Un-regularizing: approximate proximal point and faster stochastic algorithms for empirical risk minimization

Roy Frostig RF@CS.STANFORD.EDU
Stanford University

Rong Ge RONGGE@MICROSOFT.COM
Sham M. Kakade SKAKADE@MICROSOFT.COM
Microsoft Research, New England

Aaron Sidford SIDFORD@MIT.EDU
MIT

Abstract

We develop a family of accelerated stochastic algorithms that optimize sums of convex functions. Our algorithms improve upon the fastest running time for empirical risk minimization (ERM), and in particular linear least-squares regression, across a wide range of problem settings.

To achieve this, we establish a framework, based on the classical *proximal point algorithm*, useful for accelerating recent fast stochastic algorithms in a black-box fashion. Empirically, we demonstrate that the resulting algorithms exhibit notions of stability that are advantageous in practice. Both in theory and in practice, the provided algorithms reap the computational benefits of adding a large strongly convex regularization term, without incurring a corresponding bias to the original ERM problem.

1. Introduction

A general optimization problem central to machine learning is that of *empirical risk minimization* (ERM): finding a predictor or regressor that minimizes a sum of loss functions defined by a data sample. In this paper, we focus on *empirical risk minimization of linear predictors*: given a set of n data points $a_i, \ldots, a_n \in \mathbb{R}^d$ and convex loss functions $\phi_i : \mathbb{R} \to \mathbb{R}$ for $i = 1, \ldots, n$, solve

$$\min_{x \in \mathbb{R}^n} F(x), \quad \text{where} \quad F(x) \stackrel{\text{def}}{=} \textstyle\sum_{i=1}^n \phi_i(a_i^\mathsf{T} x). \qquad (1)$$

Proceedings of the 32^{nd} International Conference on Machine Learning, Lille, France, 2015. JMLR: W&CP volume 37.

This problem underlies supervised learning (*e.g.* the training of logistic regressors when $\phi_i(z) = \log(1 + e^{-zb_i})$ and their regularized form when $\phi_i(z) = \log(1 + e^{-zb_i}) + \frac{\gamma}{2n}\|x\|_2^2$) and captures the widely-studied problem of linear least-squares regression when $\phi_i(z) = \frac{1}{2}(z - b_i)^2$.

Over the past five years, problems such as (1) have received increased attention, with a recent burst of activity in the design of fast *randomized* algorithms. Iterative methods that randomly sample the ϕ_i have been shown to outperform standard first-order methods under mild assumptions (Bottou & Bousquet, 2008; Johnson & Zhang, 2013; Xiao & Zhang, 2014; Defazio et al., 2014; Shalev-Shwartz & Zhang, 2014).

Despite the breadth of these recent results, their use in solving the ERM problem (1) lead to sub-optimal dependence on a natural notion of the problem's *condition number*. This dependence does, however, significantly impact the guarantees on running time, as high-dimensional problems encountered in practice are often poorly conditioned (due to strong correlation structure among variables). In particular, among the recent randomized algorithms, each either:

- Solves the ERM problem (1), under an assumption of strong convexity, with convergence that depends linearly on the problem's condition number (Johnson & Zhang, 2013; Defazio et al., 2014).
- Solves only an explicitly *regularized* ERM problem, $\min_x\{F(x) + \lambda r(x)\}$ where the regularizer r is a known strongly convex function and λ must be strictly positive, even when F is itself strongly convex. The first such result is due to Shalev-Shwartz & Zhang (2014) and achieves *acceleration*, *i.e.* dependence only on the square root of the regularized problem's condition number, which scales inversely with λ. Hence, taking small λ to solve the ERM problem

(where $\lambda = 0$ in effect) is not a viable option.

In this paper we show how to bridge this gap by a black-box reduction: solving the ERM problem (1), under an assumption of strong convexity, through repeated, approximate minimizations of the form $\min_x\{F(x) + \lambda r(x)\}$ for fairly large λ.

The key to our reduction is an approximate variant of the classical *proximal point algorithm* (PPA) (Rockafellar, 1976; Parikh & Boyd, 2014). Both PPA and the inner minimization procedure can then be accelerated and our analysis gives precise approximation requirements for either option. We show further practical improvements when the inner minimizer operates by a dual ascent method.

Table 1 summarizes our improvements in comparison to existing minimization procedures. Our end result is a family of algorithms, including:

1. An approximate proximal point algorithm (APPA), and a dual variant thereof, that matches the previous state-of-the-art running time and exhibits desirable empirical properties (Sections 2 and 5).
2. An accelerated variant of APPA that enjoys a running time with a square root dependence on the ERM problem's condition number (Theorem 2.6).

All analysis is performed in the common setting where each ϕ_i is smooth and the sum F is strongly convex (*e.g.* as in overdetermined linear regression).

Several of the algorithmic tools in this paper are similar in principle to (and sometimes appear indirectly in) work scattered throughout the machine learning and optimization literature – from classical treatments of error-tolerant PPA (Rockafellar, 1976; Güler, 1992) to the effective proximal term used by Shalev-Shwartz & Zhang (2014) in enabling their acceleration. By analyzing these as separate tools, and by bookkeeping the error requirements that they impose, we are able to assemble them into algorithms achieving an improved runtime.

Our techniques also extend naturally to the following more general optimization problem, which is fundamental in the theory of convex optimization,

$$\min_{x\in\mathbb{R}^d} \textstyle\sum_{i=1}^n \psi_i(x), \quad \text{where} \quad \psi_i : \mathbb{R}^d \to \mathbb{R}, \qquad (2)$$

and which covers ERM problems for multiclass and structured prediction. To simplify exposition and comparison to related work we focus on (1); extensions to (2) are made apparent in Section 2.

For the remainder of the introduction, we present the formal setup and cover previous approaches and running times. Section 2 provides the reduction framework and algorithms, then states the running time guarantees provided by the framework. Section 3 provides analysis for the simplest among the algorithms (the other two are analyzed in the appendix). Finally, Section 4 discusses implementation concerns, and Section 5 concludes with an empirical analysis. Most proofs are deferred to the appendix.

Empirical risk minimization		
Algorithm	Running time	Problem
GD	$dn^2\kappa\log(\epsilon_0/\epsilon)$	F
Accel. GD	$dn^{3/2}\sqrt{\kappa}\log(\epsilon_0/\epsilon)$	F
SAG, SVRG	$dn\kappa\log(\epsilon_0/\epsilon)$	F
SDCA	$dn\kappa'\log(\epsilon_0/\epsilon)$	$F+\lambda r$
Acc. SDCA, APCG	$dn\sqrt{\kappa'}\log(\epsilon_0/\epsilon)$	$F+\lambda r$
This work	$dn\sqrt{\kappa}\log(\epsilon_0/\epsilon)$	F

Linear least-squares regression		
Algorithm	Running time	Problem
Pseudoinverse	$nd^{\omega-1}$	$\|Ax-b\|_2^2$
Row sampling	$(nd+d^\omega)\log(\epsilon_0/\epsilon)$	$\|Ax-b\|_2^2$
Rand. Kaczmarz	$dn\kappa\log(\epsilon_0/\epsilon)$	$Ax=b$
Accel. coord.	$dn\sqrt{\kappa}\log(\epsilon_0/\epsilon)$	$Ax=b$
This work	$dn\sqrt{\kappa}\log(\epsilon_0/\epsilon)$	$\|Ax-b\|_2^2$

Table 1. Theoretical performance comparison on ERM and linear regression. Running times hold in expectation for randomized algorithms. In the "problem" column for ERM, F marks algorithms that can optimize the ERM objective (1), while $F+\lambda r$ marks those that only solve the explicitly regularized problem. For linear regression, $Ax=b$ marks algorithms that only solve consistent linear systems, whereas $\|Ax-b\|_2^2$ marks those that more generally minimize the squared loss. The constant ω is the exponent of the matrix multiplication running time (currently below 2.373 (Williams, 2012)).

1.1. Formal setup

Consider ERM (1) in the following common setting:

Assumption 1.1 (Regularity)**.** *Each loss function ϕ_i is L-smooth, i.e. for all $x, y \in \mathbb{R}$,*

$$\phi(y) \le \phi(x) + \phi'(x)(y-x) + \tfrac{L}{2}(y-x)^2,$$

and the sum F is μ-strongly convex, i.e. for all $x, y \in \mathbb{R}^d$,

$$F(x) \ge F(x) + \nabla F(x)^\mathsf{T}(y-x) + \tfrac{\mu}{2}\|y-x\|_2^2.$$

Let $R \stackrel{\text{def}}{=} \max_i \|a_i\|_2$, $A \in \mathbb{R}^{n\times d}$ be the matrix whose i'th row is a_i^T, and $\kappa = \lceil LR^2/\mu \rceil$ denote the *condition number*.

Although many algorithms are designed for special cases of the ERM objective F where there is some known, exploitable structure to the problem, our aim is to study the most general case subject to Assumption 1.1. To standardize the comparison among algorithms, we consider the following generic model of interaction with F:

Assumption 1.2 (Computational model). *For any $i \in [n]$ and $x \in \mathbb{R}^d$, two possible primitive operations are:*

- *For $b \in \mathbb{R}$, compute the gradient of $x \mapsto \phi_i(a_i^\mathsf{T} x - b)$.*
- *For $b \in \mathbb{R}$, $c \in \mathbb{R}^d$, minimize $\phi_i(a_i^\mathsf{T} x) + b\|x - c\|_2^2$.*

We refer to these operations, as well as to the evaluation of $\phi_i(a_i^\mathsf{T} x)$, as single accesses *to ϕ_i, and assume that these operations can be computed in $O(d)$ time.*

Notation Denote $[n] \stackrel{\text{def}}{=} \{1, \ldots, n\}$. Denote the optimal value of a convex function by $f^{\text{opt}} = \min_x f(x)$, and, when f is clear from context, let x^{opt} denote a minimizer. A point x' is an *ϵ-approximate minimizer* of f if $f(x') - f^{\text{opt}} \leq \epsilon$. The Fenchel dual of a convex function $f : \mathbb{R}^k \to \mathbb{R}$ is $f^* : \mathbb{R}^k \to \mathbb{R}$ defined by $f^*(y) = \sup_{x \in \mathbb{R}^k}\{\langle y, x\rangle - f(x)\}$.

1.2. Running times and related work

Table 1 compares our results with the running time of both classical and recent algorithms for solving the ERM problem (1) and linear least-squares regression.

In the more general context of the ERM problem, GD refers to canonical gradient descent on F, Accel. GD is Nesterov's accelerated gradient decent (Nesterov, 1983; 2004), SVRG is the stochastic variance-reduced gradient of Johnson & Zhang (2013), SAG is the stochastic average gradient of Roux et al. (2012) and Defazio et al. (2014), SDCA is the stochastic dual coordinate ascent of Shalev-Shwartz & Zhang (2013), Acc. SDCA is the accelerated proximal SDCA of Shalev-Shwartz & Zhang (2014) and APCG is the accelerated coordinate algorithm of Lin et al. (2014). The latter three algorithms are more restrictive in that they only solve the explicitly regularized problem $F + \lambda r$, even if F is itself strongly convex (such algorithms run in time inversely proportional to λ).

The running time of the algorithms are presented based on the setting considered in this paper, *i.e.* under Assumptions 1.1 and 1.2. Many of the algorithms can be applied in more general settings (*e.g.* even if the function F is not strongly convex) and have different convergence guarantees in those cases. The running times are characterized by four parameters: d is the data dimension, n is the number of samples, $\kappa = \lceil LR^2/\mu \rceil$ is the condition number (for $F + \lambda r$ the condition number $\kappa' = \lceil LR^2/\lambda \rceil$ is used) and ϵ_0/ϵ is the ratio between the initial and desired accuracy. Running times are stated per $\tilde{O}$-notation; factors that depend poly-logarithmically on n, κ, and κ' are ignored.

For the linear least-squares regression problem, there is greater variety in the algorithms that apply. For comparison we include the use of a Moore-Penrose pseudoinverse to compute a solution in closed form via the standard normal equations, algorithms based on the randomized Kaczmarz method (Strohmer & Vershynin, 2009; Needell et al., 2014) and their accelerated variant (Lee & Sidford, 2013), and algorithms based on subspace embedding or row sampling (Nelson & Nguyen, 2013; Li et al., 2013; Cohen et al., 2015). Some Kaczmarz-based methods can only solve the more restrictive problem of consistent systems (finding x satisfying $Ax = b$) rather than minimize the squared loss $\|Ax - b\|_2^2$. The running times depend on the same four parameters $n, d, \kappa, \epsilon_0/\epsilon$ as before, except for computing the closed-form pseudoinverse, which for simplicity we consider "exact," independent of initial and target errors ϵ_0/ϵ.

Condition numbers in machine learning The condition number of the ERM problem (1) captures notions of data complexity such as variable correlation. Machine learning problems are typically high-dimensional with highly correlated variables, which implies large κ. On the other hand these problems often need not be optimized to a precision far below the statistical noise level of $O(1/n)$, so $\log(\epsilon_0/\epsilon) = O(\log n)$. Hence, in the statistically interesting regime, condition number dependence often comprises the main concern in runtime bounds.

2. Main results

This section describes our framework for iteratively applying and accelerating certain minimization algorithms. When instantiated with recent fast algorithms we obtain, under Assumptions 1.1 and 1.2, algorithms guaranteed to solve the ERM problem in time $\tilde{O}(nd\sqrt{\kappa}\log(1/\epsilon))$.

Our framework stems from a critical insight of the classical *proximal point algorithm (PPA)* or *proximal iteration*: to minimize F (or more generally, any convex function) it suffices to iteratively minimize

$$f_{s,\lambda}(x) = F(x) + \tfrac{\lambda}{2}\|x - s\|_2^2 \tag{3}$$

for $\lambda > 0$ and proper choice of *center* $s \in \mathbb{R}^d$. PPA iteratively applies the update rule $x^{(t+1)} \leftarrow \operatorname{argmin}_x f_{x^{(t)},\lambda}(x)$ and converges to the minimizer of F. The minimization in the update is known as the *proximal operator* (Parikh & Boyd, 2014), and we refer to it in the sequel as the *inner* minimization problem.

We establish three distinct types of *approximate* proximal point algorithms, *i.e.* algorithms that do not require full inner minimization. Each enables the use of a different existing fast algorithm as its inner minimizer, in turn yielding several ways to obtain our improved ERM running time:

- Section 2.1 provides APPA: an algorithm most similar to PPA, but requiring inner minimization by only a *fixed* multiplicative constant in each iteration.
- Section 2.2 provides Accelerated APPA: an accelerated version of APPA. Its instantiation with SVRG (Johnson & Zhang, 2013) as an inner minimizer

Algorithm 1 Approximate PPA (APPA)

input $x^{(0)} \in \mathbb{R}^d$, $\lambda > 0$
input primal $(\frac{2(\lambda+\mu)}{\mu}, \lambda)$-oracle $\mathcal{P}$
  for $t = 1, \ldots, T$ **do**
    $x^{(t)} \leftarrow \mathcal{P}(x^{(t-1)})$
  end for
output $x^{(T)}$

achieves the accelerated runtime for the *general* ERM problem (2).

- Section 2.3 provides Dual APPA: an algorithm whose approximate inner minimizers operate on the dual $f_{s,\lambda}$, with warm starts between iterations. Dual APPA enables several inner minimizers that are otherwise incompatible with APPA. Its instantiation with APCG (Lin et al., 2014) or with Accelerated Proximal SDCA (Shalev-Shwartz & Zhang, 2014) as an inner minimizer achieves the accelerated runtime for the ERM problem (1).

2.1. An approximate proximal point algorithm

To design APPA, we quantify the error that can be tolerated of an inner minimizer, while accounting for the computational cost of ensuring such error. The abstraction we use is the following notion of inner approximation:

Definition 2.1. *An algorithm $\mathcal{P}$ is a* primal (c, λ)-oracle *if, given $x \in \mathbb{R}^d$, it outputs $\mathcal{P}(x)$ that is a $([f_{x,\lambda}(x) - f_{x,\lambda}^{opt}]/c)$-approximate minimizer of $f_{x,\lambda}$ in time $\mathcal{T}_{\mathcal{P}}$.*[1]

In other words, a primal oracle is an algorithm initialized at x that reduces the error of $f_{x,\lambda}$ by a $1/c$ fraction, in time that depends on λ, and c, and regularity properties of F.

Typical iterative first-order algorithms, such as those in Table 1, yield primal (c, λ)-oracles with runtimes $\mathcal{T}_{\mathcal{P}}$ that scale inversely in λ or $\sqrt{\lambda}$, and logarithmically in c. For instance:

Theorem 2.2 (SVRG (Johnson & Zhang, 2013))**.** *SVRG is a primal (c, λ)-oracle with runtime complexity $\mathcal{T}_{\mathcal{P}} = O(nd\lceil \frac{LR^2}{\mu+\lambda} \rceil \log c)$.*

Note, moreover, that SVRG is a primal oracle even for the *general* ERM problem (2).

APPA (Algorithm 1) takes any primal oracle and queries it repeatedly. We prove the following lemma to guarantee a geometric convergence rate for the iterates produced in this manner (proved in Section 3).

Lemma 2.3 (Contraction in APPA)**.** *Fix any $c' \in (0, 1)$, $x \in \mathbb{R}^d$, and primal $(\frac{\lambda+\mu}{c'\mu}, \lambda)$-oracle $\mathcal{P}$. If $x' = \mathcal{P}(x)$, then*[2]

$$F(x') - F^{opt} \leq \frac{\lambda + c'\mu}{\lambda + \mu} \left(F(x) - F^{opt}\right). \tag{4}$$

Lemma 2.3 in turn implies the following runtime bound for APPA.

Theorem 2.4 (Un-regularizing in APPA)**.** *Given a primal $(\frac{2(\mu+\lambda)}{\mu}, \lambda)$-oracle $\mathcal{P}$, Algorithm 1 minimizes the general ERM problem (2) to within accuracy ϵ in time*

$$O\left(\mathcal{T}_{\mathcal{P}} \frac{\mu + \lambda}{\mu} \log \frac{\epsilon_0}{\epsilon}\right)$$

Corollary 2.5. *Instantiating Theorem 2.4 with SVRG (Johnson & Zhang, 2013) as the primal oracle and taking $\lambda = \mu$ yields the running time bound $O(nd\kappa \log(\frac{2(\mu+\lambda)}{\mu}) \log(\epsilon_0/\epsilon))$ for the general ERM problem (2).*

2.2. Accelerated APPA

We show how to generically accelerate APPA by developing Algorithm 2, Accelerated APPA. In a sense, it uses inner minimizers more efficiently, but requires a more precise constant minimization factor. We prove a contraction lemma, analogous to Lemma 2.3, as part of a complete analysis of Accelerated APPA in Appendix D. As before, it implies the following runtime bound.

Theorem 2.6 (Un-regularizing in Accelerated APPA)**.** *Given a primal $\left(4(\frac{2\lambda+\mu}{\mu})^{3/2}, \lambda\right)$-oracle $\mathcal{P}$ for $\lambda \geq 2\mu$, Algorithm 2 minimizes the general ERM problem (2) to within accuracy ϵ in time*

$$O\left(\mathcal{T}_{\mathcal{P}} \sqrt{\frac{\mu + \lambda}{\mu}} \log \frac{\epsilon_0}{\epsilon}\right).$$

Algorithm 2 Accelerated APPA

input $x^{(0)} \in \mathbb{R}^d$, $\mu > 0$, $\lambda > 2\mu$
input primal $(4\rho^{3/2}, \lambda)$-oracle $\mathcal{P}$, where $\rho = \frac{\mu+2\lambda}{\mu}$
  Define $\zeta = \frac{2}{\mu} + \frac{1}{\lambda}$
  $v^{(0)} \leftarrow x^{(0)}$
  for $t = 0, \ldots, T-1$ **do**
    $y^{(t)} \leftarrow \frac{1}{1+\rho^{-1/2}} x^{(t)} + \frac{\rho^{-1/2}}{1+\rho^{-1/2}} v^{(t)}$
    $x^{(t+1)} \leftarrow \mathcal{P}(x^{(t)})$
    $g^{(t)} \leftarrow \lambda(y^{(t)} - x^{(t+1)})$
    $v^{(t+1)} \leftarrow (1 - \rho^{-1/2}) v^{(t)} + \rho^{-1/2} \left[y^{(t)} - \zeta g^{(t)}\right]$
  end for
output $x^{(T)}$

[1] When the oracle is a randomized algorithm, we require that its outputs is, in expectation, ϵ-approximate in the same way.

[2] When the oracle is a randomized algorithm, the guarantee (4) holds in expectation over x', conditioned on x.

Algorithm 3 Dual APPA

input $x^{(0)} \in \mathbb{R}^d$, $\lambda > 0$
input dual (σ, λ)-oracle $\mathcal{D}$ (see Theorem 2.10 for σ)
  $y^{(0)} \leftarrow \hat{y}(x^{(0)})$
  for $t = 1, \dots, T$ **do**
    $y^{(t)} \leftarrow \mathcal{D}(x^{(t-1)}, y^{(t-1)})$
    $x^{(t)} \leftarrow \hat{x}_{x^{(t-1)},\lambda}(y^{(t)})$
  end for
output $x^{(T)}$

Corollary 2.7. *Instantiating Theorem 2.6 with SVRG (Johnson & Zhang, 2013) as the primal oracle and taking $\lambda = 2\mu + LR^2$ yields the running time bound $\tilde{O}(nd\sqrt{\kappa}\log(\epsilon_0/\epsilon))$ for the general ERM problem (2).*

2.3. Dual APPA

Several oracles that are well-suited inner minimizers – for F as the ERM objective (1) in particular – operate in the *regularized ERM dual*. That is, $f_{s,\lambda}$ is minimized by instead decreasing the negative dual objective $g_{s,\lambda} : \mathbb{R}^n \to \mathbb{R}$ given by

$$g_{s,\lambda}(y) = G(y) + \frac{1}{2\lambda}\|A^\mathsf{T} y\|_2^2 - s^\mathsf{T} A^\mathsf{T} y, \tag{5}$$

where $G(y) = \sum_{i=1}^n \phi_i^*(y_i)$. To make corresponding progress in the primal, dual-based algorithms make use of the *dual-to-primal* mapping, given by

$$\hat{x}_{s,\lambda}(y) = s - \tfrac{1}{\lambda} A^\mathsf{T} y, \tag{6}$$

and the *primal-to-dual* mapping, given entrywise by

$$[\hat{y}(x)]_i = \left[\frac{\partial \phi_i(z)}{\partial z}\right]\Big|_{z = a_i^\mathsf{T} x} \tag{7}$$

for $i = 1, \dots, n$ (Appendix A elaborates on duality.) Such dual algorithms include SDCA and Accelerated Proximal SDCA (Shalev-Shwartz & Zhang, 2013; 2014).

The theoretical guarantees of dual-based algorithms can usually be augmented so that they fit Definition 2.1 of a primal oracle $\mathcal{P}(x)$, under the scheme where x determines $\hat{y}(x)$ as the initial dual point. However, such an initialization scheme is not typically considered in these algorithms' analyses, so some further custom-tailored proof is needed to ensure the oracle assumption is valid in this case.

Far more importantly, such an initialization scheme has an $O(nd)$ runtime overhead per APPA iteration. A more natural scheme is to operate contiguously in the dual, or in other words to "warm start," initializing by the dual point y from the end of the previous oracle invocation. This scheme incurs only $O(d)$ overhead (see Section 4) and we find that it provides a significant speedup in practice (Section 5). Altogether, we formalize this procedure as Dual APPA (Algorithm 3), which uses a *dual oracle*:

Definition 2.8. *An algorithm $\mathcal{D}$ is a* dual (c, λ)-oracle *if, given $s \in \mathbb{R}^d$ and $y \in \mathbb{R}^n$, it outputs $\mathcal{D}(s, y)$ that is a $([g_{s,\lambda}(y) - g_{s,\lambda}^{opt}]/c)$-approximate minimizer of $g_{s,\lambda}$ in time $\mathcal{T}_\mathcal{D}$.*[1]

For instance, it is immediate from its analysis that SDCA is a valid dual oracle:

Theorem 2.9 (SDCA (Shalev-Shwartz & Zhang, 2013))**.** *SDCA is a dual (c, λ)-oracle with runtime complexity $\mathcal{T}_\mathcal{D} = \tilde{O}(nd\lceil \frac{LR^2}{\lambda} \rceil \log c)$.*

By repeatedly querying a dual oracle – producing along the way primal iterates via the dual-to-primal mapping (6) – we obtain Dual APPA, with the following runtime bound:

Theorem 2.10 (Un-regularizing in Dual APPA)**.** *Given a dual (σ, λ)-oracle $\mathcal{D}$, Algorithm 3 minimizes the ERM problem (1) to within accuracy ϵ in time*

$$\tilde{O}\left(\mathcal{T}_\mathcal{D} \frac{1}{1-r} \log \frac{\epsilon_0}{\epsilon}\right),$$

where $\sigma = O(\mathrm{poly}(n, \kappa))$ and $r < 1$ is a positive scalar depending on $\lambda/(\lambda + \mu)$.

The proof of Theorem 2.10 is given in Appendix C, including the precise definition of the numerical constants r and σ in (19) and (20).

An acceleration scheme alternative to that of Section 2.2, which obtains the same overall running time as Accelerated APPA (Algorithm 2), is to run Dual APPA using an accelerated procedure for its inner dual oracle:

Corollary 2.11. *Instantiating Theorem 2.10 with APCG (Lin et al., 2014) or with Accelerated Proximal SDCA (Shalev-Shwartz & Zhang, 2014) as the dual oracle and taking $\lambda = LR^2$ yields the running time bound $\tilde{O}(nd\sqrt{\kappa}\log(\epsilon_0/\epsilon))$.*

2.4. Discussion

Notions of error-tolerance in (primal) PPA – for both its plain and accelerated variants – have been defined and studied in prior work (Rockafellar, 1976; Güler, 1992). These mainly consider the cumulative absolute error of a given sequence of minimizers of $f_{x^{(t)},\lambda}$, assuming that such a sequence is somehow provided. Such a view falls short of providing a complete and instantiable study of runtime complexity: how, and in what time, can we produce such minimizers? Most any procedure of interest begins at some initial point, and has runtime that depends on the *relative* error ratio between its start and end. Definitions 2.1

and 2.8 capture such procedures succinctly and hence Algorithms 1, 3, and 2 fully specify families of concrete algorithms. Analytically, the main challenge for proving runtime guarantees lies in proving contractions, as in Lemma 2.3 and Theorems 2.6 and 2.10, namely in showing that a constant reduction of relative error in inner problems suffices.

As a separate note, we believe that the presentation of Accelerated APPA simplifies, and clarifies in terms of broader convex optimization theory, the "outer loop" steps used in Accelerated Proximal SDCA to enable its acceleration (Shalev-Shwartz & Zhang, 2014).

3. Convergence analysis of APPA

In this section we present a proof of Lemma 2.3, which in turn implies the geometric convergence of APPA iterates and hence the overall runtime in Theorem 2.4. The analogous proofs for Dual APPA and Accelerated APPA are significantly more complex and deferred to the appendix. However, the simple analysis carried out in this section demonstrates the conceptual core of many of the tools involved, and in particular explains the fixed relative error reduction needed in APPA, as discussed in Section 2.

Note that, as is always the case with APPA (but not with Dual APPA), the function F in this section can represent any μ-strongly convex function. In particular, it need not represent the ERM objective as it does in other sections.

First, consider exact inner minimizers. The following relates the minimum of the sub-problem $f_{s,\lambda}$ to F^{opt}.

Lemma 3.1 (Relationship between minima)**.** *For $s \in \mathbb{R}^d$ and $\lambda \geq 0$*

$$f_{s,\lambda}^{opt} - F^{opt} \leq \frac{\lambda}{\mu + \lambda} \left(F(s) - F^{opt}\right).$$

Proof. Let $x^{\text{opt}} = \operatorname{argmin}_x F(x)$ and for all $\alpha \in [0,1]$ let $x_\alpha = (1-\alpha)s + \alpha x^{\text{opt}}$. The μ-strong convexity of F implies that, for all $\alpha \in [0,1]$,

$$F(x_\alpha) \leq (1-\alpha)F(s) + \alpha F(x^{\text{opt}}) - \frac{\alpha(1-\alpha)\mu}{2}\|s - x^{\text{opt}}\|_2^2.$$

Consequently, by the definition of $f_{s,\lambda}^{\text{opt}}$,

$$\begin{aligned} f_{s,\lambda}^{\text{opt}} &\leq F(x_\alpha) + \frac{\lambda}{2}\|x_\alpha - s\|_2^2 \\ &\leq (1-\alpha)F(s) + \alpha F(x^{\text{opt}}) \\ &\quad - \frac{\alpha(1-\alpha)\mu}{2}\|s - x^{\text{opt}}\|_2^2 + \frac{\lambda\alpha^2}{2}\|s - x^{\text{opt}}\|_2^2 \end{aligned}$$

Choosing $\alpha = \frac{\mu}{\mu+\lambda}$ yields the result. □

This immediately implies contraction for the exact PPA, as it implies that in every iteration of PPA the error in F decreases by a multiplicative $\lambda/(\lambda+\mu)$. Here we show that it also implies contraction for approximate minimizers. Namely, we use the lemma above to relate approximate error reduction in $f_{s,\lambda}$ to contraction in F:

Lemma 3.2 (Error reduction implies contraction)**.** *For any $x, s \in \mathbb{R}^d$, $\lambda > 0$, and $c > 0$, if*

$$f_{s,\lambda}(x) - f_{s,\lambda}^{opt} \leq \frac{1}{c}\left(f_{s,\lambda}(s) - f_{s,\lambda}^{opt}\right), \tag{8}$$

then

$$F(x) - F^{opt} \leq \left(\frac{1}{c} + \frac{\lambda}{\mu+\lambda}\right)\left(F(s) - F^{opt}\right)$$

Proof. Note that $f_{s,\lambda}(x) \geq F(x)$ and by the same reasoning $f_{s,\lambda}^{\text{opt}} \geq F^{\text{opt}}$. By definition, $f_{s,\lambda}(s) = F(s)$, so rearranging (8) and subtracting F^{opt} from both sides implies

$$F(x) - F^{\text{opt}} \leq \frac{1}{c}\left(F(s) - F^{\text{opt}}\right) + \left(f_{s,\lambda}^{\text{opt}} - F^{\text{opt}}\right).$$

Invoking Lemma 3.1 then yields the result. □

This inequality now essentially proves Lemma 2.3:

Proof of Lemma 2.3. Suppose we have $c' \in (0,1)$ and $x \in \mathbb{R}^d$, and that $x' = \mathcal{P}(x)$ where $\mathcal{P}$ is a primal $(\frac{\lambda+\mu}{c'\mu}, \lambda)$-oracle. By definition

$$f_{x,\lambda}(x') - f_{x,\lambda}^{\text{opt}} \leq \frac{c'\mu}{\lambda+\mu}\left(f_{x,\lambda}(x) - f_{x,\lambda}^{\text{opt}}\right).$$

Applying Lemma 3.2 proves the claim. □

Remark 3.3. The proofs in this section did not require that F be smooth, or even differentiable.

4. Practical concerns

While theoretical convergence rates lay out a broad-view comparison of the algorithms in the literature, we briefly remark on some of the finer-grained differences between algorithms, which inform their implementation or empirical behavior. To match the terminology used for SVRG in Johnson & Zhang (2013), we refer to a "stage" as a single step of APPA, *i.e.* the time spent executing the inner minimization of $f_{x^{(t)},\lambda}$ or $g_{x^{(t)},\lambda}$ (as in (3) and (5)).

Re-centering overhead of Dual APPA vs. SVRG At the end of every one of its stages, SVRG pauses to compute an exact gradient by a complete pass over the dataset (costing $\Theta(nd)$ time during which n gradients are computed). Although an amortized runtime analysis hides this cost, this operation cannot be carried out in-step with the iterative

updates of the previous stage, since the exact gradient is computed at a point that is only selected at the stage's end.

Meanwhile, if each stage in Dual APPA is initialized with a valid primal-dual pair for the inner problem, Dual APPA can update the current primal point together with every dual coordinate update, in time $O(d)$, *i.e.* with negligible increase in the overhead of the update. When doing so, the corresponding data row remains fresh in cache and, unlike SVRG, no additional gradient need be computed.

Moreover, initializing each stage with a valid such primal-dual pair can be done in only $O(d)$ time. At the end of a stage where s was the center point, Dual APPA holds a primal-dual pair (x, y) where $x = \hat{x}_s(y)$. The next stage is centered at x and the dual variables initialized at y, so it remains to set up a corresponding primal point $x' = \hat{x}_x(y) = x - \frac{1}{\lambda} A^\mathsf{T} y$. This can be done by computing $x' \leftarrow 2x - s$, since we know that $x - s = -\frac{1}{\lambda} A^\mathsf{T} y$.

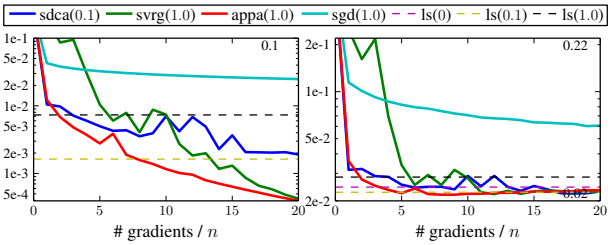
Decreasing λ APPA and Dual APPA enjoy the nice property that, as long as the inner problems are solved with enough accuracy, the algorithm does not diverge even for large choice of λ. In practice this allows us to start with a large λ and make faster inner minimizations. If we heuristically observe that the function error is not decreasing rapidly enough, we can switch to a smaller λ. Figure 3 (Section 5) demonstrates this empirically. On the contrary, algorithms like SGD or SVRG are more sensitive to the choice of λ and can suddenly diverge when it is taken too large. This makes them less amenable to (mid-run) dynamic parameter tuning.

Stable update steps Dual coordinate-wise ascent provides a convenient framework in which to derive parameter updates with data-dependent step sizes, or sometimes closed-form updates altogether (*i.e.* optimal solutions to each single-coordinate maximization sub-problem), which reduces the number of parameters needing tuning and can thereby improve stability overall.

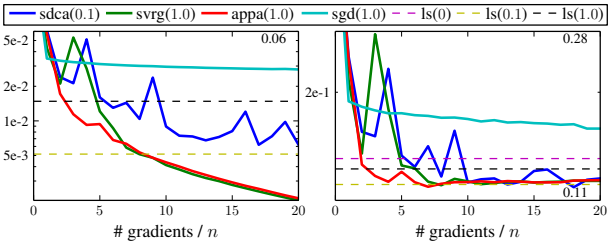
5. Empirical analysis

We experiment with Dual APPA in comparison with SDCA, SVRG, and SGD on several binary classification tasks.

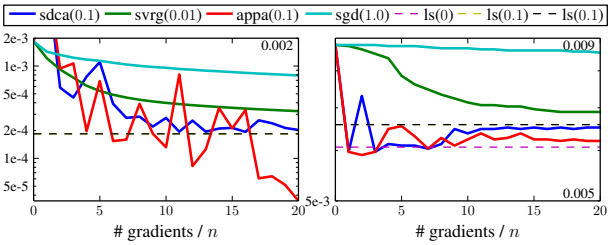
Beyond general benchmarking, the experiments also demonstrate the advantages of the unordinary "bias-variance tradeoff" presented by approximate proximal iteration: the vanishing proximal term empirically provides advantages of regularization (added strong convexity, lower variance) at a bias cost that is less severe than with typical ℓ_2 regularization. Even if some amount of ℓ_2 shrinkage is desired, Dual APPA can place yet higher weight on its ℓ_2 term, enjoy improved speed and stability, and after a few stages achieve roughly the desired bias.



(a) MNIST. Left: excess train loss $F(x) - F^{\text{opt}}$. Right: test error rate.



(b) CIFAR. Left: excess train loss $F(x) - F^{\text{opt}}$. Right: test error rate.



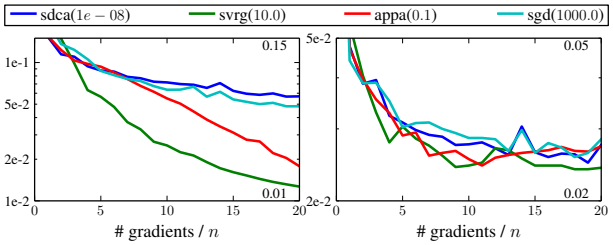
(c) Protein. Left: excess train loss $F(x) - F^{\text{opt}}$. Right: test error rate.

Figure 1. Sub-optimality curves when optimizing under squared loss $\phi_i(z) = \frac{1}{2n}(z - b_i)^2$.

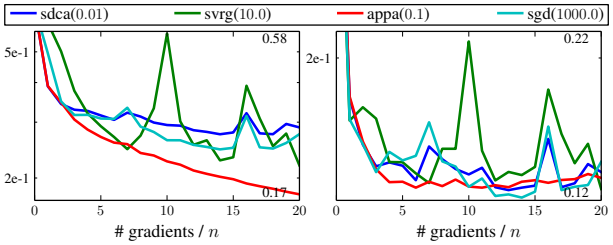
Datasets In this section we show results for three binary classification tasks, derived from MNIST,[3] CIFAR-10,[4] and Protein:[5] in MNIST we classify the digits $\{1, 2, 4, 5, 7\}$ vs. the rest, and in CIFAR we classify the animal categories vs. the automotive ones. MNIST and CIFAR are taken under non-linear feature transformations that increase the problem scale significantly: we normalize the rows by scaling the data matrix by the inverse average ℓ_2 row norm. We then take take $n/5$ random Fourier features per the randomized scheme of Rahimi & Recht (2007). This yields 12K features for MNIST (60K training examples, 10K test) and 10K for CIFAR (50K training examples, 10K test). Meanwhile, Protein is a standard pre-featurized benchmark (75 features, ∼117K training examples, ∼30K test) that we preprocess minimally by row normalization and an appended affine feature, and whose train/test split we obtain by randomly holding out 20% of the original labeled data.

Algorithms Each algorithm is parameterized by a scalar value λ analogous to the λ used in proximal iteration: λ

[3] http://yann.lecun.com/exdb/mnist/
[4] http://www.cs.toronto.edu/~kriz/cifar.html
[5] http://osmot.cs.cornell.edu/kddcup/datasets.html

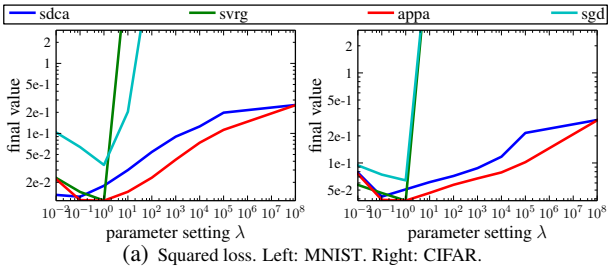


(a) MNIST. Left: train loss $F(x)$. Right: test error rate.

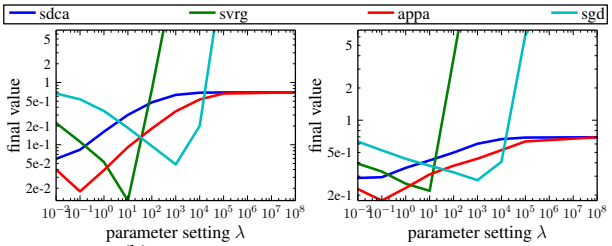


(b) CIFAR. Left: train loss $F(x)$. Right: test error rate.

Figure 2. Objective curves when optimizing under logistic loss $\phi_i(z) = \frac{1}{n}\log(1 + e^{-zb_i})$.



(a) Squared loss. Left: MNIST. Right: CIFAR.



(b) Logistic loss. Left: MNIST. Right: CIFAR.

Figure 3. Sensitivity to λ: the final objective values attained by each algorithm, after 20 stages (or the equivalent), with λ chosen at different orders of magnitude. SGD and SVRG exhibit a sharp threshold past which they easily diverge, whereas SDCA degrades more gracefully, and Dual APPA yet more so.

is the step size for SVRG, $\lambda t^{-1/2}$ is the decaying step size for SGD, and $\frac{\lambda}{2}\|x\|_2^2$ is the ridge penalty for SDCA. (See Johnson & Zhang (2013) for a comparison of SVRG to a more thoroughly tuned SGD under different decay schemes.) We use Dual APPA (Algorithm 3) with SDCA as the inner minimizer. For the algorithms with a notion of a stage – *i.e.* Dual APPA's time spent invoking the inner minimizer, SVRG's period between computing exact gradients – we set the stage size equal to the dataset size for simplicity.[6] SVRG is given an advantage in that we choose not to count its gradient computations when it computes the exact gradient between stages. All algorithms are initialized at $x = 0$. Each algorithm was run under $\lambda = 10^i$ for $i = -8, -7, \ldots, 8$, and plots report the trial that best minimized the original ERM objective.

Convergence and bias The proximal term in APPA introduces a vanishing bias for the problem (towards the initial point of $x = 0$) that provides a speedup by adding strong convexity to the problem. We investigate a natural baseline: for the purpose of minimizing the original ERM problem, how does APPA compare to solving one instance of a regularized ERM problem (using a single run of its inner optimizer)? In other words, to what extent does re-centering the regularizer over time help in solving the un-regularized problem? Intuitively, even if SDCA is run to convergence, some of the minimization is of the regularization term rather than the ERM term, hence one cannot weigh the regularization too heavily. Meanwhile, APPA can enjoy more ample strong convexity by placing a larger weight on its ℓ_2 term. This advantage is evident for MNIST and CIFAR in Figures 1 and 2: recalling that λ is the same strong convexity added both by APPA and by SDCA, we see that APPA takes λ at least an order of magnitude larger than SDCA does, to achieve faster and more stable convergence towards an ultimately lower final value.

Figure 1 also shows dashed lines corresponding to the ERM performance of the least-squares fit and of fully-optimized ridge regression, using λ as that of the best APPA and SDCA runs. These appear in the legend as "ls(λ)." They indicate lower bounds on the ERM value attainable by *any* algorithm that minimizes the corresponding regularized ERM objective. Lastly, test set classification accuracy demonstrates the extent to which a shrinkage bias is statistically desirable. In the MNIST and CIFAR holdout, we want only the small bias taken explicitly by SDCA (and effectively achieved by APPA). In the Protein holdout, we want no bias at all (again effectively achieved by APPA).

Parameter sensitivity By solving only regularized ERM inner problems, SDCA and APPA enjoy a stable response to poor specification of the biasing parameter λ. Figure 3 plots the algorithms' final value after 20 stages, against different choices of λ. Overestimating the step size in SGD or SVRG incurs a sharp transition into a regime of divergence. Meanwhile, APPA and SDCA always converge, with solution quality degrading more smoothly. APPA then exhibits an even better degradation as it overcomes an overaggressive biasing by the 20th stage.

[6] Such a choice is justified by the observation that doubling the stage size does not have noticeable effect on the results discussed.

Acknowledgments

Part of this work took place while RF and AS were at Microsoft Research, New England, and another part while AS was visiting the Simons Institute for the Theory of Computing, UC Berkeley. This work was partially supported by NSF awards 0843915 and 1111109, NSF Graduate Research Fellowship (grant no. 1122374).

References

Bottou, L. and Bousquet, O. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2008.

Cohen, M. B., Lee, Y. T., Musco, C., Musco, C., Peng, R., and Sidford, A. Uniform sampling for matrix approximation. In *Innovations in Theoretical Computer Science (ITCS)*, 2015.

Defazio, A., Bach, F., and Lacoste-Julien, S. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

Güler, O. New proximal point algorithms for convex minimization. *SIAM Journal on Optimization*, 2(4):649–664, 1992.

Johnson, R. and Zhang, T. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

Lee, Y. T. and Sidford, A. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. In *Foundations of Computer Science (FOCS)*, 2013.

Li, M., Miller, G. L., and Peng, R. Iterative row sampling. In *Foundations of Computer Science (FOCS)*, 2013.

Lin, Q., Lu, Z., and Xiao, L. An accelerated proximal coordinate gradient method. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

Needell, D., Srebro, N., and Ward, R. Stochastic gradient descent, weighted sampling, and the randomized kaczmarz algorithm. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

Nelson, J. and Nguyen, H. L. Osnap: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *Foundations of Computer Science (FOCS)*, 2013.

Nesterov, Y. A method of solving a convex programming problem with convergence rate $o(1/k^2)$. *Soviet Mathematics Doklady*, 27(2):372–376, 1983.

Nesterov, Y. *Introductory Lectures on Convex Optimization: A Basic Course*. Springer, 2004.

Parikh, N. and Boyd, S. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):123–231, 2014.

Rahimi, A. and Recht, B. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems (NIPS)*, 2007.

Rockafellar, R. T. Monotone operators and the proximal point algorithm. *SIAM Journal on Control and Optimization*, 14(5):877–898, 1976.

Roux, N. L., Schmidt, M., and Bach, F. A stochastic gradient method with an exponential convergence rate for finite training sets. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

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

Shalev-Shwartz, S. and Zhang, T. Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization. *Mathematical Programming*, pp. 1–41, 2014.

Strohmer, T. and Vershynin, R. A randomized kaczmarz algorithm with exponential convergence. *Journal of Fourier Analysis and Applications*, 15:262–278, 2009.

Williams, V. V. Multiplying matrices faster than Coppersmith-Winograd. In *Symposium on Theory of Computing (STOC)*, 2012.

Xiao, L. and Zhang, T. A proximal stochastic gradient method with progressive variance reduction. *SIAM Journal on Optimization*, 24(4):2057–2075, 2014.