text{app}}} + \underbrace{\mathbb{E}[E(f_n) - E(f^*_{\mathscr{F}})]}_{\mathscr{E}_{\text{est}}} + \underbrace{\mathbb{E}[E(\tilde{f}_n) - E(f_n)]}_{\mathscr{E}_{\text{opt}}}, \end{aligned} \tag{A.1}$$

where the expectation is taken with respect to the output of the algorithm and with respect to the training examples that we sample. This decomposition shows how three intuitive terms affect the sub-optimality:

1. $\mathscr{E}_{\text{app}}$ is the *approximation error*: it measures the effect of restricting attention to the function class $\mathscr{F}$.

2. $\mathscr{E}_{\text{est}}$ is the *estimation error*: it measures the effect of only using a finite number of samples.

3. $\mathscr{E}_{\text{opt}}$ is the *optimization error*: it measures the effect of inexactly solving the optimization problem.

While choosing the family of possible approximating functions $\mathscr{F}$ is an interesting and important issue, for the remainder of this section we will assume that we are given a fixed family. In particular, Bottou & Bousquet's assumption is that $\mathscr{F}$ is linearly-parameterized by a vector $w \in \mathbb{R}^d$, and that all quantities are bounded (x_i, y_i, and w). This means that the approximation error $\mathscr{E}_{\text{app}}$ is fixed so we can only focus on the trade-off between the estimation error $\mathscr{E}_{\text{est}}$ and the optimization error $\mathscr{E}_{\text{opt}}$.

All other sections of this work focus on the case of *finite* datasets where we can afford to do several passes through the data (*small-scale learning problems* in the language of Bottou & Bousquet). In this setting, $\mathscr{E}_{\text{est}}$ is fixed so all we can do to minimize $\mathscr{E}$ is drive the optimization error ρ as small as possible. In this section we consider the case where we do not have enough time to process all available examples, or we have an infinite number of possible examples (*large-scale learning problems* in the language of Bottou & Bousquet). In this setting, the time restriction means we need to make a trade-off between the optimization error and the estimation error: should we increase n in order to decrease the estimation error $\mathscr{E}_{\text{est}}$ or should we revisit examples to try to more quickly decrease the optimization error $\mathscr{E}_{\text{opt}}$ while keeping the estimation error fixed?

Bottou & Bousquet discuss how under various assumptions we have the variance condition

$$\forall f \in \mathscr{F} \quad \mathbb{E}\left[(\ell(f(X),Y) - \ell(f^*_{\mathscr{F}}(X),Y))^2\right] \leq c\,(E(f) - E(f^*_{\mathscr{F}}))^{2-\frac{1}{\alpha}},$$

and how this implies the bound

$$\mathscr{E} = O\left(\mathscr{E}_{\text{app}} + \left(\frac{d}{n}\log\frac{n}{d}\right)^{\alpha} + \rho\right).$$

To make the second and third terms comparable, we can take $\rho = \left(\frac{d}{n}\log\frac{n}{d}\right)^{\alpha}$. Then to achieve an accuracy of $O(\mathscr{E}_{\text{app}} + \varepsilon)$ it is sufficient to take $n = O\left(\frac{d}{\varepsilon^{1/\alpha}}\log(1/\varepsilon)\right)$

samples:

$$
\begin{aligned}
\mathscr{E} &= O\left(\mathscr{E}_{\text{app}} + \left(\frac{d}{n}\log\frac{n}{d}\right)^{\alpha} + \rho\right) \\
&= O\left(\mathscr{E}_{\text{app}} + \left(\frac{d}{n}\log\frac{n}{d}\right)^{\alpha} + \left(\frac{d}{n}\log\frac{n}{d}\right)^{\alpha}\right) \\
&= O\left(\mathscr{E}_{\text{app}} + \left(\frac{d}{n}\log\frac{n}{d}\right)^{\alpha}\right) \\
&= O\left(\mathscr{E}_{\text{app}} + \left(\frac{\varepsilon^{\frac{1}{\alpha}}}{\log(\frac{1}{\varepsilon})}\log\left(\frac{\log(\frac{1}{\varepsilon})}{\varepsilon^{\frac{1}{\alpha}}}\right)\right)^{\alpha}\right) \\
&= O\left(\mathscr{E}_{\text{app}} + \varepsilon\left(\frac{\log(\log(1/\varepsilon)) - \frac{1}{\alpha}\log(\varepsilon)}{\log(1/\varepsilon)}\right)^{\alpha}\right) \\
&= O(\mathscr{E}_{\text{app}} + \varepsilon).
\end{aligned}
$$

The results presented in Chapter 2 follow from noting that (i) the iteration cost of SVRG is $O(d)$ and (ii) that the number of iterations for SVRG to reach an accuracy of ρ is $O((n+\kappa)\log(1/\rho))$.

Appendix B

Chapter 3 Supplementary Material

B.1 Proof of Part (a) of Theorem 3.1

In this section we consider the minimization problem

$$\min_x f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x),$$

where each f_i' is L-Lipschitz continuous and each f_i is μ-strongly-convex. We will define Algorithm B.1, a variant of SAGA, by the sequences $\{x^k\}$, $\{v_k\}$, and $\{\phi_j^k\}$ given by

$$\begin{aligned} v_k &= \frac{1}{np_j}[f_{j_k}'(x^k) - f_{j_k}'(\phi_j^k)] + \frac{1}{n}\sum_{i=1}^{n} f_i'(\phi_i^k), \\ x^{k+1} &= x^k - \frac{1}{\eta} v_k, \\ \phi_j^{k+1} &= \begin{cases} f_{j_k}'(x^k) & \text{if } j = j_k, \\ \phi_j^k & \text{otherwise,} \end{cases} \end{aligned} \tag{B.1}$$

where $j_k = j$ with probability p_j. In this section we'll use the convention that $x = x^k$, that $\phi_j = \phi_j^k$, and that x^* is the minimizer of f. We first show that v_k is an

unbiased gradient estimator and derive a bound on its variance.

Lemma B.1. *We have* $\mathbb{E}[v_k] = f'(x^k)$ *and subsequently*

$$\mathbb{E}\|v_k\|^2 \leq 2\mathbb{E}\|\frac{1}{np_j}[f_j'(x) - f_j'(x^*)]\|^2 + 2\mathbb{E}\|\frac{1}{np_j}[f_j'(\phi_j) - f_j'(x^*)]\|^2.$$

Proof. We have

$$\begin{aligned}
\mathbb{E}[v_k] &= \sum_{j=1}^{n} p_{j=1}\left[\frac{1}{np_j}[f_j'(x) - f_{j_k}'(\phi_j)] + \frac{1}{n}\sum_{i=1}^{n} f_i'(\phi_i)\right] \\
&= \sum_{j=1}^{n}\left[\frac{1}{n}f_j'(x) - \frac{1}{n}f_j'(\phi_j) + \frac{p_j}{n}\sum_{i=1}^{n} f_i'(\phi_i)\right] \\
&= \frac{1}{n}\sum_{i=1}^{n} f_j'(x) - \frac{1}{n}\sum_{i=1}^{n} f_j'(\phi_j) + \sum_{i=1}^{n}[p_i]\frac{1}{n}\sum_{i=1}^{n} f_j'(\phi_j) \\
&= \frac{1}{n}\sum_{i=1}^{n} f_i'(x) = f'(x).
\end{aligned}$$

To show the second part, we use that $\mathbb{E}\|X - \mathbb{E}[X] + Y\|^2 = \mathbb{E}\|X - \mathbb{E}[X]\|^2 + \mathbb{E}\|Y\|^2$

if X and Y are independent, $\mathbb{E}\|X-\mathbb{E}[X]\|^2 \leq \mathbb{E}\|X\|^2$, and $\|x+y\|^2 \leq 2\|x\|^2+2\|y\|^2$,

$$\begin{aligned}
\mathbb{E}\|v_k\|^2 &= \mathbb{E}\|\frac{1}{np_j}[f_j'(x)-f_j'(\phi_j)]+\frac{1}{n}\sum_{i=1}^n f_i'(\phi_i)\|^2 \\
&= \mathbb{E}\|\frac{1}{np_j}[f_j'(x)-f_j'(x^*)]-f'(x)+f'(x) \\
&\quad -\frac{1}{np_j}[f_j'(\phi_j)-f_j'(x^*)]-\frac{1}{n}\sum_{i=1}^n f_i'(\phi_i))\|^2 \\
&= \mathbb{E}\|\frac{1}{np_j}[f_j'(x)-f_j'(x^*)]-f'(x) \\
&\quad -\frac{1}{np_j}[f_j'(\phi_j)-f_j'(x^*)]-\frac{1}{n}\sum_{i=1}^n f_i'(\phi_i))\|^2+\|f'(x)\|^2 \\
&\leq \mathbb{E}\|\frac{1}{np_j}[f_j'(x)-f_j'(x^*)]-f'(x)\|^2 \\
&\quad +2\mathbb{E}\|\frac{1}{np_j}[f_j'(\phi_j)-f_j'(x^*)]-\frac{1}{n}\sum_{i=1}^n f_i'(\phi_i))\|^2+\|f'(x)\|^2 \\
&\leq 2\mathbb{E}\|\frac{1}{np_j}[f_j'(x)-f_j'(x^*)]\|^2-2\|f'(x)\|^2 \\
&\quad +2\mathbb{E}\|\frac{1}{np_j}[f_j'(\phi_j)-f_j'(x^*)]\|^2+\|f'(x)\|^2 \\
&\leq 2\mathbb{E}\|\frac{1}{np_j}[f_j'(x)-f_j'(x^*)]\|^2+2\mathbb{E}\|\frac{1}{np_j}[f_j'(\phi_j)-f_j'(x^*)]\|^2.
\end{aligned}$$

□

We will also make use of the inequality

$$\langle f'(x), x^*-x\rangle \leq -\frac{\mu}{2}\|x-x^*\|^2-\frac{1}{2Ln}\sum_{i=1}^n \|f_i'(x^*)-f_i'(x)\|^2, \tag{B.2}$$

which follows from Defazio et al. (2014, Lemma 1) using that $f'(x^*)=0$ and the non-positivity of $\frac{L-\mu}{L}[f(x^*)-f(x)]$. We now give the proof of part (a) of Theorem 3.1, which we state below.

Theorem B.2 (a). *If* $\eta = \frac{4L+n\mu}{np_m}$ *and* $p_m = \min_j\{p_j\}$*, then Algorithm B.1 has*

$$\mathbb{E}[\|x^k - x^*\|_2^2] \leq \left(1 - \frac{np_m\mu}{n\mu+4L}\right)^t \left[\|x^0 - x^*\|_2^2 + \frac{2p_m}{(4L+n\mu)^2}\sum_i \frac{1}{p_i}\|\nabla f_i(x^0) - \nabla f_i(x^*)\|_2^2\right].$$

Proof. We denote the Lyapunov function T^k at iteration k by

$$T^k = \frac{1}{n}\sum_{i=1}^n \frac{1}{np_j}\|f_i'(\phi_i^k) - f_i'(x^*)\|^2 + c\|x^k - x^*\|^2.$$

We will will show that $\mathbb{E}[T^{k+1}] \leq (1 - \frac{1}{\kappa})T^k$ for some $\kappa < 1$. First, we write the expectation of the first term as

$$\begin{aligned}
&\mathbb{E}\left[\sum_i \frac{1}{n^2p_i}\|f_i'(\phi_i) - f_i'(x^*)\|^2\right] \\
&= \mathbb{E}\left[\frac{1}{n^2p_j}\|f_j'(x) - f_j'(x^*)\|^2\right] + \sum_i \frac{1}{n^2p_i}\|f_i'(\phi_i) - f_i'(x^*)\|^2 - E\left[\frac{1}{n^2p_j}\|f_j'(\phi_j) - f_j'(x^*)\|^2\right] \\
&= \frac{1}{n^2}\sum_i \|f_i'(x) - f_i'(x^*)\|^2 + \frac{1}{n^2}\sum_i \left(\frac{1}{p_i} - 1\right)\|f_i'(\phi_i) - f_i'(x^*)\|^2. \qquad \text{(B.3)}
\end{aligned}$$

Next, we simplify the other term of $\mathbb{E}[T^{k+1}]$,

$$\begin{aligned}
c\mathbb{E}\|x^{k+1} - x^*\|^2 &= c\mathbb{E}\|x - x^* - \frac{1}{\eta}v_k\|^2 \\
&= c\|x - x^*\|^2 + \frac{c}{\eta^2}\mathbb{E}\|v_k\|^2 + \frac{2c}{\eta}\langle f'(x), x - x^*\rangle
\end{aligned}$$

We now use Lemma B.1 followed by Inequality (B.2),

$$
\begin{aligned}
c\mathbb{E}\|x^{k+1}-x^*\|^2 &\leq c\|x-x^*\|^2 + \frac{c}{\eta^2}2\mathbb{E}\|\frac{1}{np_j}[f_j'(x)-f_j'(x^*)]\|^2 \\
&\quad + \frac{c}{\eta^2}2\mathbb{E}\|\frac{1}{np_j}[f_j'(\phi_j)-f_j'(x^*)]\|^2 + \frac{2c}{\eta}\langle f'(x), x-x^*\rangle \\
&\leq c(1-\frac{\mu}{\eta})\|x-x^*\|^2 + \frac{2c}{\eta^2}\mathbb{E}\|\frac{1}{np_j}(f_j'(x)-f_j'(x^*))\|^2 \\
&+ \frac{2c}{\eta^2}\mathbb{E}\|\frac{1}{np_j}(f_j'(\phi_j)-f_j'(x^*))\|^2 - \frac{c}{n\eta L}\sum_i \|f_i'(x^*)-f_i'(x)\|^2 \\
&= c(1-\frac{\mu}{\eta})\|x-x^*\|^2 + \sum_i(\frac{2c}{n^2\eta^2 p_i} - \frac{c}{n\eta L})\|f_i'(x)-f_i'(x^*)\|^2 \\
&\quad + \sum_i(\frac{2c}{n^2\eta^2 p_i})\|f_i'(\phi_i)-f_i'(x^*)\|^2.
\end{aligned}
$$

We use this to bound the expected improvement in the Lyapunov function,

$$
\begin{aligned}
\mathbb{E}[T^{k+1}] - T^k &= E[T^{k+1}] - \frac{1}{n}\sum_{i=1}^{n}\frac{1}{np_j}\|f_i'(\phi_i) - f_i'(x^*)\|^2 - c\|x-x^*\|^2 \\
&\leq \frac{1}{n^2}\sum_i \|f_i'(x) - f_i'(x^*)\|^2 + \frac{1}{n^2}\sum_i \left(\frac{1}{p_i} - 1\right)\|f_i'(\phi_i) - f_i'(x^*)\|^2 && \text{From (B.3)} \\
&\quad + c(1-\frac{\mu}{\eta})\|x-x^*\|^2 + \sum_i (\frac{2c}{n^2\eta^2 p_i} - \frac{c}{n\eta L})\|f_i'(x) - f_i'(x^*)\|^2 \\
&\quad + \sum_i (\frac{2c}{n^2\eta^2 p_i})\|f_i'(\phi_i) - f_i'(x^*)\|^2 && \text{From above} \\
&\quad - \frac{1}{n}\sum_{i=1}^{n}\frac{1}{np_j}\|f_i'(\phi_i) - f_i'(x^*)\|^2 - c\|x-x^*\|^2 && \text{Def'n of } T^k \\
&= \frac{1}{n^2}\sum_i \|f_i'(x) - f_i'(x^*)\|^2 - \frac{1}{n^2}\sum_i \|f_i'(\phi_i) - f_i'(x^*)\|^2 \\
&\quad - \frac{c\mu}{\eta}\|x-x^*\|^2 + \sum_i (\frac{2c}{n^2\eta^2 p_i} - \frac{c}{n\eta L})\|f_i'(x) - f_i'(x^*)\|^2 \\
&\quad + \sum_i (\frac{2c}{n^2\eta^2 p_i})\|f_i'(\phi_i) - f_i'(x^*)\|^2 \\
&= -\frac{1}{\kappa}T^k + \left(\frac{1}{\kappa} - \frac{\mu}{\eta}\right) c\|x - x*\|^2 && (*) \\
&\quad + \sum_i \left(\frac{2c}{n^2\eta^2 p_i} + \frac{1}{n^2} - \frac{c}{n\eta L}\right)\|f_i'(x) - f_i'(x^*)\|^2 \\
&\quad + \sum_i \left(\frac{2c}{n^2\eta^2 p_i} - \frac{1}{n^2} + \frac{1}{n^2\kappa p_i}\right)\|f_i'(\phi_i) - f_i'(x^*)\|^2 \\
&\leq -\frac{1}{\kappa}T^k + \left(\frac{1}{\kappa} - \frac{\mu}{\eta}\right)\left[c\|x - x*\|^2\right] \\
&\quad + \left(\frac{2c}{n^2\eta^2 p_m} + \frac{1}{n^2} - \frac{c}{n\eta L}\right)\left[\sum_i \|f_i'(x) - f_i'(x^*)\|^2\right] \\
&\quad + \left(\frac{2c}{n^2\eta^2 p_m} - \frac{1}{n^2} + \frac{1}{n^2\kappa p_m}\right)\left[\sum_i \|f_i'(\phi_i) - f_i'(x^*)\|^2\right],
\end{aligned}
$$

where in $(*)$ we add and subtract $\frac{1}{\kappa}T^k$ and in the last line we assumed $c \geq 0$ and used $p_i \geq p_m$. The terms in square brackets are positive, and if we can choose the

constants $\{c, \kappa, \eta\}$ to make the round brackets non-positive, we have

$$\mathbb{E}[T^{k+1}] \leq \left(1 - \frac{1}{\kappa}\right) T^k.$$

For the first expression, choosing $\kappa = \frac{\eta}{\mu}$ makes it zero. We can make the third expression zero under this choice of κ by choosing $c = \frac{\eta^2 p_m}{2} - \frac{\mu\eta}{2}$. This follows because

$$\frac{2c}{n^2\eta^2 p_m} - \frac{1}{n^2} + \frac{1}{n^2\kappa p_m} = \frac{2c}{n^2\eta^2 p_m} - \frac{1}{n^2} + \frac{\mu}{n^2\eta p_m} = 0,$$

is equivalent to

$$\frac{2c}{n^2\eta^2 p_m} = \frac{1}{n^2} - \frac{\mu}{n^2\eta p_m} \Leftrightarrow c = \frac{\eta^2 p_m}{2} - \frac{\mu\eta}{2}.$$

For the second expression, note that with our choice of c we have

$$\frac{2c}{n^2\eta^2 p_m} + \frac{1}{n^2} - \frac{c}{n\eta L} = \frac{1}{n^2} - \frac{\mu}{n^2\eta p_m} + \frac{1}{n^2} - \frac{\frac{\eta^2 p_m}{2} - \frac{\mu\eta}{2}}{n\eta L},$$

which (multiplying by n) is negative if we have

$$\frac{2}{n} + \frac{\mu}{2L} \leq \frac{\mu}{n\eta p_m} + \frac{\eta p_m}{2L}.$$

Ignoring the last term, we can choose

$$\eta = \frac{4L + n\mu}{np_m}.$$

We will also require that $c \geq 0$ to complete the proof, but this follows because $\eta \geq \frac{\mu}{p_m}$. By using that

$$c\mathbb{E}[\|x^{k+1} - x^*\|_2^2] \leq \mathbb{E}[T^{k+1}] \leq \left(1 - \frac{1}{\kappa}\right) T^k = \left(1 - \frac{\mu}{\eta}\right) T^k$$

and chaining the expectations while using the definition of η we obtain

$$\begin{aligned}\mathbb{E}[\|x^k - x^*\|_2^2] &\leq \left(1 - \frac{\mu}{\eta}\right)^k \frac{T^0}{c} \\ &= \left(1 - \frac{np_m\mu}{n\mu + 4L}\right)^k \left[\|x^0 - x^*\|^2 + \frac{1}{cn}\sum_{i=1}^n \frac{1}{np_j}\|f_i'(\phi_i^0) - f_i'(x^*)\|^2\right].\end{aligned}$$

To get the final expression, use that

$$\frac{1}{cn^2} = \frac{2}{n^2(\eta^2 p_m - \mu\eta)} \leq \frac{2}{n^2\eta^2 p_m} = \frac{2n^2 p_m^2}{n^2 p_m (4L + n\mu)^2} = \frac{2p_m}{(4L + n\mu)^2}.$$

□

B.2 Proof of Part (b) of Theorem 3.1

In this section we consider the minimization problem

$$\min_x f(x) = \frac{1}{n}\sum_{i=1}^n f_i(x),$$

where each f_i' is L_i-Lipschitz continuous and f is μ-strongly-convex. We will define Algorithm B.4 by the sequences $\{x^k\}$, $\{v_k\}$, and $\{\phi_j^k\}$ given by

$$\begin{aligned} v_k &= \frac{\bar{L}}{L_i}[f_{j_k}'(x^k) - f_{j_k}'(\phi_j^k)] + \frac{1}{n}\sum_{i=1}^n f_i'(\phi_i^k), \\ x^{k+1} &= x^k - \gamma v_k, \\ \phi_j^{k+1} &= \begin{cases} f_{r_k}'(x^k) & \text{if } j = r_k, \\ \phi_j^k & \text{otherwise,} \end{cases} \end{aligned} \tag{B.4}$$

where $j_k = j$ with probability $\frac{L_i}{\sum_{j=1}^n L_j}$ and r_k is picked uniformly at random. This is identical to Algorithm B.1, except it uses a specific choice of the p_j and the memory ϕ_j is updated based on a different random sample that is sampled uniformly. This algorithm maintains the key property that the expected step is a gradient step, $\mathbb{E}[v_k] = f'(x^k)$.

From our assumptions about f and the f_i, we have (Nesterov, 2004, see Chapter 2).

$$f_i(x) \geq f_i(y) + \left\langle f_i'(y), x-y \right\rangle + \frac{1}{2L} \left\| f_i'(x) - f_i'(y) \right\|^2, \tag{B.5}$$

and

$$f(x) \geq f(y) + \left\langle f'(y), x-y \right\rangle + \frac{\mu}{2} \left\| x-y \right\|^2. \tag{B.6}$$

We use these to derive several useful inequalities that we will use in the analysis. Adding the former times $\frac{1}{2n}$ for all i to the latter times $\frac{1}{2}$ for $y = x^*$ gives the inequality

$$\left\langle f'(x), x^* - x \right\rangle \leq f(x^*) - f(x) - \frac{\mu}{4} \left\| x^* - x \right\|^2 - \frac{1}{4n} \sum_i \frac{1}{L_i} \left\| f_i'(x^*) - f_i'(x) \right\|^2. \tag{B.7}$$

Also by applying (B.5) with $y = x^*$ and $x = \phi_i$, for each f_i and summing, we have that for all ϕ_i and x^*:

$$\frac{1}{n} \sum_i \frac{1}{L_i} \left\| f_i'(\phi_i) - f_i'(x^*) \right\|^2 \leq \frac{2}{n} \sum_i \left[f_i(\phi_i) - f(x^*) - \left\langle f_i'(x^*), \phi_i - x^* \right\rangle \right]. \tag{B.8}$$

Further, by both minimizing sides of (B.6) we obtain

$$- \left\| f'(x) \right\|^2 \leq -2\mu \left[f(x) - f(x^*) \right]. \tag{B.9}$$

We next derive a bound on the variance of the gradient estimate.

Lemma B.3. *It holds that for any ϕ_i that with x^{k+1} and x^k as given by Algorithm B.1 we have*

$$\begin{aligned} \mathbb{E} \left\| x^{k+1} - x^k \right\|^2 \quad &\leq \quad 2\gamma^2 \frac{\bar{L}}{n} \sum_i \frac{1}{L_i} \left\| f_j'(\phi_j^k) - f_j'(x^*) \right\|^2 \\ &\quad + 2\gamma^2 \frac{\bar{L}}{n} \sum_i \frac{1}{L_i} \left\| f_j'(x^k) - f_j'(x^*) \right\|^2 - \eta^2 \left\| f'(x^k) \right\|^2. \end{aligned}$$

Proof. We again follow the SAGA argument closely here

$$
\begin{aligned}
& \mathbb{E}\left\|x^{k+1}-x^k\right\|^2 \\
= \;& \gamma^2\mathbb{E}\left\|\frac{\bar{L}}{L_j}\left[f_j'(\phi_j^k)-f_j'(x^k)\right]-\frac{1}{n}\sum_{i=1}^n f_i'(\phi_i^k)\right\|^2 \\
= \;& \gamma^2\mathbb{E}\left\|\frac{\bar{L}}{L_j}\left[f_j'(\phi_j^k)-f_j'(x^*)\right]-\frac{1}{n}\sum_{i=1}^n f_i'(\phi_i^k)-\frac{\bar{L}}{L_j}\left[f_j'(x^k)-f_j'(x^*)\right]-f'(x^k)\right\|^2 \\
& +\eta^2\left\|f'(x^k)\right\|^2 \\
\leq \;& 2\gamma^2\mathbb{E}\left\|\frac{\bar{L}}{L_j}\left[f_j'(\phi_j^k)-f_j'(x^*)\right]-\frac{1}{n}\sum_{i=1}^n f_i'(\phi_i^k)\right\|^2 \\
& +2\gamma^2\mathbb{E}\left\|\frac{\bar{L}}{L_j}\left[f_j'(x^k)-f_j'(x^*)\right]-f'(x^k)\right\|^2+\eta^2\left\|f'(x^k)\right\|^2 \\
\leq \;& 2\gamma^2\mathbb{E}\left\|\frac{\bar{L}}{L_j}\left[f_j'(\phi_j^k)-f_j'(x^*)\right]\right\|^2 \\
& +2\gamma^2\mathbb{E}\left\|\frac{\bar{L}}{L_j}\left[f_j'(x^k)-f_j'(x^*)\right]\right\|^2-\eta^2\left\|f'(x^k)\right\|^2.
\end{aligned}
$$

We can expand those expectations as follows:

$$
\begin{aligned}
\mathbb{E}\left\|\frac{\bar{L}}{L_i}\left[f_j'(\phi_j^k)-f_j'(x^*)\right]\right\|^2 &= \frac{1}{n\bar{L}}\sum_i L_i\left\|\frac{\bar{L}}{L_i}\left[f_j'(\phi_j^k)-f_j'(x^*)\right]\right\|^2 \\
&= \frac{\bar{L}}{n}\sum_i\frac{1}{L_i}\left\|\left[f_j'(\phi_j^k)-f_j'(x^*)\right]\right\|^2,
\end{aligned}
$$

and similarly for $\mathbb{E}\left\|\frac{\bar{L}}{L_i}\left[f_j'(x^k)-f_j'(x^*)\right]\right\|^2$. □

We now give the proof of part (b) of Theorem 1, which we state below.

Theorem B.4 (b). *If $\gamma=\frac{1}{4L}$, then Algorithm B.4 has*

$$
E\left[\left\|x^k-x^*\right\|^2\right]\leq\left(1-\min\left\{\frac{1}{3n},\frac{\mu}{8\bar{L}}\right\}\right)^k\left[\left\|x^k-x^*\right\|^2+\frac{n}{2\bar{L}}\left(f(x^0)-f(x^*)\right)\right].
$$

Proof. We define the Lyapunov function as

$$T^k = \frac{1}{n}\sum_i f_i(\phi_i^k) - f(x^*) - \frac{1}{n}\sum_i \left\langle f_i'(x^*), \phi_i^k - x^* \right\rangle + c\left\|x^k - x^*\right\|^2.$$

The expectations of the first terms in T^{k+1} are straightforward to simplify:

$$\begin{aligned}
\mathbb{E}\left[\frac{1}{n}\sum_i f_i(\phi_i^{k+1})\right] &= \frac{1}{n}f(x^k) + \left(1-\frac{1}{n}\right)\frac{1}{n}\sum_i f_i(\phi_i^k), \\
\mathbb{E}\left[-\frac{1}{n}\sum_i \left\langle f_i'(x^*), \phi_i^{k+1} - x^* \right\rangle\right] &= -\left(1-\frac{1}{n}\right)\frac{1}{n}\sum_i \left\langle f_i'(x^*), \phi_i^k - x^* \right\rangle.
\end{aligned}$$

Note that these terms make use of the uniformly sampled $\phi_r^{k+1} = x^k$ value. For the change in the last term of T^k we expand the quadratic and apply $\mathbb{E}[x^{k+1}] = x^k - \eta f'(x^k)$ to simplify the inner product term:

$$\begin{aligned}
& c\mathbb{E}\left\|x^{k+1} - x^*\right\|^2 \\
=& c\mathbb{E}\left\|x^k - x^* + x^{k+1} - x^k\right\|^2 \\
=& c\left\|x^k - x^*\right\|^2 + 2c\mathbb{E}\left[\left\langle x^{k+1} - x^k, x^k - x^* \right\rangle\right] + c\mathbb{E}\left\|x^{k+1} - x^k\right\|^2 \\
=& c\left\|x^k - x^*\right\|^2 - 2c\eta\left\langle f'(x^k), x^k - x^* \right\rangle + c\mathbb{E}\left\|x^{k+1} - x^k\right\|^2.
\end{aligned}$$

We now apply Lemma B.3 to bound the error term $c\mathbb{E}\left\|x^{k+1} - x^k\right\|^2$, giving:

$$\begin{aligned}
& c\mathbb{E}\left\|x^{k+1} - x^*\right\|^2 \\
\leq\ & c\left\|x^k - x^*\right\|^2 - c\eta^2\left\|f'(x^k)\right\|^2 \\
& -2c\eta\left\langle f'(x^k), x^k - x^* \right\rangle \\
& +2c\gamma^2\frac{\bar{L}}{n}\sum_i \frac{1}{L_i}\left\|f_i'(\phi_i^k) - f_i'(x^*)\right\|^2 + 2c\gamma^2\frac{\bar{L}}{n}\sum_i \frac{1}{L_i}\left\|f_i'(x^k) - f_i'(x^*)\right\|^2.
\end{aligned}$$

Now we bound $-2c\gamma\langle f'(x), x - x^*\rangle$ with (B.7) and then apply (B.8) to bound

$\mathbb{E}\left\|f_j'(\phi_j) - f_j'(x^*)\right\|^2$:

$$
\begin{aligned}
c\mathbb{E}\left\|x^{k+1} - x^*\right\|^2 \leq & \left(c - \frac{1}{2}c\eta\mu\right)\left\|x^k - x^*\right\|^2 \\
& + \left(2c\eta^2\bar{L} - \frac{1}{2}c\gamma\right)\frac{1}{n}\sum_i \frac{1}{L_i}\left\|f_i'(x^k) - f_i'(x^*)\right\|^2 - c\eta^2\left\|f'(x^k)\right\|^2 \\
& - 2c\eta\left[f(x^k) - f(x^*)\right] \\
& + \left(4c\eta^2\bar{L}\right)\frac{1}{n}\sum_i \left[f_i(\phi_i) - f_i(x^*) - \left\langle f_i'(x^*), \phi_i - x^*\right\rangle\right].
\end{aligned}
$$

We can now combine the bounds we have derived for each term in T, and pull out a fraction $\frac{1}{\kappa}$ of T^k (for any κ at this point). Together with (B.9) this yields:

$$
\begin{aligned}
\mathbb{E}[T^{k+1}] - T^k \leq & -\frac{1}{\kappa}T^k + \left(\frac{1}{n} - 2c\eta - 2c\eta^2\mu\right)\left[f(x^k) - f(x^*)\right] \\
& + \left(\frac{1}{\kappa} + 4c\eta^2\bar{L} - \frac{1}{n}\right)\left[\frac{1}{n}\sum_i f_i(\phi_i^k) - f(x^*) - \frac{1}{n}\sum_i \left\langle f_i'(x^*), \phi_i^k - x^*\right\rangle\right] \\
& + \left(\frac{1}{\kappa} - \frac{1}{2}\eta\mu\right)c\left\|x^k - x^*\right\|^2 + \left(2\eta\bar{L} - \frac{1}{2}\right)c\eta\frac{1}{n}\sum_i \frac{1}{L_i}\left\|f_i'(x^k) - f_i'(x^*)\right\|^2.
\end{aligned}
\tag{B.10}
$$

Note that the term in square brackets in the second row is positive in light of (B.8). We now attempt to find constants that satisfy the required relations. We start with naming the constants that we need to be non-positive:

$$
\begin{aligned}
c_1 &= \frac{1}{n} - 2c\eta - 2c\eta^2\mu, \\
c_2 &= \frac{1}{\kappa} + 4c\eta^2\bar{L} - \frac{1}{n}, \\
c_3 &= \frac{1}{\kappa} - \frac{1}{2}\eta\mu, \\
c_4 &= 2\eta\bar{L} - \frac{1}{2}.
\end{aligned}
$$

Recall that we are using the step size $\gamma = 1/4\bar{L}$, and thus $c_4 = 0$. Setting c_1 to zero

gives

$$c = \frac{1}{2\gamma(1-\gamma\mu)n},$$

which is positive since $\gamma\mu < 1$. Now we look at the restriction that $c_2 \leq 0$ places on κ:

$$\begin{aligned}
\frac{1}{\kappa} + 4c\eta^2\bar{L} - \frac{1}{n} &= \frac{1}{\kappa} + \frac{4\gamma\bar{L}}{2(1-\gamma\mu)n} - \frac{1}{n} \\
&= \frac{1}{\kappa} + \frac{1}{2(1-\gamma\mu)n} - \frac{1}{n} \\
&= \frac{1}{\kappa} + \frac{1}{2(1-\mu/4\bar{L})n} - \frac{1}{n} \\
&\leq \frac{1}{\kappa} + \frac{1}{2(1-\bar{L}/4\bar{L})n} - \frac{1}{n} \\
&= \frac{1}{\kappa} + \frac{2}{3n} - \frac{1}{n} \\
&= \frac{1}{\kappa} - \frac{1}{3n},
\end{aligned}$$

$$\therefore \frac{1}{\kappa} \leq \frac{1}{3n}.$$

We also have the restriction from $c_3 = \frac{1}{\kappa} - \frac{1}{2}\eta\mu$ of

$$\frac{1}{\kappa} \leq \frac{\mu}{8\bar{L}},$$

therefore we can take

$$\frac{1}{\kappa} = \min\left\{\frac{1}{3n}, \frac{\mu}{8\bar{L}}\right\}.$$

Note that $c\left\|x^k - x^*\right\|^2 \leq T^k$, and therefore by chaining expectations and plugging in constants we get:

$$E\left[\left\|x^k - x^*\right\|^2\right] \leq \left(1 - \min\left\{\frac{1}{3n}, \frac{\mu}{8\bar{L}}\right\}\right)^k \left[\left\|x^k - x^*\right\|^2 + \frac{n}{2\bar{L}}\left(f(x^0) - f(x^*)\right)\right].$$

□

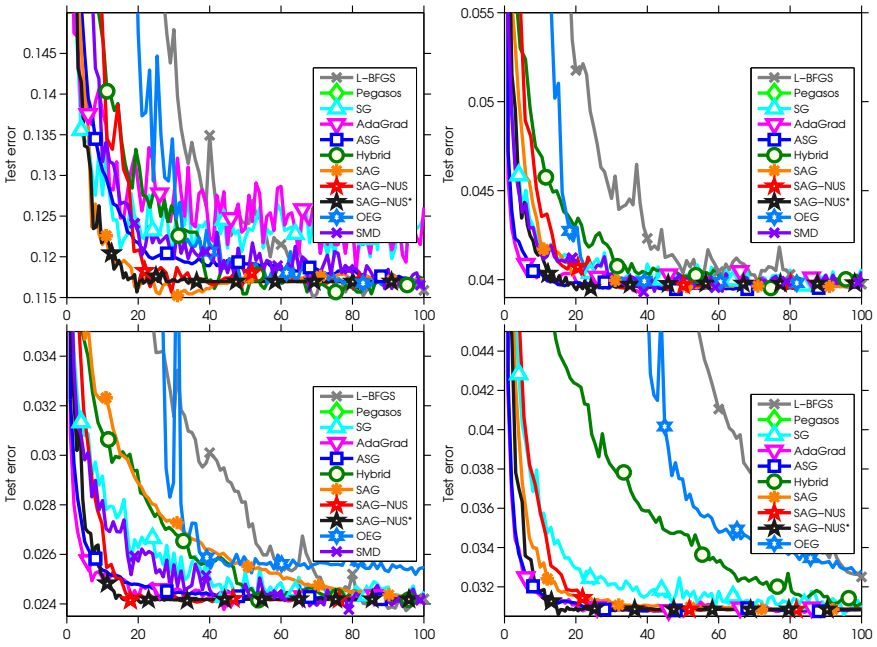

Figure B.1: Test error against effective number of passes for different deterministic, stochastic, and semi-stochastic optimization strategies (this figure is best viewed in colour). Top-left: OCR, Top-right: CoNLL-2000, bottom-left: CoNLL-2002, bottom-right: POS-WSJ.

B.3 Test Error Plots for All Methods

In the main body we only plotted test error for a subset of the methods. In Figure B.1 we plot the test error of all methods considered in Figure 1. Note that Pegasos does not appear on the plot (despite being in the legend) because its values exceed the maximum plotted values. In these plots we see that the SAG-NUS methods perform similarly to the best among the optimally-tuned stochastic gradient methods in terms of test error, despite the lack of tuning required to apply these methods.

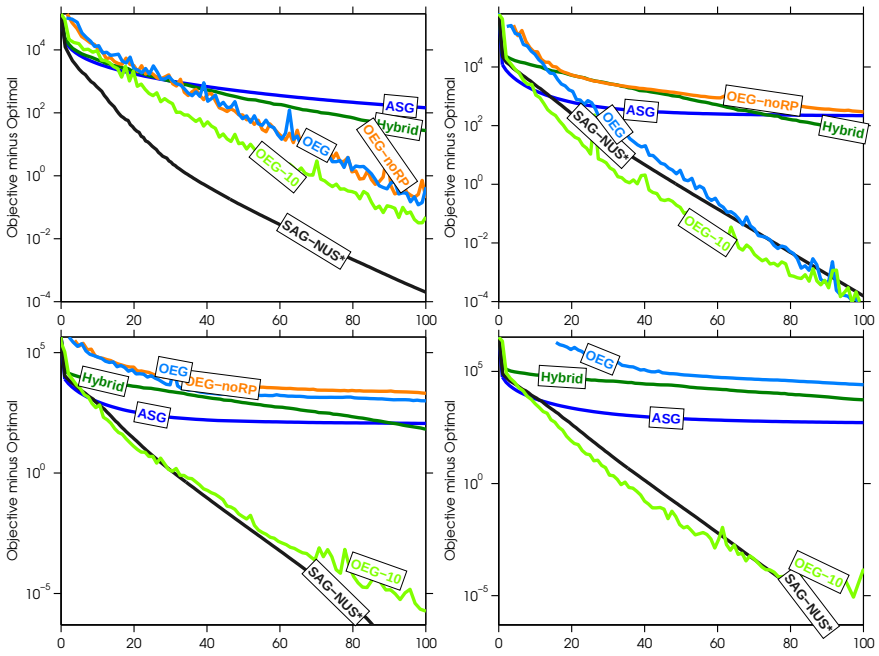

Figure B.2: Objective minus optimal objective value against effective number of passes for different variants of OEG. Top-left: OCR, Top-right: CoNLL-2000, bottom-left: CoNLL-2002, bottom-right: POS-WSJ.

B.4 Improved Results for OEG

Owing to the high variance of the performance of the OEG method, we explored whether better performance could be obtained with the OEG method. The two most salient observations from these experiments where that (i) utilizing a random permutation on the first pass through the data seems to be crucial to performance, and (ii) that better performance could be obtained on the two datasets where OEG performed poorly by using a different initialization. In particular, better performance could be obtained by initializing the parts with the correct labels to a larger value, such as 10. In Figure B.2, we plot the performance of the OEG method without using the random permutation (*OEG-noRP*) as well as OEG with this initialization (*OEG-10*). Removing the random permutation makes OEG perform much worse on one of the datasets, while using the different initialization makes OEG perform nearly as well as SAG-NUS* on the datasets where previously it performed poorly

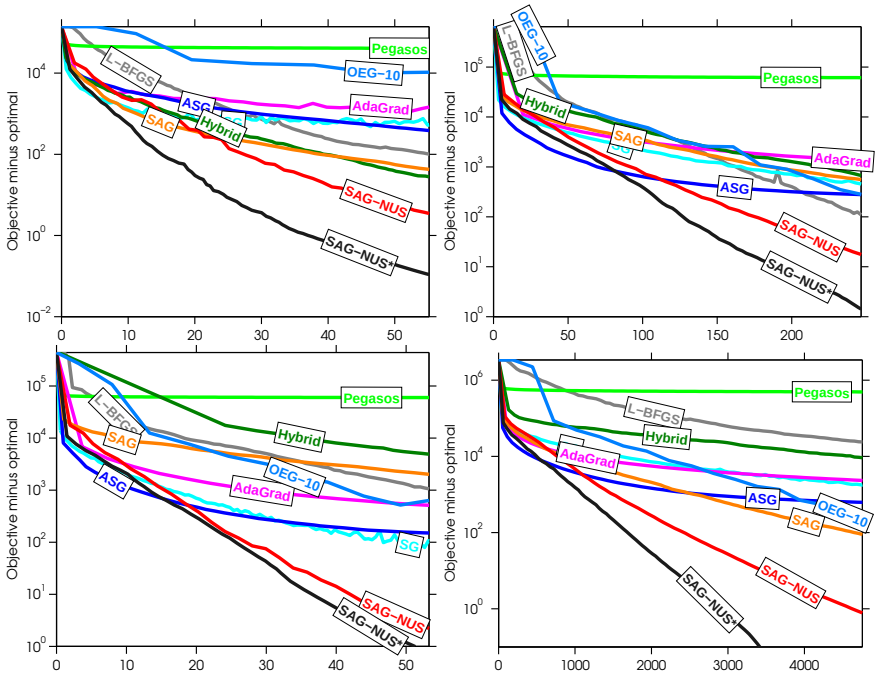

Figure B.3: Objective minus optimal objective value against time for different deterministic, stochastic, and semi-stochastic optimization strategies. Top-left: OCR, Top-right: CoNLL-2000, bottom-left: CoNLL-2002, bottom-right: POS-WSJ.

(although it does not make up the performance gap on the remaining data set). Performance did not further improve by using even larger values in the initialization, and using a value that was too large lead to numerical problems.

B.5 Runtime Plots

In the main body we plot the performance against the effective number of passes as an implementation-independent way of comparing the different algorithms. In all cases except SMD, we implemented a C version of the method and also compared the running times of our different implementations. This ties the results to the hardware used to perform the experiments and to our specific implementation, and thus says little about the runtime in different hardware settings or different implemen-

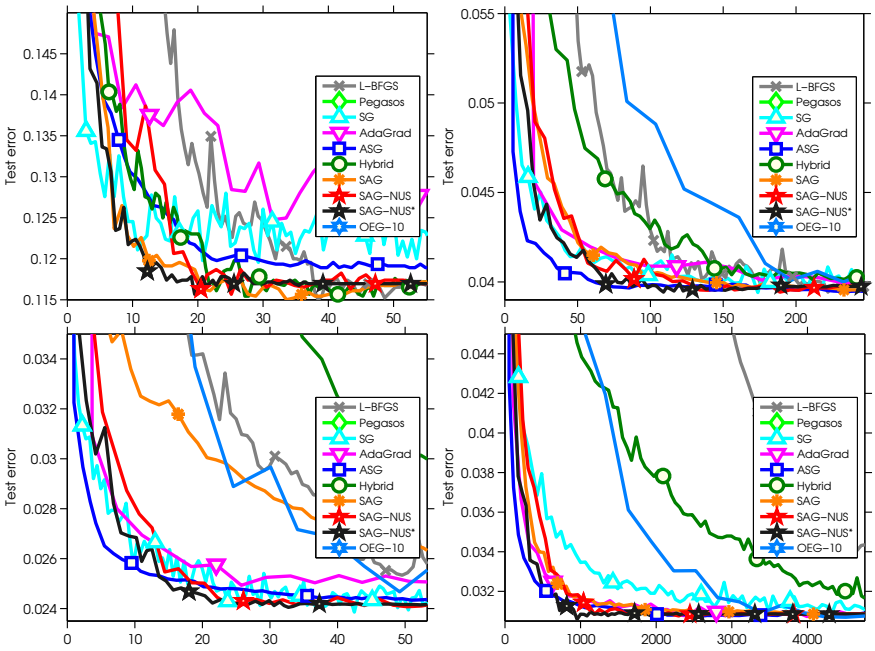

Figure B.4: Test error against time for different deterministic, stochastic, and semi-stochastic optimization strategies. Top-left: OCR, Top-right: CoNLL-2000, bottom-left: CoNLL-2002, bottom-right: POS-WSJ.

tations, but does show the practical performance of the methods in this particular setting. We plot the training objective against runtime in Figure B.3 and the test error in Figure B.4. In general, the runtime plots show the exact same trends as the plots against the effective number of passes. However, we note several small differences:

- *AdaGrad* performs slightly worse in terms of runtime. This seems to be due to the extra square root operators needed to implement the method.

- *Hybrid* performs worse in terms of runtime, although it was still faster than the *L-BFGS* method. This seems to be due to the higher relative cost of applying the L-BFGS update when the batch size is small.

- *OEG* performed much worse in terms of runtime, even with the better initialization from the previous section.

Finally, we note that these implementations are available on the first author's web-page:
http://www.cs.ubc.ca/~schmidtm/Software/SAG4CRF.html

Appendix C

Chapter 4 Supplementary Material

C.1 Proof of Theorem 4.2

First we restate Theorem 4.2 from Chapter 4.

Theorem C.1. *If each f_i is geodesically L-smooth and f is geodesically μ-strongly convex over the Riemannian manifold $\mathscr{M}$, the MASAGA algorithm with the constant step size $\eta = \frac{2\mu+\sqrt{\mu^2-8\rho(1+\alpha)\zeta L^2}}{4(1+\alpha)\zeta L^2}$ converges linearly while satisfying the following:*

$$\mathbb{E}\left[\mathrm{d}^2(x_t, x^*)\right] \leq (1-\rho)^t \Upsilon^0,$$

where $\rho = \min\{\frac{\mu^2}{8(1+\alpha)\zeta L^2}, \frac{1}{n} - \frac{1}{\alpha n}\}$, $\Upsilon^0 = 2\alpha\zeta\eta^2 \sum_{i=1}^n \|M^0[i] - \Gamma_{x^*}^{x_0}[\nabla f_i(x^*)]\|_2^2 + \mathrm{d}^2(x_0, x^*)$ is a positive scalar, and $\alpha > 1$ is a constant.

Proof. Let $\delta_t = \mathrm{d}^2(x_t, x^*)$. First we find an upper-bound for $\mathbb{E}\left[\|v_t\|_2^2\right]$.

$$\begin{aligned}
\mathbb{E}\left[\|v_t\|_2^2\right] &= \mathbb{E}\left[\|\nabla f_{i_t}(x_t) - \Gamma_{x_0}^{x_t}\left[M^t[i_t] - \hat{\mu}\right]\|_2^2\right] \\
&= \mathbb{E}\left[\|\nabla f_{i_t}(x_t) - \Gamma_{x^*}^{x_t}\left[\nabla f_{i_t}(x^*)\right] - \Gamma_{x_0}^{x_t}\left[M^t[i_t] - \Gamma_{x^*}^{x_0}\left[\nabla f_{i_t}(x^*)\right] - \hat{\mu}\right]\|_2^2\right] \\
&\leq 2\mathbb{E}\left[\|\nabla f_{i_t}(x_t) - \Gamma_{x^*}^{x_t}\left[\nabla f_{i_t}(x^*)\right]\|_2^2\right] \\
&+ 2\mathbb{E}\left[\|\Gamma_{x_0}^{x_t}\left[M^t[i_t] - \Gamma_{x^*}^{x_0}\left[\nabla f_{i_t}(x^*)\right] - \hat{\mu}\right]\|_2^2\right] \\
&\leq 2\mathbb{E}\left[\|\nabla f_{i_t}(x_t) - \Gamma_{x^*}^{x_t}\left[\nabla f_{i_t}(x^*)\right]\|_2^2\right] \\
&+ 2\mathbb{E}\left[\|M^t[i_t] - \Gamma_{x^*}^{x_0}\left[\nabla f_{i_t}(x^*)\right]\|_2^2\right] \\
&\leq 2L^2\delta_t + 2\mathbb{E}\left[\|M^t[i_t] - \Gamma_{x^*}^{x_0}\left[\nabla f_{i_t}(x^*)\right]\|_2^2\right].
\end{aligned}$$

The first inequality is due to $(a+b)^2 \leq 2a^2 + 2b^2$ and the second one is from the variance upper-bound inequality, in other words $\mathbb{E}\left[x^2 - \mathbb{E}\left[x\right]^2\right] \leq \mathbb{E}\left[x^2\right]$. The last inequality comes from the geodesic Lipschitz smoothness of each f_i. Note that the expectation is taken with respect to i_t.

$$\begin{aligned}
\mathbb{E}\left[\delta_{t+1}\right] &\leq \mathbb{E}\left[\delta_t - 2\left\langle v_t, \mathrm{Exp}_{x_t}^{-1}(-x^*)\right\rangle + \zeta\eta^2\|v_t\|_2^2\right] \\
&= \delta_t - 2\eta\left\langle \nabla f(x_t), \mathrm{Exp}_{x_t}^{-1}(-x^*)\right\rangle + \zeta\eta^2\mathbb{E}\left[\|v_t\|_2^2\right] \\
&\leq \delta_t - \eta\mu\delta_t + \zeta\eta^2\mathbb{E}\left[\|v_t\|_2^2\right] \\
&\leq (1-\mu\eta)\delta_t + \zeta\eta^2\left[2L^2\delta_t + 2\mathbb{E}\left[\|M^t[i_t] - \Gamma_{x^*}^{x_0}\left[\nabla f_{i_t}(x^*)\right]\|_2^2\right]\right] \\
&= (1 - \mu\eta + 2\zeta L^2\eta^2)\delta_t + 2\zeta\eta^2\Psi_t.
\end{aligned}$$

The first inequality is due to the trigonometric distance bound, the second one is due to the strong convexity of f, and the last one is due to the upper-bound of v_t. Ψ_t is defined as follows:

$$\Psi_t = \frac{1}{n}\sum_{i=1}^{n}\|M^t[i] - \Gamma_{x^*}^{x_0}\left[\nabla f_i(x^*)\right]\|_2^2.$$

We define the Lyaponov function

$$\Upsilon^t = \delta_t + c\Psi_t,$$

for some $c > 0$. Note that $\Upsilon^t \geq 0$, since both δ_t and Ψ_t are positive or zero. Next

we find an upper-bound for $\mathbb{E}[\Psi_{t+1}]$.

$$\begin{aligned}
\mathbb{E}[\Psi_{t+1}] &= \frac{1}{n}(\frac{1}{n}\sum_{i=1}^{n}\|\nabla f_i(x_t) - \Gamma_{x^*}^{x_t}[\nabla f_i(x^*)]\|_2^2) \\
&\quad + (1-\frac{1}{n})(\frac{1}{n}\sum_{i=1}^{n}\|M^t[i] - \Gamma_{x^*}^{x_0}[\nabla f_i(x^*)]\|_2^2) \\
&= \frac{1}{n}(\frac{1}{n}\sum_{i=1}^{n}\|\nabla f_i(x_t) - \Gamma_{x^*}^{x_t}[\nabla f_i(x^*)]\|_2^2) + (1-\frac{1}{n})\Psi_t \\
&\leq \frac{L^2}{n}\delta_t + (1-\frac{1}{n})\Psi_t.
\end{aligned}$$

The inequality is due to the geodesic Lipschitz smoothness of f_i. Then, for some positive $\rho \leq 1$ we have the following inequality:

$$\begin{aligned}
\mathbb{E}\left[\Upsilon^{t+1}\right] - (1-\rho)\Upsilon^t \leq (1-\mu\eta+2\zeta L^2\eta^2 - (1-\rho) + \frac{cL^2}{n})\delta_t \\
+ (2\zeta\eta^2 - c(1-\rho) + c(1-\frac{1}{n}))\Psi_t. \qquad \text{(C.1)}
\end{aligned}$$

In the right hand side of Inequality C.1, δ_t and Ψ_t are positive by construction. If the coefficients of δ_t and Ψ_t in the right hand side of the Inequality C.1 are negative, we would have $\mathbb{E}\left[\Upsilon^{t+1}\right] \leq (1-\rho)\Upsilon^t$. More precisely, we require

$$2\zeta\eta^2 - c(1-\rho) + c(1-\frac{1}{n}) \leq 0, \qquad \text{(C.2)}$$

$$1-\mu\eta+2\zeta L^2\eta^2 - (1-\rho) + \frac{cL^2}{n} \leq 0. \qquad \text{(C.3)}$$

To satisfy Inequality C.2 we require $\rho \leq \frac{1}{n} - \frac{2\zeta\eta^2}{c}$. If we set $c = 2\alpha n\zeta\eta^2$ for some $\alpha > 1$, then $\rho \leq \frac{1}{n} - \frac{1}{\alpha n}$, which satisfies our requirement. If we replace the value of c in Inequality C.3, we will get:

$$\begin{aligned}
&\rho - \mu\eta + 2\zeta L^2\eta^2 + 2\alpha\zeta L^2\eta^2 \leq 0, \\
&\eta \in (\eta^- = \frac{2\mu - \sqrt{\mu^2 - 8\rho(1+\alpha)\zeta L^2}}{4(1+\alpha)\zeta L^2}, \eta^+ = \frac{2\mu + \sqrt{\mu^2 - 8\rho(1+\alpha)\zeta L^2}}{4(1+\alpha)\zeta L^2}).
\end{aligned}$$

To ensure the term under the square root is positive, we also need $\rho < \frac{\mu^2}{8(1+\alpha)\zeta L^2}$.

Finally, if we set $\rho = \min\{\frac{\mu^2}{8(1+\alpha)\zeta L^2}, \frac{1}{n} - \frac{1}{\alpha n}\}$ and $\eta = \eta^+$, then we have:

$$\mathbb{E}\left[\Upsilon^{t+1}\right] \leq (1-\rho)^{t+1}\Upsilon^0,$$

where Υ^0 is a scalar. Since $\Psi_t > 0$ and $\mathbb{E}\left[\delta_{t+1}\right] \leq \mathbb{E}\left[\Upsilon^{t+1}\right]$, we get the required bound:

$$\mathbb{E}\left[\delta_{t+1}\right] \leq (1-\rho)^{t+1}\Upsilon^0.$$

□

C.2 Proof of Theorem 4.4

First we restate Theorem 4.4 from Chapter 4.

Theorem C.2. *When each f_i is geodesically L_i-smooth and f is geodesically μ-strongly convex on the manifold $\mathscr{M}$, the MASAGA algorithm with the defined non-uniform sampling scheme and the constant step size $\eta = \frac{2\mu + \sqrt{\mu^2 - 8\rho(\bar{L}+\alpha L)\frac{\zeta}{\gamma}\bar{L}}}{4(\bar{L}+\alpha L)\frac{\zeta}{\gamma}\bar{L}}$ converges linearly as follows:*

$$\mathbb{E}\left[\mathrm{d}^2(x_t, x^*)\right] \leq (1-\rho)^t\Upsilon^0,$$

where $\rho = \min\{\frac{\gamma\mu^2}{8(1+\alpha)\zeta L\bar{L}}, \frac{\gamma}{n} - \frac{\gamma}{\alpha n}\}$, $\gamma = \frac{\min\{L_i\}}{\bar{L}}$, $L = \max\{L_i\}$, $\bar{L} = \frac{1}{n}\sum_{i=1}^n L_i$, and $\alpha > 1$ is a constant, and $\Upsilon^0 = \frac{2\alpha\zeta\eta^2}{\gamma}\sum_{i=1}^n \frac{\bar{L}}{L_i}\|M^0[i] - \Gamma_{x^*}^{x_0}\left[\nabla f_i(x^*)\right]\|_2^2 + \mathrm{d}^2(x_0, x^*)$ are positive scalars.

Proof. Let $\delta_t = \mathrm{d}^2(x_t, x^*)$. First let us find an upper bound for $\mathbb{E}\left[\|\frac{\bar{L}}{L_{i_t}} v_t\|_2^2\right]$:

$$
\begin{aligned}
\mathbb{E}\left[\|\frac{\bar{L}}{L_{i_t}} v_t\|_2^2\right] &= \mathbb{E}\left[(\frac{\bar{L}}{L_{i_t}})^2 \|\nabla f_{i_t}(x_t) - \Gamma_{x_0}^{x_t}\left[M^t[i_t] - \hat{\mu}\right]\|_2^2\right] \\
&= \mathbb{E}[(\frac{\bar{L}}{L_{i_t}})^2 \|\nabla f_{i_t}(x_t) - \Gamma_{x^*}^{x_t}[\nabla f_{i_t}(x^*)] \\
&\qquad - \Gamma_{x_0}^{x_t}\left[M^t[i_t] - \Gamma_{x^*}^{x_0}[\nabla f_{i_t}(x^*)] - \hat{\mu}\right]\|_2^2] \\
&\leq 2\mathbb{E}\left[(\frac{\bar{L}}{L_{i_t}})^2 \|\nabla f_{i_t}(x_t) - \Gamma_{x^*}^{x_t}[\nabla f_{i_t}(x^*)]\|_2^2\right] \\
&+ 2\mathbb{E}\left[\|(\frac{\bar{L}}{L_{i_t}})(\Gamma_{x_0}^{x_t}\left[M^t[i_t] - \Gamma_{x^*}^{x_0}[\nabla f_{i_t}(x^*)] - \hat{\mu}\right])\|_2^2\right] \\
&\leq 2\mathbb{E}\left[(\frac{\bar{L}}{L_{i_t}})^2 \|\nabla f_{i_t}(x_t) - \Gamma_{x^*}^{x_t}[\nabla f_{i_t}(x^*)]\|_2^2\right] \\
&+ 2\mathbb{E}\left[(\frac{\bar{L}}{L_{i_t}})^2 \|M^t[i_t] - \Gamma_{x^*}^{x_0}[\nabla f_{i_t}(x^*)]\|_2^2\right] \\
&\leq 2\mathbb{E}\left[(\frac{\bar{L}}{L_{i_t}})^2 L_{i_t}^2 \delta_t\right] + 2\mathbb{E}\left[(\frac{\bar{L}}{L_{i_t}})^2 \|M^t[i_t] - \Gamma_{x^*}^{x_0}[\nabla f_{i_t}(x^*)]\|_2^2\right] \\
&= 2\bar{L}^2\delta_t + 2\mathbb{E}\left[(\frac{\bar{L}}{L_{i_t}})^2 \|M^t[i_t] - \Gamma_{x^*}^{x_0}[\nabla f_{i_t}(x^*)]\|_2^2\right] \\
&= 2\bar{L}^2\delta_t + 2\frac{1}{n}\sum_{i=1}^{n}(\frac{\bar{L}}{L_i})\|M^t[i_t] - \Gamma_{x^*}^{x_0}[\nabla f_{i_t}(x^*)]\|_2^2.
\end{aligned}
$$

The first inequality is due to $(a+b)^2 \leq 2a^2 + 2b^2$ and the second one is from the variance upper-bound inequality, i.e., $\mathbb{E}\left[x^2 - \mathbb{E}[x]^2\right] \leq \mathbb{E}\left[x^2\right]$. The last inequality comes from the Lipschitz smoothness of each f_i. The last equality is due to fact that we sample each f_i with probability $\frac{L_i}{n\bar{L}}$.

$$
\begin{aligned}
\mathbb{E}\left[\delta_{t+1}\right] &\leq \mathbb{E}\left[\delta_t - 2\left\langle \frac{\bar{L}}{L_{i_t}} v_t, \mathrm{Exp}_{x_t}^{-1}\left(-x^*\right)\right\rangle + \zeta\eta^2 \| \frac{\bar{L}}{L_{i_t}} v_t \|_2^2\right] \\
&= \delta_t - 2\eta\left\langle \nabla f(x_t), \mathrm{Exp}_{x_t}^{-1}\left(-x^*\right)\right\rangle + \zeta\eta^2 \mathbb{E}\left[\| \frac{\bar{L}}{L_{i_t}} v_t \|_2^2\right] \\
&\leq \delta_t - \eta\mu\delta_t + \zeta\eta^2 \mathbb{E}\left[\| \frac{\bar{L}}{L_{i_t}} v_t \|_2^2\right] \\
&\leq (1-\mu\eta)\delta_t + \zeta\eta^2\left[2\bar{L}^2\delta_t + 2\frac{1}{n}\sum_{i=1}^{n}(\frac{\bar{L}}{L_i})\| M^t[i_t] - \Gamma_{x^*}^{x_0}\left[\nabla f_i(x^*)\right]\|_2^2\right] \\
&= (1-\mu\eta+2\zeta\bar{L}^2\eta^2)\delta_t + 2\zeta\eta^2\Psi_t,
\end{aligned}
$$

where the first inequality is due to the trigonometric distance bound, the second one is due to strong convexity of f, and the last one is due to upper-bound of v_t. Let

$$
\Psi_t = \frac{1}{n}\sum_{i=1}^{n}(\frac{\bar{L}}{L_i})\| M^t[i] - \Gamma_{x^*}^{x_0}\left[\nabla f_i(x^*)\right]\|_2^2.
$$

Now let us define the Lyaponov function:

$$
\Upsilon^t = \delta_t + c\,\Psi_t,
$$

for some $c > 0$. Next we have to find an upper bound for $\mathbb{E}\left[\Psi_{t+1}\right]$.

$$
\begin{aligned}
\mathbb{E}\left[\Psi_{t+1}\right] &= \frac{1}{n}\left(\sum_{i=1}^{n}\frac{L_i}{n\bar{L}}(\frac{\bar{L}}{L_i})\| \nabla f_i(x_t) - \Gamma_{x^*}^{x_t}\left[\nabla f_i(x^*)\right]\|_2^2\right) \\
&\quad + \left(\frac{1}{n}\sum_{i=1}^{n}(1-\frac{L_i}{n\bar{L}})\| M^t[i_t] - \Gamma_{x^*}^{x_0}\left[\nabla f_i(x^*)\right]\|_2^2\right) \\
&= \frac{1}{n}\left(\frac{1}{n}\sum_{i=1}^{n}\| \nabla f_i(x_t) - \Gamma_{x^*}^{x_t}\left[\nabla f_i(x^*)\right]\|_2^2\right) + (1-\frac{\gamma}{n})\Psi_t \\
&\leq \frac{1}{n}\left(\frac{1}{n}\sum_{i=1}^{n}L_i^2\delta_t\right) + (1-\frac{\gamma}{n})\Psi_t \\
&\leq \frac{L\bar{L}}{n}\delta_t + (1-\frac{\gamma}{n})\Psi_t,
\end{aligned}
$$

where $\gamma = \frac{\min\{L_i\}}{\bar{L}}$. The first inequality is due to geodesic Lipschitz smoothness of f_i. Then for some positive $\rho \leq 1$ we have the following inequality

$$\begin{aligned}\mathbb{E}\left[\Upsilon^{t+1}\right] - (1-\rho)\Upsilon^t \leq (1 - \mu\eta + 2\zeta\bar{L}^2\eta^2 - (1-\rho) + \frac{cL\bar{L}}{n})\delta_t \\ + (2\zeta\eta^2 - c(1-\rho) + c(1-\frac{\gamma}{n}))\Psi_t. \end{aligned} \tag{C.4}$$

So if the coefficients of δ_t and Ψ_t in the right hand side of Inequality C.4 are negative, we have $\mathbb{E}\left[\Upsilon^{t+1}\right] \leq (1-\rho)\Upsilon^t$. More precisely, we need

$$2\zeta\eta^2 - c(1-\rho) + c(1-\frac{\gamma}{n}) \leq 0, \tag{C.5}$$

$$1 - \mu\eta + 2\zeta\bar{L}^2\eta^2 - (1-\rho) + \frac{cL\bar{L}}{n} \leq 0. \tag{C.6}$$

To satisfy C.5, we need

$$\begin{aligned}2\zeta\eta^2 + c\rho - \frac{c\gamma}{n} \leq 0, \\ \rho \leq \frac{\gamma}{n} - \frac{2\zeta\eta^2}{c}.\end{aligned}$$

Let $c = \frac{2\alpha n\zeta\eta^2}{\gamma}$ for some $\alpha > 1$, then $\rho \leq \frac{\gamma}{n} - \frac{\gamma}{\alpha n}$. Now we replace c with its value in C.6 and we get

$$\begin{aligned}&\rho - \mu\eta + 2\zeta\bar{L}^2\eta^2 + 2\alpha\frac{\zeta}{\gamma}L\bar{L}\eta^2 \leq 0, \\ &\eta \in \left(\eta^- = \frac{2\mu - \sqrt{\mu^2 - 8\rho(\bar{L}+\alpha L)\frac{\zeta}{\gamma}\bar{L}}}{4(\bar{L}+\alpha L)\frac{\zeta}{\gamma}L^2}, \eta^+ = \frac{2\mu + \sqrt{\mu^2 - 8\rho(\bar{L}+\alpha L)\frac{\zeta}{\gamma}\bar{L}}}{4(\bar{L}+\alpha L)\frac{\zeta}{\gamma}\bar{L}}\right).\end{aligned}$$

To get the term under square root be positive, we need that $\rho < \frac{\mu^2}{8(\bar{L}+\alpha L)\frac{\zeta}{\gamma}\bar{L}} < \frac{\mu^2}{8(1+\alpha)\frac{\zeta}{\gamma}L\bar{L}}$ holds. Finally if we set $\rho = \min\{\frac{\gamma\mu^2}{8(1+\alpha)\zeta L\bar{L}}, \frac{\gamma}{n} - \frac{\gamma}{\alpha n}\}$ and $\eta = \eta^+$, then we have:

$$\mathbb{E}\left[\Upsilon^{t+1}\right] \leq (1-\rho)^{t+1}\Upsilon^0,$$

where Υ^0 is a scalar. Since $\Psi_t > 0$, then $\mathbb{E}[\delta_{t+1}] \leq \mathbb{E}\left[\Upsilon^{t+1}\right]$ and we get:

$$\mathbb{E}[\delta_{t+1}] \leq (1-\rho)^{t+1}\Upsilon^0.$$

□

Appendix D

Chapter 5 Supplementary Material

D.1 Examples of Splitting for Variational-Gaussian Inference

We give detailed derivations for the splitting-examples shown in Section 5.2.1 in Chapter 5. As in Chapter 5, we denote the Gaussian posterior distribution by $q(\mathbf{z}|\lambda) := \mathcal{N}(\mathbf{z}|\mathbf{m},\mathbf{V})$, so that $\lambda = \{\mathbf{m},\mathbf{V}\}$ with $\mathbf{m}$ being the mean and $\mathbf{V}$ being the covariance matrix.

D.1.1 Gaussian Process (GP) Models

Consider GP models for N input-output pairs $\{y_n, \mathbf{x}_n\}$ indexed by n. Let $z_n := f(\mathbf{x}_n)$ be the latent function drawn from a GP with a zero-mean function and a covariance function $\kappa(\mathbf{x},\mathbf{x}')$. We denote the Kernel matrix obtained on the data $\mathbf{x}_n$ for all n by $\mathbf{K}$.

We use a non-Gaussian likelihood $p(y_n|z_n)$ to model the output, and assume that each y_n is independently sampled from this likelihood given $\mathbf{z}$. The joint-

distribution over $\mathbf{y}$ and $\mathbf{z}$ is shown below:

$$p(\mathbf{y},\mathbf{z}) = \prod_{n=1}^{N} p(y_n|z_n)\mathcal{N}(\mathbf{z}|0,\mathbf{K}). \tag{D.1}$$

The ratio required for the lower bound is shown below, along with the split, where non-Gaussian terms are in $\tilde{p}_d$ and Gaussian terms are in $\tilde{p}_e$:

$$\frac{p(\mathbf{y},\mathbf{z})}{q(\mathbf{z}|\mathbf{m},\mathbf{V})} = \underbrace{\prod_{n=1}^{N} p(y_n|z_n)}_{\tilde{p}_d(\mathbf{z}|\lambda)} \underbrace{\frac{\mathcal{N}(\mathbf{z}|0,\mathbf{K})}{\mathcal{N}(\mathbf{z}|\mathbf{m},\mathbf{V})}}_{\tilde{p}_e(\mathbf{z}|\lambda)}. \tag{D.2}$$

By substituting in Eq. (5.1) of Chapter 5, we can obtain the lower bound $\underline{\mathscr{L}}$ after a few simplifications, as shown below:

$$\underline{\mathscr{L}}(\mathbf{m},\mathbf{V}) := \mathbb{E}_{q(\mathbf{z})}\left[\log \frac{p(\mathbf{y},\mathbf{z})}{q(\mathbf{z}|\mathbf{m},\mathbf{V})}\right], \tag{D.3}$$

$$= \mathbb{E}_{q(\mathbf{z})}\left[\sum_{n=1}^{N} \log p(y_n|z_n)\right] + \mathbb{E}_{q(\mathbf{z})}\left[\log \frac{\mathcal{N}(\mathbf{z}|0,\mathbf{K})}{\mathcal{N}(\mathbf{z}|\mathbf{m},\mathbf{V})}\right], \tag{D.4}$$

$$= \underbrace{\sum_{n=1}^{N} \mathbb{E}_q[\log p(y_n|z_n)]}_{-f(\lambda)} - \underbrace{\mathbb{D}_{KL}[\mathcal{N}(\mathbf{z}|\mathbf{m},\mathbf{V}) \,\|\, \mathcal{N}(\mathbf{z}|0,\mathbf{K})]}_{h(\lambda)}. \tag{D.5}$$

The assumption A2 is satisfied since the KL divergence is convex in both $\mathbf{m}$ and $\mathbf{V}$. This is clear from the expression of the KL divergence:

$$D_{KL}[\mathcal{N}(\mathbf{z}|\mathbf{m},\mathbf{V})||\mathcal{N}(\mathbf{z}|0,\mathbf{K})] = \tfrac{1}{2}[-\log|\mathbf{V}\mathbf{K}^{-1}| + \mathrm{Tr}(\mathbf{V}\mathbf{K}^{-1}) + \mathbf{m}^T\mathbf{K}^{-1}\mathbf{m} - D] \tag{D.6}$$

where D is the dimensionality of $\mathbf{z}$. Convexity with respect to $\mathbf{m}$ follows from the fact that the above is quadratic in $\mathbf{m}$ and $\mathbf{K}$ is positive semi-definite. Convexity with respect to $\mathbf{V}$ follows due to concavity of $\log|\mathbf{V}|$ (trace is linear, so does not matter).

Assumption A1 depends on the choice of the likelihood $p(y_n|z_n)$, but is usually satisfied. The simplest example is a Gaussian likelihood for which the function f

takes the following form:

$$f(\mathbf{m},\mathbf{V}) = \sum_{n=1}^{N} \mathbb{E}_q[-\log p(y_n|z_n)] = \sum_{n=1}^{N} \mathbb{E}_q[-\log \mathcal{N}(y_n|z_n,\sigma^2)] \tag{D.7}$$

$$= \sum_{n=1}^{N} \tfrac{1}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}\left[(y_n - m_n)^2 + v_n\right], \tag{D.8}$$

where m_n is the n'th element of $\mathbf{m}$ and v_n is the n'th diagonal entry of $\mathbf{V}$. This clearly satisfies A1, since the objective is quadratic in $\mathbf{m}$ and linear in $\mathbf{V}$.

Here is an example where A1 is not satisfied: for Poisson likelihood $\log p(y_n|z_n) = \exp[y_n z_n - e^{z_n}]/y_n!$ with rate parameter equal to e^{z_n}, the function f takes the following form:

$$f(\mathbf{m},\mathbf{V}) = \sum_{n=1}^{N} \mathbb{E}_q[-\log p(y_n|z_n)] = \sum_{n=1}^{N} [-y_n m_n + e^{m_n + v_n/2} + \log(y_n!)] \tag{D.9}$$

whose derivative is not Lipschitz continuous since exponential functions are not Lipschitz.

D.1.2 Generalized Linear Models (GLMs)

We now describe a split for generalized linear models. We model the output y_n by using an exponential family distribution whose natural-parameter is equal to $\eta_n := \mathbf{x}_n^T \mathbf{z}$. Assuming a standard Gaussian prior over $\mathbf{z}$, the joint distribution can be written as follows:

$$p(\mathbf{y},\mathbf{z}) := \prod_{n=1}^{N} p(y_n|\mathbf{x}_n^T\mathbf{z})\mathcal{N}(\mathbf{z}|0,\mathbf{I}). \tag{D.10}$$

A similar split can be obtained by putting non-conjugate terms $p(y_n|\mathbf{x}_n^T\mathbf{z})$ in $\tilde{p}_d$ and the rest in $\tilde{p}_e$:

$$\frac{p(\mathbf{y},\mathbf{z})}{q(\mathbf{z}|\lambda)} = \underbrace{\prod_{n=1}^{N} p(y_n|\mathbf{x}_n^T\mathbf{z})}_{\tilde{p}_d(\mathbf{z}|\lambda)} \underbrace{\frac{\mathcal{N}(\mathbf{z}|0,\mathbf{I})}{\mathcal{N}(\mathbf{z}|\mathbf{m},\mathbf{V})}}_{\tilde{p}_e(\mathbf{z}|\lambda)}.$$

The lower bound can be shown to be the following:

$$\underline{\mathscr{L}}(\mathbf{m},\mathbf{V}) := \underbrace{\sum_{n=1}^{N} \mathbb{E}_q[\log p(y_n|\mathbf{x}_n^T\mathbf{z})]}_{-f(\lambda)} - \underbrace{\mathbb{D}_{KL}[\mathscr{N}(\mathbf{z}|\mathbf{m},\mathbf{V}) \,||\, \mathscr{N}(\mathbf{z}|0,\mathbf{I})]}_{h(\lambda)}, \tag{D.11}$$

which is very similar to the GP case. Therefore, Assumptions A1 and A2 will follow with similar arguments.

D.1.3 Correlated Topic Model (CTM)

We consider text documents with a vocabulary size N. Let $\mathbf{z}$ be a length K real-valued vector which follows a Gaussian distribution shown in (D.12). Given $\mathbf{z}$, a topic t_n is sampled for the n'th word using a multinomial distribution shown in (D.13). Probability of observing a word in the vocabulary is then given by (D.14).

$$p(\mathbf{z}|\theta) = \mathscr{N}(\mathbf{z}|\mu,\Sigma), \tag{D.12}$$

$$p(t_n = k|\mathbf{z}) = \frac{\exp(z_k)}{\sum_{j=1}^{K}\exp(z_j)}, \tag{D.13}$$

$$p(\text{Observing a word v}|t_n,\theta) = \beta_{v,t_n}. \tag{D.14}$$

Here β is a $N \times K$ real-valued matrix with non-negative entries and columns that sum to 1. The parameter set for this model is given by $\theta = \{\mu,\Sigma,\beta\}$. We can marginalize out t_n and obtain the data-likelihood given $\mathbf{z}$,

$$p(\text{Observing a word v}|\mathbf{z},\theta) = \sum_{k=1}^{K} p(\text{Observing a word v}|t_n = k,\theta)p(t_n = k|\mathbf{z}) \tag{D.15}$$

$$= \sum_{k=1}^{K} \beta_{vk}\frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}. \tag{D.16}$$

Given that we observe n'th word y_n times, we can write the following joint distribution:

$$p(\mathbf{y},\mathbf{z}) := \prod_{n=1}^{N}\left[\sum_{k=1}^{K}\beta_{n,k}\frac{e^{z_k}}{\sum_j e^{z_j}}\right]^{y_n}\mathcal{N}(\mathbf{z}|\mu,\Sigma). \tag{D.17}$$

We can then use the following split:

$$\frac{p(\mathbf{y},\mathbf{z})}{q(\mathbf{z}|\lambda)} = \underbrace{\prod_{n=1}^{N}\left[\sum_{k=1}^{K}\beta_{n,k}\frac{e^{z_k}}{\sum_j e^{z_j}}\right]^{y_n}}_{\tilde{p}_d(\mathbf{z}|\lambda)}\underbrace{\frac{\mathcal{N}(\mathbf{z}|\mu,\Sigma)}{\mathcal{N}(\mathbf{z}|\mathbf{m},\mathbf{V})}}_{\tilde{p}_e(\mathbf{z}|\lambda)},$$

where μ,Σ are parameters of the Gaussian prior and $\beta_{n,k}$ are parameters of K multinomials.

The lower bound is shown below:

$$\begin{aligned}\underline{\mathscr{L}}(\mathbf{m},\mathbf{V}) := &\sum_{n=1}^{N} y_n\left\{\mathbb{E}_q\left[\log\left(\sum_{k=1}^{K}\beta_{n,k}e^{z_k}\right)\right]\right\} - W\mathbb{E}_q\left\{\log\left[\sum_{j=1}^{K}e^{z_j}\right]\right\}\\ &- \mathbb{D}_{KL}[\mathcal{N}(\mathbf{z}|\mathbf{m},\mathbf{V})\,\|\,\mathcal{N}(\mathbf{z}|0,\mathbf{I})],\end{aligned} \tag{D.18}$$

where $W=\sum_n y_n$ is the total number of words. The top line is the function $[-f(\lambda)]$ while the bottom line is $[-h(\lambda)]$.

There are two intractable expectations in f, each involving expectation of a log-sum-exp function. Wang and Blei (2013) use the Delta method and Laplace method to approximate these expectations. In contrast, in PG-SVI algorithm, we use Monte Carlo to approximate the gradient of these functions.

D.2 Proof of Theorems 5.1 and 5.2

We first prove Theorem 5.2. Theorem 5.1 is obtained as a special case of it. Our proof technique is borrowed from Ghadimi et al. (2014). We extend their results to general divergence functions.

We denote the proximal projection at λ_k with gradient $\mathbf{g}$ and step-size β by,

$$\mathscr{P}(\lambda_k, \mathbf{g}, \beta) := \frac{1}{\beta}(\lambda_k - \lambda_{k+1}), \tag{D.19}$$

$$\text{where } \lambda_{k+1} = \underset{\lambda \in \mathscr{S}}{\operatorname{argmin}}\ \lambda^T \mathbf{g} + h(\lambda) + \frac{1}{\beta}\mathbb{D}(\lambda \,\|\, \lambda_k). \tag{D.20}$$

The following lemma gives a bound on the norm of $\mathscr{P}(\lambda_k, \mathbf{g}, \beta)$.

Lemma D.1. *The following holds for any $\lambda_k \in \mathscr{S}$, any real-valued vector $\mathbf{g}$ and $\beta > 0$.*

$$\mathbf{g}^T \mathscr{P}(\lambda_k, \mathbf{g}, \beta) \geq \alpha ||\mathscr{P}(\lambda_k, \mathbf{g}, \beta)||^2 + \frac{1}{\beta}[h(\lambda_{k+1}) - h(\lambda_k)]. \tag{D.21}$$

Proof. By taking the gradient of $\lambda^T \mathbf{g} + \frac{1}{\beta}\mathbb{D}(\lambda \,\|\, \lambda_k)$ and picking any sub-gradient ∇h of h at λ_{k+1}, the corresponding sub-gradient of the right hand side of ((D.20)) is given as follows:

$$\mathbf{g} + \nabla h(\lambda_{k+1}) + \frac{1}{\beta}\nabla_\lambda \mathbb{D}(\lambda_{k+1} \,\|\, \lambda_k). \tag{D.22}$$

We use this to derive the optimality condition of ((D.20)). For any λ, the following holds from the optimality condition:

$$(\lambda - \lambda_{k+1})^T \left[\mathbf{g} + \nabla h(\lambda_{k+1}) + \frac{1}{\beta}\nabla_\lambda \mathbb{D}(\lambda_{k+1} \,\|\, \lambda_k)\right] \geq 0. \tag{D.23}$$

Letting $\lambda = \lambda_k$,

$$(\lambda_k - \lambda_{k+1})^T \left[\mathbf{g} + \nabla h(\lambda_{k+1}) + \frac{1}{\beta}\nabla_\lambda \mathbb{D}(\lambda_{k+1} \,\|\, \lambda_k)\right] \geq 0, \tag{D.24}$$

which implies

$$\mathbf{g}^T(\lambda_k - \lambda_{k+1}) \geq \frac{1}{\beta}(\lambda_{k+1} - \lambda_k)^T \nabla_\lambda \mathbb{D}(\lambda_{k+1} \,\|\, \lambda_k) + h(\lambda_{k+1}) - h(\lambda_k) \tag{D.25}$$

$$\geq \frac{\alpha}{\beta}||\lambda_{k+1} - \lambda_k||^2 + h(\lambda_{k+1}) - h(\lambda_k). \tag{D.26}$$

The first line follows from Assumption A2 (convexity of h), and the second line follows from Assumption A6. □

Now, we are ready to prove Theorem 5.2:

Proof. Let $\tilde{g}_{\lambda,k} := \mathscr{P}(\lambda_k, \nabla f(\lambda_k), \beta_k)$. Since ∇f is L-smooth (Assumption A1), for any $k = 0, 1, \ldots, t-1$ we have,

$$\begin{aligned} f(\lambda_{k+1}) &\leq f(\lambda_k) + \langle \nabla f(\lambda_k), \lambda_{k+1} - \lambda_k \rangle + \frac{L}{2}\|\lambda_{k+1} - \lambda_k\|_2^2 \\ &= f(\lambda_k) - \beta_k \left\langle \nabla f(\lambda_k), \tilde{g}_{\lambda,k} \right\rangle + \frac{L}{2}\beta_k^2 \|\tilde{g}_{\lambda,k}\|_2^2 \\ &\leq f(\lambda_k) - \beta_k \alpha \|\tilde{g}_{\lambda,k}\|_2^2 - [h(\lambda_{k+1}) - h(\lambda_k)] + \frac{L}{2}\beta_k^2 \|\tilde{g}_{\lambda,k}\|_2^2. \end{aligned}$$

The second line follows from the definition of $\mathscr{P}$ and the last line is due to Lemma D.1. Rearranging the terms and using $-\underline{\mathscr{L}} = f + h$ we get:

$$\begin{aligned} & -\underline{\mathscr{L}}(\lambda_{k+1}) + \underline{\mathscr{L}}(\lambda_k) \leq -[\beta_k \alpha - \frac{L}{2}\beta_k^2]\|\tilde{g}_{\lambda,k}\|_2^2, \\ \Rightarrow \qquad & \underline{\mathscr{L}}(\lambda_{k+1}) - \underline{\mathscr{L}}(\lambda_k) \geq [\beta_k \alpha - \frac{L}{2}\beta_k^2]\|\tilde{g}_{\lambda,k}\|_2^2. \end{aligned}$$

Summing these term for all $k = 0, 1, \ldots t-1$, we get the following:

$$\underline{\mathscr{L}}(\lambda_{t-1}) - \underline{\mathscr{L}}(\lambda_0) \geq \sum_{k=0}^{t-1} [\beta_k \alpha - \frac{L}{2}\beta_k^2]\|\tilde{g}_{\lambda,k}\|_2^2.$$

By noting that the global maximum of the lower bound always upper bounds any other value, we get $\underline{\mathscr{L}}(\lambda_*) - \underline{\mathscr{L}}(\lambda_0) \geq \underline{\mathscr{L}}(\lambda_{t-1}) - \underline{\mathscr{L}}(\lambda_0)$. Using this,

$$\begin{aligned} & \underline{\mathscr{L}}(\lambda_*) - \underline{\mathscr{L}}(\lambda_0) \geq \sum_{k=0}^{t-1} [\beta_k \alpha - \frac{L}{2}\beta_k^2]\|\tilde{g}_{\lambda,k}\|_2^2, \\ \Rightarrow \qquad & \min_{k=0,1,\ldots,t-1} \|\tilde{g}_{\lambda,k}\|_2^2 [\sum_{k=0}^{t-1} [\beta_k \alpha - \frac{L}{2}\beta_k^2]] \leq \underline{\mathscr{L}}(\lambda_*) - \underline{\mathscr{L}}(\lambda_0). \end{aligned}$$

Since we assume at least one of $\beta_k < 2\alpha/L$, we can divide by the summation term,

to get the following:

$$\min_{k=0,1,\ldots,t-1} \|\tilde{g}_{\lambda,k}\|_2^2 \leq \frac{\underline{\mathscr{L}}(\lambda_*) - \underline{\mathscr{L}}(\lambda_1)}{\sum_{k=0}^{t-1}[\beta_k\alpha - \frac{L}{2}\beta_k^2]},$$

which proves Theorem 5.2. □

Theorem 5.1 can be obtained by simply plugging in $\beta_k = \alpha/L$,

$$\min_{k=0,1,\ldots,t-1} \|\tilde{g}_{\lambda,k}\|_2^2 \leq \frac{C_0}{\sum_{k=0}^{t-1}[\frac{\alpha^2}{L} - \frac{\alpha^2}{2L}]} = \frac{2C_0L}{\alpha^2 t}.$$

D.3 Proof of Theorem 5.3

We will first prove the following theorem, which gives a similar result to Theorem 5.2 but for a stochastic gradient $\widehat{\nabla} f$.

Theorem D.2. *If we choose the step-size β_k such that $0 < \beta_k \leq 2\alpha_*/L$ with $\beta_k < 2\alpha_*/L$ for at least one k, then,*

$$\mathbb{E}_{R,\xi}[\|G_R\|_2^2] \leq \frac{C_0 + \frac{c\sigma^2}{2}\sum_{k=0}^{t-1}\frac{\beta_k}{M_k}}{\sum_{k=0}^{t-1}\left(\alpha_*\beta_k - L\beta_k^2/2\right)}, \tag{D.27}$$

where the expectation is taken over $R \in \{0,1,2,\ldots,t-1\}$ which is a discrete random variable drawn from the probability mass function

$$Prob(R=k) = \frac{\alpha_*\beta_k - L\beta_k^2/2}{\sum_{k=0}^{t-1}\left(\alpha_*\beta_k - L\beta_k^2/2\right)},$$

and over $\xi := \{\xi_1, \xi_2, \ldots, \xi_{t-1}\}$ with ξ_k is the noise in the stochastic approximation $\widehat{\nabla} f$.

Proof. Let $\tilde{g}_{\lambda,k} := \mathscr{P}(\lambda_k, \widehat{\nabla} f(\lambda_k), \beta_k)$, $\delta_k := \widehat{\nabla} f(\lambda_k) - \nabla f(\lambda_k)$. Since ∇f is L-

smooth, for any $k = 0, 1, \ldots, t$ we have,

$$f(\lambda_{k+1}) \leq f(\lambda_k) + \langle \nabla f(\lambda_k), \lambda_{k+1} - \lambda_k \rangle + \frac{L}{2}\|\lambda_{k+1} - \lambda_k\|_2^2 \tag{D.28}$$

$$= f(\lambda_k) - \beta_k \left\langle \nabla f(\lambda_k), \tilde{g}_{\lambda,k} \right\rangle + \frac{L}{2}\beta_k^2 \|\tilde{g}_{\lambda,k}\|_2^2 \tag{D.29}$$

$$= f(\lambda_k) - \beta_k \left\langle \widehat{\nabla} f(\lambda_k), \tilde{g}_{\lambda,k} \right\rangle + \frac{L}{2}\beta_k^2 \|\tilde{g}_{\lambda,k}\|_2^2 + \beta_k \left\langle \delta_k, \tilde{g}_{\lambda,k} \right\rangle \tag{D.30}$$

where we have used the definition of $\tilde{g}_{\lambda,k}$ and δ_k. Now using Lemma D.1 on the second term and Cauchy-Schwarz for the last term, we get the following:

$$f(\lambda_{k+1}) \leq f(\lambda_k) - \left[\alpha\beta_k \|\tilde{g}_{\lambda,k}\|_2^2 + h(\lambda_{k+1}) - h(\lambda_k)\right] + \frac{L}{2}\beta_k^2 \|\tilde{g}_{\lambda,k}\|_2^2 + \beta_k \|\delta_k\|_2 \|\tilde{g}_{\lambda,k}\|_2. \tag{D.31}$$

After rearranging and using Young's inequality $\|\delta_k\|_2 \|\tilde{g}_{\lambda,k}\|_2 \leq (c/2)\|\delta_k\|_2^2 + 1/(2c)\|\tilde{g}_{\lambda,k}\|_2^2$ given a constant $c > 0$, we get

$$-\underline{\mathscr{L}}(\lambda_{k+1}) \leq -\underline{\mathscr{L}}(\lambda_k) - \alpha\beta_k \|\tilde{g}_{\lambda,k}\|_2^2 + \frac{L}{2}\beta_k^2 \|\tilde{g}_{\lambda,k}\|_2^2 + \frac{\beta_k}{2c}\|\tilde{g}_{\lambda,k}\|_2^2 + \frac{\beta_k c}{2}\|\delta_k\|_2^2 \tag{D.32}$$

$$= -\underline{\mathscr{L}}(\lambda_k) - \left((\alpha - 1/(2c))\beta_k - \frac{L}{2}\beta_k^2\right)\|\tilde{g}_{\lambda,k}\|_2^2 + \frac{c\beta_k}{2}\|\delta_k\|_2^2. \tag{D.33}$$

Now considering $c > 1/(2\alpha)$, $\alpha_* = \alpha - 1/(2c)$ and $\beta_k \leq \frac{2\alpha_*}{L}$, and summing up both sides for iteration $k = 0, 1 \ldots, t-1$, we obtain

$$\sum_{k=0}^{t-1} \left(\alpha_*\beta_k - \frac{L}{2}\beta_k^2\right)\|\tilde{g}_{\lambda,k}\|_2^2 \leq \underline{\mathscr{L}}^* - \underline{\mathscr{L}}(\lambda_0) + \sum_{k=0}^{t-1} \frac{c\beta_k}{2}\|\delta_k\|_2^2. \tag{D.34}$$

Now by taking expectation with respect to ξ on both sides and using the fact that $\mathbb{E}_\xi \|\delta_k\|_2^2 \leq \frac{\sigma^2}{M_k}$ by assumption $A3$ and $A4$, we get

$$\sum_{k=0}^{t-1} \left(\alpha_*\beta_k - \frac{L}{2}\beta_k^2\right)\mathbb{E}_\xi\|\tilde{g}_{\lambda,k}\|_2^2 \leq C_0 + \frac{c\sigma^2}{2}\sum_{k=0}^{t-1}\frac{\beta_k}{M_k}. \tag{D.35}$$

Writing the expectation with respect to R and $\mathbf{x}$ we get

$$\mathbb{E}_{R,\xi}[\|\tilde{g}_{\lambda_k,R}\|_2^2] = \frac{\sum_{k=0}^{t-1}\left(\alpha_*\beta_k - \frac{L}{2}\beta_k^2\right)\mathbb{E}_\xi\|\tilde{g}_{\lambda,k}\|_2^2}{\sum_{k=0}^{t-1}\left(\alpha_*\beta_k - \frac{L}{2}\beta_k^2\right)}, \tag{D.36}$$

whose numerator is the left side of (D.35). Dividing (D.35) by $\sum_{k=0}^{t}\left(\alpha_*\beta_k - \frac{L}{2}\beta_k^2\right)$ and using this in (D.36) we get the result. □

By substituting $\beta_k = \alpha_*/L$ and $M_k = M$ in (D.27),

$$\mathbb{E}_{R,\xi}[\|G_R\|_2^2] \leq \frac{C_0 + \frac{c\sigma^2}{2}\sum_{k=0}^{t-1}\frac{\beta_k}{M_k}}{\sum_{k=0}^{t-1}\left(\alpha_*\beta_k - L\beta_k^2/2\right)} \tag{D.37}$$

$$= \frac{C_0 + \frac{c\sigma^2\alpha_* t}{2LM}}{\frac{\alpha_*^2 t}{2L}} = \left(\frac{2LC_0}{\alpha_*^2 t} + \frac{c\sigma^2}{M\alpha^*}\right) \tag{D.38}$$

The probability distribution for R reduces to a uniform distribution in this case, with the probability of each iteration being $1/t$. This proves Theorem D.2.

D.4 Derivation of Closed-form Updates for the GP Model

The PG-SVI iterations $\lambda_{k+1} = \min_{\lambda\in\mathscr{S}} \lambda^T\left[\widehat{\nabla} f(\lambda_k)\right] + h(\lambda) + \frac{1}{\beta_k}\mathbb{D}(\lambda\|\lambda_k)$ takes the following form for the GP model, as discussed in Section 5.5 of Chapter 5:

$$\begin{aligned}(\mathbf{m}_{k+1}, \mathbf{V}_{k+1}) = \underset{\mathbf{m},\mathbf{V}\succ 0}{\operatorname{argmin}}\ & (m_n\alpha_{n_k,k} + \tfrac{1}{2}v_n\gamma_{n_k,k}) \\ & + D_{KL}\left[\mathscr{N}(\mathbf{z}|\mathbf{m},\mathbf{V})||\mathscr{N}(\mathbf{z}|0,\mathbf{K})\right] \\ & + \frac{1}{\beta_k}D_{KL}\left[\mathscr{N}(\mathbf{z}|\mathbf{m},\mathbf{V})||\mathscr{N}(\mathbf{z}|\mathbf{m}_k,\mathbf{V}_k)\right],\end{aligned} \tag{D.39}$$

where n_k is the example selected in k'th iteration. We will now show that its solution can be obtained in closed-form.

D.4.1 Full Update of $\mathbf{V}_{k+1}$

We first derive the full update of $\mathbf{V}_{k+1}$. The KL divergence between two Gaussian distributions is given as follows:

$$D_{KL}\left[\mathcal{N}(\mathbf{z}|\mathbf{m},\mathbf{V})||\mathcal{N}(\mathbf{z}|0,\mathbf{K})\right] = -\tfrac{1}{2}[\log|\mathbf{V}\mathbf{K}^{-1}| - \mathrm{Tr}(\mathbf{V}\mathbf{K}^{-1}) - \mathbf{m}^T\mathbf{K}^{-1}\mathbf{m} + D]. \tag{D.40}$$

Using this, we expand the last two terms of (D.39) to get the following,

$$\begin{aligned}
&-\tfrac{1}{2}\left[\log|\mathbf{V}\mathbf{K}^{-1}| - \mathrm{Tr}(\mathbf{V}\mathbf{K}^{-1}) - \mathbf{m}^T\mathbf{K}^{-1}\mathbf{m} + D\right] \\
&-\tfrac{1}{2}\frac{1}{\beta_k}\left[\log|\mathbf{V}\mathbf{K}^{-1}| - \mathrm{Tr}\{\mathbf{V}\mathbf{V}_k^{-1}\} - (\mathbf{m}-\mathbf{m}_k)^T\mathbf{V}_k^{-1}(\mathbf{m}-\mathbf{m}_k) + D\right] \\
&= -\tfrac{1}{2}\left[\left(1+\frac{1}{\beta_k}\right)\log|\mathbf{V}| - \mathrm{Tr}\{\mathbf{V}(\mathbf{K}^{-1}+\frac{1}{\beta_k}\mathbf{V}_k^{-1})\} - \mathbf{m}^T\mathbf{K}^{-1}\mathbf{m}\right. \\
&\qquad \left. -\frac{1}{\beta_k}(\mathbf{m}-\mathbf{m}_k)^T\mathbf{V}_k^{-1}(\mathbf{m}-\mathbf{m}_k) + \left(1+\frac{1}{\beta_k}\right)(D-\log|\mathbf{K}|)\right].
\end{aligned} \tag{D.41}$$

Taking derivative of (D.39) with respect to $\mathbf{V}$ at $\mathbf{V}=\mathbf{V}_{k+1}$ and setting it to zero, we get the following (here $\mathbf{I}_n$ is a matrix with all zeros, except the n'th diagonal element which is set to 1):

$$\Rightarrow \quad -\left(1+\frac{1}{\beta_k}\right)\mathbf{V}_{k+1}^{-1} + \left(\mathbf{K}^{-1}+\frac{1}{\beta_k}\mathbf{V}_k^{-1}\right) + \gamma_{n_k,k}\mathbf{I}_{n_k} = 0, \tag{D.42}$$

$$\Rightarrow \quad \mathbf{V}_{k+1}^{-1} = \frac{1}{1+\beta_k}\mathbf{V}_k^{-1} + \frac{\beta_k}{1+\beta_k}\left(\mathbf{K}^{-1}+\gamma_{n_k,k}\mathbf{I}_{n_k}\right), \tag{D.43}$$

$$\Rightarrow \quad \mathbf{V}_{k+1}^{-1} = r_k\mathbf{V}_k^{-1} + (1-r_k)\left(\mathbf{K}^{-1}+\gamma_{n_k,k}\mathbf{I}_{n_k}\right), \tag{D.44}$$

which gives us the update of $\mathbf{V}_{k+1}$ for $r_k := 1/(1+\beta_k)$.

D.4.2 Avoiding a full update of $\mathbf{V}_{k+1}$

A full update will require storing the matrix $\mathbf{V}_{k+1}$. Fortunately, we can avoid storing the full matrix and still do an exact update. The key point here is to notice that to compute the stochastic gradient in the next iteration we only need one diagonal element of $\mathbf{V}_{k+1}$ rather than the whole matrix. Specifically, if we sample n_{k+1}'th example at the iteration $k+1$, then we need to compute $v_{n_{k+1},k+1}$ which is

the n_{k+1}'th diagonal element of $\mathbf{V}_{k+1}$. This can be done by solving one linear equation, as we show in this section. Specifically, we show that the following updates can be used to compute $v_{n_{k+1},k+1}$:

$$v_{n_{k+1},k+1} = \kappa_{n_{k+1},n_{k+1}} - \kappa_{n_{k+1}}^T \left(\mathbf{K} + [\text{diag}(\widetilde{\gamma}_k)]^{-1}\right)^{-1} \kappa_{n_{k+1}}, \tag{D.45}$$

where $\widetilde{\gamma}_k = r_k \widetilde{\gamma}_{k-1} + (1-r_k)\gamma_{n_k,k} 1_{n_k}$ (1_n is a vector of all zeros except its n'th entry which is equal to 1). We start the recursion with $\widetilde{\gamma}_0 = \varepsilon$ where ε is a small positive number.

We will now show that $\mathbf{V}_k$ can be reparameterized in terms of a vector $\widetilde{\gamma}_k$ which contains accumulated weighted sum of the gradient $\gamma_{n_j,j}$, for all $j \leq k$. To show this, we recursively substitute the update of $\mathbf{V}_j$ for $j < k+1$, as shown below (recall that n_k is the example selected at the k'th iteration). The second line is obtained by substituting the full update of $\mathbf{V}_k$ by using (D.44). The third line is obtained after a few simplifications. The fourth line is obtained by substituting the update of $\mathbf{V}_{k-1}$ and a few simplifications.

$$\begin{aligned}
\mathbf{V}_{k+1}^{-1} &= r_k \mathbf{V}_k^{-1} + (1-r_k)\left[\mathbf{K}^{-1} + \gamma_{n_k,k}\mathbf{I}_{n_k}\right] \\
&= r_k\left[r_{k-1}\mathbf{V}_{k-1}^{-1} + (1-r_{k-1})\left(\mathbf{K}^{-1} + \gamma_{n_{k-1},k-1}\mathbf{I}_{n_{k-1}}\right)\right] \\
&\quad + (1-r_k)\left[\mathbf{K}^{-1} + \gamma_{n_k,k}\mathbf{I}_{n_k}\right] \\
&= r_k r_{k-1}\mathbf{V}_{k-1}^{-1} + (1 - r_k r_{k-1})\mathbf{K}^{-1} \\
&\quad + \left[r_k(1-r_{k-1})\gamma_{n_{k-1},k-1}\mathbf{I}_{n_{k-1}} + (1-r_k)\gamma_{n_k,k}\mathbf{I}_{n_k}\right] \\
&= r_k r_{k-1} r_{k-2}\mathbf{V}_{k-2}^{-1} + (1 - r_k r_{k-1} r_{k-2})\mathbf{K}^{-1} \\
&\quad + [r_k r_{k-1}(1-r_{k-2})\gamma_{n_{k-2},k-2}\mathbf{I}_{n_{k-2}} \\
&\quad + r_k(1-r_{k-1})\gamma_{n_{k-1},k-1}\mathbf{I}_{n_{k-1}} + (1-r_k)\gamma_{n_k,k}\mathbf{I}_{n_k}].
\end{aligned}$$

This update expresses $\mathbf{V}_{k+1}$ in terms of $\mathbf{V}_{k-2}$, $\mathbf{K}$, and gradients of the data example selected at $k, k-1$, and $k-2$. Continuing in this fashion until $k=0$, we can write

the update as follows:

$$\begin{aligned}\mathbf{V}_{k+1}^{-1} = t_k\mathbf{V}_0^{-1} + (1-t_k)\mathbf{K}^{-1} + [&r_k r_{k-1}\dots r_3 r_2(1-r_1)\gamma_{n_1,1}\mathbf{I}_{n_1} \\ &+ r_k r_{k-1}\dots r_4 r_3(1-r_2)\gamma_{n_2,2}\mathbf{I}_{n_2} \\ &+ r_k r_{k-1}\dots r_5 r_4(1-r_3)\gamma_{n_3,3}\mathbf{I}_{n_2} + \dots \\ &+ r_k r_{k-1}(1-r_{k-2})\gamma_{n_{k-2},k-2}\mathbf{I}_{n_{k-2}} \\ &+ r_k(1-r_{k-1})\gamma_{n_{k-1},k-1}\mathbf{I}_{n_{k-1}} + (1-r_k)\gamma_{n_k,k}\mathbf{I}_{n_k}],\end{aligned} \tag{D.46}$$

where t_k is the product of $r_k, r_{k-1}, \dots, r_0$. We can write the updates more compactly by defining the accumulation of the gradients $\gamma_{n_j,j}$ for all $j \le k$ by a vector $\widetilde{\gamma}_k$,

$$\mathbf{V}_{k+1}^{-1} = t_k\mathbf{V}_0^{-1} + (1-t_k)\mathbf{K}^{-1} + \mathrm{diag}(\widetilde{\gamma}_k)$$

The vector $\widetilde{\gamma}_k$ can be obtained by using a recursion. We illustrate this below, where we have grouped the terms in (D.46) to show the recursion for $\widetilde{\gamma}_k$ (here 1_n is a vector with all zero entries except n'th entry which is set to 1):

$$\begin{aligned}&r_k r_{k-1}\dots r_6 r_5 r_4 r_3 r_2(1-r_1)\gamma_{n_1,1}\mathbf{1}_{n_1} &&= \widetilde{\gamma}_1 \\ +&r_k r_{k-1}\dots r_6 r_5 r_4 r_3(1-r_2)\gamma_{n_2,2}\mathbf{1}_{n_2} &&= \widetilde{\gamma}_2 \\ +&r_k r_{k-1}\dots r_6 r_5 r_4(1-r_3)\gamma_{n_3,3}\mathbf{1}_{n_3} &&= \widetilde{\gamma}_3 \\ +&r_k r_{k-1}\dots r_6 r_5(1-r_4)\gamma_{n_4,4}\mathbf{1}_{n_4} &&= \widetilde{\gamma}_4 \\ +&r_k r_{k-1}\dots r_6(1-r_5)\gamma_{n_5,5}\mathbf{1}_{n_5} &&= \widetilde{\gamma}_5 \\ &\vdots\end{aligned}$$

Therefore, $\widetilde{\gamma}_k$ can be recursively updated as follows:

$$\widetilde{\gamma}_k = r_k\widetilde{\gamma}_{k-1} + (1-r_k)\gamma_{n_k,k}1_{n_k}, \tag{D.47}$$

with an initialization $\widetilde{\gamma}_0 = \varepsilon$ where ε is a small constant to avoid numerical issues. If we set $\mathbf{V}_0 = \mathbf{K}$, then the formula simplifies to the following:

$$\mathbf{V}_{k+1}^{-1} = \mathbf{K}^{-1} + \mathrm{diag}(\widetilde{\gamma}_k), \tag{D.48}$$

which is completely specified by $\widetilde{\gamma}_k$, eliminating the need to compute and store

$\mathbf{V}_{k+1}$. The n_{k+1}'th diagonal element can be obtained by using Matrix Inversion Lemma, which gives us the update (D.45).

D.4.3 Update of m

Taking derivative of (D.39) with respect to $\mathbf{m}$ at $\mathbf{m} = \mathbf{m}_{k+1}$ and setting it to zero, we get the following (here 1_n is a vector with all zero entries except n'th entry which is set to 1):

$$
\begin{aligned}
\Rightarrow \quad & -\mathbf{K}^{-1}\mathbf{m}_{k+1} - \frac{1}{\beta_k}\mathbf{V}_k^{-1}(\mathbf{m}_{k+1} - \mathbf{m}_k) - \alpha_{n_k,k}1_{n_k} = 0, \\
\Rightarrow \quad & -\left[\mathbf{K}^{-1} + \frac{1}{\beta_k}\mathbf{V}_k^{-1}\right]\mathbf{m}_{k+1} + \frac{1}{\beta_k}\mathbf{V}_k^{-1}\mathbf{m}_k - \alpha_{n_k,k}1_{n_k} = 0, \\
\Rightarrow \quad & \mathbf{m}_{k+1} = \left[\mathbf{K}^{-1} + \frac{1}{\beta_k}\mathbf{V}_k^{-1}\right]^{-1}\left[\frac{1}{\beta_k}\mathbf{V}_k^{-1}\mathbf{m}_k - \alpha_{n_k,k}1_{n_k}\right], \\
\Rightarrow \quad & \mathbf{m}_{k+1} = \left[(1-r_k)\mathbf{K}^{-1} + r_k\mathbf{V}_k^{-1}\right]^{-1}\left[-(1-r_k)\alpha_{n_k,k}1_{n_k} + r_k\mathbf{V}_k^{-1}\mathbf{m}_k\right],
\end{aligned}
$$

where the last step is obtained using the fact that $1/\beta_k = r_k/(1-r_k)$.

We simplify as shown below. The second line is obtained by adding and subtracting $(1-r_k)\mathbf{K}^{-1}\mathbf{m}_k$ in the square bracket at the right. In the the third line, we take $\mathbf{m}_k$ out. The fourth line is obtained by plugging in the updates of $\mathbf{V}_k^{-1} = \mathbf{K}^{-1} + \mathrm{diag}(\widetilde{\gamma}_k)$. The fifth line is obtained by using Matrix-Inversion lemma, and the sixth line is obtained by taking $\mathbf{K}^{-1}$ out of the right-most term.

$$
\begin{aligned}
\mathbf{m}_{k+1} &= \left[(1-r_k)\mathbf{K}^{-1} + r_k\mathbf{V}_k^{-1}\right]^{-1}\left[-(1-r_k)\alpha_{n_k,k}1_{n_k} + r_k\mathbf{V}_k^{-1}\mathbf{m}_k\right] \\
&= \left[(1-r_k)\mathbf{K}^{-1} + r_k\mathbf{V}_k^{-1}\right]^{-1}\left[(1-r_k)\{-\mathbf{K}^{-1}\mathbf{m}_k - \alpha_{n_k,k}1_{n_k}\}\right. \\
&\qquad \left. + \{(1-r_k)\mathbf{K}^{-1} + r_k\mathbf{V}_k^{-1}\}\mathbf{m}_k\right] \\
&= \mathbf{m}_k + (1-r_k)\left[(1-r_k)\mathbf{K}^{-1} + r_k\mathbf{V}_k^{-1}\right]^{-1}\left(-\mathbf{K}^{-1}\mathbf{m}_k - \alpha_{n_k,k}1_{n_k}\right) \\
&= \mathbf{m}_k - (1-r_k)\left[\mathbf{K}^{-1} + r_k\mathrm{diag}(\widetilde{\gamma}_{k-1})\right]^{-1}\left(\mathbf{K}^{-1}\mathbf{m}_k + \alpha_{n_k,k}1_{n_k}\right) \\
&= \mathbf{m}_k - (1-r_k)\left[\mathbf{K} - \mathbf{K}\left(\mathbf{K} + \mathrm{diag}(r_k\widetilde{\gamma}_{k-1})^{-1}\right)^{-1}\mathbf{K}\right]\left(\mathbf{K}^{-1}\mathbf{m}_k + \alpha_{n_k,k}1_{n_k}\right) \\
&= \mathbf{m}_k - (1-r_k)\left[\mathbf{I} - \mathbf{K}\left(\mathbf{K} + \mathrm{diag}(r_k\widetilde{\gamma}_{k-1})^{-1}\right)^{-1}\right](\mathbf{m}_k + \alpha_{n_k,k}\kappa_{n_k}) \\
&= \mathbf{m}_k - (1-r_k)(\mathbf{I} - \mathbf{K}\mathbf{B}_k^{-1})(\mathbf{m}_k + \alpha_{n_k,k}\kappa_{n_k}),
\end{aligned}
$$

where $\mathbf{B}_k := \mathbf{K} + [\text{diag}(r_k \widetilde{\gamma}_{k-1})]^{-1}$.

Since $r_k \widetilde{\gamma}_{k-1}$ and $\widetilde{\gamma}_k$ differ only slightly (by the new example gradient γ_{n_k}, we can instead use the following approximate update:

$$\mathbf{m}_{k+1} = \mathbf{m}_k - (1 - r_k)(\mathbf{I} - \mathbf{K}\mathbf{A}_k^{-1})(\mathbf{m}_k + \alpha_{n_k,k}\kappa_{n_k}), \tag{D.49}$$

where $\mathbf{A}_k := \mathbf{K} + [\text{diag}(\widetilde{\gamma}_k)]^{-1}$.

D.5 Closed-form Updates for GLMs

We rewrite the lower bound as

$$-\underline{\mathscr{L}}(\mathbf{m}, \mathbf{V}) := \underbrace{\sum_{n=1}^{N} f_n(\widetilde{m}_n, \widetilde{v}_n)}_{f(m,V)} + \underbrace{\mathbb{D}_{KL}[\mathscr{N}(\mathbf{z}|\mathbf{m}, \mathbf{V}) \,\|\, \mathscr{N}(\mathbf{z}|0, \mathbf{I})]}_{h(\mathbf{m},\mathbf{V})}, \tag{D.50}$$

where $f_n(\widetilde{m}_n, \widetilde{v}_n) := -\mathbb{E}_q[\log p(y_n|\mathbf{x}_n^T \mathbf{z})]$ with $\widetilde{m}_n := \mathbf{x}_n^T$ and $\widetilde{v}_n := \mathbf{x}_n^T \mathbf{V} \mathbf{x}_n$. We can compute a stochastic approximation to the gradient of f by randomly selecting an example n_k (choosing $M = 1$) and using a Monte Carlo gradient approximation to the gradient of f_{n_k}. Similar to GP, we define the following as our gradients of function f_n:

$$\alpha_{n_k,k} := N \nabla_{\widetilde{m}_{n_k}} f_{n_k}(\widetilde{m}_{n_k}, \widetilde{v}_{n_k}), \quad \gamma_{n_k,k} := 2N \nabla_{\widetilde{v}_{n_k}} f_{n_k}(\widetilde{m}_{n_k}, \widetilde{v}_{n_k}) \tag{D.51}$$

The PG-SVI iteration can be written as follows:

$$\begin{aligned}(\mathbf{m}_{k+1}, \mathbf{V}_{k+1}) = \underset{\mathbf{m}, \mathbf{V} \succ 0}{\text{argmin}} \; & (\widetilde{m}_n \alpha_{n_k,k} + \tfrac{1}{2} \widetilde{v}_n \gamma_{n_k,k}) + D_{KL}[\mathscr{N}(\mathbf{z}|\mathbf{m}, \mathbf{V}) || \mathscr{N}(\mathbf{z}|0, \mathbf{I})] \\ & + \frac{1}{\beta_k} D_{KL}[\mathscr{N}(\mathbf{z}|\mathbf{m}, \mathbf{V}) || \mathscr{N}(\mathbf{z}|\mathbf{m}_k, \mathbf{V}_k)]. \end{aligned} \tag{D.52}$$

Using a similar derivation to the GP model, we can show that the following updates will give us the solution:

$$\begin{aligned}
\widetilde{\gamma}_k &= r_k \widetilde{\gamma}_{k-1} + (1-r_k)\gamma_{n_k,k} 1_{n_k}, \\
\widetilde{\mathbf{m}}_{k+1} &= \widetilde{\mathbf{m}}_k - (1-r_k)(\mathbf{I} - \mathbf{K}\mathbf{A}_k^{-1})(\mathbf{m}_k + \alpha_{n_k,k}\kappa_{n_k}), \\
\widetilde{v}_{n_{k+1},k+1} &= \kappa_{n_{k+1},n_{k+1}} - \kappa_{n_{k+1}}^T \mathbf{A}_k^{-1} \kappa_{n_{k+1}},
\end{aligned} \tag{D.53}$$

where $\mathbf{K} = \mathbf{X}\mathbf{X}^T$ and $\widetilde{\mathbf{m}}_k := \mathbf{X}^T\mathbf{m}$.