Mini-Batch Primal and Dual Methods for SVMs

Martin Takáč MARTIN.TAKI@GMAIL.COM
University of Edinburgh, JCMB, King's Buildings, EH9 3JZ, Edinburgh, UK

Avleen Bijral ABIJRAL@TTIC.EDU
Toyota Technological Institute at Chicago, 6045 S. Kenwood Ave., Chicago, Illinois 60637, USA

Peter Richtárik PETER.RICHTARIK@ED.AC.UK
University of Edinburgh, JCMB, King's Buildings, EH9 3JZ, Edinburgh, UK

Nathan Srebro NATI@UCHICAGO.EDU
Toyota Technological Institute at Chicago, 6045 S. Kenwood Ave., Chicago, Illinois 60637, USA

Abstract

We address the issue of using mini-batches in stochastic optimization of SVMs. We show that the same quantity, the *spectral norm of the data*, controls the parallelization speedup obtained for both primal stochastic subgradient descent (SGD) and stochastic dual coordinate ascent (SCDA) methods and use it to derive novel variants of mini-batched SDCA. Our guarantees for both methods are expressed in terms of the original nonsmooth primal problem based on the hinge-loss.

1. Introduction

Stochastic optimization approaches have been shown to have significant theoretical and empirical advantages in training linear Support Vector Machines (SVMs), as well as in many other learning applications, and are often the methods of choice in practice. Such methods use a single, randomly chosen, training example at each iteration. In the context of SVMs, approaches of this form include primal stochastic gradient descent (SGD) (e.g., Pegasos, Shalev-Shwartz et al. 2011, NORMA, Zhang 2004, and dual stochastic coordinate ascent (e.g., SDCA, Hsieh et al. 2008, RCDM, Richtárik & Takáč 2013).

However, the inherent sequential nature of such approaches becomes a problematic limitation for parallel and distributed computations as the predictor must be updated after each training point is processed, providing very little opportunity for parallelization. A popular remedy is to use *mini-batches*. That is, to use several training points at each iteration, instead of just one, calculating the update based on each point separately and aggregating the updates. The question is then whether basing each iteration on several points can indeed reduce the number of required iterations, and thus yield parallelization speedups.

In this paper, we consider using mini-batches with Pegasos (SGD on the primal objective) and with Stochastic Dual Coordinate Ascent (SDCA). We show that for *both methods*, the quantity that controls the speedup obtained using mini-batching/parallelization is the *spectral norm of the data*.

In Section 3 we provide the first analysis of mini-batched Pegasos (with the original, non-smooth, SVM objective) that provably leads to parallelization speedups (Theorem 1). The idea of using mini-batches with Pegasos is not new, and is discussed already by Shalev-Shwartz et al. (2011), albeit without a theoretical justification. The original analysis does not benefit from using mini-batches—the same number of iterations is required even when large mini-batches are used, there is no speedup, and the serial runtime (overall number of operations, in this case data accesses) increases linearly with the mini-batch size. In fact, no parallelization speedup can be guaranteed based only on a bound on the radius of the data, as in the original Pegasos analysis. Instead, we provide a refined analysis based on the spectral norm of the data.

We then move on to SDCA (Section 4). We show the situation is more involved, and a modification to the method is necessary. SDCA has been consistently

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

shown to outperform Pegasos in practice (Hsieh et al., 2008; Shalev-Shwartz et al., 2011), and is also popular as it does not rely on setting a step-size as in Pegasos. It is thus interesting and useful to obtain mini-batch variants of SDCA as well. We first show that a naive mini-batching approach for SDCA can fail, in particular when the mini-batch size is large relative to the spectral norm (Section 4.1). We then present a "safe" variant, and an analysis that establishes the same spectral-norm-dependent parallelization speedups as for Pegasos (Section 4.2). Similar to a recent analysis of non-mini-batched SDCA by Shalev-Shwartz & Zhang (2012), we establish a guarantee on the duality gap, and thus also on the suboptimality of the *primal* SVM objective, when using mini-batched SDCA (Theorem 2). We then go on to describe a more aggressive, adaptive, method for mini-batched SDCA, which is based on the analysis of the "safe" approach, and which we show often outperforms it in practice (Section 4.3, with experiments in Section 5).

For simplicity of presentation we focus on the hinge loss, as in the SVM objective. However, *all our results for both Pegasos and SDCA are valid for any Lipschitz continuous loss function.*

Related Work. Several recent papers consider the use of mini-batches in stochastic gradient descent, as well as stochastic dual averaging and stochastic mirror descent, when minimizing a *smooth* loss function (Dekel et al., 2012; Agarwal & Duchi, 2011; Cotter et al., 2011). These papers establish parallelization speedups for *smooth* loss minimization with mini-batches, possibly with the aid of some "acceleration" techniques, and without relying on, or considering, the spectral norm of the data. However, these results do not apply to SVM training, where the objective to be minimized is the non-smooth hinge loss. In fact, the only data assumption in these papers is an assumption on the radius of the data, which is *not* enough for obtaining parallelization guarantees when the loss is non-smooth.

Our contribution is thus orthogonal to prior work, showing that it is possible to obtain parallelization speedups even for non-smooth objectives, but only with a dependence on the spectral norm. We also analyze SDCA, which is a substantially different method from the methods analyzed in these papers. It is interesting to note that a bound of the spectral norm could perhaps indicate that it is easier to "smooth" the objective, and thus allow obtaining results similar to ours (i.e. on the suboptimality of the original non-smooth objective) by smoothing the objective and relying on mini-batched smooth SGD, where the spectral norm might control how well the smoothed loss captures the original loss. But we are not aware of any analysis of this nature, nor whether such an analysis is possible.

There has been some recent work on mini-batched coordinate descent methods for ℓ_1-regularized problems (and, more generally, regularizes by a separable convex function), similar to the SVM dual. Bradley et al. (2011) presented and analyzed SHOTGUN, a parallel coordinate descent method for ℓ_1-regularized problems, showing linear speedups for mini-batch sizes bounded in terms of the spectral norm of the data. The analysis does not directly apply to the SVM dual because of the box constraints, but is similar in spirit. Furthermore, Bradley et al. (2011) do not discuss a "safe" variant which is applicable for any mini-batch size, and only study the analogue of what we refer to as "naive" mini-batching (Section 4.1). More directly related is recent work of Richtárik & Takáč (2013; 2012); Tappenden et al. (2013); Richtárik & Takáč which provided a theoretical framework and analysis for a more general setting than SHOTGUN, that includes also the SVM dual as a special case. However, guarantees in this framework, as well as those of Bradley et al. (2011), are only on the dual suboptimality (in our terminology), and not on the more relevant primal suboptimality, i.e., the suboptimality of the original SVM problem we are interested in. Our theoretical analysis builds on that of Richtárik & Takáč (2012), combined with recent ideas of Shalev-Shwartz & Zhang (2012) for "standard" (serial) SDCA, to obtain bounds on the duality gap and primal suboptimality.

2. Support Vector Machines

We consider the optimization problem of training a linear[1] Support Vector Machine (SVM) based on n labeled training examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \pm 1$. We use $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n] \in \mathbb{R}^{d \times n}$ to denote the matrix of training examples. We assume the data is normalized such that $\max_i \|\mathbf{x}_i\| \leq 1$, and thus suppress the dependence on $\max_i \|\mathbf{x}_i\|$ in all results. Training a SVM corresponds to finding a linear predictor $\mathbf{w} \in \mathbb{R}^d$ with low ℓ_2-norm $\|\mathbf{w}\|$ and small (empirical) average hinge loss $\hat{L}(\mathbf{w}) := \frac{1}{n}\sum_{i=1}^n \ell(y_i \langle \mathbf{w}, \mathbf{x}_i \rangle)$, where $\ell(z) := [1-z]_+ = \max\{0, 1-z\}$. This bi-

[1]Since both Pegasos and SDCA can be kernelized, all methods discussed are implementable also with kernels, and all our results hold. However, the main advantage of SGD and SDCA is where the feature map is given explicitly, and so we focus our presentation on this setting.

objective problem can be serialized as

$$\min_{\mathbf{w}\in\mathbb{R}^d}\left[\mathbf{P}(\mathbf{w}) := \tfrac{1}{n}\textstyle\sum_{i=1}^n \ell(y_i\left\langle \mathbf{w},\mathbf{x}_i\right\rangle) + \tfrac{\lambda}{2}\left\|\mathbf{w}\right\|^2\right], \quad (1)$$

where $\lambda > 0$ is a regularization trade-off parameter. It is also useful to consider the dual of (1):

$$\max_{\boldsymbol{\alpha}\in\mathbb{R}^n, 0\le\boldsymbol{\alpha}_i\le 1}\left[\mathbf{D}(\boldsymbol{\alpha}) := \tfrac{-\boldsymbol{\alpha}^\top\mathbf{Q}\boldsymbol{\alpha}}{2\lambda n^2} + \tfrac{1}{n}\textstyle\sum_{i=1}^n\boldsymbol{\alpha}_i\right], \quad (2)$$

$$\mathbf{Q}\in\mathbb{R}^{n\times n}, \quad \mathbf{Q}_{i,j} = y_i y_j\left\langle \mathbf{x}_i,\mathbf{x}_j\right\rangle; \quad (3)$$

$\mathbf{Q}$ is the Gram matrix of the (labeled) data. The (primal) optimum of (1) is given by $\mathbf{w}^* = \frac{1}{\lambda n}\sum_{i=1}^n \boldsymbol{\alpha}_i^* y_i\mathbf{x}_i$, where $\boldsymbol{\alpha}^*$ is the (dual) optimum of (2). It is thus natural to associate with each dual solution $\boldsymbol{\alpha}$ a primal solution (i.e., a linear predictor)

$$\mathbf{w}(\boldsymbol{\alpha}) := \tfrac{1}{\lambda n}\textstyle\sum_{i=1}^n\boldsymbol{\alpha}_i y_i\mathbf{x}_i. \quad (4)$$

We will be discussing "mini-batches" of size b, represented by *random subsets* $A\subseteq\langle n\rangle := \{1,2,\ldots,n\}$ of examples, drawn uniformly at random from all subsets of $\langle n\rangle$ of cardinality b. Whenever we draw such a subset, we for simplicity write $A\in\mathrm{Rand}(b)$. By $\mathbf{Q}_A\in\mathbb{R}^{b\times b}$ we denote the submatrix of $\mathbf{Q}$ corresponding to rows and columns indexed by A, $\mathbf{v}_A\in\mathbb{R}^b$ to denote a similar restriction of a vector $\mathbf{v}\in\mathbb{R}^n$, and $\mathbf{v}_{[A]}\in\mathbb{R}^n$ for the "censored" vector where entries inside A are as in $\mathbf{v}$ and entries outside A are zero. The average hinge loss on examples in A is denoted by

$$\hat{L}_A(\mathbf{w}) := \tfrac{1}{b}\textstyle\sum_{i\in A}\ell(y_i\left\langle \mathbf{w},\mathbf{x}_i\right\rangle). \quad (5)$$

3. Mini-Batches in Primal Stochastic Gradient Descent Methods

Algorithm 1 Pegasos with Mini-Batches

Input: $\{(\mathbf{x}_i,y_i)\}_{i=1}^n$, $\lambda>0$, $b\in\langle n\rangle$, $T\ge 1$
Initialize: set $\mathbf{w}^{(1)} = \mathbf{0}\in\mathbb{R}^d$
for $t=1$ **to** T **do**
  Choose random mini-batch $A_t\in\mathrm{Rand}(b)$
  $\eta_t = \frac{1}{\lambda t}$, $A_t^+ = \{i\in A_t\ :\ y_i\langle\mathbf{w}^{(t)},\mathbf{x}_i\rangle < 1\}$
  $\mathbf{w}^{(t+1)} = (1-\eta_t\lambda)\mathbf{w}^{(t)} + \frac{\eta_t}{b}\sum_{i\in A_t^+} y_i\mathbf{x}_i$
end for
Output: $\bar{\mathbf{w}}^{(T)} = \frac{2}{T}\sum_{t=\lfloor T/2\rfloor+1}^T\mathbf{w}^{(t)}$

Pegasos is an SGD approach to solving (1), where at each iteration the iterate $\mathbf{w}^{(t)}$ is updated based on an unbiased estimator of a sub-gradient of the objective $\mathbf{P}(\mathbf{w})$. Whereas in a "pure" stochastic setting, the subgradient is estimated based on only a *single* training example, in our mini-batched variation (Algorithm 1) at each iteration we consider the partial objective:

$$\mathbf{P}_t(\mathbf{w}) := \hat{L}_{A_t}(\mathbf{w}) + \tfrac{\lambda}{2}\left\|\mathbf{w}\right\|^2, \quad (6)$$

where $A_t\in\mathrm{Rand}(b)$. We then calculate the subgradient of the partial objective $\mathbf{P}_t$ at $\mathbf{w}^{(t)}$:

$$\boldsymbol{\nabla}^{(t)} := \nabla\mathbf{P}_t(\mathbf{w}^{(t)}) \overset{(6)}{=} \nabla\hat{L}_{A_t}(\mathbf{w}^{(t)}) + \lambda\mathbf{w}^{(t)}, \quad (7)$$

where

$$\nabla\hat{L}_A(\mathbf{w}) \overset{(5)}{=} -\tfrac{1}{b}\textstyle\sum_{i\in A}\chi_i(\mathbf{w})y_i\mathbf{x}_i \quad (8)$$

and $\chi_i(\mathbf{w}) := 1$ if $y_i\left\langle\mathbf{w},\mathbf{x}_i\right\rangle < 1$ and 0 otherwise (indicator for not classifying example i correctly with a margin). The next iterate is obtained by setting $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta_t\boldsymbol{\nabla}^{(t)}$. We can now write

$$\mathbf{w}^{(t+1)} \overset{(7)\pm(8)}{=} (1-\eta_t\lambda)\mathbf{w}^{(t)} + \tfrac{\eta_t}{b}\sum_{i\in A_t}\chi_i(\mathbf{w}^{(t)})y_i\mathbf{x}_i. \quad (9)$$

Analysis of mini-batched Pegasos rests on bounding the norm of the subgradient estimates $\boldsymbol{\nabla}^{(t)}$. The standard Pegasos analysis uses the bound $\|\nabla\hat{L}_A(\mathbf{w})\| \le \frac{1}{b}\sum_{i\in A}\|\chi_i(\mathbf{w})y_i\mathbf{x}_i\| \le \frac{1}{b}\sum_{i\in A}1 = 1$. From (7) we then get $\|\boldsymbol{\nabla}^{(t)}\|\le\lambda\|\mathbf{w}^{(t)}\|+1$; the standard Pegasos analysis follows. This bound relies only on the assumption $\max_i\|\mathbf{x}_i\|\le 1$, and is the tightest bound without further assumptions on the data.

The core novel observation here is that the expected (square) norm of $\nabla\hat{L}_A$ can be bounded in terms of (an upper bound on) *the spectral norm of the data:*

$$\sigma^2 \ge \tfrac{1}{n}\left\|\mathbf{X}\right\|^2 = \tfrac{1}{n}\|\textstyle\sum_i\mathbf{x}_i\mathbf{x}_i^\top\| \overset{(3)}{=} \tfrac{1}{n}\left\|\mathbf{Q}\right\|, \quad (10)$$

where $\|\cdot\|$ denotes the spectral norm (largest singular value) of a matrix. In order to bound $\nabla\hat{L}_A$, we first perform the following calculation, introducing the key quantity β_b, useful also in the analysis of SDCA.

Lemma 1. *For any* $\mathbf{v}\in\mathbb{R}^n$, $\tilde{\mathbf{Q}}\in\mathbb{R}^{n\times n}$, $A\in\mathrm{Rand}(b)$,

$$\mathbb{E}[\mathbf{v}_{[A]}^\top\tilde{\mathbf{Q}}\mathbf{v}_{[A]}] = \tfrac{b}{n}[(1-\tfrac{b-1}{n-1})\textstyle\sum_{i=1}^n\tilde{\mathbf{Q}}_{ii}\mathbf{v}_i^2 + \tfrac{b-1}{n-1}\mathbf{v}^\top\tilde{\mathbf{Q}}\mathbf{v}].$$

Moreover, if $\tilde{\mathbf{Q}}_{ii}\le 1$ *for all* i *and* $\frac{1}{n}\|\tilde{\mathbf{Q}}\|\le\sigma^2$, *then*

$$\mathbb{E}[\mathbf{v}_{[A]}^\top\tilde{\mathbf{Q}}\mathbf{v}_{[A]}] \le \tfrac{b}{n}\beta_b\left\|\mathbf{v}\right\|^2, \text{ where}$$

$$\beta_b := 1 + \tfrac{(b-1)(n\sigma^2-1)}{n-1}. \quad (11)$$

Proof.

$$\begin{aligned}\mathbb{E}[\mathbf{v}_{[A]}^\top\tilde{\mathbf{Q}}\mathbf{v}_{[A]}] &= \mathbb{E}[\textstyle\sum_{i\in A}\mathbf{v}_i^2\tilde{\mathbf{Q}}_{ii} + \sum_{i,j\in A, i\ne j}\mathbf{v}_i\mathbf{v}_j\tilde{\mathbf{Q}}_{ij}]\\ &\overset{(*)}{=} b\mathbb{E}_i[\mathbf{v}_i^2\tilde{\mathbf{Q}}_{ii}] + b(b-1)\mathbb{E}_{i,j}[\mathbf{v}_i\mathbf{v}_j\tilde{\mathbf{Q}}_{ij}]\\ &= \tfrac{b}{n}\textstyle\sum_i\tilde{\mathbf{Q}}_{ii}\mathbf{v}_i^2 + \tfrac{b(b-1)}{n(n-1)}\mathbf{v}^\top(\tilde{\mathbf{Q}}-\mathrm{diag}(\tilde{\mathbf{Q}}))\mathbf{v}\\ &= \tfrac{b}{n}[(1-\tfrac{b-1}{n-1})\textstyle\sum_i\tilde{\mathbf{Q}}_{ii}\mathbf{v}_i^2 + \tfrac{b-1}{n-1}\mathbf{v}^\top\tilde{\mathbf{Q}}\mathbf{v}],\end{aligned}$$

where in $(*)$ the expectations are over i,j chosen uniformly at random without replacement. Now using $\tilde{\mathbf{Q}}_{ii}\le 1$ and $\|\tilde{\mathbf{Q}}\|\le n\sigma^2$, we can further write

$$\le \tfrac{b}{n}[(1-\tfrac{b-1}{n-1})\left\|\mathbf{v}\right\|^2 + \tfrac{b-1}{n-1}n\sigma^2\left\|\mathbf{v}\right\|^2] = \tfrac{b}{n}\beta_b\left\|\mathbf{v}\right\|^2. \square$$

We can now apply Lemma 1 to $\nabla \hat{L}_A$:

Lemma 2. *For any $\mathbf{w} \in \mathbb{R}^d$ and $A \in \mathrm{Rand}(b)$ we have $\mathbb{E}[\|\nabla \hat{L}_A(\mathbf{w})\|^2] \leq \frac{\beta_b}{b}$, where β_b is as in Lemma 1.*

Proof. If $\chi \in \mathbb{R}^n$ is the vector with entries $\chi_i(\mathbf{w})$, then

$$\mathbb{E}[\|\nabla \hat{L}_A(\mathbf{w})\|^2] \overset{(8)}{=} \mathbb{E}[\|\tfrac{1}{b}\textstyle\sum_{i\in A}\boldsymbol{\chi}_i y_i \mathbf{x}_i\|^2] \overset{(3)}{=} \tfrac{1}{b^2}\mathbb{E}[\boldsymbol{\chi}_{[A]}^\top \mathbf{Q} \boldsymbol{\chi}_{[A]}] \overset{(\text{Lem1})}{\leq} \tfrac{1}{b^2}\tfrac{b}{n}\beta_b \|\boldsymbol{\chi}\|^2 \leq \tfrac{\beta_b}{b}. \quad \square$$

Using the by-now standard analysis of SGD for strongly convex functions, we obtain the main result of this section:

Theorem 1. *After T iterations of Pegasos with mini-batches (Algorithm 1), we have that for the averaged iterate $\bar{\mathbf{w}}^{(T)} = \frac{2}{T}\sum_{t=\lfloor T/2\rfloor+1}^{T} \mathbf{w}^{(t)}$:*

$$\mathbb{E}\left[\mathbf{P}(\bar{\mathbf{w}}^{(T)})\right] - \inf_{\mathbf{w}\in\mathbb{R}^d} \mathbf{P}(\mathbf{w}) \leq \tfrac{\beta_b}{b} \cdot \tfrac{30}{\lambda T}.$$

Proof. Unrolling (9) with $\eta_t = 1/(\lambda t)$ yields

$$\mathbf{w}^{(t)} = -\tfrac{1}{\lambda(t-1)}\textstyle\sum_{\tau=1}^{t-1} g^{(\tau)}, \tag{12}$$

where $g^{(\tau)} := \nabla \hat{L}_{A_\tau}(\mathbf{w}^{(\tau)})$. Using the inequality $\|\sum_{\tau=1}^{t-1} g^{(\tau)}\|^2 \leq (t-1)\sum_{\tau=1}^{t-1}\|g^{(\tau)}\|^2$, we now get

$$\mathbb{E}[\|\mathbf{w}^{(t)}\|^2] \overset{(12)}{\leq} \textstyle\sum_{\tau=1}^{t-1} \frac{\mathbb{E}[\|g^{(\tau)}\|^2]}{\lambda^2(t-1)} \overset{(\text{Lem2})}{\leq} \frac{\beta_b}{\lambda^2 b}, \tag{13}$$

$$\mathbb{E}[\|\boldsymbol{\nabla}^{(t)}\|^2] \overset{(7)+(\text{Lem2})}{\leq} 2(\lambda^2\mathbb{E}[\|\mathbf{w}^{(t)}\|^2] + \tfrac{\beta_b}{b}) \overset{(13)}{\leq} 4\tfrac{\beta_b}{b}.$$

The performance guarantee is now given by the analysis of SGD with tail averaging (Theorem 5 of Rakhlin et al. 2012, with $\alpha = \frac{1}{2}$ and $G^2 = 4\frac{\beta_b}{b}$). $\square$

Parallelization speedup. When $b = 1$ we have $\beta_b = 1$ (see (11)) and Theorem 1 agrees with the standard (serial) Pegasos analysis[2] (Shalev-Shwartz et al., 2011). For larger mini-batches, the guarantee depends on the quantity β_b, which in turn depends on the spectral norm σ^2. Since $\frac{1}{n} \leq \sigma^2 \leq 1$, we have $1 \leq \beta_b \leq b$.

The worst-case situation is at a degenerate extreme, when all data points lie on a single line, and so $\sigma^2 = 1$ and $\beta_b = b$. In this case Lemma 2 degenerates to the worst-case bound of $\mathbb{E}[\|\nabla \hat{L}_A(\mathbf{w})\|^2] \leq 1$, and in Theorem 1 we have $\frac{\beta_b}{b} = 1$, indicating that using larger mini-batches does not help at all, and the same number of parallel iterations is required.

However, when $\sigma^2 < 1$, and so $\beta_b < 1$, we see a benefit in using mini-batches in Theorem 1, corresponding to a parallelization speedup of $\frac{b}{\beta_b}$. The best situation is when $\sigma^2 = \frac{1}{n}$, and so $\beta_b = 1$, which happens when all training points are orthogonal. In this case there is never any interaction between points in the mini-batch, and using a mini-batch of size b is just as effective as making b single-example steps. When $\beta_b = 1$ we indeed see that the speedup speedup is equal to the number of mini-batches, and that the behavior in terms of the number of data accesses (equivalently, serial runtime) bT, does not depend on b; that is, even with larger mini-batches, we require no more data accesses, and we gain linearly from being able to perform the accesses in parallel. The case $\sigma^2 = \frac{1}{n}$ is rather extreme, but even for intermediate values $\frac{1}{n} < \sigma^2 < 1$ we get speedup. In particular, as long as $b \leq \frac{1}{\sigma^2}$, we have $\beta_b \leq 2$, and an essentially linear speedup. Roughly speaking, $\frac{1}{\sigma^2}$ captures the number of examples in the mini-batch beyond which we start getting significant interactions between points.

[2] Except that we avoid the logarithmic factor by relying on tail averaging and a more modern SGD analysis.

4. Mini-Batches in Dual Stochastic Coordinate Ascent Methods

An alternative stochastic method to Pegasos is Stochastic Dual Coordinate Ascent (SDCA, Hsieh et al. 2008), aimed to solve the dual problem (2). At each iteration we choose a single training example $(\mathbf{x}_i, y_i)$, uniformly at random, corresponding to a single dual variable (coordinate) $\boldsymbol{\alpha}_i = e_i^\top \boldsymbol{\alpha}$. Subsequently, $\boldsymbol{\alpha}_i$ is updated so as to maximize the (dual) objective, keeping all other coordinates of $\boldsymbol{\alpha}$ unchanged and maintaining the box constraints. At iteration t, the update $\boldsymbol{\delta}_i^{(t)}$ to $\boldsymbol{\alpha}_i^{(t)}$ is computed via

$$\begin{aligned}
\boldsymbol{\delta}_i^{(t)} &:= \operatorname*{arg\,max}_{0\leq \boldsymbol{\alpha}_i^{(t)}+\delta\leq 1} \mathbf{D}(\boldsymbol{\alpha}^{(t)} + \delta e_i) \\
&\overset{(2)}{=} \operatorname*{arg\,max}_{0\leq \boldsymbol{\alpha}_i^{(t)}+\delta\leq 1} (\lambda n - (\mathbf{Q}e_i)^\top \boldsymbol{\alpha}^{(t)})\delta - \tfrac{\mathbf{Q}_{i,i}}{2}\delta^2 \\
&= \mathrm{clip}_{[-\boldsymbol{\alpha}_i^{(t)}, 1-\boldsymbol{\alpha}_i^{(t)}]} \tfrac{\lambda n - (\mathbf{Q}e_i)^\top \boldsymbol{\alpha}^{(t)}}{\mathbf{Q}_{i,i}} \\
&\overset{(3),(4)}{=} \mathrm{clip}_{[-\boldsymbol{\alpha}_i^{(t)}, 1-\boldsymbol{\alpha}_i^{(t)}]} \tfrac{\lambda n(1 - y_i\langle \mathbf{w}(\boldsymbol{\alpha}^{(t)}), \mathbf{x}_i\rangle)}{\|\mathbf{x}_i\|^2},
\end{aligned} \tag{14}$$

where clip_I is projection onto the interval I. Variables $\boldsymbol{\alpha}_j^{(t)}$ for $j \neq i$ are unchanged. Hence, a single iteration has the form $\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)} + \boldsymbol{\delta}_i^{(t)} e_i$. Similar to a Pegasos update, at each iteration a single, random, training point is considered, the "response" $y_i \langle \mathbf{w}(\boldsymbol{\alpha}^{(t)}), \mathbf{x}_i\rangle$ is calculated (this operation dominates the computational effort), and based on the response, a multiple of $\mathbf{x}_i$ is added to the weight vector $\mathbf{w}$ (corresponding to changing $\boldsymbol{\alpha}_i$). The two methods thus involve fairly similar operations at each iteration, with essentially identical computational costs. They differ

in that in Pegasos, $\boldsymbol{\alpha}_i$ is changed according to some pre-determined step-size, while SDCA changes it optimally so as to maximize the dual objective (and maintain dual feasibility); there is no step-size parameter.

SDCA was suggested and studied empirically by Hsieh et al. (2008), where empirical advantages over Pegasos were often observed. In terms of a theoretical analysis, by considering the dual problem (2) as an ℓ_1-regularized, box-constrained quadratic problem, it is possible to obtain guarantees on the *dual* suboptimality, $\mathbf{D}(\boldsymbol{\alpha}^*) - \mathbf{D}(\boldsymbol{\alpha}^{(t)})$, after a finite number of SDCA iterations (Shalev-Shwartz & Tewari, 2011; Nesterov, 2012; Richtárik & Takáč, 2013). However, such guarantees do *not* directly imply guarantees on the *primal* suboptimality of $\mathbf{w}(\boldsymbol{\alpha}^{(t)})$. Recently, Shalev-Shwartz & Zhang (2012) bridged this gap, and provided guarantees on $\mathbf{P}(\mathbf{w}(\boldsymbol{\alpha}^{(t)})) - \mathbf{P}(\mathbf{w}^*)$ after a finite number of SDCA iterations. These guarantees serve as the starting point for our theoretical study.

4.1. Naive Mini-Batching

A naive approach to parallelizing SDCA using mini-batches is to compute $\boldsymbol{\delta}_i^{(t)}$ in parallel, according to (14), for all $i \in A_t$, all based on the current iterate $\boldsymbol{\alpha}^{(t)}$, and then update $\boldsymbol{\alpha}_i^{(t+1)} = \boldsymbol{\alpha}_i^{(t)} + \boldsymbol{\delta}_i^{(t)}$ for $i \in A_t$, and keep $\boldsymbol{\alpha}_j^{(t+1)} = \boldsymbol{\alpha}_j^{(t)}$ for $j \notin A_t$. However, not only might this approach not reduce the number of required iterations, it might actually *increase* the number of required iterations. This is because the dual objective need *not improve monotonically* (as it does for "pure" SDCA), and even not converge.

To see this, consider an extreme situation with only two identical training examples: $\mathbf{Q} = \left[\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right]$, $\lambda = \frac{1}{n} = \frac{1}{2}$ and mini-batch size $b = 2$ (i.e., in each iteration we use both examples). If we start with $\boldsymbol{\alpha}^{(0)} = \mathbf{0}$ with $\mathbf{D}(\boldsymbol{\alpha}^{(0)}) = 0$ then $\boldsymbol{\delta}_1^{(0)} = \boldsymbol{\delta}_2^{(0)} = 1$ and following the naive approach we have $\boldsymbol{\alpha}^{(1)} = (1,1)^T$ with objective value $\mathbf{D}(\boldsymbol{\alpha}^{(1)}) = 0$. In the next iteration $\boldsymbol{\delta}_1^{(1)} = \boldsymbol{\delta}_2^{(1)} = -1$ which brings us back to $\boldsymbol{\alpha}^{(2)} = \mathbf{0}$. So the algorithm will alternate between those two solutions with objective value $\mathbf{D}(\boldsymbol{\alpha}) = 0$, while at the optimum $\mathbf{D}(\boldsymbol{\alpha}^*) = \mathbf{D}((0.5, 0.5)^\top) = 0.25$.

This is of course a simplistic toy example, but the same phenomenon will occur when a large number of training examples are identical or highly correlated. This can also be observed empirically in some of our experiments discussed later, e.g., in Figure 2.

The problem here is that since we update each $\boldsymbol{\alpha}_i$ independently to its optimal value *as if all other coordinates were fixed*, we are ignoring interactions between the updates. As we see in the extreme example above, two different $i, j \in A_t$, might suggest essentially the same change to $\mathbf{w}(\boldsymbol{\alpha}^{(t)})$, but we would then perform this update *twice*, overshooting and yielding a new iterate which is actually worse then the previous one.

4.2. Safe Mini-Batching

Properly accounting for the interactions between coordinates in the mini-batch would require jointly optimizing over all $\boldsymbol{\alpha}_i$, $i \in A_t$. This would be a very powerful update and no-doubt reduce the number of required iterations, but would require solving a box-constrained quadratic program, with a quadratic term of the form $\boldsymbol{\delta}_A^\top \mathbf{Q}_A \boldsymbol{\delta}_A$, $\boldsymbol{\delta}_A \in \mathbb{R}^b$, at each iteration. This quadratic program cannot be distributed to different machines, each handling only a single data point.

Instead, we propose a "safe" variant, where the term $\boldsymbol{\delta}_A^\top \mathbf{Q}_A \boldsymbol{\delta}_A$ is approximately bounded by the separable surrogate $\beta \left\|\boldsymbol{\delta}_A\right\|^2$, for some $\beta > 0$ which we will discuss later. That is, the update is given by:

$$\begin{aligned} \boldsymbol{\delta}_i^{(t)} &:= \operatorname*{arg\,max}_{0 \le \boldsymbol{\alpha}_i^{(t)} + \delta \le 1} (\lambda n - (\mathbf{Q} e_i)^\top \boldsymbol{\alpha}^{(t)})\delta - \tfrac{\beta}{2}\delta^2 \\ &= \operatorname{clip}_{[-\boldsymbol{\alpha}_i^{(t)}, 1-\boldsymbol{\alpha}_i^{(t)}]} \frac{\lambda n (1 - y_i \langle w(\boldsymbol{\alpha}^{(t)}), \mathbf{x}_i \rangle)}{\beta}, \quad (15) \end{aligned}$$

with $\boldsymbol{\alpha}_i^{(t+1)} = \boldsymbol{\alpha}_i^{(t)} + \boldsymbol{\delta}_i^{(t)}$ for $i \in A_t$, and $\boldsymbol{\alpha}_j^{(t+1)} = \boldsymbol{\alpha}_j^{(t)}$ for $j \notin A_t$. In essence, $\frac{1}{\beta}$ serves as a step-size, where we are now careful not to take steps so big that they will accumulate together and overshoot the objective. If handling only a single point at each iteration, such a short-step approach is not necessary, we do not need a step-size, and we can take a "full step", setting $\boldsymbol{\alpha}_i$ optimally ($\beta = 1$). But with the potential for interaction between coordinates updated in parallel, we must use a smaller step.

We will first rely on the bound (10), and establish that the choice $\beta = \beta_b$ as in (11) provides for a safe step size. To do so, we consider the dual objective at $\boldsymbol{\alpha} + \boldsymbol{\delta}$,

$$\mathbf{D}(\boldsymbol{\alpha} + \boldsymbol{\delta}) = -\frac{\boldsymbol{\alpha}^\top \mathbf{Q} \boldsymbol{\alpha} + 2\boldsymbol{\alpha}^\top \mathbf{Q} \boldsymbol{\delta} + \boldsymbol{\delta}^\top \mathbf{Q} \boldsymbol{\delta}}{2\lambda n^2} + \sum_{i=1}^n \frac{\boldsymbol{\alpha}_i + \boldsymbol{\delta}_i}{n}, \quad (16)$$

and the following separable approximation to it:

$$\mathbf{H}(\boldsymbol{\delta}, \boldsymbol{\alpha}) := -\frac{\boldsymbol{\alpha}^\top \mathbf{Q} \boldsymbol{\alpha} + 2\boldsymbol{\alpha}^\top \mathbf{Q} \boldsymbol{\delta} + \beta_b \left\|\boldsymbol{\delta}\right\|^2}{2\lambda n^2} + \sum_{i=1}^n \frac{\boldsymbol{\alpha}_i + \boldsymbol{\delta}_i}{n}, \quad (17)$$

in which $\beta_b \left\|\boldsymbol{\delta}\right\|^2$ replaces $\boldsymbol{\delta}^\top \mathbf{Q} \boldsymbol{\delta}$. Our update (15) with $\beta = \beta_b$ can be written as $\boldsymbol{\delta} = \arg \max_{\boldsymbol{\delta}: 0 \le \boldsymbol{\alpha} + \boldsymbol{\delta} \le 1} \mathbf{H}(\boldsymbol{\delta}, \boldsymbol{\alpha})$ (we then use the coordinates $\boldsymbol{\delta}_i$ for $i \in A$ and ignore the rest). We are essentially performing parallel coordinate ascent on $\mathbf{H}(\boldsymbol{\delta}, \boldsymbol{\alpha})$ instead of on $\mathbf{D}(\boldsymbol{\alpha} +$

$\boldsymbol{\delta}$). To understand this approximation, we note that $\mathbf{H}(\mathbf{0},\boldsymbol{\alpha}) = \mathbf{D}(\boldsymbol{\alpha})$, and show that $\mathbf{H}(\boldsymbol{\delta},\boldsymbol{\alpha})$ provides an expected lower bound on $\mathbf{D}(\boldsymbol{\alpha}+\boldsymbol{\delta})$:

Lemma 3. *For any $\boldsymbol{\alpha},\boldsymbol{\delta} \in \mathbb{R}^n$ and $A \in \text{Rand}(b)$,*

$$\mathbb{E}_A[\mathbf{D}(\boldsymbol{\alpha}+\boldsymbol{\delta}_{[A]})] \geq (1-\tfrac{b}{n})\mathbf{D}(\boldsymbol{\alpha}) + \tfrac{b}{n}\mathbf{H}(\boldsymbol{\delta},\boldsymbol{\alpha}).$$

Proof. Examining (16) and (17), the terms that do not depend on $\boldsymbol{\delta}$ are equal on both sides. For the linear term in $\boldsymbol{\delta}$, we have that $\mathbb{E}[\boldsymbol{\delta}_{[A]}] = \frac{b}{n}\boldsymbol{\delta}$, and again we have equality on both sides. For the quadratic term we use Lemma 1 which yields $\mathbb{E}[\boldsymbol{\delta}_{[A]}^\top \mathbf{Q}\boldsymbol{\delta}_{[A]}] \leq \frac{b}{n}\beta_b \|\boldsymbol{\delta}\|^2$, and after negation establishes the desired bound. □

Inequalities of this general type are also studied in (Richtárik & Takáč, 2012) (see Sections 3 and 4). Based on the above lemma, we can modify the analysis of Shalev-Shwartz & Zhang (2012) to obtain (see complete proof in the supplementary material):

Theorem 2. *Consider the SDCA updates given by (15), with $A_t \in \text{Rand}(b)$, starting from $\boldsymbol{\alpha}^{(0)} = \mathbf{0}$ and with $\beta = \beta_b$ (given in eq. (11)). For any $\epsilon > 0$ and*

$$t_0 \geq \max\{0, \lceil \tfrac{n}{b}\log(\tfrac{2\lambda n}{\beta_b})\rceil\}, \tag{18}$$

$$T_0 \geq t_0 + \tfrac{\beta_b}{b}\left[\tfrac{4}{\lambda\epsilon} - 2\tfrac{n}{\beta_b}\right]_+, \tag{19}$$

$$T \geq T_0 + \max\{\lceil \tfrac{n}{b}\rceil, \tfrac{\beta_b}{b}\tfrac{1}{\lambda\epsilon}\}, \tag{20}$$

$$\bar{\boldsymbol{\alpha}} := \tfrac{1}{T-T_0}\textstyle\sum_{t=T_0}^{T-1}\boldsymbol{\alpha}^{(t)}, \tag{21}$$

we have

$$\mathbb{E}[\mathbf{P}(\mathbf{w}(\bar{\boldsymbol{\alpha}}))] - \mathbf{P}(\mathbf{w}^*) \leq \mathbb{E}[\mathbf{P}(\mathbf{w}(\bar{\boldsymbol{\alpha}})) - \mathbf{D}(\bar{\boldsymbol{\alpha}})] \leq \epsilon.$$

By Theorem 2, the # of iterations of mini-batched SDCA, sufficient to reach *primal* suboptimality ϵ, is

$$\tilde{O}\left(\tfrac{n}{b} + \tfrac{\beta_b}{b}\cdot\tfrac{1}{\lambda\epsilon}\right), \tag{22}$$

We observe the *same speedup* as in the case of mini-batched Pegasos: factor of $\frac{b}{\beta_b}$, with an essentially linear speedup when $b \leq \frac{1}{\sigma^2}$. It is interesting to note that the quantity β_b only affects the second, ϵ-dependent, term in (22). The "fixed cost" term, which essentially requires a full pass over the data, is *not* affected by β_b, and is *always* scaled down by b.

4.3. Aggressive Mini-Batching

Using $\beta = \beta_\sigma$ is safe, but might be too conservative. In particular, we used the spectral norm to bound $\boldsymbol{\delta}^\top\mathbf{Q}\boldsymbol{\delta} \leq \|\mathbf{Q}\|\|\boldsymbol{\delta}\|^2$ in Lemma 3 (through Lemma 1), but this is a worst case bound over all possible vectors, and might be loose for the relevant vectors $\boldsymbol{\delta}$. Relying on a worst-case bound might mean we are taking much smaller steps then we could be. Furthermore, the approach we presented thus far relies on knowing the spectral norm of the data, or at least a bound on the spectral norm (recall (10)), in order to set the step-size. Although it is possible to estimate this quantity by sampling, this can certainly be inconvenient.

Algorithm 2 SDCA with Mini-Batches (aggressive)

Input: $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$, $\lambda > 0$, $b \in \mathbb{R}^d$, $T \geq 1$, $\gamma = 0.95$
Initialize: set $\boldsymbol{\alpha}^{(0)} = \mathbf{0}$, $\mathbf{w}^{(0)} = \mathbf{0}$, $\beta^{(0)} = \beta_b$
for $t = 0$ **to** T **do**
  Choose $A_t \in \text{Rand}(b)$
  For $i \in A_t$, compute $\tilde{\boldsymbol{\delta}}_i$ from (15) using $\beta = \beta^{(t)}$
  Sum $\zeta := \sum_{i\in A_t}\tilde{\boldsymbol{\delta}}_i^2$ and $\tilde{\Delta} := \sum_{i\in A_t}\tilde{\boldsymbol{\delta}}_i y_i \mathbf{x}_i$
  Compute $\rho = \text{clip}_{[1,\beta_b]}(\|\tilde{\Delta}\|^2/\zeta)$
  For $i \in A_t$, compute $\boldsymbol{\delta}_i$ from (15) using $\beta = \rho$.
  $\beta^{(t+1)} := (\beta^{(t)})^\gamma \rho^{1-\gamma}$
  if $\mathbf{D}(\boldsymbol{\alpha}^{(t)} + \boldsymbol{\delta}_{[A_t]}) > \mathbf{D}(\boldsymbol{\alpha}^{(t)})$ **then**
    $\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)} + \boldsymbol{\delta}_{[A_t]}$,
    $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \frac{1}{\lambda n}\sum_{i\in A_t}\boldsymbol{\delta}_i y_i \mathbf{x}_i$
  else
    $\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}^{(t)}$, $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)}$
  end if
end for

Instead, we suggest a more aggressive variant of mini-batched SDCA which gradually adapts β based on the actual values of $\|\boldsymbol{\delta}_{[A_t]}^{(t)}\|^2$ and $\boldsymbol{\delta}_{[A_t]}^{(t)}\mathbf{Q}\boldsymbol{\delta}_{[A_t]}^{(t)}$. In Section 5 one can observe the advantages of this aggressive strategy. In this variant, at each iteration we calculate the ratio $\rho = \tilde{\boldsymbol{\delta}}_{[A]}^\top\mathbf{Q}\tilde{\boldsymbol{\delta}}_{[A]}^\top/\|\tilde{\boldsymbol{\delta}}_{[A]}\|^2$, and nudge the step size towards it by updating it to a weighted geometric average of the previous step size and "optimal" step size based on the step $\boldsymbol{\delta}$ considered. One complication is that due to the box constraints, not only the magnitude but also the direction of the step $\boldsymbol{\delta}$ depends on the step-size β, leading to a circular situation. The approach we take is as follows: we maintain a "current step size" β. At each iteration, we first calculate a tentative step $\tilde{\boldsymbol{\delta}}_A$, according to (15), with the current β. We then calculate ρ according to this step direction, and update β to $\beta^\gamma\rho^{1-\gamma}$ for some pre-determined parameter $0 < \gamma < 1$ that controls how quickly the step-size adapts. But, instead of using $\tilde{\boldsymbol{\delta}}$ calculated with the previous β, we actually re-compute $\boldsymbol{\delta}_A$ using the step-size ρ. We note that this means the ratio ρ does *not* correspond to the step $\boldsymbol{\delta}_A$ actually taken, but rather to the tentative step $\tilde{\boldsymbol{\delta}}_A$. We could potentially continue iteratively updating ρ according to $\boldsymbol{\delta}_A$ and $\boldsymbol{\delta}_A$ according to ρ, but we found that this does not improve performance significantly and is not worth the extra computational effort. This aggressive strategy is

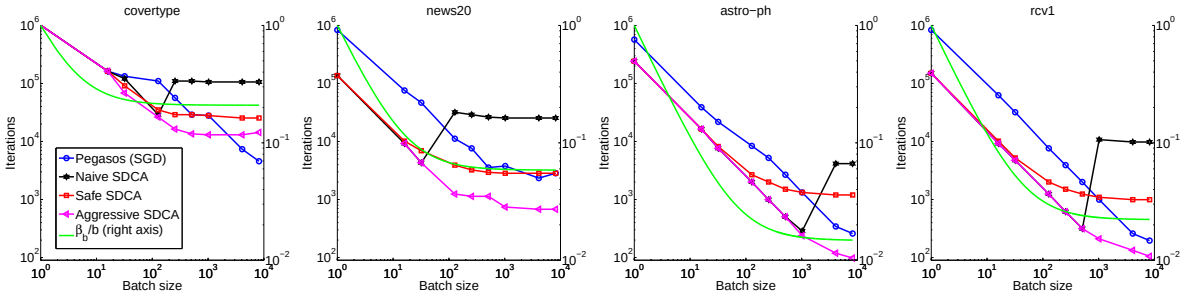

Figure 1. Number of iterations (left vertical axis) needed to find a 0.001-accurate *primal* solution for different mini-batch sizes b (horizontal axis). The leading factor in our analysis, β_b/b, is plotted on the right vertical axis.

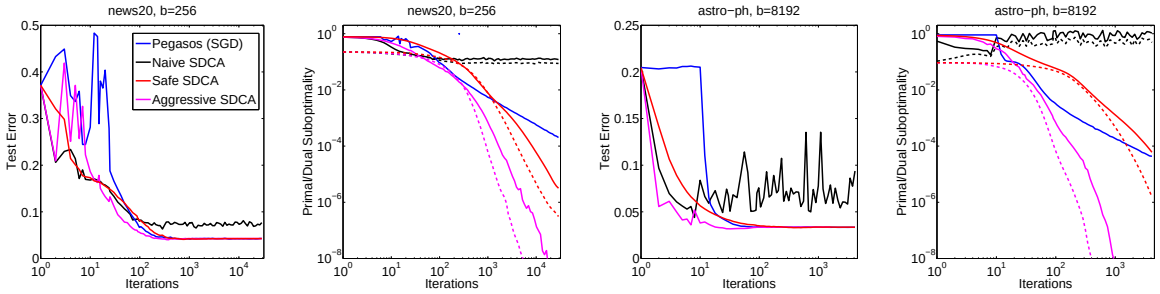

Figure 2. Evolutions of primal (solid) and dual (dashed) sub-optimality and test error for news20 and astro-ph datasets. Instead of tail averaging, in the experiments we used decaying averaging with $\bar{\mathbf{w}}^{(t)} = 0.9\bar{\mathbf{w}}^{(t-1)} + 0.1\mathbf{w}^{(t)}$.

summarized in Algorithm 2. Note that we initialize $\beta = \beta_b$, and also constrain β to remain in the range $[1, \beta_b]$, but we can use a very crude upper bound σ^2 for calculating β_b. Also, in our aggressive strategy, we refuse steps that do not actually increase the dual objective, corresponding to overly aggressive step sizes.

Carrying out the aggressive strategy requires computing $\tilde{\boldsymbol{\delta}}_{[A]}^\top \mathbf{Q}\tilde{\boldsymbol{\delta}}_{[A]}$ and the dual objective efficiently and in parallel. The main observation here is that:

$$\tilde{\boldsymbol{\delta}}_{[A]}^\top \mathbf{Q}\tilde{\boldsymbol{\delta}}_{[A]} = \|\textstyle\sum_{i\in A}\tilde{\boldsymbol{\delta}}_i y_i \mathbf{x_i}\|^2 \tag{23}$$

and so the main operation to be performed is an aggregation of $\sum_{i\in A}\tilde{\boldsymbol{\delta}}_i y_i \mathbf{x}_i$, similar to the operation required in mini-batched Pegasos. As for the dual objective, it can be written as $\mathbf{D}(\boldsymbol{\alpha}) = -\|\mathbf{w}(\boldsymbol{\alpha})\|^2 - \frac{1}{n}\|\boldsymbol{\alpha}\|_1$ and can thus be readily calculated if we maintain $\mathbf{w}(\boldsymbol{\alpha})$, its norm, and $\|\boldsymbol{\alpha}\|_1$.

5. Experiments

Figure 1 shows the required number of iterations (corresponding to the parallel runtime) required for achieving a primal suboptimality of 0.001 using Pegasos, naive SDCA, safe SDCA and aggressive SDCA, on four benchmark datasets detailed in Table 1, using different mini-batch sizes. Also shown (on an independent scale; right axis) is the leading term $\frac{\beta_b}{b}$ in our complexity results. The results confirm the advantage of SDCA over Pegasos, at least for $b = 1$, and that both Pegasos and SDCA enjoy nearly-linear speedups, at least for small batch sizes. Once the mini-batch size is such that $\frac{\beta_b}{b}$ starts flattening out (corresponding to $b \approx \frac{1}{\sigma^2}$, and so significant correlations inside each mini-batch), the safe variant of SDCA follows a similar behavior and does not allow for much parallelization speedup beyond this point, but at least does not deteriorate like the naive variant. Pegasos and the aggressive variant do continue showing speedups beyond $b \approx \frac{1}{\sigma^2}$. The experiments clearly demonstrate the aggressive modification allows SDCA to continue enjoying roughly the same empirical speedups as Pegasos, even for large mini-batch sizes, maintaining an advantage throughout. It is interesting to note that the aggressive variant continues improving even past the point of failure of the naive variant, thus establishing that it is empirically important to adjust the step-size to achieve a balance between safety and progress.

In Figure 2 we depict the evolution of solutions using the various methods for two specific data sets. Here we can again see the relative behavior of the methods, as well as clearly see the failure of the naive approach, which past some point causes the objective to deteriorate and does not converge to the optimal solution.

6. Summary and Discussion

Contribution. Our contribution in this paper is twofold: (i) we identify the spectral norm of the data,

and through it β_b, as the important quantity controlling guarantees for mini-batched/parallelized Pegasos (primal method) and SDCA (dual method). We provide the first analysis of mini-batched Pagasos, with the non-smooth hinge-loss, that shows speedups, and we analyze for the first time mini-batched SDCA with guarantees expressed in terms of the primal problem (hence, our mini-batched SDCA is a primal-dual method); (ii) based on our analysis, we present novel variants of mini-batched SDCA which are necessary for achieving speedups similar to those of Pegasos, and thus open the door to effective mini-batching using the often-empirically-better SDCA.

Related work. Our safe SDCA mini-batching approach is similar to the parallel coordinate descent methods of Bradley et al. (2011) and Richtárik & Takáč (2012), but we provide an analysis in terms of the primal SVM objective, which is the more relevant object of interest. Furthermore, Bradley et al.'s analysis does *not* use a step-size and is thus limited only to small enough mini-batches—if the spectral norm is unknown and too large a mini-batch is used, their method might not converge. Richtárik & Takáč's method does incorporate a *fixed* step-size, similar to our safe variant, but as we discuss this step-size might be too conservative for achieving the true potential of mini-batching. In the context of SVMs with mini-batches, the analysis of Duchi et al. (2012a;b) is valid in small dimensions d, implying nearly linear speedups up to a batch size that gets worse as d increases. On the other hand, our analysis is dimension independent, allows for infinite dims (as in the kernelized case) or extremely high d (e.g., when the feature vectors are very sparse).

Generality. We chose to focus on Pegasos and SDCA with regularized hinge-loss minimization, but all our results remain unchanged for any Lipschitz loss functions. Furthermore, Lemma 2 can also be used to establish identical speedups for mini-batched SGD optimization of $\min_{\|\mathbf{w}\|\leq B}\hat{L}(\mathbf{w})$, as well as for direct stochastic approximation of the population objective (generalization error) $\min L(\mathbf{w})$. In considering the population objective, the sample size is essentially infinite, we sample with replacements (from the population), σ^2 is a bound on the second moment of the data distribution, and $\beta_b = 1 + (b-1)\sigma^2$.

Table 1. Datasets and regularization parameters λ used; "%" is percent of features which are non-zero. *cov* is the forest covertype dataset of Shalev-Shwartz et al. (2011), *astro-ph* consists of abstracts of papers from physics also of Shalev-Shwartz et al. (2011), *rcv1* is from the Reuters collection and *news20* is from the 20 news groups both obtained from libsvm collection (Libsvm).

Data	# train	# test	# dim	%	$10^3\lambda$
cov	522,911	58,101	54	22	0.010
rcv1	20,242	677,399	47,236	0.16	0.100
ast.-ph	29,882	32,487	99,757	0.08	0.050
news20	15,020	4,976	1,355,191	0.04	0.125

Experiments. Our experiments confirm the empirical advantages of SDCA over Pegasos, previously observed without mini-batching. However, we also point out that in order to perform mini-batched SDCA effectively, a step-size is needed, detracting from one of the main advantages of SDCA over Pegasos. Furthermore, in the safe variant, this stepsize needs to be set according to the spectral norm (or bound on the spectral norm), with too small a setting for β (i.e., too large steps) possibly leading to non-convergence, and too large a setting for β yielding reduced speedups. In contrast, the Pegasos stepsize is *independent* of the spectral norm, and in a sense Pegasos adapts implicitly (see, e.g., its behavior compared to aggressive SDCA in the experiments). We do provide a more aggressive variant of SDCA, which does match Pegasos's speedups empirically, but this requires an explicit heuristic adaptation of the stepsize.

Parallel Implementation. In this paper we analyzed the iteration complexity, and behavior of the iterates, of mini-batched Pegasos and SDCA. Unlike "pure" (b=1) Pegasos and SDCA, which are not amenable to parallelization, using mini-batches does provide opportunities for it. Of course, actually achieving good parallelization speedups on a specific architecture in practice requires an efficient parallel, possibly distributed, implementation of the iterations. In this regard, we point out that the core computation required for both Pegasos and SDCA is that of computing $\sum_{i\in A} g_i(\langle \mathbf{w}, \mathbf{x}_i\rangle)\mathbf{x}_i$, where g is some scalar function. Parallelizing such computations efficiently in a distributed environment has been studied by e.g., Dekel et al. (2012); Hsu et al. (2011); their methods can be used here too. Alternatively, one could also consider asynchronous or delayed updates (Agarwal & Duchi, 2011; Niu et al., 2011).

Acknowledgments

The work of MT and PR was supported by EPSRC grants EP/J020567/1 (Algorithms for Data Simplicity), EP/I017127/1 (Mathematics for Vast Digital Resources) and by NAIS (funded by EPSRC grant EP/G036136/1 and the Scottish Funding Council). Collaboration on this work was also supported in part by a Google research award.

References

Agarwal, A. and Duchi, J. Distributed delayed stochastic optimization. In *NIPS*, 2011.

Bradley, J.K., Kyrola, A., Bickson, D., and Guestrin, C. Parallel coordinate descent for l1-regularized loss minimization. In *ICML*, 2011.

Cotter, A., Shamir, O., Srebro, N., and Sridharan, K. Better mini-batch algorithms via accelerated gradient methods. In *NIPS*, 2011.

Dekel, O., Gilad-Bachrach, R., Shamir, O., and Xiao, L. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13: 165–202, 2012.

Duchi, John, Bartlett, Peter L., and Wainwright, Martin J. Randomized smoothing for (parallel) stochastic optimization. In *ICML*, 2012a.

Duchi, John, Bartlett, Peter L., and Wainwright, Martin J. Randomized smoothing for stochastic optimization. *SIAM Journal on Optimization*, 22(2): 674–701, June 2012b.

Hsieh, C-J., Chang, K-W., Lin, C-J., Keerthi, S.S., and Sundarajan, S. A dual coordinate descent method for large-scale linear svm. In *ICML*, 2008.

Hsu, D., Karampatziakis, N., Langford, J., and Smola, A. Parallel online learning. *arXiv:1103.4204*, 2011.

Libsvm. *Datasets*. http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html.

Nesterov, Yu. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM J. Optimization*, 22:341–362, 2012.

Niu, F., Recht, B., Re, C., and Wright, S. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In Shawe-Taylor, J., Zemel, R.S., Bartlett, P., Pereira, F.C.N., and Weinberger, K.Q. (eds.), *NIPS 24*, pp. 693–701. 2011.

Rakhlin, A., Shamir, O., and Sridharan, K. Making gradient descent optimal for strongly convex stochastic optimization. *ArXiv:1109.5647*, 2012.

Richtárik, P. and Takáč, M. Distributed coordinate descent methods for big data optimization. Technical report.

Richtárik, P. and Takáč, M. Parallel coordinate descent methods for big data optimization. *ArXiv:1212.0873*, 2012.

Richtárik, P. and Takáč, M. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, 2013. doi: 10.1007/s10107-012-0614-z.

Shalev-Shwartz, S. and Tewari, A. Stochastic Methods for l1-regularized Loss Minimization. *JMLR*, 12: 1865–1892, 2011.

Shalev-Shwartz, S. and Zhang, T. Stochastic dual coordinate ascent methods for regularized loss minimization. *ArXiv:1209.1873*, 2012.

Shalev-Shwartz, S.S., Singer, Y., Srebro, N., and Cotter, A. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical Programming: Series A and B- Special Issue on Optimization and Machine Learning*, pp. 3–30, 2011.

Tappenden, R., Richtárik, P., and Gondzio, J. Inexact coordinate descent: complexity and preconditioning. *ArXiv:1304.5530*, 2013.

Zhang, T. Solving large scale linear prediction using stochastic gradient descent algorithms. In *ICML*, 2004.