Second Order Stochastic Optimization for Machine Learning in Linear Time

Naman Agarwal[*] Brian Bullins[†] Elad Hazan [‡]

October 17, 2016

Abstract

First-order stochastic methods are the state-of-the-art in large-scale machine learning optimization owing to efficient per-iteration complexity. Second-order methods, while able to provide faster convergence, have been much less explored due to the high cost of computing the second-order information. In this paper we develop second-order stochastic methods for optimization problems in machine learning that match the per-iteration cost of gradient based methods, and in certain settings improves upon the overall running time upon the state-of-the-art. Furthermore, our algorithm has the desirable property of being implementable in time linear in the sparsity of the input data.

[*]namana@cs.princeton.edu, Computer Science, Princeton University
[†]bbullins@cs.princeton.edu, Computer Science, Princeton University
[‡]ehazan@cs.princeton.edu, Computer Science, Princeton University

1 Introduction

In recent literature stochastic first-order optimization has taken the stage as the primary workhorse for training learning models, due in large part to its affordable computational costs which are linear (in the data representation) per iteration. The main research effort devoted to improving the convergence rates of first-order methods have introduced elegant ideas and algorithms in recent years, including adaptive regularization [DHS11], variance reduction [JZ13, DBLJ14], dual coordinate ascent [SSZ13], and many more.

In contrast, second-order methods have typically been much less explored in large scale machine learning (ML) applications due to their prohibitive computational cost per iteration which requires computation of the Hessian in addition to a matrix inversion. These operations are infeasible for large scale problems in high dimensions.

In this paper we propose a family of novel second-order algorithms, LiSSA (Linear time Stochastic Second-Order Algorithm) for convex optimization that attain fast convergence rates while also allowing for an implementation with linear time per-iteration cost, matching the running time of the best known gradient-based methods. Moreover, in the setting where the number of training examples (m) is much larger than the underlying dimension (d), we show that our algorithm has provably faster running time than the best known gradient based methods.

Formally, the main optimization problem we are concerned with is the Empirical Risk Minimization (ERM) problem:

$$\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x}) = \min_{\mathbf{x}\in\mathbb{R}^d} \left\{ \frac{1}{m}\sum_{k=1}^{m} f_k(\mathbf{x}) + R(\mathbf{x}) \right\}$$

where each $f_k(\mathbf{x})$ is a convex function and $R(\mathbf{x})$ is a convex regularizer. The above optimization problem is the standard objective minimized in most supervised learning settings. Examples include Logistic Regression, SVMs, etc. A common aspect of many applications of ERM in ML is that the loss function $f_i(\mathbf{x})$ is of the form $l(\mathbf{x}^T\mathbf{v}_i, y_i)$ where $(\mathbf{v}_i, y_i)$ is the i^{th} training example-label pair. We call such functions Generalized Linear Models (GLM) and will restrict our attention to this case. We will assume that the regularizer is an ℓ_2 regularizer [1], typically $\|\mathbf{x}\|^2$.

In this paper we focus on second-order optimization methods (Newton's Method) where in each iteration, the underlying principle is to move to the minimizer of the second-order Taylor approximation at any point. Formally, the Newton method update at a point $\mathbf{x}_t$ is given by

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \nabla^{-2} f(\mathbf{x}_t)\nabla f(\mathbf{x}_t) \tag{1}$$

Newton's method has certain desirable properties such as the updates being independent of the choice of coordinate system and the Hessian providing the necessary regularization based on the curvature at the present point. Indeed, Newton's method can be shown to achieve quadratic convergence [Nes04] eventually. Although Newton's method comes with good theoretical guarantees, the complexity per step grows roughly as $\Omega(md^2 + d^3)$ (the former term for computing the Hessian and the latter for inversion) making it prohibitive in practice. Our main contribution is a suite of algorithms each of which performs an approximate Newton update based on stochastic Hessian

[1] It can be seen that some of these assumptions can be relaxed in our analyses, but since these choices are standard in ML, we make these assumptions to simplify the discourse.

information and is implementable in linear $O(d)$ time. These algorithms improve upon the state-of-the-art first-order methods both in theory and practice. In the following we give a summary of our results. We propose two algorithms, LiSSA and LiSSA-Sample.

LiSSA: (Algorithm 1) is a practical stochastic second-order algorithm based on a novel estimator of the Hessian inverse, leading to an efficient Newton Step (Equation 1). The estimator is based on the well known Taylor approximation of the inverse (Fact 2) and is described formally in Section 3.1. We prove the following informal theorem about LiSSA.

Theorem (Informal)**.** *LiSSA returns a point $\mathbf{x}_t$ such that $f(\mathbf{x}_t) \leq \min_{\mathbf{x}^*} f(\mathbf{x}^*) + \varepsilon$ in total time*

$$\tilde{O}\left((m + S_1\kappa)\, d \log\left(\frac{1}{\varepsilon}\right)\right).$$

where κ is the underlying condition number of the problem and S_1 is a bound on the variance of the estimator.

The precise version of the above theorem appears as Theorem 3.3. In theory, the best bound we can show for S_1 is $O(\kappa^2)$, but in practice we observe that setting S_1 to be a small constant is sufficient. Therefore LiSSA as evidenced by the theorem above enjoys a convergence rate comparable to the state-of-the-art first-order methods. We provide a detailed comparison of our results with existing first-order and second-order methods in Section 1.2. Moreover we demonstrate in Section 7 that in practice LiSSA performs well compared to popular first-order methods. We also show that LiSSA runs in time proportional to input sparsity making it an attractive method for high dimensional sparse data.

LiSSA-Sample: This variant brings together efficient first-order algorithms with Matrix Sampling techniques [LMP13, CLM+15] to achieve better runtime guarantees than the state-of-the-art in convex optimization for ML in the regime when $m > d$. Specifically we prove the following theorem:

Theorem (Informal)**.** *LiSSA-Sample returns a point $\mathbf{x}_t$ such that $f(\mathbf{x}_t) \leq \min_{\mathbf{x}^*} f(\mathbf{x}^*) + \varepsilon$ in total time*

$$\tilde{O}\left(m + \sqrt{\kappa d}\right) d \log^2\left(\frac{1}{\varepsilon}\right) \log\log\left(\frac{1}{\varepsilon}\right).$$

The above result strictly improves upon the best known running time for first order methods achieved by acceleration when we are in the setting when $\kappa > m > d$.

In all of our results stated above κ corresponds to the condition number of the underlying problem. In particular we assume some strong convexity for the underlying problem. This is a standard assumption which is usually enforced by the addition of the ℓ_2 regularizer. In stating our results formally we stress on the nuances between different notions of the condition number (ref. Section 2), and we state our results precisely with respect to these notions. In general, all of our generalization/relaxations of the condition number are smaller than $\frac{1}{\lambda}$ where λ is the coefficient of the added ℓ_2 regularizer which is usually taken to be the condition number of the problem. The condition of strong convexity has been relaxed in literature by introducing proximal methods. It is an interesting direction to adapt our results in those setting which we leave for future work.

We also remark that all of our results focus on the very high accuracy regime. In general the benefits of linear convergence and second-order methods can be seen to be effective only when considerably small error is required. This is also the case for recent advances in fast first-order methods where their improvement over stochastic gradient descent becomes apparent only in the high accuracy regime, and our experiments too demonstrate that second-order methods improve upon fast first-order methods in the regime of very high accuracy.

We further consider the special case when the function f is self-concordant. Self-concordant functions are a sub-class of convex functions which have been extensively studied in convex optimization literature in the context of Interior Point Methods [Nem04]. For self-concordant functions we propose an algorithm (Algorithm 5) which achieves linear convergence with running time guarantees independent of the condition number. We prove the formal running time guarantee as Theroem 6.2.

We remark that even though we prove that LISSA-Sample has the best running time in theory in the appropriate regime of parameters, we believe our main contribution to be a demonstration of the fact that second-order methods are comparable and can often be better than first-order methods for convex optimization in the large data regime even when measured on the absolute scale of running time both in theory and in practice.

1.1 Overview of Techniques

LiSSA: The key idea underlying LiSSA is the use of the Taylor expansion to construct a natural estimator of the inverse. Indeed, as can be seen from the description of the estimator in Section 3.1, the estimator we construct becomes unbiased as we include more and more terms in the series. We note that this is not the case with estimators that were considered in previous works such as [EM15], and so we therefore consider our estimator to be more natural. In the implementation of the algorithm we achieve the optimal bias/variance trade-off by truncating the series appropriately.

An important observation underlying our linear time $O(d)$ step is that for GLM functions, $\nabla^2 f_i(\mathbf{x})$ has the form $\alpha \mathbf{v}_i \mathbf{v}_i^T$ where α is a scalar dependent on $\mathbf{v}_i^T \mathbf{x}$. A single step of LiSSA requires us to efficiently compute $\nabla^2 f_i(\mathbf{x})\mathbf{b}$ for a given vector $\mathbf{b}$. In this case it can be seen that the matrix-vector product reduces to a vector-vector product, giving us an $O(d)$ time update.

LiSSA-Sample: LiSSA-Sample is based on Algorithm 2 which represents a general family of algorithms that couples the quadratic minimization view of Newton's method with any efficient first-order method. In essence Newton's method allows us to reduce (up to $\log \log$ factors) the optimization of a general convex function to solving intermediate quadratic or ridge regression problems. Such a reduction is useful in two ways.

First, as we demonstrate through our algorithm LiSSA-Sample, the quadratic nature of ridge regression problems allows us to leverage powerful sampling techniques, leading to an improvement over the running time of the best known accelerated first-order method. On a high level this improvement comes from the fact that when solving system of m linear equations in d dimensions, a constant number of passes through the data is enough to reduce the system to $O(d \log(d))$ equations. We carefully couple this principle and the computation required with accelerated first-order methods to achieve the running times for LISSA-Sample. The result for the quadratic sub-problem (ridge regression) is stated in Theorem 5.1, and the result for convex optimization is stated in Theorem 5.2.

The second advantage of the reduction to quadratic sub-problems comes from the observation that the intermediate quadratic sub-problems can be better conditioned than the function itself, allowing us a better choice of the step size in practice. We define these local notions of condition number formally in Section 2 and summarize the typical benefits for such algorithms in Theorem 4.1. We note that in theory this is not a significant improvement; however, in practice we believe that this could be significant and lead to runtime improvements as evidenced by our experiments for LiSSA.

To achieve the bound for LiSSA-Sample we extend the definition and procedure for sampling via leverage scores described in [CLM$^+$15] to the case when the matrix is given as a sum of PSD matrices and not just rank one matrices. We reformulate and reprove the theorems proved in [CLM$^+$15] in this context. These may be of independent interest.

1.2 Comparison with Related Work

In this section we aim to provide a short summary of the key ideas and results underlying optimization methods for large scale ML. Our main purpose in this section is to compare and contrast our results with the state-of-the-art methods in terms of the theoretical guarantees of the running time to the best of our knowledge. We divide the summary into three high level principles: first-order Gradient based methods, second-order Hessian based methods, and Quasi-Newton methods. For the sake of brevity we will restrict our summary to results in the case when the objective is strongly convex, which as justified above is usually ensured in practice by the addition of an appropriate regularizer. In such settings the main focus is usually to obtain algorithms which have provably linear convergence and fast implementations.

First Order Methods First-order methods have dominated the space of optimization algorithms for ML owing largely to the fact that they can always be implemented in time proportional to the underlying dimension (or sparsity). Vanilla gradient descent is well known to converge linearly to the optimum with a rate of convergence that is dependent inversely upon the condition number of the objective. In the large data regime, Stochastic first-order methods (introduced and analyzed first in [RM51]) have proven especially successful. Vanilla SGD, however, converges sub-linearly even in the strongly convex setting. A significant advancement in terms of the running time of first-order methods was achieved recently by a clever merging of stochastic gradient descent with its full version to provide *variance reduction*. The representative algorithms in this space are SAGA [RSB12, DBLJ14] and SVRG [JZ13, ZMJ13]. The key technical achievement of the above algorithms is to relax the running time dependence on m (the number of training examples) and κ (the condition number) from a product to a sum. Another algorithm which achieves similar running time guarantees is based on dual coordinate ascent, SDCA [SSZ13].

Further improvements over SAGA, SDCA and SVRG have been obtained by applying the classical idea of *acceleration* emerging from the seminal work of Nesterov [Nes83]. The progression of work here includes an accelerated version of SDCA [SSZ16], Catalyst [LMH15] which provides a generic framework to accelerate first order algorithms, and Katyusha [Zhu16] which introduces the concept of negative momentum to extend acceleration for variance reduced algorithms beyond the strongly convex setting. The key technical achievement of accelerated methods in general is to reduce the dependence on condition number from linear to a square root. We summarize these results in Table 1.

LiSSA places itself naturally into the space of fast first-order methods by having a running time dependence that is comparable to SAGA/SVRG (ref. Table 1). In LISSA-Sample we leverage the quadratic structure of the sub-problem for which efficient sampling techniques have been developed in the literature and utilize accelerated first-order methods to improve the running time in the case when the underlying dimension is much smaller than the number of training examples. Indeed, to the best of our knowledge LISSA-Sample is the fastest (theoretically) known algorithm under the condition $m >> d$. Such an improvement seems out of hand for the present first-order methods as it seems to strongly leverage the quadratic nature of the sub-problem to reduce its size. We summarize these results in Table 1.

Algorithm	Runtime	Reference
SVRG, SAGA, SDCA	$(md + O(\hat{\kappa}d)) \log(\frac{1}{\varepsilon})$	[JZ13, ZMJ13, RSB12, SSZ13]
LiSSA	$(md + O(\hat{\kappa}_l)S_1) \log(\frac{1}{\varepsilon})$	
AccSDCA, Catalyst, Katyusha	$\tilde{O}\left(md + d\sqrt{\hat{\kappa}m}\right) \log(\frac{1}{\varepsilon})$	[SSZ16, LMH15, Zhu16]
LiSSA-Sample	$\tilde{O}\left(md + d\sqrt{\kappa_{sample} d}\right) \log^2(\frac{1}{\varepsilon})$	

Table 1: Run time comparisons. For definitions of the various notions of condition number refer to Section 2

Second Order Methods Second-order methods such as Newton's method have classically been utilized in optimization in many different settings including development of Interior Point Methods [Nem04] for general convex programming. The key advantage of Newton's method is the linear-quadratic convergence rate which is achieved by the algorithm. However, if implemented naively Newton's method has two significant issues, namely that the standard analysis requires the full Hessian calculation which costs $O(md^2)$, an expense not suitable for ML applications, and the matrix inversion typically requires $O(d^3)$ time. These issues were addressed recently in the work NewSamp [EM15] which tackled the first issue by subsampling and the second issue by low rank projections. We improve upon [EM15] by defining a more *natural* estimator for the Hessian inverse and by demonstrating the estimator can be computed in time proportional to $O(d)$. We also point the reader to the works of [Mar10, BCNN11] which incorporate the idea of taking samples of the Hessian; however, these works do not provide precise running time guarantees on their proposed algorithm based on problem specific parameters.

Quasi-Newton Methods The expensive computation of the Newton step has also been tackled via estimation of the curvature from the change in gradients. These algorithms are generally known as Quasi-Newton methods stemming from the seminal BFGS algorithm [Bro70, Fle70, Gol70, Sha70]. We would like to point the reader to the book [NW06] which is an excellent reference for the algorithm and its limited memory variant (L-BFGS). The more recent work in this area has focused on analyzing Stochastic Quasi-Newton methods which were proposed and analyzed in various different settings by [SYG07, MR14, BHNS14]. These works typically achieve sub-linear convergence to the optimum. A significant advancement in this line of work was provided by [MNJ16] who propose an algorithm based on L-BFGS by incorporating ideas from variance reduction to achieve linear convergence to the optimum in the strongly convex setting. Although the algorithm achieves linear convergence, the running time of the algorithm depends very poorly on the condition number (as acknowledged by the authors). Indeed, in applications that interest us, the condition number is not necessarily a constant as is typically assumed to be the case of the theoretical results in [MNJ16].

Our key observation of linear time Hessian-vector products computations for ML applications provides evidence that in such instances, getting true Hessian information is efficient enough to alleviate the need for Quasi-Newton information via gradients.

1.3 Organization of the Paper

The paper is organized as follows. We first present the necessary definitions, notations and conventions adopted by us through the paper in Section 2. We then describe our estimator for LiSSA, as well as state and prove the convergence guarantee for LiSSA in Section 3. After presenting a generic procedure to couple first-order methods with Newton's method in 4, we present LiSSA-Sample and the associated fast quadratic solver in Section 5. We then present our results regarding self-concordant functions in Section 6. Finally we present an experimental evaluation of LiSSA in Section 7.

2 Preliminaries

We adopt the convention of denoting vectors and scalars in lowercase, matrices in uppercase, and vectors in boldface. We will use $\|\cdot\|$ without a subscript to denote the ℓ_2 norm for vectors and the spectral norm for matrices. Throughout the paper we denote $\mathbf{x}^* \triangleq \operatorname{argmin}_{\mathbf{x}\in\mathcal{K}} f(\mathbf{x})$. A convex function f is defined to be α-strongly convex and β-smooth if, for all $\mathbf{x}, \mathbf{y}$,

$$\nabla f(\mathbf{x})^\top(\mathbf{y}-\mathbf{x}) + \frac{\beta}{2}\|\mathbf{y}-\mathbf{x}\|^2 \geq f(\mathbf{y}) - f(\mathbf{x}) \geq \nabla f(\mathbf{x})^\top(\mathbf{y}-\mathbf{x}) + \frac{\alpha}{2}\|\mathbf{y}-\mathbf{x}\|^2.$$

The following is a well known fact about the inverse of a matrix A s.t. $\|A\| \leq 1$ and $A \succeq 0$:

$$A^{-1} = \sum_{i=0}^{\infty}(I-A)^i. \tag{2}$$

Definition of Condition Numbers We now define several measures for the condition number of a function f. The differences between these notions are subtle and we use them to precisely characterize the running time for our algorithms [2].

For an α-strongly convex and β-smooth function f, the well notion of the condition number of the function is defined as $\kappa(f) \triangleq \frac{\beta}{\alpha}$ or κ when the function is clear from the context. Note that by definition this corresponds to the following notion

$$\kappa \triangleq \frac{\max_x \lambda_{\max}(\nabla^2 f)}{\min_x \lambda_{\min}(\nabla^2 f)}$$

We define a slightly relaxed notion of condition number where the *max* moves out of the fraction above. We refer to this notion as a *local* condition number κ_l as compared to the *global* condition

[2] During initial reading we suggest the reader to skip the subtlety with these notions with the knowledge that all of these notions are smaller than the pessimistic bound one can achieve by considering a value proportional to $O(\lambda^{-1})$ where λ is the coefficient with the ℓ_2 regularizer.

number κ defined above. It follows that $\kappa_l \leq \kappa$

$$\kappa_l \triangleq \max_{\mathbf{x}} \frac{\lambda_{\max}(\nabla^2 f(\mathbf{x}))}{\lambda_{\min}(\nabla^2 f(\mathbf{x}))}$$

The above notions are defined for any general function f but in the case of ERM functions of the form $f(\mathbf{x}) = \frac{1}{m}\sum_{k=1}^{m} f_k(\mathbf{x})$, a further distinction is made with regard to the component functions. We refer to such definitions of the condition number with $\hat{\kappa}$. In such cases one typically assumes the following bound on each component $\beta_{max}(\mathbf{x}) \triangleq \max_k \lambda_{max}(\nabla^2 f_k(\mathbf{x}))$. The running times of algorithms like SVRG depend on the following notion of condition number

$$\hat{\kappa} = \frac{\max_{\mathbf{x}} \beta_{\max}(\mathbf{x})}{\min_{\mathbf{x}} \lambda_{\min}(\nabla^2 f(\mathbf{x}))}$$

Similarly, we define a notion of local condition number for $\hat{\kappa}$, namely

$$\hat{\kappa}_l \triangleq \max_{\mathbf{x}} \frac{\beta_{max}(\mathbf{x})}{\lambda_{min}(\nabla^2 f(\mathbf{x}))}$$

As before one can immediately see that $\hat{\kappa}_l \leq \hat{\kappa}$

For our (admittedly pessimistic) bounds on the variance we also need a per component strong convexity bound $\alpha_{min}(\mathbf{x}) \triangleq \min_k \lambda_{min}(\nabla^2 f_k(\mathbf{x}))$. We can now define

$$\hat{\kappa}_l^{max} \triangleq \max_{\mathbf{x}} \frac{\beta_{max}(\mathbf{x})}{\alpha_{min}(\mathbf{x})}$$

Assumptions: In light of the previous definitions we make the following assumptions about the given function $f = \frac{1}{m}\sum_{k=1}^{m} f_k(\mathbf{x})$ to make the analysis easier. Firstly we assume that the regularizer term has been divided equally and included in f_k. We further assume that each $\nabla^2(f_k) \preceq I$ [3]. we also assume that f is α-strongly convex and β-smooth, $\hat{\kappa}_l$ is the associated local condition number and $\nabla^2 f$ has a Lipschitz constant bounded by M.

We now collect key concepts and pre-existing results that we use for our analysis in the rest of the paper.

Matrix Concentration The following lemma is a standard concentration of measure result for sums of independent matrices [4]. An excellent reference for this material is [Tro12].

Theorem 2.1 (Matrix Bernstein [Tro12])**.** *Consider a finite sequence $\{X_k\}$ of independent, random, Hermitian matrices with dimension d. Assume that*

$$\mathbf{E}[X_k] = 0 \text{ and } \|X_k\| \leq R$$

Define $Y = \sum_k X_k$. Then we have for all $t \geq 0$

$$\mathbf{Pr}\left(\|Y\| \geq t\right) \leq d \exp\left(\frac{-t^2}{4R^2}\right)$$

[3]The scaling is without loss of generality even when looking at additive errors since due to the linear convergence this gets picked up in the log-term.

[4]The theorem in the reference states the inequality for more nuanced bounded variance case. We only state the simpler bounded spectral norm case which suffices for our purposes.

Accelerated SVRG The following theorem was proved in [LMH15].

Theorem 2.2. *Given a function $f(\mathbf{x}) = sum_{i=1}^{m} f_i(\mathbf{x})$ with condition number κ, the accelerated version of SVRG via Catalyst [LMH15] finds an ε approximate minimum with probability $1-\delta$ in time*

$$\tilde{O}(md + min(\sqrt{\kappa m}, \kappa)d) \log\left(\frac{1}{\varepsilon}\right)$$

Sherman-Morrison Formula The following is a well known expression for writing the inverse of rank one perturbations of matrices

$$\left(A + \mathbf{v}\mathbf{v}^T\right)^{-1} = A^{-1} - \frac{A^{-1}\mathbf{v}\mathbf{v}^T A^{-1}}{1 + \mathbf{v}^T A^{-1}\mathbf{v}}$$

3 LiSSA: Linear (time) Stochastic Second-Order Algorithm

3.1 Estimators for the Hessian Inverse

Based on a recursive reformulation of the Taylor expansion (Equation 2) we now describe an unbiased estimator of the Hessian. For a matrix A define A_j^{-1} as the first j terms in the Taylor expansion, i.e.,

$$A_j^{-1} = \sum_{i=0}^{j} (I - A)^i \text{ or equivalently } A_j^{-1} = I + (I - A)A_{j-1}^{-1}.$$

Note that $\lim_{j\to\infty} A_j^{-1} \to A^{-1}$. Using the above recursive formulation, we now describe an unbiased estimator of $\nabla^{-2} f$ by deriving an unbiased estimator $\tilde{\nabla}^{-2} f_j$ for $\nabla^{-2} f_j$.

Definition 3.1 (Estimator)**.** *Given j independent and unbiased samples $\{X_1 \ldots X_j\}$ of the Hessian $\nabla^2 f$, define $\{\tilde{\nabla}^{-2} f_0 \ldots \tilde{\nabla}^{-2} f_j\}$ recursively as follows:*

$$\tilde{\nabla}^{-2} f_0 = I \text{ and } \tilde{\nabla}^{-2} f_t = I + (I - X_j)\tilde{\nabla}^{-2} f_{t-1}.$$

It can be readily seen that $\mathbf{E}[\tilde{\nabla}^{-2} f_j] = \nabla^{-2} f_j$ and therefore $\mathbf{E}[\tilde{\nabla}^{-2} f_j] \to \nabla^{-2} f$ as $j \to \infty$ giving us an unbiased estimator in the limit.

Remark 3.2. *One can also define and analyze a simpler (non-recursive) estimator based on directly sampling terms from the series Equation (2). Theoretically one can get similar guarantees for the estimator; however, empirically our proposed estimator is superior in performance.*

3.2 Algorithm

In this section we describe our main algorithm LiSSA (Algorithm 1). Our algorithm runs in two phases: in the first phase it runs any efficient first-order method FO for T_1 steps to shrink the function value to the regime where we can then show linear convergence for our algorithm. It further uses the estimator defined in Definition 3.1 in place of the true Hessian inverse to take Newton steps. We use two parameters S_1, S_2 to define the Newton step. S_2 represents the depth to which we capture the Taylor expansion. S_1 represents the number of unbiased estimators of the Hessian inverse we average to get better concentration for our estimator. In the algorithm, we compute the average Newton step directly, which can be computed in linear time as observed earlier, instead of estimating the Hessian inverse.

Algorithm 1 LiSSA: Linear (time) Stochastic Second-Order Algorithm

```
**Input:** $T$, $f(\mathbf{x}) = \sum_{k=1}^{m} f_k(\mathbf{x})$, $S_1$, $S_2$, $T_1$
$\mathbf{x}_1 = FO(f(\mathbf{x}), T_1)$
**for** $t = 1$ to $T$ **do**
  **for** $i = 1$ to $S_1$ **do**
    $X_{[i,0]} = \nabla f(\mathbf{x}_t)$
    **for** $j = 1$ to $S_2$ **do**
      Sample $\tilde{\nabla}^2 f_{[i,j]}(\mathbf{x}_t)$ uniformly from $\{\nabla^2 f_k(\mathbf{x}_t) \mid k \in [m]\}$
      $X_{[i,j]} = \nabla f(\mathbf{x}_t) + (I - \tilde{\nabla}^2 f_{[i,j]}(\mathbf{x}_t))X_{[i,j-1]}$
    **end for**
    $X_{[i]} = X_{[i,S_2]}$
  **end for**
  $X_t = 1/S_1 \left(\sum_{i=1}^{S_1} X_{[i,S_2]}\right)$
  $\mathbf{x}_{t+1} = \mathbf{x}_t - X_t$
**end for**
**return** $\mathbf{x}_{T+1}$
```

3.3 Main Theorem

In this section we present our main theorem which analyzes the convergence properties of LiSSA. Define $FO(M, \hat{\kappa}_l)$ to be the total time required by a first order algorithm to achieve accuracy $\frac{1}{4\hat{\kappa}_l M}$.

Theorem 3.3. *Consider Algorithm 1. Set parameters as follows:* $T_1 = FO(M, \hat{\kappa}_l)$, $S_1 = O((\hat{\kappa}_l^{max})^2 \ln(\frac{d}{\delta}))$, $S_2 \geq 2\hat{\kappa}_l \ln(4\hat{\kappa}_l)$. *The following guarantee holds for every* $t \geq T_1$ *with probability* $1-\delta$,

$$\|\mathbf{x}_{t+1} - \mathbf{x}^*\| \leq \frac{\|\mathbf{x}_t - \mathbf{x}^*\|}{2}.$$

Moreover, we have that each step of the algorithm takes at most $\tilde{O}(md + (\hat{\kappa}_l^{max})^2 \hat{\kappa}_l d^2)$ *time. Additionally if* f *is GLM, then each step of the algorithm can be run in time* $O(md + (\hat{\kappa}_l^{max})^2 \hat{\kappa}_l d)$.

As an immediate corollary we get that:

Corollary 3.4. *For a GLM function* $f(\mathbf{x})$ *Algorithm 1 returns a point* $\mathbf{x}_t$ *such that with probability at least* $1-\delta$

$$f(\mathbf{x}_t) \leq \min_{\mathbf{x}^*} f(\mathbf{x}^*) + \varepsilon$$

in total time $\tilde{O}(m + (\hat{\kappa}_l^{max})^2 \hat{\kappa}_l) d \ln\left(\frac{1}{\varepsilon}\right)$ *for* $\varepsilon \to 0$.

In the above theorems $\tilde{O}$ hides log factors of $\kappa, d, \frac{1}{\delta}$. We note that our bound $(\hat{\kappa}_l^{max})^2$ on the variance is very pessimistic and can likely be improved to a more average quantity. However, since it does not have much of an effect in practice, we have not tried to optimize it further.

We now prove our main theorem about the convergence of LiSSA Theorem 3.3.

Proof of Theorem 3.3. Note that since we use a first order algorithm to get a solution of accuracy at least $\frac{1}{4\hat{\kappa}_l M}$ we have that

$$\|\mathbf{x}_1 - \mathbf{x}^*\| \leq \frac{1}{4\hat{\kappa}_l M} \tag{3}$$

As can be easily seen from Definition 3.1 a single step of our algorithm is equivalent to $\mathbf{x}_{t+1} = \mathbf{x}_t - \tilde{\nabla}^{-2} f(\mathbf{x}_t)\nabla f(\mathbf{x}_t)$ where $\tilde{\nabla}^{-2} f(\mathbf{x}_t)$ is the average of S_1 independent estimators $\tilde{\nabla}^{-2} f(\mathbf{x}_t)_{S_2}$. We now make use of the following lemma.

Lemma 3.5. *Let $\mathbf{x}_{t+1} = \mathbf{x}_t - \tilde{\nabla}^{-2} f(\mathbf{x}_t)\nabla f(\mathbf{x}_t)$, as per a single iteration of Algorithm 1, and S_1, S_2 are as defined in Algorithm 1. Then if we choose $S_2 \geq 2\hat{\kappa}_l \ln(2\hat{\kappa}_l)$ we get the following guarantee on the convergence rate for every step with probability $1 - \delta$*

$$\|\mathbf{x}_{t+1} - \mathbf{x}^*\| \leq \gamma \|\mathbf{x}_t - \mathbf{x}^*\| + M\|\nabla^{-2} f(\mathbf{x}_t)\|\|\mathbf{x}_t - \mathbf{x}^*\|^2$$

where $\gamma = 16\hat{\kappa}_l^{max}\sqrt{\frac{\ln(d\delta^{-1})}{S_1}} + \frac{1}{16}$.

Substituting the values of S_1 and S_2, combining Equation (3) and Lemma 3.5, and noting that $\|\nabla^{-2} f(\mathbf{x}_t)\| \leq \hat{\kappa}_l$, we have that at the start of the Newton phase the following inequality holds

$$\|\mathbf{x}_{t+1} - \mathbf{x}^*\| \leq \frac{\|\mathbf{x}_t - \mathbf{x}^*\|}{4} + M\hat{\kappa}_l^{max}\|\mathbf{x}_t - \mathbf{x}^*\|^2 \leq \frac{\|\mathbf{x}_t - \mathbf{x}^*\|}{2}$$

It can be shown via induction that the above property holds for all $t \geq T_1$ which concludes the proof. □

We now provide a proof of Lemma 3.5.

Proof of Lemma 3.5. Define $\chi(\mathbf{x}_t) = \int_0^1 \nabla^2 f(\mathbf{x}^* + \tau(\mathbf{x}_t - \mathbf{x}^*))d\tau$. Note that $\nabla f(\mathbf{x}_t) = \chi(\mathbf{x}_t)(\mathbf{x}_t - \mathbf{x}^*)$. Following an analysis similar to that of Nesterov [Nes04], we have that

$$\begin{aligned} \|\mathbf{x}_{t+1} - \mathbf{x}^*\| &= \|\mathbf{x}_t - \mathbf{x}^* - \tilde{\nabla}^{-2} f(\mathbf{x}_t)\nabla f(\mathbf{x}_t)\| \\ &= \|\mathbf{x}_t - \mathbf{x}^* - \tilde{\nabla}^{-2} f(\mathbf{x}_t)\chi(\mathbf{x}_t)(\mathbf{x}_t - \mathbf{x}^*)d\tau\| \\ &\leq \|I - \tilde{\nabla}^{-2} f(\mathbf{x}_t)\chi(\mathbf{x}_t)\|\|\mathbf{x}_t - \mathbf{x}^*\| \end{aligned}$$

Following from the previous equations we have that

$$\frac{\|\mathbf{x}_{t+1} - \mathbf{x}^*\|}{\|\mathbf{x}_t - \mathbf{x}^*\|} \leq \|I - \tilde{\nabla}^{-2} f(\mathbf{x}_t)\chi(\mathbf{x}_t)\| = \|\underbrace{I - \nabla^{-2} f(\mathbf{x}_t)\chi(\mathbf{x}_t)}_{a} - \underbrace{\left(\tilde{\nabla}^{-2} f(\mathbf{x}_t) - \nabla^{-2} f(\mathbf{x}_t)\right)\chi(\mathbf{x}_t)}_{b}\|$$

We now analyze the above two terms a, b separately.

$$\begin{aligned} \|a\| &= \|I - \nabla^{-2} f(\mathbf{x}_t)\chi(\mathbf{x}_t)\| \\ &\leq \|\nabla^{-2} f(\mathbf{x}_t)\int_0^1 \nabla^2 f(\mathbf{x}_t) - \nabla^2 f(\mathbf{x}^* + \tau(\mathbf{x}_t - \mathbf{x}^*))d\tau\| \\ &\leq M\|\nabla^{-2} f(\mathbf{x}_t)\|\|\mathbf{x}_t - \mathbf{x}^*\| \end{aligned}$$

The second inequality follows from the Lipschitz bound on the Hessian. The second term can be bounded as follows

$$\|b\| = \left(\|\left(\tilde{\nabla}^{-2} f(\mathbf{x}_t) - \nabla^{-2} f(\mathbf{x})\right)\|\|\chi(\mathbf{x}_t)\|\right) \leq \gamma$$

The previous claim follows from Lemma 3.6 which shows a concentration bound on the sampled estimator and by noting that due to our assumption on the function we have that $\forall \mathbf{x}\ \|\nabla^2 f(\mathbf{x})\| \leq 1$ and hence $\|\chi(\mathbf{x})\| \leq 1$.

Putting the above two bounds together and using triangle inequality we get that

$$\frac{\|\mathbf{x}_{t+1} - \mathbf{x}^*\|}{\|\mathbf{x}_t - \mathbf{x}^*\|} \leq M\|\nabla^{-2} f(\mathbf{x}_t)\|\|\mathbf{x}_t - \mathbf{x}^*\| + \gamma$$

which concludes the proof. $\square$

Lemma 3.6. *Let $\tilde{\nabla}^{-2} f(\mathbf{x}_t)$ be the average of S_1 independent samples of $\tilde{\nabla}^{-2} f(\mathbf{x}_t)_{S_2}$ defined in 3.1 and used in the per step update of Algorithm 1 and let $\nabla^2 f(\mathbf{x}_t)$ be the true Hessian. If we set $S_2 \geq 2\hat{\kappa}_l \ln(\hat{\kappa}_l S_1)$, then we have that,*

$$\mathbf{Pr}\left(\|\tilde{\nabla}^{-2} f(\mathbf{x}_t) - \nabla^{-2} f(\mathbf{x}_t)\| > 16\hat{\kappa}_l^{max}\sqrt{\frac{\ln(\frac{d}{\delta})}{S_1}} + 1/16\right) \leq \delta$$

Proof of Lemma 3.6. First note the following statement which is a straightforward implication of our construction of the estimator

$$\mathbf{E}[\tilde{\nabla}^{-2} f_i(\mathbf{x}_t)] = \mathbf{E}[\tilde{\nabla}^{-2} f_i(\mathbf{x}_t)_{S_2}] = \sum_{i=1}^{S_2} (I - \nabla^2 f(\mathbf{x}_t))^i$$

We also know from Equation 2 that for matrices X such that $\|X\| \leq 1$

$$X^{-1} = \sum_{i=0}^{\infty} (I - X)^i$$

Since we have scaled the function such that $\|\nabla^2 f_k\| \leq 1$, it follows that

$$\nabla^{-2} f(\mathbf{x}_t) = \mathbf{E}\left[\tilde{\nabla}^{-2} f_i(\mathbf{x}_t)\right] + \sum_{i=S_2}^{\infty} (I - \nabla^2 f(\mathbf{x}_t))^i \tag{4}$$

Also note that we have $\nabla^2 f(\mathbf{x}_t) \succeq \frac{I}{\hat{\kappa}_l}$ which implies that $\|I - \nabla^2 f(\mathbf{x}_t)\| \leq 1 - \frac{1}{\hat{\kappa}_l}$. Observing the second term in the above equation,

$$\begin{aligned}
\|\sum_{i=S_2}^{\infty} (I - \nabla^2 f(\mathbf{x}_t))^i\| &\leq \|(I - \nabla^2 f(\mathbf{x}_t))\|^{S_2} \left(\sum_{i=0}^{\infty} \|I - \nabla^2 f(\mathbf{x}_t)\|^i\right) \\
&\leq (1 - \frac{1}{\hat{\kappa}_l})^{S_2} \left(\sum_{i=0}^{\infty} (1 - \frac{1}{\hat{\kappa}_l})^i\right) \\
&\leq (1 - \frac{1}{\hat{\kappa}_l})^{S_2} \hat{\kappa}_l \\
&\leq \exp\left(-\frac{S_2}{\hat{\kappa}_l}\right) \hat{\kappa}_l
\end{aligned}$$

Since we have chosen $S_2 \geq 2\hat{\kappa}_l ln(4\hat{\kappa}_l)$ we get that the above term is bounded by 1/16. We will now show using Matrix Bernstein inequality 2.1 that the estimate $\tilde{\nabla}^{-2} f$ is concentrated around its expectation. To apply the inequality we first need to bound the spectral norm of each random variable. Towards that end we note that $\tilde{\nabla}^{-2} f_{S_2}$ has maximum spectral norm bounded by

$$\|\tilde{\nabla}^{-2} f_{S_2}\| \leq \sum_{i=0}^{S_2} (1 - 1/\hat{\kappa}_l^{max})^i \leq \hat{\kappa}_l^{max}$$

We can now apply the Matrix Bernstein inequality 2.1 which gives the following

$$\mathbf{Pr}\left(\|\tilde{\nabla}^{-2} f - \mathbf{E}[\tilde{\nabla}^{-2} f]\| > \varepsilon\right) \leq 2d\, exp\left(\frac{-\varepsilon^2 S_1}{64(\hat{\kappa}_l^{max})^2}\right)$$

Setting $\varepsilon = 16\hat{\kappa}_l^{max}\sqrt{\frac{ln(\frac{d}{\delta})}{S_1}}$ gives us that the probability above is bounded by δ. Now putting together the bounds and Equation (4) we get the required result.

□

3.4 Leveraging Sparsity

A key property of real-world datasets is that although the input is a high dimensional vector, the number of non-zero entries is typically very low. The following theorem shows that LiSSA can be implemented in a way to leverage the underlying sparsity of the data. Our key observation is that for GLM functions, the rank one Hessian-vector product can be performed in $O(s)$ time where s is the sparsity of the input $\mathbf{x}_k$.

Theorem 3.7. *For GLM functions Algorithm 1 returns a point $\mathbf{x}_t$ such that with probability at least $1 - \delta$*

$$f(\mathbf{x}_t) \leq \min_{\mathbf{x}^*} f(\mathbf{x}^*) + \varepsilon$$

in total time $\tilde{O}(ms + (\hat{\kappa}_l^{max})^2 \hat{\kappa}_l s) \ln\left(\frac{1}{\varepsilon}\right)$ for $\varepsilon \to 0$.

We will prove the following theorem after which Theorem 3.7 follows immediately.

Theorem 3.8. *Consider Algorithm 1, let f be of the form described above, and let s be such that the number of non zero entries in $\mathbf{x}_i$ is bounded by s. Then each step of the algorithm can be implemented in time $O(ms + (\kappa_l^{max})^2 \kappa_l s)$.*

Proof of Theorem 3.8. Proof by induction. Let $c_0 = 1$, $\mathbf{v}_0 = \mathbf{0}$, and consider the update rules $c_{j+1} = 1 + (1-\lambda)c_j$ and $\mathbf{v}_{j+1} = (1-\lambda)\mathbf{v}_j - \tilde{\nabla}^2 f_{[i,j+1]}(\mathbf{x})(c_j \nabla f(\mathbf{x}) + \mathbf{v}_j)$, where λ is the regularization parameter. For the base case, note that $X_{[i,0]} = c_0 \nabla f(\mathbf{x}) + \mathbf{v}_0 = \nabla f(\mathbf{x})$, as is the case in Algorithm 1. Furthermore, suppose $X_{[i,j]} = c_j \nabla f(\mathbf{x}) + \mathbf{v}_j$. Then we see that

$$\begin{aligned}
X_{[i,j+1]} &= \nabla f(\mathbf{x}) + (I - \lambda I - \tilde{\nabla}^2 f_{[i,j+1]}(\mathbf{x}))X_{[i,j]} \\
&= \nabla f(\mathbf{x}) + ((1-\lambda)I - \tilde{\nabla}^2 f_{[i,j+1]}(\mathbf{x}))(c_j \nabla f(\mathbf{x}) + \mathbf{v}_j) \\
&= (1 + (1-\lambda)c_j)\nabla f(\mathbf{x}) + (1-\lambda)(\mathbf{v}_j) - \tilde{\nabla}^2 f_{[i,j+1]}(\mathbf{x})(c_j \nabla f(\mathbf{x}) + \mathbf{v}_j) \\
&= c_{j+1}\nabla f(\mathbf{x}) + \mathbf{v}_{j+1}
\end{aligned}$$

Note that updating c_{j+1} takes constant time, and $\tilde{\nabla}^2 f_{[i,j+1]}(\mathbf{x})(c_j \nabla f(\mathbf{x}))$ and $\tilde{\nabla}^2 f_{[i,j+1]}(\mathbf{x})\mathbf{v}_j$ can each be calculated in $O(s)$ time. It can also be seen that each products gives an s-sparse vector, so subtracting them from $(1-\lambda)(\mathbf{v}_j)$ takes $O(s)$ time. Thus, the overall time to update $\mathbf{v}_{j+1}$ is $O(s)$. Since $\nabla f(\mathbf{x})$ can be calculated in $O(ms)$ time, and since $\mathbf{v}_0$ is 0-sparse which implies the number of non-zero entries of $\mathbf{v}_j$ is at most js, it follows that the total time to calculate X_t is $O(ms + (\kappa_l^{max})^2 \kappa_l s)$.

□

4 LiSSA: Extensions

In this section we first describe a family of algorithms which generically couple first-order methods as sub-routines with second order methods. In particular we formally describe the algorithm LiSSA-Quad (Algorithm 2) and provide its runtime guarantee (Theorem 4.1). The key idea underlying this algorithm is that Newton's method essentially reduces a convex optimization problem to solving intermediate quadratic subproblems given by the second-order Taylor approximation at a point Q_t given by

$$Q_t(\mathbf{y}) = f(\mathbf{x}_{t-1}) + \nabla f(\mathbf{x}_{t-1})^T \mathbf{y} + \frac{\mathbf{y}^T \nabla^2 f(\mathbf{x}_{t-1}) \mathbf{y}}{2}$$

The above ideas provides an alternative implementation of our estimator for $\nabla^{-2} f(\mathbf{x})$ used in LiSSA. Consider running Gradient Descent on the above quadratic Q_t and let $\mathbf{y}_t^i$ be the i^{th} step in this process. By definition we have that

$$\mathbf{y}_t^{i+1} = \mathbf{y}_t^i - \nabla Q_t(\mathbf{y}_t^i) = (I - \nabla^2 f(\mathbf{x}_t))\mathbf{y}_t^i - \nabla f(\mathbf{x}_t)$$

It can be seen that the above expression corresponds exactly to the steps taken in LiSSA, the difference being that we use a sample of the Hessian instead of the true Hessian. Therefore LiSSA can also be interpreted as doing a partial stochastic gradient descent on the quadratic Q_t. It is partial because we have a precise estimate of gradient of the function f and a stochastic estimate for the Hessian. We note that this is essential for the linear convergence guarantees we show for LiSSA.

The above interpretation indicates towards any first-order linearly convergent scheme being used for approximating the minimizer of the quadratic Q_t. In particular consider any algorithm *ALG* which given a convex quadratic function Q_t and an error value ε produces a point $\mathbf{y}$ such that

$$\|\mathbf{y} - \mathbf{y}_t^*\| \leq \varepsilon \tag{5}$$

with probability at least $1-\delta_{ALG}$, where $\mathbf{y}_t^* = \operatorname{argmin} Q_t$. Let the total time taken by the algorithm *ALG* to produce the point be $T_{ALG}(\varepsilon, \delta_{ALG})$. For our applications we require *ALG* to be linearly convergent, i.e. T_{ALG} is proportional to $\log(\frac{1}{\varepsilon})$ with probability at least $1-\delta_{ALG}$.

Given such an algorithm *ALG*, LiSSA-Quad described in Algorithm 2 generically implements the above idea, modifying LiSSA by replacing the inner loop with the given algorithm *ALG*. The following is a meta-theorem about the convergence properties of LiSSA-Quad.

Theorem 4.1. *Given the function $f(\mathbf{x}) = \sum f_i(\mathbf{x})$ which is α strongly convex, let $\mathbf{x}^*$ be the minimizer of the function and $\{\mathbf{x}_t\}$ be defined as in Algorithm 2. Suppose the algorithm ALG satisfies condition (5) with probability $1-\delta_{ALG}$ under the appropriate setting of parameters ALG_{params}. Set the parameters in the algorithm as follows: $T_1 = T_{ALG}(1/4\alpha M)$, $T = \log\log(1/\varepsilon)$, $\delta_{ALG} = \delta/T$,*

Algorithm 2 LiSSA-Quad

```
**Input:** $T$, $f(\mathbf{x}) = \sum_{k=1}^{m} f_k(\mathbf{x})$, $ALG$, $ALG_{params}$, $T_1$, $\varepsilon$
$\mathbf{x}_0 = \text{ALG}\ (f(\mathbf{x}), T_1)$
**for** $t = 1$ to $T$ **do**
    $Q_t(\mathbf{y}) = \nabla f(\mathbf{x}_{t-1})^T \mathbf{y} + \frac{\mathbf{y}^T \nabla^2 f(\mathbf{x}_t) \mathbf{y}}{2}$
    $\mathbf{x}_t = A(Q_t, \varepsilon^2, A_{params})$
**end for**
**return** $\mathbf{x}_{T+1}$
```

where ε is the final error guarantee one wishes to achieve. Then we have that after T steps with probability at least $1-\delta$

$$\min_{t=\{1\ldots T\}} \|\mathbf{x}_t - \mathbf{x}^*\| \leq \varepsilon.$$

In particular, LiSSA-Quad(ALG) produces a point x such that

$$\|\mathbf{x} - \mathbf{x}^*\| \leq \varepsilon$$

in total time $O(T_{ALG}(\varepsilon, \delta_{ALG}) \log\log(1/\varepsilon))$ with probability at least $1-\delta$ for $\varepsilon \to 0$.

Note that for GLM functions, the $\nabla Q_t(\mathbf{y})$ at any point can be computed in time linear in d. In particular a full gradient of Q_t can be computed in time $O(md)$ and a stochastic gradient(corresponding to a stochastic estimate of the Hessian) in time $O(d)$. Therefore a natural choice for the algorithm ALG in the above are first-order algorithms which are linearly convergent, for example SVRG, SDCA, Acc-SDCA. Choosing a first order algorithm FO gives us a family of algorithms LISSA-Quad(FO), each with running time comparable to the running time of the underlying FO upto logarithmic factors. The following corollary summarizes the typical running time guarantees for LiSSA-Quad(FO) when FO is Acc-SVRG.

Corollary 4.2. *Given a GLM function $f(\mathbf{x})$, if ALG is replaced by Acc-SVRG [LMH15] then under a suitable setting of parameters, LiSSA-Quad produces a point $\mathbf{x}$ such that*

$$f(\mathbf{x}) - f(\mathbf{x}^*) \leq \varepsilon$$

with probability at least $1-\delta$, in total time $\tilde{O}(m + min\left\{\sqrt{\hat{\kappa}_l m}, \hat{\kappa}_l\right\}) d \log(1/\varepsilon) \log\log(1/\varepsilon)$.

Here the $\tilde{O}$ above hides logarithmic factors in κ, d, δ, but not in ε. Note that the above running times depend upon the condition number $\hat{\kappa}_l$ which as described in Section 2 can potentially provide better dependence compared to its global counterpart. In practice this difference could lead to faster running time for LiSSA-Quad as compared to the underlying first-order algorithm FO. We now provide a proof for Theorem 4.1.

Proof of Theorem 4.1. We run the algorithm A to achieve accuracy ε^2 on each of the intermediate quadratic functions Q_t and set $\delta_A = \delta/T$ which implies via a union bound that for all $t \leq T$

$$\|\mathbf{x}_{t+1} - \mathbf{x}_t^*\| \leq \varepsilon^2 \tag{6}$$

with probability at least $1-\delta$.

Assume that $\forall t < T$, $\|\mathbf{x}_t - \mathbf{x}^*\| \geq \varepsilon$ (otherwise the theorem is trivially true). Using the Newton's method analysis as before we get that $\forall t \leq T$

$$\begin{aligned}
\|\mathbf{x}_{t+1} - \mathbf{x}^*\| &\leq \|\mathbf{x}_t^* - \mathbf{x}^*\| + \|\mathbf{x}_{t+1} - \mathbf{x}_t^*\| \\
&\leq \|\mathbf{x}_t - \nabla^{-2} f(\mathbf{x}_t) \nabla f(\mathbf{x}_t) - \mathbf{x}^*\| + \|\mathbf{x}_{t+1} - \mathbf{x}_t^*\| \\
&\leq \frac{M}{4\alpha} \|\mathbf{x}_t - \mathbf{x}^*\|^2 + \varepsilon^2 \\
&\leq \left(\frac{M}{4\alpha} + 1 \right) \|\mathbf{x}_t - \mathbf{x}^*\|^2
\end{aligned}$$

where the second inequality follows from the analysis in the proof of Theorem 3.3 and Equation 6. We know that $\|\mathbf{x}_0 - \mathbf{x}_t\| \leq \sqrt{\frac{\alpha}{M}}$ from the initial run of the first order algorithm FO. Applying the above inductively and using the value of T prescribed by the theorem statement we get that $\|\mathbf{x}_T - \mathbf{x}^*\| \leq \varepsilon$. $\square$

5 Runtime Improvement through Fast Qudratic Solvers

The previous section establishes the reduction from general convex optimization to quadratic functions. In this section we show how we can leverage the fact that for quadratic functions the running time for accelerated first order methods can be improved in the regime when $\kappa > m >> d$. In particular we show the following theorem.

Theorem 5.1. *Given a vector $\mathbf{b} \in \mathbb{R}^d$ and a matrix $A = \frac{1}{m} \sum A_i$ where each A_i is of the form $A_i = \mathbf{v}_i \mathbf{v}_i^T + \lambda I$ for some $\mathbf{v}_i \in \mathbb{R}^d, \|\mathbf{v}_i\| \leq 1$ and $\lambda \geq 0$ a fixed parameter, Algorithm 4 computes a vector $\tilde{\mathbf{v}}$ such that $\|A^{-1}\mathbf{b} - \tilde{\mathbf{v}}\| \leq \varepsilon$ with probability at least $1 - \delta$ in total time*

$$\tilde{O}\left(md \log\left(\frac{1}{\varepsilon}\right) + \left(d + \sqrt{\kappa_{sample}(A) d} \right) d \log^2\left(\frac{1}{\varepsilon}\right) \right).$$

$\tilde{O}()$ contains factors logarithmic in $m, d, \kappa_{sample}(A), \|b\|, \delta$.

$\kappa_{sample}(A)$ is the condition number of an $O(d\log(d))$ sized sample of A and is formally defined in Equation (11). We can now use Algorithm 4 to compute an approximate Newton step by setting $A = \nabla^2 f(x)$ and $\mathbf{b} = \nabla f(x)$. We therefore propose LiSSA-Sample to be a variant of LiSSA-Quad where Algorithm 4 is used as the subroutine ALG and any first order algorithm can be used in the initial phase. The following theorem bounding the running time of LiSSA-Sample follows immediately from Theorem 4.1 and Theorem 5.1.

Theorem 5.2. *Given a GLM function $f(\mathbf{x}) = \sum_i f_i(\mathbf{x})$, let $\mathbf{x}^* = \operatorname{argmin} f(\mathbf{x})$. LiSSA-Sample produces a point $\mathbf{x}$ such that*

$$\|\mathbf{x} - \mathbf{x}^*\| \leq \varepsilon$$

with probability at least $1 - \delta$ in total time

$$\tilde{O}\left(\left(md \log\left(\frac{1}{\varepsilon}\right) + \left(d + \sqrt{\kappa_{sample}(f) d} \right) d \log^2\left(\frac{1}{\varepsilon}\right) \right) \log\log\left(\frac{1}{\varepsilon}\right) \right).$$

$\tilde{O}()$ contains factors logarithmic in $m, d, \kappa_{sample}(f), G, \delta$.

5.1 Fast Quadratic Solver - Outline

In this section we provide a short overview of Algorithm 4. To simplify the discussion, lets consider the case when we have to compute $A^{-1}\mathbf{b}$ for a $d \times d$ matrix A given as $A = \sum_{i=1}^{m} \mathbf{v}_i \mathbf{v}_i^T = VV^T$ where the i^{th} column of V is $\mathbf{v}_i$. The computation can be recast as minimization of a convex quadratic function $Q(\mathbf{y}) = \frac{\mathbf{y}^T A \mathbf{y}}{2} + \mathbf{b}^T y$ and can be solved up to accuracy ε in total time $\left(m + \sqrt{\kappa(A) m}\right) d \log(1/\varepsilon)$ as can be seen from Theorem 2.2 Algorithm 4 improves upon the running time bound in the case when $m > d$. In the following we provide a high level outline of the procedure which is formally described as Algorithm 4.

- Given A we will compute a low complexity constant spectral approximation B of A. Specifically $B = \sum_{i=1}^{O(d\log(d))} \mathbf{u}_i \mathbf{u}_i^T$ and $B \preceq A \preceq 2B$. This is achieved by techniques developed in Matrix Sampling/Sketching literature, especially [CLM+15]. The procedure requires solving a constant number of $O(d \log(d))$ sized linear systems which we do via accelerated SVRG.

- We use B as preconditioner and compute $BA^{-1}\mathbf{b}$ by minimizing the quadratic $\frac{\mathbf{y}^T AB^{-1}\mathbf{y}}{2} + \mathbf{b}^T \mathbf{y}$. Note that this quadratic is well conditioned and can be minimized using gradient descent. In order to compute the gradient of the quadratic which is given by $AB^{-1}\mathbf{y}$ we again use Accelerated SVRG to solve a linear system in B

- Finally we compute $A^{-1}\mathbf{b} = B^{-1}BA^{-1}\mathbf{b}$ using Accelerated SVRG to solve a linear system solver in B.

In the rest of the section we formally describe the procedure outlined and provide the necessary definitions. One key nuance we must take into account is the fact that based on our assumption we have included the regularizer term into the component functions. Due to this, the Hessian does not necessarily look like a sum of rank one matrices. Of course one can decompose the Identity matrix that appears due to the regularizer as a sum of rank one matrices but note that the procedure above requires that each of the sub-samples must have good condition number too in order to solve linear systems on them with Accelerated SVRG. Therefore the subsamples generated must look like sub-samples formed from the Hessians of component functions. For this purpose we extend the procedure and the definition for leverage scores described in [CLM+15] to the case when the matrix is given as a sum of PSD matrices and not just rank one matrices. We reformulate and reprove the basic theorems proved in [CLM+15] in this context. To maintain computational efficiency of the procedure we then make use of the fact that each of the PSD matrices actually is a rank one matrix plus the Identity matrix. We now provide the necessary preliminaries for the description of the algorithm and its analysis.

5.2 Preliminaries for Fast Quadratic Solver

For all the definitions and preliminaries below assume we are given a $d \times d$ PSD matrix $A \triangleq \sum_{i=1}^{m} A_i$ where A_i are also PSD matrices. Let $A \cdot B \triangleq Tr(B^T A)$ be the standard matrix dot product. Given two matrices A and B we say B is a λ spectral approximation of A if $\frac{1}{\lambda} A \preceq B \preceq A$.

Definition 5.3 (Generalized Matrix Leverage Scores)**.**

$$\tau_i(A) \triangleq A^+ \cdot A_i \tag{7}$$

$$\tau_i^B(A) \triangleq B^+ \cdot A_i \tag{8}$$

The following facts follows easily

$$\sum_{i=1}^{n} \tau_i(A) = Tr(\sum A^+ A_i) = Tr(A^+ A) = rank(A) \leq d \tag{9}$$

Fact 5.4. *If B is a λ spectral approximation of A then $\tau_i(A) \leq \tau_i^B(A) \leq \lambda\tau_i(A)$*

Given A of the form $A = \frac{1}{m}\sum A_i$ we define a sample of A of size r as the following. Consider a subset of indices of size r, $I = \{i_1 \dots i_r\} \subseteq [m]$. For every such sample I and given a weight vector $\mathbf{w} \in \mathbb{R}^r$, we can associate the following matrix

$$Sample(\mathbf{w}, I) \triangleq \sum_{j \in I} w_j A_j \tag{10}$$

When the sample is unweighted i.e. $\mathbf{w} = \mathbb{1}$ we will simply denote the above as $Sample(I)$. We can now define $\kappa_{sample}(A, r)$ to be

$$\kappa_{sample}(A, r) \triangleq \max_{I:|I| \geq r} \kappa(Sample(r, I)) \tag{11}$$

In words this is the worst case condition number of an unweighted sub-sample of size r of the matrix A. For our results we will be concerned with a $\Omega(d\log(d))$ sized sub-sample, i.e. the quantity $\kappa_{sample}(A, O(d\log(d)))$. We remind the reader that by definition κ_{sample} for Hessians of the functions we are concerned with is always larger than $1/\lambda$ where λ is the coefficient of the ℓ_2 regularizer.

The following lemma is a generalization of the leverage score sampling lemma (Lemma 4 [CLM+15]). The proof is very similar to the original proof in [CLM+15] and is included in the Appendix for completeness.

Lemma 5.5 (Spectral Approximation via Leverage Score Sampling)**.** *Given an error parameter $0 < \varepsilon < 1$, let $\mathbf{u}$ be a vector of leverage score overestimates, i.e. $\tau_i(A) \leq \mathbf{u}_i$, for all $i \in [m]$. Let α be a sampling rate parameter and let c be a fixed positive constant. For each matrix A_i, we define a sampling probability $p_i(\alpha) = min\{1, \alpha\mathbf{u}_i c\log d\}$. Let I be a random sample of indices drawn from $[m]$ by sampling each index with probability $p_i(\alpha)$. Define the weight vector $\mathbf{w}(\alpha)$ be the vector such that $\mathbf{w}(\alpha)_i = \frac{1}{p_i(\alpha)}$. By definition of weighted samples we have that*

$$Sample(\mathbf{w}(\alpha), I) = \sum_{i=1}^{m} \frac{1}{p_i(\alpha)} A_i \mathbb{1}_{x_i \sim p_i(\alpha)}(x_i = 1)$$

where x_i is a Bernoulli random variable with probability $p_i(\alpha)$.

If we set $\alpha = \varepsilon^{-2}$, $S = Sample(\mathbf{w}(\alpha), I)$ is formed by atmost $\sum_i \min\{1, \alpha\mathbf{u}_i c\log(d)\} \leq \alpha c\log(d)\|\mathbf{u}\|_1$ entries in the above sum and $\frac{1}{1+\varepsilon}S$ is a $\frac{1+\varepsilon}{1-\varepsilon}$ spectral approximation for A with probability at least $1 - d^{-c/3}$.

The following theorem is an analogue of the key theorem regarding uniform sampling (Theorem 1 [CLM+15]). The proof is identical to the original proof and is included in the Appendix for completeness.

Theorem 5.6 (Uniform Sampling)**.** *Given any $A = \sum_{i=1}^m A_i$ as defined above, let $S = \sum_{j=1}^r X_j$ be formed by unformly sampling r matrices $X_1 \dots X_m \sim \{A_i\}$ without repetition. Define*

$$\tilde{\tau}_i^S(A) = \begin{cases} \tau_i^S(A) & \text{if } \exists\, j \ \text{ s.t. } X_j = A_i \\ \tau_i^{S+A_i}(A) & \text{otherwise} \end{cases}$$

Then $\tilde{\tau}_i^S(A) \geq \tau_i(A)$ for all i and

$$\mathbb{E}\left[\sum_{i=1}^n \tilde{\tau}_i^S(A)\right] \leq \frac{md}{r}$$

Unlike the case of rank one matrices as in [CLM+15], it is not immediately clear how to efficiently compute $\tilde{\tau}_i$ as you cannot directly apply the Sherman Morrison formula. In our case though, since we have that each $A_i = \mathbf{v}_i \mathbf{v}_i^T + \lambda I$ we define slightly different estimates which are efficiently computable and prove that they work as well.

Theorem 5.7. *Given any $A = \sum_{i=1}^m A_i$ where each A_i is of the form $A_i = \mathbf{v}_i \mathbf{v}_i^T + \lambda I$. Let $S = \sum_{j=1}^m X_j$ be formed by unformly sampling r matrices $X_1 \dots X_m \sim \{A_i\}$ without repetition. Define*

$$\hat{\tau}_i^S(A) = \begin{cases} \mathbf{v}_i^T S^+ \mathbf{v}_i^T + \frac{d}{r} & \text{if } \exists\, j \ \text{ s.t. } X_j = A_i \\ \frac{1}{1+\frac{1}{\mathbf{v}_i^T (S+\lambda I)^+ \mathbf{v}_i^T}} + \frac{d}{r} & \text{otherwise} \end{cases}$$

Then $\hat{\tau}_i^S(A) \geq \tau_i(A)$ for all i and

$$\mathbb{E}\left[\sum_{i=1}^n \hat{\tau}_i^S(A)\right] \leq O\left(\frac{md}{r}\right)$$

Proof of Theorem 5.7. We will prove that

$$\frac{d}{m} \geq \hat{\tau}_i^S(A) - \tilde{\tau}_i^S(A) \geq 0 \tag{12}$$

Consider the case when $\exists\, j \ \ s.t.\ X_j = A_i$. Now by definition of $\tilde{\tau}_i^S(A)$

$$\tilde{\tau}_i^S(A) = \tau_i^S(A) = S^+ \cdot A_i = S^+ \cdot (\mathbf{v}_i \mathbf{v}_i^T + \lambda I) = \mathbf{v}_i^T S^+ \mathbf{v}_i + \lambda S^+ \cdot \lambda I \leq \mathbf{v}_i^T S^+ \mathbf{v}_i^T + (r\lambda I)^+ \cdot I = \mathbf{v}_i^T S^+ \mathbf{v}_i^T + \frac{d}{r} = \hat{\tau}_i^S(A)$$

The above follows by noting that $S \succeq r\lambda I$. It also follows from the definition that $\hat{\tau}^S(A)_i - \tilde{\tau}^S(A)_i \leq \frac{d}{r}$.

In the other case a similar inequality can be shown by noting via the Sherman Morrison Formula that

$$\mathbf{v}_i^T (S + \lambda I + \mathbf{v}_i \mathbf{v}_i^T)^+ \mathbf{v}_i = \mathbf{v}_i^T \left((S+\lambda I)^+ - \frac{(S+\lambda I)^+ \mathbf{v}_i \mathbf{v}_i^T (S+\lambda I)^+}{1 + \mathbf{v}_i^T (S+\lambda I)^+ \mathbf{v}_i} \right) \mathbf{v}_i = \frac{1}{1 + \frac{1}{\mathbf{v}_i^T (S+\lambda I)^+ \mathbf{v}_i^T}}$$

This proves Equation (12). A direct application of Theorem 5.6 now finishes the proof. $\square$

Algorithm 3 REPEATED HALVING

```
1: Input: $A = \sum_{i=1}^{m}(\mathbf{v}_i\mathbf{v}_i^T + \lambda I)$
2: Output: $B$ an $O(dlog(d))$ size weighted sample of $A$ and $B \preceq A \preceq 2B$
3: Take a uniformly random unweighted sample of size $\frac{m}{2}$ of $A$ to form $A'$
4: if $A'$ has size $> O(d\log(d))$ then
5:     Recursively compute an 2-spectral approximation $\tilde{A}'$ of $A'$
6: end if
7: Compute estimates $\gamma_i$ of generalized leverage scores $\{\hat{\tau}_i^{A'}(A)\}$ s.t. the following are satisfied
       $$\gamma_i \geq \hat{\tau}_i^{A'}(A)$$
       $$\sum \gamma_i \leq \sum 16\hat{\tau}_i^{A'}(A) + 1$$
8: Use these estimates to sample matrices from $A$ to form $B$
```

5.3 Algorithms

In the following we formally state the two sub-procedures Algorithm 4 which solves the required system and Algorithm 3 which is the sampling routine for reducing the size of the system.

We prove the following theorem regarding the above algorithm REPEATED HALVING 3.

Theorem 5.8. *REPEATED HALVING (Algorithm 3) produces a $\tilde{O}(d)$ sized sample B such that $\frac{B}{2} \preceq A \preceq 2B$ and can be implemented in total time*

$$\tilde{O}\left(md + d^2 + \sqrt{\kappa_{sample}(O(d\log(d)))d}\right)$$

The $\tilde{O}$ in the above contains factors logarithmic in $m, d, \|A\|_F$. Note that the Frobenius norm of A is bounded by $d\|A\|$ which is bounded in our case by d.

Algorithm 4 Fast Quadratic Solver (FQS)

```
1: Input: $A = \sum_{i=1}^{m}(\mathbf{v}_i\mathbf{v}_i^T + \lambda I)$, b, $\varepsilon$
2: Output : $\tilde{\mathbf{v}}$ s.t. $\|A^{-1}\mathbf{b} - \tilde{\mathbf{v}}\| \leq \varepsilon$
3: Compute $B$ s.t. $2B \succeq A \succeq B$ using REPEATED HALVING(Algorithm 3)
4: $Q(\mathbf{y}) = \frac{\mathbf{y}^T AB^{-1}\mathbf{y}}{2} + \mathbf{b}^T\mathbf{y}$
5: Compute $\hat{\mathbf{y}}$ such that $\|\hat{\mathbf{y}} - \text{argmin}\, Q(\mathbf{y})\| \leq \frac{\varepsilon}{4\|B^{-1}\|}$
6: Output $\tilde{\mathbf{v}}$ such that $\|B^{-1}\hat{\mathbf{y}} - \tilde{\mathbf{v}}\| \leq \varepsilon/2$
```

We first provide the proof of Theorem 5.1 using Theorem 5.8 and then provide the proof for Theorem 5.8. For the purpose of clarity of discourse we hide the terms that appear due to the probabilistic part of the Lemmas. To bound the total probability of failure we can take a union bound and those terms show up in the log.

Proof of 5.1. We will first prove correctness of the algorithm which follows from noting that $\text{argmin}\, Q(\mathbf{y}) = BA^{-1}b$. Therefore we have that

$$\varepsilon/2 \geq \|B^{-1}(\hat{\mathbf{y}} - \text{argmin}\, Q(\mathbf{y}))\| = \|B^{-1}\hat{\mathbf{y}} - A^{-1}b\|$$

A simple application of triangle inequality proves that $\|A^{-1}b - \tilde{\mathbf{v}}\| \leq \varepsilon$.

Running Time Analysis We now prove that the algorithm can be implemented efficiently. We first note the following two simple facts: since B is a 2-approximation of A, $\kappa(B) \leq 2\kappa(A)$ and also that the quadratic $Q(\mathbf{y})$ is 1-strongly convex and 2-smooth. Next, we consider the running time of implementing Step 5. For this purpose we will perform gradient descent over the quadratic $Q\mathbf{y})$. Note that the gradient of Q is given by

$$\nabla Q(\mathbf{y}) = AB^{-1}\mathbf{y} - \mathbf{b}$$

Let $\mathbf{y}_0 = 0$, $\tilde{\varepsilon} = \left(\frac{\varepsilon}{4\|B^{-1}\|}\right)^2$ and G_Q be a bound on the gradient of the quadratic Q($\mathbf{y}$). Let

$$\mathbf{v}_t \text{ be such that } \|B^{-1}\mathbf{y}_{t-1} - \mathbf{v}_t\| \leq \frac{\tilde{\varepsilon}}{100\|A\|G_Q}$$

$$\mathbf{y}_{t+1} = \mathbf{y}_t - \eta\left(A\mathbf{v}_t - \mathbf{b}\right)$$

Define $h_t = Q(\mathbf{y}_t) - \min Q(\mathbf{y})$. Using the standard analysis of gradient descent we show that the following holds true for $t > 0$

$$h_t \leq \max\{\tilde{\varepsilon}, (0.9)^t h_0\}$$

This follows directly from the gradient descent analysis which we outline below. To make the analysis easier we define a *true* gradient descent series.

$$\mathbf{z}_{t+1} = \mathbf{y}_t - \eta\nabla Q(\mathbf{y}_t)$$

Note that

$$\|\mathbf{z}_{t+1} - \mathbf{y}_{t+1}\| = \|A\left(\mathbf{v}_t - B^{-1}\mathbf{y}_t\right)\| \leq \frac{\tilde{\varepsilon}}{10G_Q}$$

We now have that

$$\begin{aligned}
h_{t+1} - h_t &= Q(\mathbf{y}_{t+1}) - Q(\mathbf{y}_t) \\
&\leq \langle\nabla Q(\mathbf{y}_t), \mathbf{y}_{t+1} - \mathbf{y}_t\rangle + \frac{\beta}{2}\|\mathbf{y}_{t+1} - \mathbf{y}_t\|^2 \\
&= \langle\nabla Q(\mathbf{y}_t), \mathbf{z}_{t+1} - y_t\rangle + \langle\nabla Q(\mathbf{y}_t), \mathbf{y}_{t+1} - \mathbf{z}_{t+1}\rangle + \frac{\beta}{2}\|\mathbf{z}_{t+1} - \mathbf{y}_t + \mathbf{y}_{t+1} - \mathbf{z}_{t+1}\|^2 \\
&\leq -\eta\|\nabla Q(\mathbf{y}_t)\|^2 + \langle\nabla Q(\mathbf{y}_t), \mathbf{y}_{t+1} - \mathbf{z}_{t+1}\rangle + \beta\eta^2\|\nabla Q(\mathbf{y}_t)\|^2 + \beta\|\mathbf{y}_{t+1} - \mathbf{z}_{t+1}\|^2 \\
&\leq \frac{1}{\beta}\|\nabla Q(\mathbf{y}_t)\|^2 + \frac{2}{10\beta}\left((\|\nabla Q(\mathbf{y}_t)\| + 1)\tilde{\varepsilon}\right) \\
&\leq -\frac{\alpha}{\beta}h_t + \frac{\tilde{\varepsilon}}{50\beta}
\end{aligned}$$

where $\beta \geq 1$ and $\alpha \leq 2$ are the strong convexity and smoothness parameters of $Q(\mathbf{y})$. Therefore we have that

$$h_{t+1} \leq 0.75h_t + 0.08\tilde{\varepsilon}$$

Using the inductive assumption that $\|h_t\| \leq \max\{\tilde{\varepsilon}, (0.9)^t h_0\}$ it follows easily that

$$h_{t+1} \leq \max\{\tilde{\varepsilon}, (0.9)^{t+1} h_0\}$$

Using the above inequality it follows that after $t \geq O(\log(\frac{1}{\tilde{\varepsilon}}))$ we have that h_t is less than equal to $\tilde{\varepsilon}$. By the 1-strong convexity of $Q(\mathbf{y})$ we have that if $h_t \leq \tilde{\varepsilon}$ then

$$\|\mathbf{y}_t - \operatorname{argmin} Q(\mathbf{y})\| \leq \sqrt{\tilde{\varepsilon}} \leq \frac{\varepsilon}{4\|B\|^{-1}}$$

The running time of the above sub-procedure is bounded by the time to multiply a vector with A, which takes $O(md)$ time, and the time required to compute $\mathbf{v}_t$, which involves solving a linear system in B at each step. Finally in Step 6 we once again compute the solution of a linear system in B. Combining these we get that the total running time is

$$\tilde{O}(md + LIN(B, \hat{\varepsilon})) \log(\frac{1}{\tilde{\varepsilon}})$$

where $\hat{\varepsilon} = \frac{\varepsilon}{400\|A\|\|B\|^{-1} G_Q} = \Omega\left(\frac{\varepsilon}{\kappa(A) G_Q}\right)$. Now we can bound $LIN(B, \varepsilon)$ by $\tilde{O}(d^2 + d\sqrt{\kappa(A)d}) log(1/\varepsilon)$ by using Acc-SVRG to solve the linear system and by noting that B is an $O(d\log(d))$ sized 2-approximation sample of A. The bound G_Q can be taken as a bound on the gradient of the quadratic at the start of the procedure as the norm of gradient only shrinks along the gradient descent path. Therefore it is enough to take $G_Q \leq \|b\|$. This finishes the proof.

$\square$

Proof of Theorem 5.8. The correctness of the algorithm is immediate from Theorems 5.7 and 5.5. The key challenge in the proof lies in proving that there is an efficient way to implement the algorithm which we describe next.

Runtime Analysis To analyze the running time of the algorithm we consider the running time of the computation required in step 7 of the algorithm. We will show that there exists an efficient algorithm to compute numbers γ_i such that

$$\gamma_i \geq \hat{\tau}_i^{A'}(A)$$

$$\sum \gamma_i \leq \sum 16\hat{\tau}_i^{A'}(A) + 8k\varepsilon\|A\|_F^2$$

First note that by recursion $\tilde{A}'$ is a 2-spectral approximation of A'. Therefore we have that

$$\hat{\tau}_i^{A'}(A) \leq \hat{\tau}_i^{\tilde{A}'}(A) \leq 2\hat{\tau}_i^{A'}(A)$$

Also we know that A' is an $O(d\log(d))$ sized weighted sub-sample of A', therefore it is of the form

$$\tilde{A}' = \sum_{i=1}^{O(d\log(d))} \mathbf{v}_i \mathbf{v}_i^T + \lambda I = \sum_{i=1}^{O(d\log(d))} b_i b_i^T$$

where such a decomposition can be achieved easily via decomposing the identity into canonical vectors. Therefore any such $\tilde{A}'$ can be written BB^T where the columns of B are b_i.

To compute $\hat{\tau}_i^{\tilde{A}'}(A)$, we need to compute the value of $\mathbf{v}_i^T(\tilde{A}')^+\mathbf{v}_i = \|B^T(\tilde{A}')^+\mathbf{v}_i\|_2^2$. Indeed computing this directly for all vectors v_i may be inefficient, but one can compute a constant approximation using the following procedure outlined in [CLM+15].

To compute a constant approximation, we randomly sample a Gaussian matrix G consisting of k rows and compute instead the norms $\gamma'(i) = \|GB^T(\tilde{A}')^+\mathbf{v}_i\|_2^2$. By the Johnson Lindenstrauss Lemma setting $k = O(\log(md))$ we get that with high probability we have that

$$1/2\|B^T(\tilde{A}')^+\mathbf{v}_i\|_2^2 \leq \gamma'(i) \leq 2\|B^T(\tilde{A}')^+\mathbf{v}_i\|_2^2$$

Consider the following procedure for the computation of $\gamma'(i)$.

1. Compute the matrix $G' = GB^T$ first which takes $\tilde{O}(kd^2)$ time.
2. For each row G'_i of G' compute a vector G''_i such that $\|G''_i - G'_i\tilde{A}'^+\|^2 \leq \varepsilon$. This takes a total time of $kLIN(\tilde{A}', \varepsilon)$
3. For each v_i compute $\gamma''_i \triangleq \sum_{j=1}^{k} < G''_j, \mathbf{v}_i >^2$. This takes a total time of $O(kmd)$

Here $LIN(S, \varepsilon)$ is the running time of solving a linear system in S with error ε. Therefore the total running time of the above algorithm is (substituting $k = O(\log(md))$)

$$O(md + d^2 + LIN(\tilde{A}', \varepsilon))$$

It is now easy from the definitions of γ''_i and γ'_i to see that

$$\gamma''_i \in \left[\gamma'_i - k\varepsilon\|\mathbf{v}_i\|^2, \gamma'_i + k\varepsilon\|\mathbf{v}_i\|^2\right]$$

Setting $\gamma_i = 4(\gamma''_i + k\varepsilon\|\mathbf{v}_i\|^2)$, it easily follows from the definitions that

$$\gamma_i \geq \hat{\tau}_i^{A'}(A)$$

$$\sum \gamma_i \leq \sum 16\hat{\tau}_i^{A'}(A) + 8k\varepsilon\|A\|_F^2$$

Now setting ε to be $\frac{1}{8k\|V\|_F^2}$ satisfies the required inequality for γ_i. This implies that when sampling from γ_i we will have a two approximation and the number of matrices will be bounded by $O(d\log(d))$.

To solve the linear systems required above we use Accelerated SVRG (Theorem 2.2) which ensures that the running time of $LIN(S, \varepsilon) \leq \tilde{O}(d^2 + d\sqrt{\kappa(S)d})\log(\frac{1}{\varepsilon})$. This follows by noting that in our case S is a sum of $O(d\log(d))$ matrices. To bound the condition number of S note that it is a 2-approximation of some unweighted sample of the matrix A of size greater than $O(dlog(d))$. Therefore we have that $\kappa(S) \leq \kappa_{sample}(A)$.

Putting the above arguments together we get that the total running time of the procedure is bounded by

$$\tilde{O}(md + (d^2 + d\sqrt{\kappa_{sample(A)}d}))$$

which proves the theorem.

□

6 Condition Number Independent Algorithms

In this section we state our main result regarding self-concordant functions and provide the proof. Following is the definition of a self-concordant function.

Definition 6.1 (Self-Concordant Functions)**.** *Let $\mathcal{K} \subseteq \mathbb{R}^n$ be a non-empty open convex set, and and let $f : \mathcal{K} \mapsto \mathbb{R}$ be a C^3 convex function. Then, f is said to be self-concordant if*

$$|\nabla^3 f(\mathbf{x})[\mathbf{h}, \mathbf{h}, \mathbf{h}]| \leq 2(\mathbf{h}^\top \nabla^2 f(\mathbf{x})\mathbf{h})^{3/2},$$

where we have

$$\nabla^k f(\mathbf{x})[\mathbf{h}_1, \ldots, \mathbf{h}_k] \triangleq \frac{\partial^k}{\partial t_1 \ldots \partial t_k}|_{t_1=\cdots=t_k} f(\mathbf{x} + t_1\mathbf{h}_1 + \cdots + t_k\mathbf{h}_k).$$

Our main Theorem regarding self-concordant functions is as follows

Theorem 6.2. *Let $0 < r < 1$, let $\gamma \geq 1$, and let ν be a constant depending on γ, r. Set $\eta = 10(1-r)^2$, $S_1 = c_r = \frac{50}{(1-r)^4}$, where c_r is a constant depending on r, and $T = \frac{f(\mathbf{x}_1) - f(\mathbf{x}^*)}{\nu}$. Then, after $t > T$ the following linear convergence guarantee holds between two full gradient steps for Algorithm 5*

$$\mathbf{E}\left[f(w_s) - f(w^*)\right] \leq \frac{1}{2}\mathbf{E}\left[f(w_{s-1}) - f(w^*)\right]$$

Moreover, the time complexity of each full gradient step is $O(md + c_r d^2)$ where c_r is a constant depending on r (independent of the condition number).

To describe the Algorithm we first present the requisite definitions/basic concepts regarding self concordant functions.

6.1 Self-Concordant Functions Preliminaries

In this section we define and collect some of the well known properties of self-concordant functions. An excellent reference for this material is the lecture notes on this subject by Nemirovski [Nem04]. We begin by defining self-concordant functions.

Another key object in the analysis of self concordant functions is the notion of a Dikin Ellipsoid, which is the unit ball around a point in the norm given by the Hessian $\|\cdot\|_{\nabla^2 f}$ at the point. We will refer to this norm as the *local norm* around a point and denote it as $\|\cdot\|_{\mathbf{x}}$. Formally,

Definition 6.3 (Dikin ellipsoid)**.** *The Dikin ellipsoid of radius r centered at a point $\mathbf{x}$ is defined as*

$$W_r(\mathbf{x}) \triangleq \{\mathbf{y} \mid \|\mathbf{y} - \mathbf{x}\|_{\nabla^2 f(\mathbf{x})} \leq r\}$$

One of the key properties of self-concordant functions that we use is that inside the Dikin ellipsoid, the function is well conditioned with respect to the local norm at the center. The next lemma makes this formal. The proof of this lemma can be found in [Nem04].

Lemma 6.4 (See [Nem04])**.** *For all $\mathbf{h}$ such that $\|\mathbf{h}\|_{\mathbf{x}} < 1$ we have that*

$$(1 - \|\mathbf{h}\|_{\mathbf{x}})^2 \nabla^2 f(\mathbf{x}) \preceq \nabla^2 f(\mathbf{x} + \mathbf{h}) \preceq \frac{1}{(1 - \|\mathbf{h}\|_{\mathbf{x}})^2} \nabla^2 f(\mathbf{x})$$

The next lemma establishes that the function is smooth in the Dikin ellipsoid. The proof of this lemma can be found in [Nem04]

Lemma 6.5 (See [Nem04]). *For all* $\mathbf{h}$ *such that* $\|\mathbf{h}\|_{\mathbf{x}} < 1$ *we have that*

$$f(\mathbf{x}+\mathbf{h}) \leq f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{h} \rangle + \rho\left(\|\mathbf{h}\|_{\mathbf{x}}\right)$$

where $\rho(x) \triangleq -\ln(1-x) - x$

Another key quantity which is used both as a potential function as well as a dampening for the step size in the analysis of Newton's method in general is the Newton Decrement which is defined as $\lambda(\mathbf{x}) \triangleq \|\nabla f(\mathbf{x})\|_{\mathbf{x}}^{*} = \sqrt{\nabla f(\mathbf{x})^{\top} \nabla^{-2} f(\mathbf{x}) \nabla f(\mathbf{x})}$. The following lemma quantifies how $\lambda(\mathbf{x})$ behaves as a potential by showing that once it drops below 1, it ensures that the minimum of the function lies in the current Dikin ellipsoid. This is the property which we use crucially in our analysis. The proof can be found in [Nem04].

Lemma 6.6 (See [Nem04]). *If* $\lambda(\mathbf{x}) < 1$ *then*

$$\|\mathbf{x} - \mathbf{x}^{*}\|_{\mathbf{x}} \leq \frac{\lambda(\mathbf{x})}{1 - \lambda(\mathbf{x})}$$

6.2 Condition Number Independent Algorithms

In this section we describe an efficient linearly convergent Algorithm (5) for optimization of self-concordant functions for which the running time is independent of the condition number. We have not tried to optimize the complexity of algorithms in terms of d in this section as our main focus is to make them condition number independent.

The key idea here is the ellipsoidal view of Newton's method whereby we show that after making a constant number of full Newton steps, one can identify an ellipsoid and a norm such that the function is well conditioned with respect to the norm in the ellipsoid. This is depicted in Figure 1.

At this point one can run any first order algorithm one wishes. We in particular choose SVRG and prove its fast convergence. Algorithm 6 (described in the Appendix) states the modified SVRG routine for general norms used in Algorithm 5.

We now state and prove the following theorem regarding the convergence of Algorithm 5.

Algorithm 5 EllipsoidCover

```
1: Input : Self-concordant $f(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x})$, $T$, $r \in \{0,1\}$, initial point $\mathbf{x}_1 \in \mathcal{K}$, $\gamma > 1$, $S_1$, $\eta$, $T$
2: Initialize: $\mathbf{x}_{\text{curr}} = \mathbf{x}_1$
3: while $\lambda(\mathbf{x}_{\text{curr}}) > \frac{r}{1+r}$ do
4:     $step = \gamma(1 + \lambda(\mathbf{x}_{\text{curr}}))$
5:     $\mathbf{x}_{\text{curr}} = \mathbf{x}_{\text{curr}} - \frac{1}{step} \nabla^{-2} f(\mathbf{x}_{\text{curr}}) \nabla f(\mathbf{x}_{\text{curr}})$
6: end while
7: $\mathbf{x}_{T+1} = \mathbf{N-SVRG}(W_r(\mathbf{x}), f(\mathbf{x}), \nabla^2 f(\mathbf{x}_{\text{curr}}), S_1, \eta, T)$
8: return $\mathbf{x}_{T+1}$
```

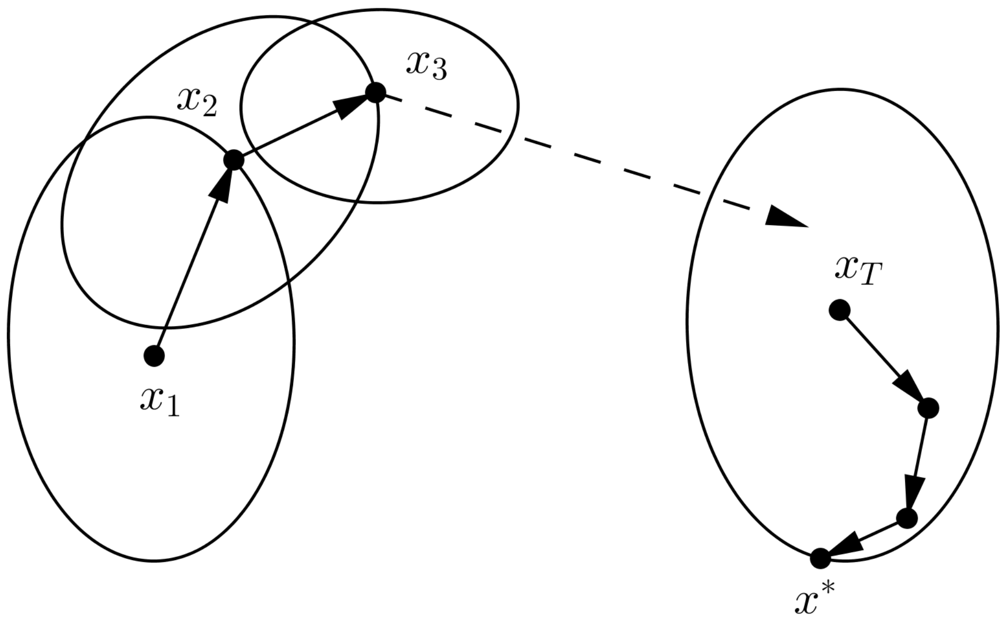

Figure 1: Visualization of the steps taken in Algorithm 5.

Proof. It follows from Lemma 6.8 that at $t = \frac{f(\mathbf{x}_1)-f(\mathbf{x}^*)}{\nu}$ the minimizer is contained in the Dikin Ellipsoid of radius r where ν is a constant depending on γ, r. This coupled with Lemma 6.7 shows that the function satisfies the following property with respect to $W_{\mathbf{x}_t}$

$$\forall \mathbf{x} \in W_{\mathbf{x}_t} \;\; (1-r)^2 \nabla^2 f(\mathbf{x}_T) \preceq \nabla^2 f(\mathbf{x}) \preceq (1-r)^{-2} \nabla^2 f(\mathbf{x}_T)$$

Using the above fact and Lemma 6.9 and substituting for the parameters concludes the proof. □

We describe the Algorithm N-SVRG 6 in the Appendix section A.3. Since the algorithm and the following lemmas are minor variations of their original versions we include the proofs of the following lemmas in the Appendix section A.3 for completeness.

Lemma 6.7. *Let f be a self-concordant function over $\mathcal{K}$, let $0 < r < 1$, and consider $\mathbf{x} \in \mathcal{K}$. Let $W_r(\mathbf{x})$ be the Dikin ellipsod of radius r and let Let $\alpha = (1-r)^2$ and $\beta = (1-r)^{-2}$. Then, for all $\mathbf{h}$ s.t. $\mathbf{x} + \mathbf{h} \in W_r(\mathbf{x})$,*

$$\alpha \nabla^2 f(\mathbf{x}) \preceq \nabla^2 f(\mathbf{x} + \mathbf{h}) \preceq \beta \nabla^2 f(\mathbf{x})$$

Lemma 6.8. *Let f be a self-concordant function over $\mathcal{K}$, let $0 < r < 1$, let $\gamma \geq 1$, and consider following the damped Newton step as described in Algorithm 5 with initial point $\mathbf{x}_1$. Then, the number of steps t of the algorithm before the minimizer of f is contained in the Dikin ellipsoid of radius r of the current iterate, i.e. $\mathbf{x}^* \in W_r(\mathbf{x}_t)$, is at most*

$$t = \frac{f(\mathbf{x}_1) - f(\mathbf{x}^*)}{\nu},$$

where ν is a constant depending on γ, r.

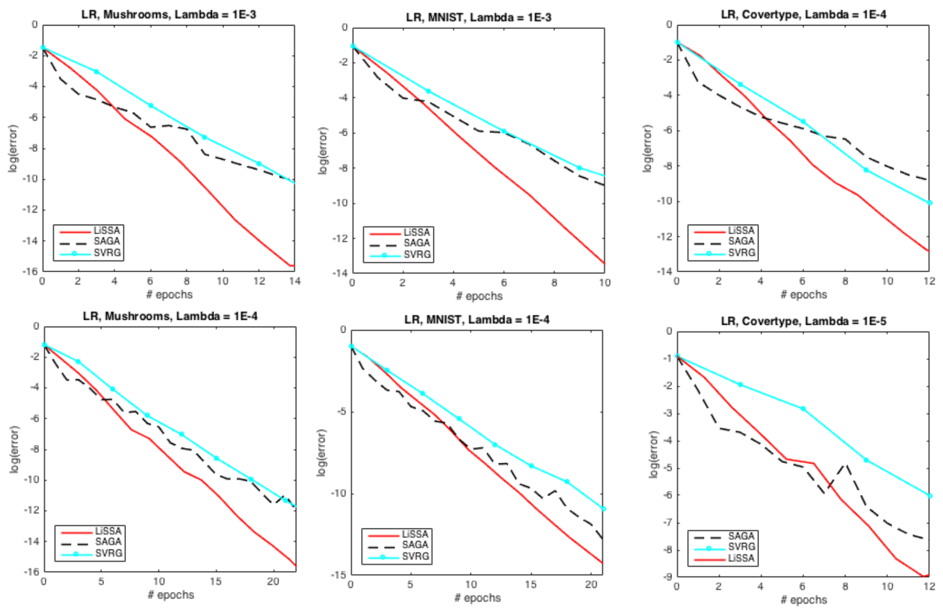

Figure 2: Performance of LiSSA as compared to a variety of related optimization methods for different datasets and choices of regularization parameter λ. $S_1 = 1$, $S_2 \sim \kappa \ln(\kappa)$

Lemma 6.9. *Let f be a convex function. Suppose there exists a convex set $\mathcal{K}$ and a positive semi-definite matrix A such that $\forall \mathbf{x} \in \mathcal{K}$ $\alpha A \preceq \nabla^2 f(\mathbf{x}) \preceq \beta A$. Then the following holds between two full gradient steps of Algorithm 6*

$$\mathbf{E}\left[f(\mathbf{x}_s) - f(\mathbf{x}^*)\right] \leq \left(\frac{1}{\alpha\eta(1-2\eta\beta)n} + \frac{2\eta\beta}{(1-2\eta\beta)}\right)\mathbf{E}\left[f(\mathbf{x}_{s-1}) - f(\mathbf{x}^*)\right]$$

7 Experiments

In this section we present experimental evaluation for our theoretical results. We perform the experiments for a classification task over two labels using the Logistic Regression objective function with the ℓ_2 regularizer. For all of the classification tasks we choose two values of λ : $\frac{1}{m}$ and $\frac{10}{m}$, where m is the number of training examples. We perform the above classification tasks over three data sets: MNIST, CoverType, Mushrooms. Figure 2 displays the log-error achieved by LiSSA as compared to two standard first order algorithms SVRG/SAGA in terms of number of passes over the data. Figure 3 presents the performance of LiSSA as compared to NewSamp [EM15] and vanilla Newton's method both over time as well as iterations.

7.1 Experiment Details

In this section we describe our experiments and choice of parameters in detail. Table 2 provides details regarding the datasets we picked to do the experiments. To make sure our functions are scaled (such that the norm of the Hessian is bounded) we scale the above dataset points to unit norm.

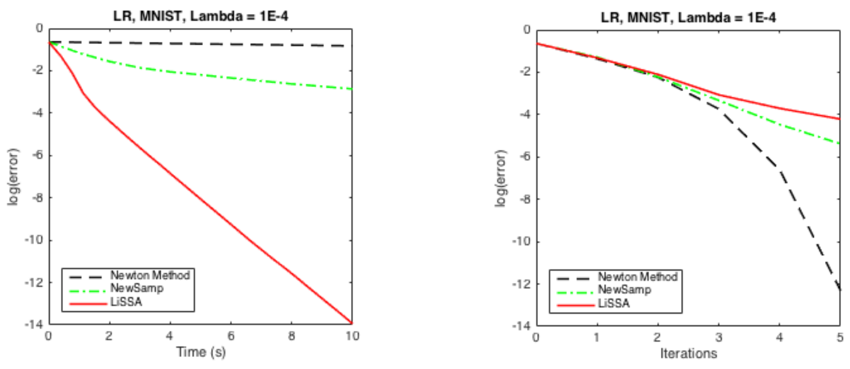

Figure 3: Convergence of LiSSA over time/iterations for LR with MNIST, as compared to NewSamp and Newton's method.

Table 2: A description of datasets used in the experiment

Data set	m	d	Reference
MNIST4-9	11791	784	[LC10]
Mushrooms	8124	112	[Lic13]
CoverType	100000	54	[BD99]

7.2 Comparison with Standard Algorithms

In Figures 2 and 4 we present comparisons between the efficiency of our algorithm with different standard and popular algorithms. On the Y axis in both cases we plot the $log(currentValue - OptimumValue)$. We obtained the optimum value for each case by running our algorithm for a long enough time until it converged to the point of machine precision.

Epoch Comparison: In Figure 2 we compare LiSSA with SVRG and SAGA in terms of the accuracy achieved in terms of the number of passes over the data. To compute the number of passes in SVRG/SAGA we make sure that the inner stochastic gradient iteration in both the algorithms counts as exactly 1 pass (this is because even though it accesses gradients at two different points one of them can be stored from before in both cases). The outer full gradient in SVRG counts as one complete pass over the data. We set the number of inner iterations of SVRG to $2m$ for the case when $\lambda = 1/m$ and parameter tune the number of inner iterations when $\lambda = 10/m$. The stepsizes for all the algorithms are parameter tuned by an exhaustive search over the parameters.

Time Comparison: For the comparison with respect to time (Figure 4) we consider the following algorithms : AdaGrad[DHS11], BFGS[Bro70, Fle70, Gol70, Sha70], Gradient Descent, SGD, SVRG[JZ13] and SAGA[DBLJ14]. The $\log(Error)$ is plotted as a function of the time elapsed from the start of the run for each algorithm. We next describe our choice of parameters for the algorithms. For AdaGrad we used the faster diagonal scaling version proposed by [DHS11]. We implemented the basic version of BFGS with backtracking line search. For Gradient Descent in each experiment we find a reasonable step size using parameter tuning. For Stochastic Gradient

Descent we use the variable step size $\eta_t = \gamma/\sqrt{t}$ which is usually the prescribed step size, and we hand tune the parameter γ. The parameters for SVRG and SAGA were chosen in the same way as before.

Choice of Parameters for LiSSA: To pick the parameters for our algorithm, we observe that it exhibits smooth behavior even in the case of $S_1 = 1$, so this is used for the experiment. However, we observe that increasing S_2 has a positive effect on the convergence of the algorithm up to a certain point, as a higher S_2 leads to a larger per-iteration cost. This behavior is consistent with the theoretical analysis. We summarize the comparison between the per-iteration convergence and the value of S_2 in Figure 5. As the theory predicts S_2 to be of the order $\kappa \ln(\kappa)$, for our experiments we determine an estimate for κ and set S_2 to around $\kappa \ln(\kappa)$. This value is typically equal to m in our experiments. We observe that setting S_2 in this way resulted in the experimental results displayed in Figure 2.

Comparison with Second-Order Methods: In this subsection we present details about the comaprison between LiSSA, NewSamp [EM15] and vanilla Newton's method displayed in Figure 3. We perform this experiment on the MNIST database and show the convergence properties of all three algorithms over time as well as over iterations. We could not replicate the results of NewSamp on all of our datasets as it sometimes seems to diverge in our experiments. For Logistic Regression on the MNIST dataset we could get it to converge by setting the value of S_1 to be slightly higher. We observe as is predicted by the theory that when compared with respect to number of iterations, NewSamp and LiSSA perform similarly while vanilla Newton performs the best as it shows a quadratic convergence. This can be seen in Figure 3. However when we consider the performance in terms of time for these algorithms we see that LiSSA has a significant advantage.

7.3 Acknowledgements

The authors would like to thank Zeyuan Allen-Zhu, Dan Garber, Haipeng Luo and David McAllester for several illuminating discussions and suggestions. We would especially like to thank Yin Tat Lee for discussions regarding matrix sampling.

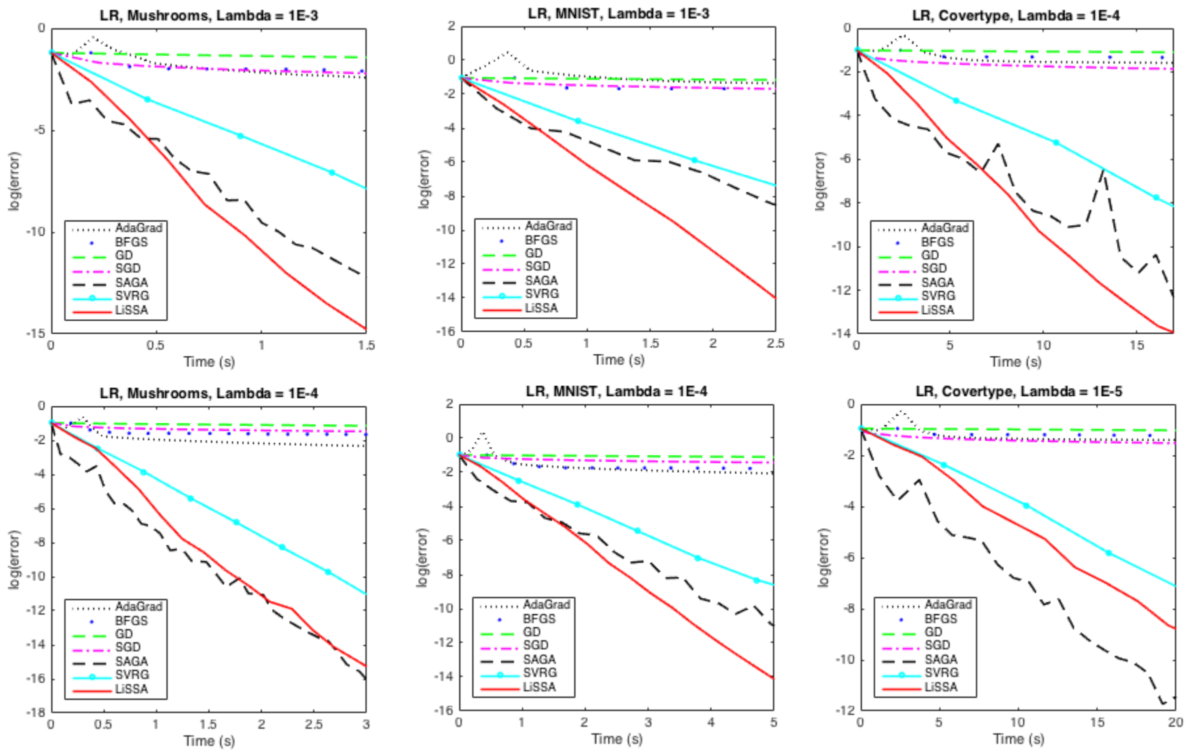

Figure 4: Performance of LiSSA as compared to a variety of related optimization methods for different datasets and choices of regularization parameter λ. $S_1 = 1$, $S_2 \sim \kappa \ln(\kappa)$

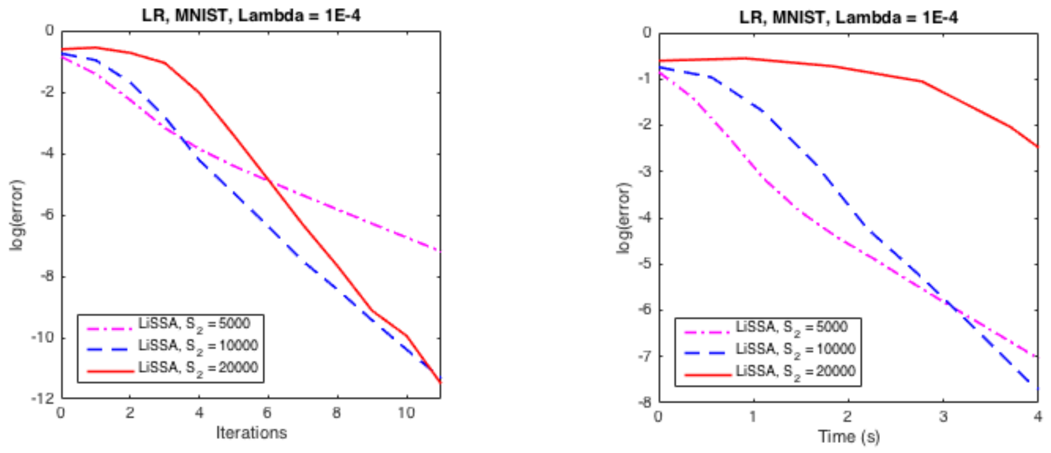

Figure 5: Differing convergence rates for LiSSA based on different choices of the S_2 parameter.

References

[BCNN11] Richard H Byrd, Gillian M Chin, Will Neveitt, and Jorge Nocedal. On the use of stochastic hessian information in optimization methods for machine learning. *SIAM Journal on Optimization*, 21(3):977–995, 2011.

[BD99] Jock A Blackard and Denis J Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture*, 24(3):131–151, 1999.

[BHNS14] Richard H Byrd, SL Hansen, Jorge Nocedal, and Yoram Singer. A stochastic quasi-newton method for large-scale optimization. *arXiv preprint arXiv:1401.7020*, 2014.

[Bro70] Charles G Broyden. The convergence of a class of double-rank minimization algorithms 2. the new algorithm. *IMA Journal of Applied Mathematics*, 6(3):222–231, 1970.

[CLM$^+$15] Michael B. Cohen, Yin Tat Lee, Cameron Musco, Christopher Musco, Richard Peng, and Aaron Sidford. Uniform sampling for matrix approximation. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, Rehovot, Israel, 2015*, pages 181–190, 2015.

[DBLJ14] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems*, 2014.

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

[EM15] Murat A Erdogdu and Andrea Montanari. Convergence rates of sub-sampled newton methods. In *Advances in Neural Information Processing Systems*, pages 3034–3042, 2015.

[Fle70] Roger Fletcher. A new approach to variable metric algorithms. *The computer journal*, 13(3), 1970.

[Gol70] Donald Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of computation*, 24(109):23–26, 1970.

[JZ13] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*, pages 315–323, 2013.

[LC10] Y LeCun and C Cortes. MNIST handwritten digit database. 2010.

[Lic13] M. Lichman. UCI machine learning repository, 2013.

[LMH15] Hongzhou Lin, Julien Mairal, and Zaïd Harchaoui. A universal catalyst for first-order optimization. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, Montreal, Quebec, Canada*, pages 3384–3392, 2015.

[LMP13] Mu Li, Gary L. Miller, and Richard Peng. Iterative row sampling. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, 2013.

[Mar10] James Martens. Deep learning via hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 735–742, 2010.

[MNJ16] Philipp Moritz, Robert Nishihara, and Michael I. Jordan. A linearly-convergent stochastic L-BFGS algorithm. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016*, pages 249–258, 2016.

[MR14] Aryan Mokhtari and Alejandro Ribeiro. RES: regularized stochastic BFGS algorithm. *IEEE Trans. Signal Processing*, 62(23):6089–6104, 2014.

[Nem04] AS Nemirovskii. Interior point polynomial time methods in convex programming. *Lecture Notes*, 2004.

[Nes83] Yurii Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.

[Nes04] Y Nesterov. *Introductory lectures on convex optimization.* Springer Science & Business Media, 2004.

[NW06] Jorge Nocedal and Stephen Wright. *Numerical optimization.* Springer Science & Business Media, 2006.

[RM51] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[RSB12] Nicolas L Roux, Mark Schmidt, and Francis R Bach. A stochastic gradient method with an exponential convergence _rate for finite training sets. In *Advances in Neural Information Processing Systems*, 2012.

[Sha70] David F Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of computation*, 24(111):647–656, 1970.

[SSZ13] Shai Shalev-Shwartz and Tong Zhang. Stochastic dual coordinate ascent methods for regularized loss. *The Journal of Machine Learning Research*, 14(1):567–599, 2013.

[SSZ16] Shai Shalev-Shwartz and Tong Zhang. Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization. *Mathematical Programming*, 155(1-2):105–145, 2016.

[SYG07] Nicol N. Schraudolph, Jin Yu, and Simon Gnter. A stochastic quasi-newton method for online convex optimization. In *In Proceedings of 11th International Conference on Artificial Intelligence and Statistics*, 2007.

[Tro12] Joel A Tropp. User-friendly tail bounds for sums of random matrices. *Foundations of computational mathematics*, 12(4):389–434, 2012.

[Zhu16] Zeyuan Allen Zhu. Katyusha: Accelerated variance reduction for faster SGD. *CoRR*, abs/1603.05953, 2016.

[ZMJ13] Lijun Zhang, Mehrdad Mahdavi, and Rong Jin. Linear convergence with condition number independent access of full gradients. In *Advances in Neural Information Processing Systems*, pages 980–988, 2013.

A Appendix

A.1 Proof of Lemma 5.5

The proof is almost identical to the proof of Lemma 4 in [CLM$^+$15]. As in there we will use the following lemma on matrix concentration. This appears as Lemma 11 in [CLM$^+$15].

Lemma A.1. *Let $Y_1 \dots Y_k$ be independent random positive semidefinite matrices of size $d \times d$. Let $Y = \sum Y_i$ and let $Z = E[Y]$. If $Y_i \preceq R.Z$ then*

$$P\left[\sum Y_i \preceq (1-\varepsilon)Z\right] \leq de^{\frac{\varepsilon^2}{2R}}$$

$$P\left[\sum Y_i \succeq (1+\varepsilon)Z\right] \leq de^{\frac{\varepsilon^2}{3R}}$$

For every matrix A_i choose $Y_i = \frac{A_i}{p_i}$ with probability p_i and 0 otherwise. Therefore we need to bound $\sum Y_i$. Note that $E[Y_i] = A$. We will now show that

$$\forall i \;\; Y_i \preceq \frac{1}{c \log d\varepsilon^{-2}} A$$

which will finish the proof by a direct application of Lemma A.1. First we will show that

$$\frac{A_i}{\tau_i(A)} \preceq A$$

We need to show that $\forall \mathbf{x} \;\; \mathbf{x}^T A_i \mathbf{x} \leq \tau_i(A) \mathbf{x}^T A \mathbf{x}$. By noting that A_i are PSD we have that if $\mathbf{x}^T A \mathbf{x} = 0$ then $\forall i, \mathbf{x}^T A_i \mathbf{x} = 0$. Therefore we can now consider $\mathbf{x} = A^{+/2}\mathbf{y}$. Therefore we need to show that

$$\mathbf{y} A^{+/2} A_i A^{+/2} \mathbf{y} \leq \tau_i(A) \|y\|^2$$

This is true because the maximum eigenvalue of $A^{+/2} A_i A^{+/2}$ is bounded by its $Trace(A^{+/2} A_i A^{+/2})$ (due to PSDness) which is equal to $\tau_i(A)$ by definition. Therefore when $p_i < 1$ i.e. $\alpha c \mathbf{u}_i \log(d) < 1$, the facts $\tau_i(A) \leq \mathbf{u}_i$ immediately provide

$$Y_i \preceq \frac{1}{c \log d\varepsilon^{-2}} A$$

When $p_i = 1$ the above does not hold but we can essentially replace $Y_i = A_i$ with $c \log d\varepsilon^{-2}$ variables each equal to $\frac{A_i}{c \log d\varepsilon^{-2}}$ each being sampled with probability 1. This does not change $E[\sum Y_i]$ but proves concentration. Now a direct application of Lemma A.1 finishes the proof. Also note that a standard chernoff bound proves the required bound on the sample size.

A.2 Proof of Lemma 5.6

The proof is almost identical to the proof of Theorem 1 in [CLM+15]. We include the proof here for completeness. Define S_i to be S if $\exists j \;\; X_j = A_i$ and $S + A_i$ otherwise. Now since $S^i \preceq A$ we have that $\tilde{\tau}_i^S(A) \geq \tau_i(A)$. To bound the expected sum we break it down into two parts

$$\sum_i \tilde{\tau}_i^S(A) = \sum_{i \in S} \tilde{\tau}_i^S(A) + \sum_{i \notin S} \tilde{\tau}_i^S(A)$$

The first part is a sum of S's leverage scores and is therefore bounded by d. To bound the second term, consider a random process that first selects S, then selects a random $i \notin S$ and returns $\tilde{\tau}_i^S(A)$. There are always exactly $m - r$ such i, so the value returned by this random process is, in expectation, exactly equal to $\frac{1}{m-r} E[\sum_{i \notin S} \tilde{\tau}_i^S(A)]$. This random process is also equivalent to randomly selecting a set S of size $r+1$, then randomly choosing an $i \in S'$ and returning its leverage score! In expectation it is therefore equal to the average leverage score in S. S' has size $r+1$ and its leverage scores sum to its rank, so we can bound its average leverage score by $\frac{d}{r+1}$. Overall we have that

$$E[\sum_i \tilde{\tau}_i^S(A)] = d + \frac{d(m-r)}{r+1} \leq O(\frac{md}{r})$$

A.3 Condition Number Independent Algorithms - Proofs

A.3.1 Proof of Ellipsoidal Cover Lemma

Proof of Lemma 6.8. Let $\lambda(\mathbf{x})$ be the Newton decrement at $\mathbf{x}$. By Lemma 6.6 we know that if $\lambda(\mathbf{x}) \leq \frac{r}{1+r} < 1$, then

$$||\mathbf{x} - \mathbf{x}^*||_{\mathbf{x}} \leq \frac{\lambda(\mathbf{x})}{1 - \lambda(\mathbf{x})} = r.$$

Consider a single iteration of the algorithm at time t. If $\lambda(\mathbf{x}_t) \leq \frac{r}{1+r}$, then we may conclude that $\mathbf{x}^* \in W_r(\mathbf{x}_t)$. Therefore, it is only when $\lambda(\mathbf{x}_t) > \frac{r}{1+r}$ that $\mathbf{x}^*$ may not be contained within $W_r(\mathbf{x}_t)$. Since our update is of the form

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \frac{1}{\gamma(1 + \lambda(\mathbf{x}_t))} \nabla^{-2} f(\mathbf{x}_t) \nabla f(\mathbf{x}_t),$$

We have the following

$$\begin{aligned}
f(\mathbf{x}_{t+1}) &\leq f(\mathbf{x}_t) - \frac{1}{\gamma(1 + \lambda(\mathbf{x}_t))} \nabla f(\mathbf{x}_t)^\top ([\nabla^2 f(\mathbf{x}_t)]^{-1} \nabla f(\mathbf{x}_t)) + \rho\left(\frac{\lambda(\mathbf{x}_t)}{\gamma(1 + \lambda(\mathbf{x}_t))}\right) \\
&= f(\mathbf{x}_t) - \frac{(\lambda(\mathbf{x}_t))^2}{\gamma(1 + \lambda(\mathbf{x}_t))} + \rho\left(\frac{\lambda(\mathbf{x}_t)}{\gamma(1 + \lambda(\mathbf{x}_t))}\right) \\
&= f(\mathbf{x}_t) - \frac{(\lambda(\mathbf{x}_t))^2}{\gamma(1 + \lambda(\mathbf{x}_t))} - \ln\left(1 - \frac{\lambda(\mathbf{x}_t)}{\gamma(1 + \lambda(\mathbf{x}_t))}\right) - \frac{\lambda(\mathbf{x}_t)}{\gamma(1 + \lambda(\mathbf{x}_t))} \\
&= f(\mathbf{x}_t) - \frac{\lambda(\mathbf{x}_t)}{\gamma} + \ln\left(1 + \frac{\lambda(\mathbf{x}_t)}{\gamma + (\gamma - 1)\lambda(\mathbf{x}_t)}\right)
\end{aligned}$$

where the first inequality follows from Lemma 6.5. It now follows that

$$\begin{aligned}
f(\mathbf{x}_t) - f(\mathbf{x}_{t+1}) &\geq \frac{\lambda(\mathbf{x}_t)}{\gamma} - \ln\left(1 + \frac{\lambda(\mathbf{x}_t)}{\gamma + (\gamma-1)\lambda(\mathbf{x}_t)}\right) \\
&\geq \frac{\lambda(\mathbf{x}_t)}{\gamma} - \frac{\lambda(\mathbf{x}_t)}{\gamma + (\gamma-1)\lambda(\mathbf{x}_t)} \\
&= \lambda(\mathbf{x}_t)\left(\frac{1}{\gamma} - \frac{1}{\gamma + (\gamma-1)\lambda(\mathbf{x}_t)}\right) \\
&> \frac{r}{1+r}\left(\frac{1}{\gamma} - \frac{1}{\gamma + \frac{(\gamma-1)r}{1+r}}\right) \\
&> 0
\end{aligned}$$

where the second inequality comes from the fact that, for all $x \in \mathbb{R}$, $\ln(1+x) \leq 1+x$. Let $\nu = \frac{r}{1+r}\left(\frac{1}{\gamma} - \frac{1}{\gamma+\frac{(\gamma-1)r}{1+r}}\right)$. Then we see that after

$$\frac{f(\mathbf{x}_1) - f(\mathbf{x}^*)}{\nu}$$

steps, we can guarantee that we have arrived at $\mathbf{x}_t$ such that $\mathbf{x}^* \in W_r(\mathbf{x}_t)$. □

Algorithm 6 N-SVRG

```
Input: Convex set $\mathcal{K}$, $f(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x})$, Norm $A \succeq 0$, update frequency $S_1$, and learning rate $\eta$
  Initialize $\tilde{\mathbf{x}}_0$
  for $s = 1, 2 \ldots$ do
    $\tilde{\mathbf{x}} = \tilde{\mathbf{x}}_{s-1}$
    $\tilde{\mu} = \frac{1}{n}\sum_{k=1}^{m} \nabla f_k(\tilde{\mathbf{x}})$
    $\mathbf{x}_0 = \tilde{\mathbf{x}}$
    for $t = 1$ to $S_1$ do
      Randomly choose $i_t \in \{1 \ldots m\}$
      $g = \nabla\psi_{i_t}(\mathbf{x}_{t-1}) - \nabla\psi_{i_t}(\tilde{\mathbf{x}}) + \tilde{\mu}$
      $\mathbf{x}_t = \Pi_{\mathcal{K}}^{A}\left(\mathbf{x}_{t-1} - \eta A^{-1} g\right)$
    end for
    $\tilde{\mathbf{x}}_s = \mathbf{x}_t$ for randomly chosen $t \in \{1, \ldots, m\}$
  end for
```

A.3.2 Proof of SVRG Lemma

Proof of Lemma 6.9. The proof follows the original proof of SVRG [JZ13] with a few modifications to take into account the general norm. For any i, consider

$$g_i(\mathbf{x}) = f_i(\mathbf{x}) - f_i(\mathbf{x}^*) - \langle \nabla f_i(\mathbf{x}^*), \mathbf{x} - \mathbf{x}^* \rangle$$

We know that $g_i(\mathbf{x}^*) = \min_{\mathbf{x}} g_i(\mathbf{x})$ since $\nabla g_i(\mathbf{x}) = 0$. Therefore

$$\begin{aligned} 0 = g_i(\mathbf{x}^*) &\leq \min_\eta \left(g_i(\mathbf{x} - \eta A^{-1}\nabla g_i(\mathbf{x}^*))\right) \\ &\leq \min_\eta \left(g_i(\mathbf{x}) - \eta\|g_i(\mathbf{x})\|^2_{A^{-1}} + 0.5\beta\eta^2\|g_i(\mathbf{x})\|^2_{A^{-1}}\right) \\ &= g_i(\mathbf{x}) - \frac{1}{2\beta}\|g_i(\mathbf{x})\|^2_{A^{-1}} \end{aligned}$$

The second inequality follows from smoothness of g_i (Note that g_i is as smooth as f_i). Summing the above inequality over i and setting $\mathbf{x}^*$ to be the minimum of f and hence $\nabla f(\mathbf{x}^*) = 0$ we get

$$\frac{\sum_{i=1}^m \|\nabla f_i(\mathbf{x}) - \nabla f_i(\mathbf{x}^*)\|^2_{A^{-1}}}{2m\beta} \leq f(\mathbf{x}) - f(\mathbf{x}^*)$$

Define $v_t = \nabla f_{i_t}(\mathbf{x}_{t-1}) - \nabla f_{i_t}(\tilde{\mathbf{x}}) + \tilde{\mu}$. Conditioned on $\mathbf{x}_{t-1}$ we can take the expectation with respect to i_t and obtain

$$\begin{aligned} \mathbf{E}\left[\|v_t\|^2_{A^{-1}}\right] &\leq 2\mathbf{E}\left[\|\nabla f_{i_t}(\mathbf{x}_{t-1}) - \nabla f_{i_t}(\mathbf{x}^*)\|^2_{A^{-1}}\right] + 2\mathbf{E}\left[\|\nabla f_{i_t}(\tilde{\mathbf{x}}) - \nabla f_{i_t}(\mathbf{x}^*) - \nabla f(\tilde{\mathbf{x}})\|^2_{A^{-1}}\right] \\ &\leq 2\mathbf{E}\left[\|\nabla f_{i_t}(\mathbf{x}_{t-1}) - \nabla f_{i_t}(\mathbf{x}^*)\|^2_{A^{-1}}\right] + 2\mathbf{E}\left[\|\nabla f_{i_t}(\tilde{\mathbf{x}}) - \nabla f_{i_t}(\mathbf{x}^*)\|^2_{A^{-1}}\right] \\ &\leq 4\beta\left(f(\mathbf{x}_{t-1}) - f(\mathbf{x}^*) + f(\tilde{\mathbf{x}}) - f(\mathbf{x}^*)\right) \end{aligned}$$

The above set inequalities follow by noting the following three facts.

- $\|a + b\|^2_{A^{-1}} \leq 2\|a\|^2_{A^{-1}} + 2\|b\|^2_{A^{-1}}$
- $\tilde{\mu} = \nabla f(\tilde{\mathbf{x}})$
- $\mathbf{E}\left[\|X - \mathbf{E}X\|^2_{A^{-1}}\right] \leq \mathbf{E}\|X\|^2_{A^{-1}}$

Now note that conditioned on $\mathbf{x}_{t-1}$ we have that $\mathbf{E}v_t = \nabla f(\mathbf{x}_{t-1})$ and therefore

$$\begin{aligned} \mathbf{E}\|\mathbf{x}_t - \mathbf{x}^*\|^2_A &\leq \|\mathbf{x}_{t-1} - \eta A^{-1}v_t - \mathbf{x}^*\|^2_A \\ &= \|\mathbf{x}_{t-1} - \mathbf{x}^*\|^2_A - 2\eta\langle \mathbf{x}_{t-1} - \mathbf{x}^*, AA^{-1}\mathbf{E}v_t\rangle + \eta^2\mathbf{E}\|v_t\|^2_{A^{-1}} \\ &\leq \|\mathbf{x}_{t-1} - \mathbf{x}^*\|^2_A - 2\eta\langle \mathbf{x}_{t-1} - \mathbf{x}^*, \nabla f(\mathbf{x}_{t-1})\rangle + 4\beta\eta^2\left(f(\mathbf{x}_{t-1}) - f(\mathbf{x}^*) + f(\tilde{\mathbf{x}}) - f(\mathbf{x}^*)\right) \\ &\leq \|\mathbf{x}_{t-1} - \mathbf{x}^*\|^2_A - 2\eta\left(f(\mathbf{x}_{t-1}) - f(\mathbf{x}^*)\right) + 4\beta\eta^2\left(f(\mathbf{x}_{t-1}) - f(\mathbf{x}^*) + f(\tilde{\mathbf{x}}) - f(\mathbf{x}^*)\right) \\ &= \|\mathbf{x}_{t-1} - \mathbf{x}^*\|^2_A - 2\eta(1 - 2\eta\beta)\left(f(\mathbf{x}_{t-1}) - f(\mathbf{x}^*)\right) + 4\beta\eta^2\left(f(\tilde{\mathbf{x}}) - f(\mathbf{x}^*)\right) \end{aligned}$$

The first inequality uses the Pythagorean inequality for norms, the second inequality uses the derived inequality for $\mathbf{E}\|v_t\|^2_{A^{-1}}$ and the second inequality uses the convexity of $f(\mathbf{x})$. Consider a fixed stage s, so that $\tilde{\mathbf{x}} = \tilde{\mathbf{x}}_{s-1}$ and $\tilde{\mathbf{x}}_s$ are selected after all the updates have been completed. By summing the previous inequality over t and taking an expectation over the history we get

$$\begin{aligned} \mathbf{E}\|\mathbf{x}_n - \mathbf{x}^*\|^2_A + 2\eta(1 - 2\eta\beta)n\mathbf{E}\left[f(\mathbf{x}_s) - f(\mathbf{x}^*)\right] &\leq \mathbf{E}\|\mathbf{x}_0 - \mathbf{x}^*\|^2_A + 4\beta n\eta^2\mathbf{E}\left[f(\tilde{\mathbf{x}}) - f(\mathbf{x}^*)\right] \\ &= \|\tilde{\mathbf{x}} - \mathbf{x}^*\|^2_A + 4\beta n\eta^2\mathbf{E}\left[f(\tilde{\mathbf{x}}) - f(\mathbf{x}^*)\right] \\ &\leq 2\frac{\mathbf{E}\left[f(\tilde{\mathbf{x}}) - f(\mathbf{x}^*)\right]}{\alpha} + 4\beta n\eta^2\mathbf{E}\left[f(\tilde{\mathbf{x}}) - f(\mathbf{x}^*)\right] \\ &= 2\left(\frac{1}{\alpha} + 2\beta n\eta^2\right)\mathbf{E}\left[f(\tilde{\mathbf{x}}) - f(\mathbf{x}^*)\right] \end{aligned}$$

The second inequality uses the strong convexity property. Therefore we have that

$$\mathbf{E}\left[f(\mathbf{x}_s) - f(\mathbf{x}^*)\right] \leq \left(\frac{1}{\alpha\eta(1-2\eta\beta)n} + \frac{2\eta\beta}{(1-2\eta\beta)}\right)\mathbf{E}\left[f(\mathbf{x}_{s-1}) - f(\mathbf{x}^*)\right]$$

$\square$