Efficient Mini-batch Training for Stochastic Optimization

Mu Li[1,2], Tong Zhang[2,3], Yuqiang Chen[2], Alexander J. Smola[1,4]
[1]Carnegie Mellon University [2]Baidu, Inc. [3]Rutgers University [4]Google, Inc.
muli@cs.cmu.edu, tzhang@stat.rutgers.edu, chenyuqiang@baidu.com, alex@smola.com

ABSTRACT

Stochastic gradient descent (SGD) is a popular technique for large-scale optimization problems in machine learning. In order to parallelize SGD, minibatch training needs to be employed to reduce the communication cost. However, an increase in minibatch size typically decreases the rate of convergence. This paper introduces a technique based on approximate optimization of a conservatively regularized objective function within each minibatch. We prove that the convergence rate does not decrease with increasing minibatch size. Experiments demonstrate that with suitable implementations of approximate optimization, the resulting algorithm can outperform standard SGD in many scenarios.

1. INTRODUCTION

The recent years have witnessed a rapid growth of data in variety and volume. The sheer amount of data has led to increasing interest in scalable optimization. Stochastic gradient descent (SGD) is one of the most popular methods. It has been successfully applied to large scale natural language processing [11], deep learning [7], matrix factorization [10], image classification [17], and latent variable models [22].

Traditional SGD processes one example per iteration. This sequential nature makes SGD challenging for distributed inference. A common practical solution is to employ mini-batch training, which aggregates multiple examples at each iteration. However, the synchronization cost of mini-batch training is potentially still too large for large scale applications. For instance, in a distributed implementation, machines may need to communicate with each other for every mini-batch in order to synchronize the shared variables, such as gradients or parameters [8]. Given that both bandwidth and latency of networks are often 100x worse than physical memory, this overhead cannot be ignored.

Although large mini-batches are preferable to reduce the communication cost, they may slow down convergence rate in practice [4]. That is, if SGD converges by T iterations, the mini-batch training with batch size b may need more than T/b iterations. The increase in computation diminishes the benefits of the reduced communication cost due to large b. In addition, the I/O costs increases if the data is too large to fit into memory so that one need to fetch the minibatch from disk or network [25].

This paper considers the problem that we want to use *large mini-batches* to reduce *communication cost* but at the same time retain *good convergence properties*. It is known that for general convex objective functions, the convergence of SGD is $\mathcal{O}(1/\sqrt{T})$; for mini-batch SGD with minibatch size b, the convergence is $\mathcal{O}(1/\sqrt{bT} + 1/T)$ [8]. Since the total number of examples examined is bT while there is only a $\sqrt{b}$ times improvement, the convergence speed degrades with increasing minibatch size.

To address this issue we propose an alternative mini-batch update strategy that does not slow down in convergence as the mini-batch size increases. The key observation is that, when a mini-batch is large, it is desirable to solve a more complex optimization problem, rather than simply update the solution by the gradients. Specifically, in each iteration, we *solve* a conservative risk minimization subproblem. It consists of two components: the original objective function on the mini-batch and a conservative penalty. Accordingly we are able to gain more from a mini-batch before moving to the next. The conservative penalty reduces variance and prevents divergence from the previous consensus. For our goal we need two ingredients: a more sophisticated update strategy and secondly, an efficient means of solving the conservative subproblem such that the increase of computation does not overwhelm the reduced synchronization cost.

Many previous works aimed at improving mini-batch SGD optimization. [11] proposed to use asynchronous communication. [23] studied the accelerated version. At a more fundamental level, [20, 27] consider the problem of solving subproblems in parallel, followed by averaging. They can be viewed as the extreme case where the mini-batch size is the entire partition. These strategies, however, are wasteful since no communication occurs during the compute phase.

Our approach differs from previous work by the addition of a conservative penalty and the use of each data partition in a nontrivial manner beyond simple gradient computation:

- We propose a new and general way of performing mini-batch updates beyond simple parameter averaging.
- We show that the proposed algorithm has an optimal $\mathcal{O}(1/\sqrt{bT})$ convergence rate, which improves [8] when the batch size b is large. Furthermore, we show it can

KDD'14, August 24–27, 2014, New York, NY, USA.

http://dx.doi.org/10.1145/2623330.2623612 .

be improved to $\mathcal{O}(\log T/(\lambda bT) + \lambda/(\sqrt{b}T))$ for a λ-strongly convex objective function.

- We propose two strategies to solve the conservative subproblem and demonstrate how to extend them in a communication efficient distributed implementation.
- We demonstrate the efficacy of the algorithm on a large scale dataset.

2. ALGORITHM

For concreteness of our exposition we need to introduce the inference problem formally. Our goal is to solve

$$w_* = \operatorname*{argmin}_{w\in\Omega} \phi(w) \text{ where } \phi(w) = \frac{1}{n}\sum_{i=1}^{n}\phi_i(w). \quad (1)$$

Here $\phi_i : \Omega \to \mathbb{R}$ is a convex loss function and w is a shared parameter. This general form addresses a large group of machine learning problems. We give two examples:

Risk Minimization [12]: Here the objective is to minimize a loss function $\ell(x, y, w)$, such as the regression or classification error that depends on data x and label y. Moreover, one commonly adds a regularizer $c(w)$. Assume there are n example pairs $(x_1, y_1), \ldots, (x_n, y_n)$, then we obtain the objective function by setting

$$\phi_i(w) = \ell(x_i, y_i, w) + \lambda c(w) \quad (2)$$

where λ is the regularization coefficient.

Graphical Model Inference [15, 14]: In undirected graphical models (and factor graphs) the relationship between random variables can be encoded by clique potentials $\psi_C(w)$, as given by the Hammersley-Clifford theorem [1]. Assume the clique set $\mathcal{C} = \{C_1, \ldots, C_n\}$, then pseudolikelihood can be decomposed into terms

$$\phi_i(w) = \log \psi_{C_i}(w). \quad (3)$$

Note that this only applies to fully observed models. For partial observations we need to interleave this with an expectation step (or any other method for addressing nonconvex inference problems).

A much larger family of problems has been characterized by the ADMM algorithms of [2]. They can all be viewed as special cases of the above setting where different $\phi_i(w)$ only act on a subset of variables.

2.1 Mini-Batch Stochastic Gradient Descent

We begin with a brief review of a naive variant of mini-batch SGD. During training it processes a group of examples per iteration. For notational simplicity, assume that n is divisible by the number of mini-batches m. Then we partition the examples into m mini-batches, each of size $b = n/m$. Note that this assumption is not required neither for the proof nor for the implementation. Likewise, the pre-partitioning step is also not necessary in practice, however, it simplifies the exposition of what follows.

Given a random minibatch $I \subset \{1, \ldots, n\}$ of size b, we can define the objective function on I as

$$\phi_I(w) = \frac{1}{|I|}\sum_{i\in I}\phi_i(w) \text{ we have } \phi(w) = \mathbf{E}_I\left[\phi_I(w)\right]. \quad (4)$$

In the simple case that $\Omega = \mathbb{R}^d$ the mini-batch SGD employs the following stochastic update rule: at each iteration t, we pick mini-batch $I_t \subset \{1, \ldots, n\}$ of size b at random and let

$$w_t = w_{t-1} - \eta_t \nabla\phi_{I_t}(w). \quad (5)$$

Whenever Ω has a nontrivial shape we would need to add a projection step, which finds the nearest neighbor of w_t in the feasible set Ω [26].

For convex ϕ_i, this method converges to the minimum objective value at a rate of $\mathcal{O}(1/\sqrt{bT} + 1/T)$, where T is the number of iterations [8]. Although b times more examples are processed in an iteration, the mini-batch training can converge much slower than that of standard SGD with the same number of processed examples. In practice, the convergence rate slows down dramatically in terms of the number of examples processed, when we use a large mini-batch size.

2.2 Efficient Mini-Batch Training

The above empirical finding was a key motivation for our approach. To gain some intuition note that for general domains Ω the update (5) can be rewritten as an optimization problem on a mini-batch:

$$w_t = \operatorname*{argmin}_{w\in\Omega}\Big[\phi_{I_t}(w_{t-1}) + \langle\nabla\phi_{I_t}(w_{t-1}), w - w_{t-1}\rangle + \frac{1}{2\eta_t}\|w - w_{t-1}\|_2^2\Big]$$

Note that this can be regarded as an approximation of $\phi_{I_t}(w)$, the loss on the minibatch plus a conservative penalty relative to w_{t-1}. While the above optimization problem is easy to solve, the first order Taylor approximation of $\phi_{I_t}(w)$ might be too coarse to achieve sufficient progress towards the optimal solution. Such an aggressive trade-off of fast convergence in favor of computational efficiency is highly undesirable for mini-batches of large sizes: often there is high overhead of switching to the next mini-batch, e.g. due to process synchronization, data reads from disk and network communication.

Note that SGD often uses a small step size due to the variance of the randomly chosen mini-batch. However, when the size of a mini-batch increases, its variance decreases. More sophisticated methods may be used towards faster convergence rate. In this paper, we propose to update the parameter by solving the following subproblem at iteration t:

$$w_t = \operatorname*{argmin}_{w\in\Omega}\left[\phi_{I_t}(w) + \frac{\gamma_t}{2}\|w - w_{t-1}\|_2^2\right]. \quad (6)$$

It consists of two components: the first part minimizes the objective function on mini-batch I_t, aiming to achieve full utilization of this mini-batch. The second component is a conservative constraint which limits dramatic changes of the parameter to avoid overutilization.

Algorithm 4 shows the proposed algorithm. Compared to the SGD rule, we need to solve the more complex conservative subproblem for each mini-batch. For the sake of simplicity in the theoretical analysis, we assume that the optimization is performed exactly; in practice, an approximate solution will be sufficient, particularly in the early stages of inference. If the computational cost for this approximate optimization is not too expensive compared to SGD, then this method has similar overall complexity per step relative to SGD, while at the same time drastically reducing the amount of network communication required between steps.

Algorithm 1 Single node template

Input: Initial w_0, conservative coefficients $\gamma_1, \ldots, \gamma_T$

1: **for** $t = 1, \ldots, T$ **do**
2: randomly choose mini-batch $I_t \subset \{1, \ldots, n\}$ of size b
3: solve the conservative subproblem:

$$w_t = \operatorname*{argmin}_{w \in \Omega} \left[\phi_{I_t}(w) + \frac{\gamma_t}{2} \|w - w_{t-1}\|_2^2 \right].$$

4: **end for**

2.3 Theoretical Analysis

The advantage of solving the conservative subproblem (6) is that the convergence does not slow down dramatically when the mini-batch size increases. This is reflected in our main result. Before stating the theorem, we need to introduce the notion of a Bregman divergence for convex functions f as follows:

$$D_f(w; w') := f(w) - f(w') - \nabla f(w')^\top (w - w') \quad (7)$$

This is the difference between $f(w)$ and the value of the first-order Taylor expansion of f at w', when evaluated at w. The properties of Bregman divergence include:

Non-negativity: $D_f(w; w') \geq 0$

Convexity: $D_f(w; w')$ is convex with respect to w

Linearity: $D_f(w; w')$ is linear with respect to f, namely

$$D_{f+cf'}(w; w') = D_f(w; w') + cD_{f'}(w; w').$$

We need the following assumption for our theorem:

ASSUMPTION 1. *We assume that for all t:*

$$\mathbf{E}_{I_t} \left[D_\phi(w_t; w_{t-1}) \right] \leq \mathbf{E}_{I_t} \left[D_{\phi_{I_t}}(w_t; w_{t-1}) + \frac{\gamma_t}{2} \|w_t - w_{t-1}\|_2^2 \right]$$

Note that $D_\phi(w; w_{t-1}) = \mathbf{E}_{I_t} \left[D_{\phi_{I_t}}(w; w_{t-1}) \right]$ holds for general w that does not depend on I_t. However, since w_t depends on I_t we require some $\gamma_t > 0$ to satisfy the condition. Essentially, Assumption 1 bounds the amount of 'surprise' we can expect when replacing the full Bregman Divergence by one on the subset plus a conservative penalty.

Note that the assumption holds as long as we pick γ_t greater than or equal to the smoothness parameter of ϕ:

$$\phi(w) - \phi(w') - \nabla\phi(w')^\top (w - w') \leq \frac{\gamma_t}{2} \|w - w'\|_2^2.$$

In other words, the counterpart of strong convexity, namely that there exists a quadratic *upper* bound on the amount of change, suffices to guarantee this condition. In practice, however, one may be more aggressive and allow a much smaller γ_t when the mini-batch size is large. In fact, one may show that a choice of $\gamma_t = O(1/b)$ is sufficient. We have the following theorem:

THEOREM 1. *Consider the stochastic update rule (6). Assume that ϕ_i is λ-strongly convex for all i:*

$$\phi_i(w) - \phi_i(w') - \nabla\phi_i(w')^\top (w - w') \geq \frac{\lambda}{2} \|w - w'\|_2^2.$$

Under Assumption 1 and when choosing the update parameter $\gamma_t = \gamma + \lambda(t-1)$, we have for all $w_* \in \Omega$:

$$\sum_{t=1}^T \mathbf{E}[\phi(w_t) - \phi(w_*)] \leq \frac{\gamma}{2} \|w_* - w_0\|_2^2 + \frac{A^2}{b} \sum_{t=1}^T \frac{1}{\gamma_t}$$

$$\text{where } A^2 = \sup_{w \in \Omega} n^{-1} \sum_{i=1}^n \|\nabla\phi_i(w) - \nabla\phi(w)\|_2^2.$$

For convex functions, the modulus of strong convexity $\lambda = 0$ vanishes. This amounts to a constant update rate γ. In this case, choosing

$$\gamma = \sqrt{\frac{2T}{b}} \frac{A}{\|w_* - w_0\|_2}$$

minimizes the right hand side of the bound. Note that there is no a-priori guarantee that a correspondingly small γ is feasible. However, since the variance decreases with $\mathcal{O}(1/b)$ for increasing minibatch size, the scaling of $\gamma = \mathcal{O}(1/\sqrt{b})$ is appropriate. This yields the following aggregate regret bound

$$\frac{1}{T} \sum_{t=1}^T \mathbf{E}[\phi(w_t) - \phi(w_*)] \leq \frac{\sqrt{2}A}{\sqrt{Tb}} \|w_* - w_0\|_2.$$

This means that if mini-batch size is b, after T steps, we have a convergence bound of $1/\sqrt{bT}$. Therefore increasing mini-batch size does not affect convergence in terms of the number of training examples processed by the algorithm. For strongly convex $\lambda > 0$, we can achieve a regret bound of $\mathcal{O}(\log T/(\lambda bT) + \lambda/(\sqrt{b}T))$ with optimal choice of γ.

2.4 Proof of Theorem 1

For convenience, we define the regularized mini-batch loss

$$h_t(w) = \phi_{I_t}(w) + \frac{\gamma_t}{2} \|w\|_2^2.$$

Our proof relies on three lemmas. First, we upper bound $\|w_t - \bar{w}_t\|_2$, where $\bar{w}_t$ is similar to w_t except for optimizing over all examples. That is, the gradients differ via $\|\nabla\phi(\bar{w}_t) - \nabla\phi_{I_t}(\bar{w}_t)\|_2$. Next we show that the expectation of the latter, namely the variance of gradient over a mini-batch, is bounded from above by A^2/b. Finally we characterize the progress from time $t-1$ to t. The proofs of these auxiliary lemmas are presented in the Appendix.

LEMMA 1. *Let*

$$\bar{w}_t = \operatorname*{argmin}_{w \in \Omega} \left[\phi(w) + \frac{\gamma_t}{2} \|w - w_{t-1}\|_2^2 \right], \quad (8)$$

be the minimizer of the conservative version of the risk. Then the difference between the full solution $\bar{w}_t$ and the minibatch solution w_t is bounded by

$$\|w_t - \bar{w}_t\|_2 \leq \frac{1}{\gamma_t} \|\nabla\phi(\bar{w}_t) - \nabla\phi_{I_t}(\bar{w}_t)\|_2.$$

LEMMA 2. *Assume that $w \in \Omega$ and assume that we randomly choose a mini-batch I of size b independent of w. Then the expected deviation between gradients is bounded by*

$$\mathbf{E}_I \left[\|\nabla\phi_I(w) - \nabla\phi(w)\|_2^2 \right] = \frac{n-b}{n-1} \frac{B^2}{b} \leq \frac{A^2}{b}.$$

$$\text{where } B^2 = \frac{1}{n} \sum_{i=1}^n \|\nabla\phi_i(w) - \nabla\phi(w)\|_2^2.$$

LEMMA 3. *Given any $w_* \in \Omega$, the expected improvement in terms of Bregman Divergence is bounded via*

$$\begin{aligned}&\mathbf{E}\left[D_{h_t}(w_*, w_t)\right] - \mathbf{E}\left[D_{h_t}(w_*, w_{t-1})\right] \\ \leq &\phi(w_*) - \mathbf{E}\left[\phi(w_t)\right] - \mathbf{E}\left[D_\phi(w_*; w_{t-1})\right] + \frac{1}{\gamma_t}\frac{A^2}{b}. \end{aligned} \quad (9)$$

PROOF OF THEOREM 1. Under the assumption that ϕ_i is λ-strongly convex, it follows by construction that h_t is strongly convex with modulus $\gamma_t + \lambda$. Consequently the Bregman divergence is bounded by

$$D_{h_t}(w_*, w_t) \geq \frac{\gamma_t + \lambda}{2} \|w_* - w_t\|_2^2 .$$

Together with Lemma 3, we have

$$\begin{aligned}&\mathbf{E}\left[\phi(w_t) - \phi(w_*) + \frac{\gamma_{t+1}}{2}\|w_* - w_t\|_2^2\right] \\ =\ &\mathbf{E}\left[\phi(w_t) - \phi(w_*) + \frac{\gamma_t + \lambda}{2}\|w_* - w_t\|_2^2\right] \\ \leq\ &\mathbf{E}\left[\phi(w_t)\right] - \phi(w_*) + \mathbf{E}\left[D_{h_t}(w_*, w_t)\right] \\ \leq\ &\mathbf{E}\left[D_{h_t}\left(w_*, w_{t-1}\right) - D_\phi\left(w_*; w_{t-1}\right)\right] + \frac{A^2}{b\gamma_t} \\ =\ &\mathbf{E}\left[D_{\phi_{I_t}}\left(w_*, w_{t-1}\right) - D_\phi\left(w_*; w_{t-1}\right)\right] \\ &+ \frac{\gamma_t}{2}\mathbf{E}\left[\|w_* - w_{t-1}\|_2^2\right] + \frac{A^2}{b\gamma_t} \\ =\ &\frac{\gamma_t}{2}\mathbf{E}\left[\|w_* - w_{t-1}\|_2^2\right] + \frac{A^2}{b\gamma_t}\end{aligned}$$

Here the first equality follows from the definition of γ_t; the second equality follows from the definition of h_t and simple algebra; the third equality uses the fact that we are drawing I_t independently. Hence we have

$$\mathbf{E}_{I_t|w_{t-1}} D_{\phi_{I_t}}(w_*, w_{t-1}) = D_\phi(w_*; w_{t-1}).$$

Summing over $t = 1, \ldots, T$, we obtain the desired bound. □

3. PRACTICAL CONSIDERATIONS

Our analysis assumes that we solve the conservative subproblem (6) exactly; in practice, we only need to perform this optimization approximately. In this section, we propose two approximate approaches and their distributed implementation.

3.1 Approximation by Early Stopping

Optimization algorithms solving the original problem (1) can often be applied to the conservative subproblem (6): the latter consists of a part of the former with a simple quadratic term with respect to the parameter. While the most suitable optimization methods vary for different objective functions, a natural idea is to reuse the one to solve (6) but to stop it earlier. It is understood that real applications are complex; here we propose two simple but general methods that allow us to solve (6).

The first one is a direct extension of SGD. Note that, if we set $\gamma = 0$, then SGD equals to performing gradient descent with a single pass of the mini-batch with w_{t-1} as the start point. We relax the single pass constraint such that we could obtain a more accurate solution of the conservative subproblem. The algorithm, named EMSO-GD, is shown in Algorithm 2. It solves (6) by gradient descent. Standard stopping criteria can be used to achieve early stopping. For instance, we may stop when the relative objective improvement is less than a thresholds. In practice we found it most convenient to use the simplest strategy: limit the maximal iteration number L. That is, the for loop will stop if we pass the mini-batch L times. A major benefit of this strategy is to simplify the synchronization of the distributed implementation we will introduce in the next section.

Algorithm 2 EMSO-GD: solve (6) by gradient descent

Input: previous parameter w_{t-1}, mini-batch I_t
conservative coefficient γ_t, learning rate η_t
Output: new parameter w_t

1: $w_t \leftarrow w_{t-1}$
2: **for** $\ell = 0, \ldots, L$ **do**
3: update

$$w_t \leftarrow w_t - \eta_t\left(\nabla\phi_{I_t}(w_t) + \gamma_t(w_t - w_{t-1})\right) \quad (10)$$

4: **end for**

The second method is motivated by [25], where coordinate descent is applied to solve the dual form of the linear SVM in a mini-batch. Unlike [25], we directly solve the subproblem by coordinate descent in the *primal* form. Algorithm 3 shows the proposed algorithm, which is named EMSO-CD. In each time, EMSO-CD chooses a random coordinate $j \in [1, p]$, where p is the total number of coordinates, and then solves the one dimension problem

$$\operatorname*{argmin}_{w_j}\left\{\phi_{I_k}(w) + \frac{\gamma_t}{2}\|w - w_{t-1}\|_2\right\}.$$

This minimization problem may have a closed form solution. But generally it could be solved by the Newton method. Similar to EMSO-GD, we use the maximal iteration number as the early stop criteria.

Algorithm 3 EMSO-CD: solve (6) by coordinate descent

Input: previous parameter w_{t-1}, mini-batch I_t
conservative coefficient γ_t
Output: new parameter w_t

1: $w_t \leftarrow w_{t-1}$
2: **for** $\ell = 0, \ldots, Lp$ **do**
3: randomly choose coordinate $j \in [0, p]$
4: update

$$w_{t,j} \leftarrow w_{t,j} - \eta_t \frac{\nabla_j\phi_{I_t}(w_t) + \gamma_t(w_{t,j} - w_{t-1,j})}{\nabla^2_{jj}\phi_{I_t}(w_t) + \gamma_t} \quad (11)$$

5: **end for**

3.2 Distributed Model Averaging

In distributed computing, we assume there are d machines, which are connected by network. Then the conservative subproblem could be solved by all these machine together. Specifically, we first divide a mini-batch into d partitions, next assign one partition to each of the machines, and then obtain the solution via communication. One possible approach is that all machines communicate in each iteration when solving the subproblem. However, this may introduce a large amount of communication.

Instead, we propose to use a more communication friendly approach where each machine solves the conservative subproblem independently, and then all machines average their

name	KDD04	URL	CTR
# examples	146 K	2.4 M	142 M
# features	74	3.2 M	28 M
# entries	11 M	277 M	8.4 G
# features per example	73±0.8	116±17	59±26
label ratio $+1:-1$	1:111	1:2	1:15

Table 1: A collection of real datasets.

name	update	iteration	distributed
L-BFGS	[18]	batch	yes
LIBLINEAR	[9]	batch	no
Mini-Batch SGD	(5)	mini-batch	yes
EMSO-GD	(10)	mini-batch	yes
EMSO-CD	(11)	mini-batch	yes

Table 2: Evaluated Algorithms.

results. The algorithm is shown in Algorithm 4. Note that there is no restriction in terms of the choice of methods for solving the subproblems locally. In particular, the algorithms introduced in the previous section apply.

Algorithm 4 EMSO: Efficient Mini-Batch for Stochastic Optimization

Input: initialization w_0, conservative coefficients $\{\gamma_t\}_{t=1}^T$ learning rate $\{\eta_t\}_{t=1}^T$, number of machines k

1: partition examples into m mini-batches $I_1, \ldots, I_m$
2: **for** $t = 1, \ldots, T$ **do**
3: randomly choose mini-batch I_t
4: partition I_t into $I_{t,1}, \ldots, I_{t,d}$
5: **for** $i = 1, \ldots, d$ **do** {in parallel}
6: machine i get partition $I_{t,i}$
7: solve the conservative subproblem on $I_{t,i}$ by Algorithm 2 or 3 to obtain $w_t^{(i)}$
8: **end for**
9: average $w_t = \frac{1}{d}\sum_{i=1}^d w_t^{(i)}$ via communication
10: **end for**

4. EXPERIMENTS

4.1 Dataset

To evaluate the proposed algorithms, we used three binary classification datasets of varying scales, which are listed on Table 1. KDD04 comes from the particle physics task at KDD Cup 2004[1], whose goal is to classify two types of particles generated in high energy collider experiments. The URL dataset aims to detect malicious URLs[2]. CTR is a private dataset containing displayed advertisements which are randomly sampled within a period of three months. The goal is to predict whether or not an advertisement will be clicked. KDD04 is a dense dataset, while the other two are extremely sparse. The largest dataset CTR has more than 100 million examples and the raw text file size is approximately 300 GB.

4.2 Setup

For the sake of simplicity we study the case of classification via logistic regression. There the objective function can be written in the form of (1) by letting

$$\phi_i(w) = \log(1 + \exp(-y_i\langle x_i, w\rangle)).$$

Here we let (y_i, x_i) to be the i-th sample pair.

We evaluated five algorithms, as summarized in Table 2. With the exception of LibLinear all algorithms were implemented in C++ using MPI for communication. The linear algebra operations are mainly performed by eigen3[3], which is a highly efficient C++ template library.

L-BFGS This is a parallelized version of the classical memory-limited BFGS method, as described in [18]. That is, the root machine obtains subgradients from each of the client machines to aggregate into a global subgradient. Subsequently the parameters are updated and we broadcast its new value. By construction the method is a batch optimization solver.

LibLinear This is the single-machine implementation as obtained from Chih-Jen Lin's site[4]. We include it to provide a reference point to existing and well-known solvers. This is a sophisticated batch solver for convex problems.

Mini-Batch SGD This effectively computes subgradients for a small minibatch on each machine. These subgradients are then aggregated to obtain a full mini-batch supgradient. After that, we take an update on the server using (5) and rebroadcast the parameters.

We used an $\mathcal{O}(1/\sqrt{t})$ decay learning rate for the mini-batch algorithms. Specifically, we set

$$\eta_t = \eta\sqrt{\frac{\alpha}{t+\alpha}}$$

for iteration t, where constants η and α specify the initial scale and decaying speed, respectively. We performed grid search to choose the best values by examining the convergence progress. We search the range of $\eta \in \{10^0, \ldots, 10^{-5}\}$ and $\alpha \in \{10^0, \ldots, 10^4\}$.

EMSO-GD This uses the parameter-averaging approach introduced in Algorithm 4 while performing parameter updates per client via (10). It differs from the previous methods by taking the higher order information of the loss function into account when processing a minibatch.

EMSO-CD The key difference to EMSO-GD is that it uses coordinate descent to update parameters within a mini-batch. Other than that, the structure is essentially identical. For both EMSO variants we set $\lambda = 0$, the number of inner iterations to be 5 and 2, respectively, and search γ from $\{10^0, \ldots, 10^5\}$.

All experiments were carried on a cluster, where each machine is equipped with four AMD Opteron Interlagos 16 core 6272 CPUs, 128GB memory and 10Gbit Ethernet.

[1] http://osmot.cs.cornell.edu/kddcup/datasets.html
[2] http://sysnet.ucsd.edu/projects/url/
[3] http://eigen.tuxfamily.org/
[4] http://www.csie.ntu.edu.tw/~cjlin/liblinear/

4.3 Minibatch Size and Convergence

A first sanity check is to ascertain that the convergence results regarding mini-batch methods on a single node hold. For this purpose we increase the batch size from 10^3 to 10^5. The objective values after processing 10^7 examples are shown on Figure 1. As expected, when the mini-batch size increases, there is an increment of objective value for mini-batch SGD due to the rather crude approximation of the dataset by a first order Taylor expansion. That is, the convergence in terms of examples processed slows down. This degeneration is worst on the dataset KDD04, which is dense and extremely unbalanced in terms of its labels. This problem is alleviated by EMSO-GD. It performs 5 iterations of gradient descent in a mini-batch and therefore potentially gains more information about the higher order structure than SGD.

However, the convergence is much more stable when solving the conservative subproblem by coordinate descent. As can be seen from Figure 1, the objective value of EMSO-CD does not increase. It even slightly decreases, with the increasing mini-batch sizes. A possible explanation is that, even though each mini-batch is passed only twice, the coordinate descent with the previous parameter as a warm start gives satisfactory solutions to the conservative subproblem. So it provides sufficient progress to compensate for the loss due to increasing the mini-batch size.

In summary, solving conservative optimization problems on a minibatch is beneficial when compared to a naive gradient computation.

4.4 Comparison to other algorithms

In a next step we compare the algorithms listed in Table 2 by objective value versus run time. We use the same setting as Section 4.3 for the mini-batch algorithms, but only report the result with the best batch size, namely 10^5 for EMSO-CD and 10^3 for the other two algorithms. Figure 2 shows the results. It can be seen that the convergence the two batch algorithms, L-BFGS and LibLinear is similar: slow at the beginning but fast at the end. This is not too unsurprising given LibLinear's heritage.

For the mini-batch algorithms, EMSO-GD is comparable to SGD: While EMSO-GD converges faster in terms of number of minibatch iterations, it consumes 5 times more computational time than SGD. Note that, even with a larger mini-batch size, EMSO-CD is 10 times faster than the other two, and furthermore, it is faster than the batch algorithm at the end.

4.5 Minibatch Size and Synchronization Cost

Recall that a major benefit of large mini-batch sizes is the potential reduction in synchronization cost. Figure 3 shows the contribution of synchronization cost to the overall runtime of the algorithm when using 12 machines. Even with such a small number of computers the proportion is considerable. As expected, this cost decreases with increasing mini-batch size. It is due to the increase in the amount of computation between synchronization passes. Furthermore, because both EMSO-GD and EMSO-CD solve a more complex optimization problem in each mini-batch than SGD, their synchronization cost is correspondingly smaller than that of SGD. In addition, although coordinate descent passes a mini-batch twice in our experiment, it requires significant more exponentiation operations than the gradient descent (a

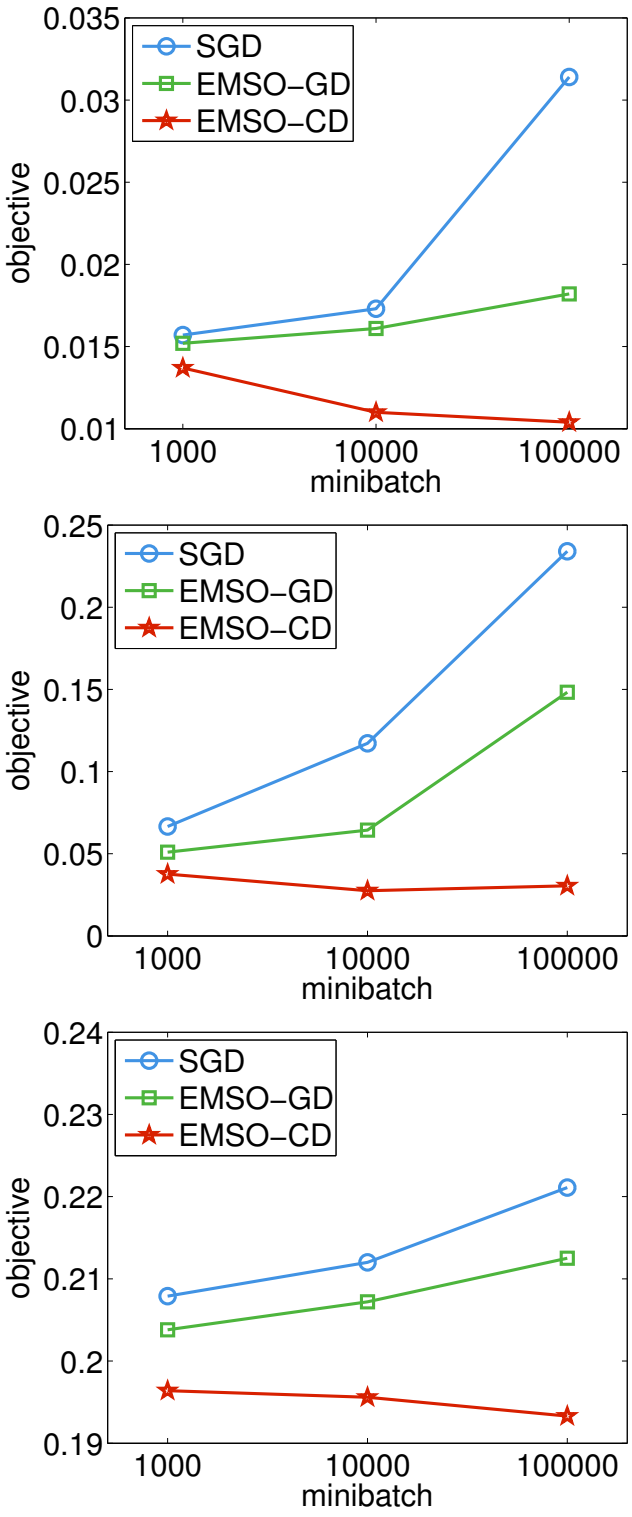

Figure 1: Objective value versus mini-batch size after in total 10^7 examples are processed in a single node. From top to bottom, datasets are KDD04, URL, and CTR, respectively, where CTR is downsampled to 4 millions examples due to the limited capacity of a single node.

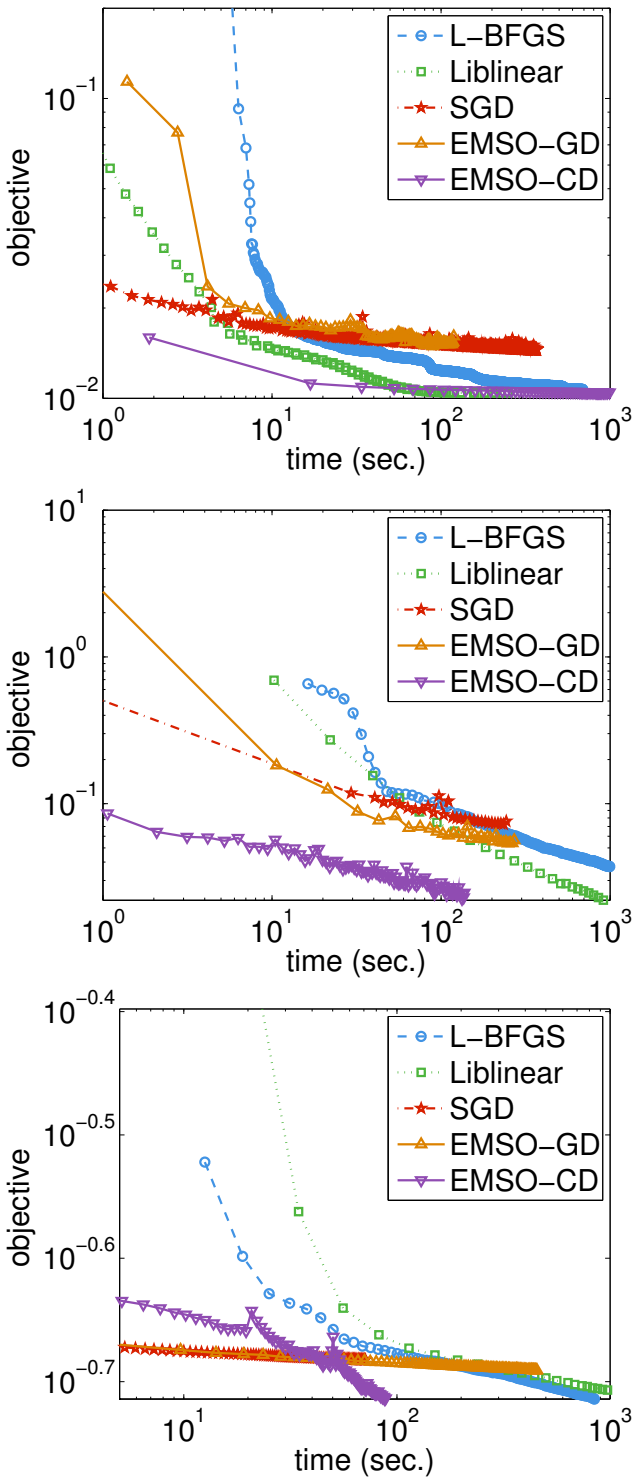

Figure 2: Objective value versus run time by a single node. Datasets are the same as Figure 1, which are KDD04, URL, and CTR from top to bottom.

fast special functions library would probably address this issue). As a result, it consumes more CPU time than gradient descent, even though the latter processes a min-batch five

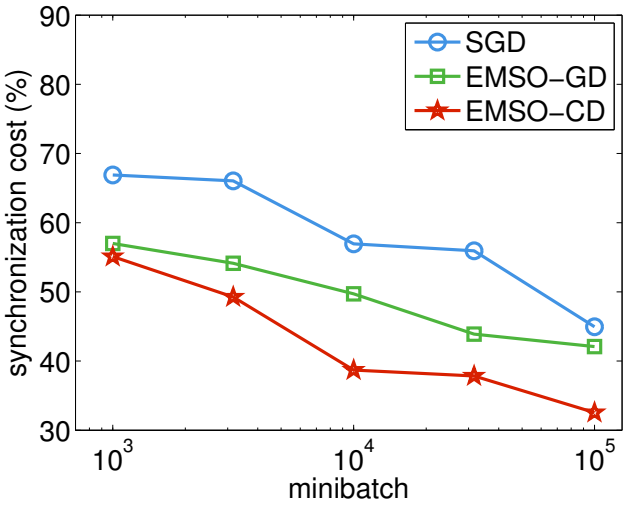

Figure 3: The fraction of synchronization cost as a function of minibatch size when communicating between 12 and 12 nodes.

#nodes	objective = 0.2 time	objective = 0.2 speedup	objective = .1972 time	objective = .1972 speedup
5	879s	1.00x	2439s	1.00x
10	499s	1.76x	1367s	1.78x
20	363s	2.42x	962s	2.54x

Table 3: Run time and speedup for EMSO-CD to reach the same objective value when running on 5, 10 and 20 nodes.

times, and therefore has a further decreased synchronization cost.

The convergence results under various mini-batch sizes are shown in Figure 4. We first fix the total number of examples processed to be 5×10^6. As can be seen from the top figure, the results are similar to the single node results of Figure 1. That is, EMSO-GD slightly improves SGD while EMSO-CD is resilient to increase the mini-batch size. The bottom of Figure 4 shows the results by fixing the run time to be 1,000 seconds. The trend then is total different. Because the portion of synchronization cost decrease, more time can be allocated to process the examples, therefore a large mini-batch size is faster. Furthermore, although SGD is comparable to EMSO-GD in the single node experiment, as shown in Figure 2, the latter outperforms the former here, due to the communication cost in distributed environment. In addition, there are clear advantages for EMSO-CD to use large batch size, as it converges faster when the mini-batch size increases.

4.6 Scalability

We conclude our experimental evaluation by assessing runtime results for varying numbers of nodes. We primarily compare the following two algorithms: EMSO-CD and L-BFGS. Figure 5 shows the convergence results. As can be seen, both algorithms benefits from an increase in the number of nodes. But L-BFGS gains more than EMSO-CD, because the former passes the whole training data in each iteration so the portion of synchronization cost is small—15% comparing 30% of EMSO-CD. However, EMSO-CD is

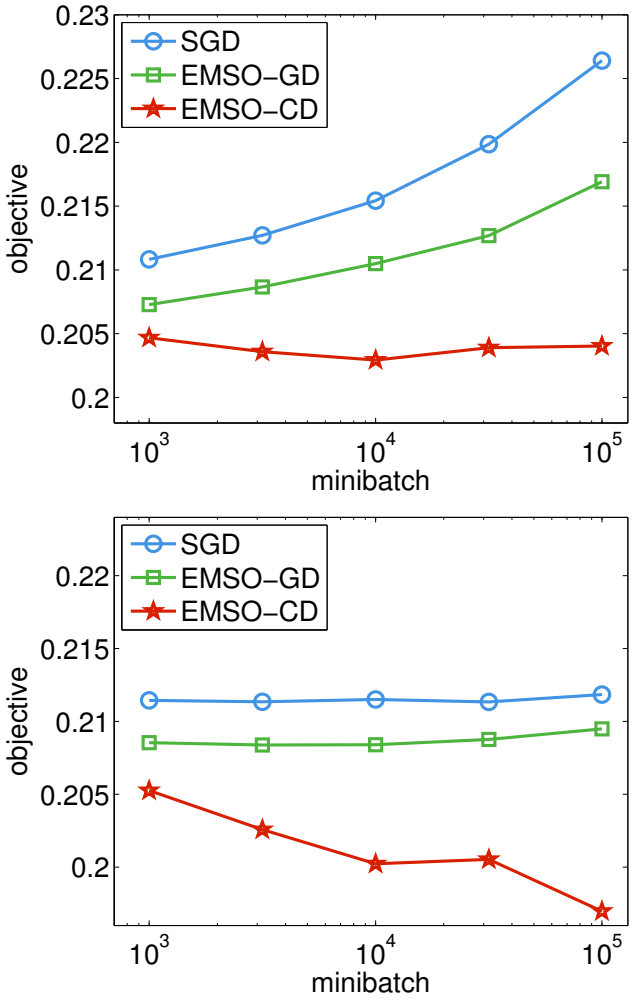

Figure 4: Objective values when varying the mini-batch size using 12 nodes. Top: for a fixed total number of examples set to 5×10^6. Bottom: for fixed runtime set to 1000 seconds.

still 10 times faster than L-BFGS, due to the faster convergence rate.

Table 3 shows the quantitative speedup for EMSO-CD to reach specific objective values. When the number of nodes doubled from 5 to 10, there is an average 1.75x speedup for both objective values, and if the nodes number is increased by 4 times, the speedup increases to 2.5x.

5. RELATED WORK AND DISCUSSION

The idea of using mini-batch in stochastic optimization has been studied by a number of researchers. For example, it was shown in [8] that distributed mini-batch gradient can achieve a convergence rate of $\mathcal{O}(1/\sqrt{Tb}+1/T)$, which is comparable to that of serial SGD when the minibatch size is small. Additional studies include [6, 24, 23].

There is also a large volume of works to improve the standard mini-batch approach. For example, [16] proposed to solve $\min_w \phi_{I_t}(w)$ directly, while [3] presented a L-BFGS style updating. In addition, [19, 13] argued to reduce the stochastic variance via gradients computed on the whole dataset.

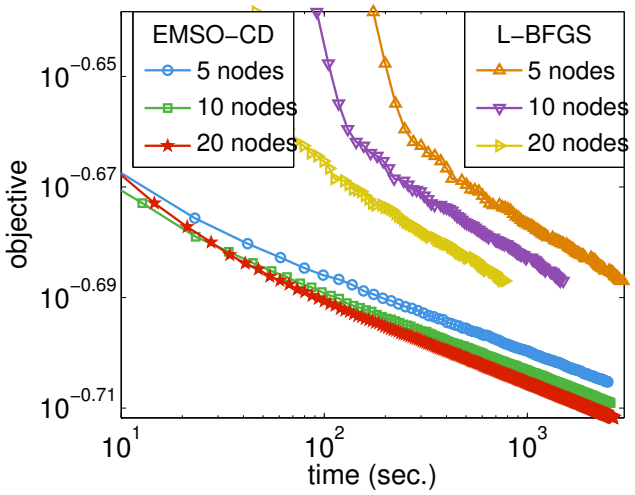

Figure 5: Objective function value versus run time for both EMSO-CD and L-BFGS using varying numbers of nodes.

Another line of research focuses on the practical performance, especially when data cannot fit into memory. For example, [25] studied solving linear SVM in the dual form by processing a block of data at each time. [21] showed that having both I/O and computational threads working together can further improve the performance. [5] studied how to select the data block.

Our work is different from previous ones in several aspects. First, we propose a novel mini-batch algorithm that solves a regularized optimization problem in primal form at each step. We show that the method can also achieve the optimal convergence rates theoretically, and presented practical implementations of the approach. The practical performance of the resulting methods outperform minibatch SGD under various scenarios.

Conclusion.

In this paper we proposed a variant of mini-batch SGD whose convergence rate does not degrade when the batch size increases. It solves a conservative subproblem in each iteration to maximize the utilization of the mini-batch while at the same time controlling the variance via a conservative constraint. We showed that it enjoys an optimal convergence rate and proposed practical distributed implementations. Furthermore, we demonstrated its efficiency on serial and distributed experiments on large scale datasets.

6. REFERENCES

[1] J. Besag. Spatial interaction and the statistical analysis of lattice systems (with discussion). *Journal of the Royal Statistical Society. Series B*, 36(2):192–236, 1974.

[2] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–123, 2010.

[3] R. Byrd, S. Hansen, J. Nocedal, and Y. Singer. A stochastic quasi-newton method for large-scale optimization. *arXiv preprint arXiv:1401.7020*, 2014.

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

[5] K.-W. Chang and D. Roth. Selective block minimization for faster convergence of limited memory large-scale linear models. In *Conference on Knowledge Discovery and Data Mining*, pages 699–707, 2011.

[6] A. Cotter, O. Shamir, N. Srebro, and K. Sridharan. Better mini-batch algorithms via accelerated gradient methods. In *NIPS*, volume 24, pages 1647–1655, 2011.

[7] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. Le, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Ng. Large scale distributed deep networks. In *Neural Information Processing Systems*, 2012.

[8] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao. Optimal distributed online prediction using mini-batches. Technical report, http://arxiv.org/abs/1012.1367, 2010.

[9] R.-E. Fan, J.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, Aug. 2008.

[10] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 69–77. ACM, 2011.

[11] K. Gimpel, D. Das, and N. A. Smith. Distributed asynchronous online learning for natural language processing. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, pages 213–222. Association for Computational Linguistics, 2010.

[12] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, New York, 2 edition, 2009.

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

[14] M. I. Jordan. *An Introduction to Probabilistic Graphical Models*. MIT Press, 2008. To Appear.

[15] F. Kschischang, B. J. Frey, and H. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.

[16] B. Kulis and P. L. Bartlett. Implicit online learning. In *Proc. Intl. Conf. Machine Learning*, 2010.

[17] Y. Lin, F. Lv, S. Zhu, M. Yang, T. Cour, K. Yu, L. Cao, and T. Huang. Large-scale image classification: fast feature extraction and svm training. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1689–1696. IEEE, 2011.

[18] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(3):503–528, 1989.

[19] D. Mahajan, S. S. Keerthi, S. Sundararajan, and L. Bottou. A parallel sgd method with strong convergence. *arXiv preprint arXiv:1311.0636*, 2013.

[20] G. Mann, R. McDonald, M. Mohri, N. Silberman, and D. Walker. Efficient large-scale distributed training of conditional maximum entropy models. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1231–1239, 2009.

[21] S. Matsushima, S. Vishwanathan, and A. Smola. Linear support vector machines via dual cached loops. In Q. Yang, D. Agarwal, and J. Pei, editors, *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD*, pages 177–185. ACM, 2012.

[22] D. Mimno, M. Hoffman, and D. Blei. Sparse stochastic inference for latent dirichlet allocation. In *International Conference on Machine Learning*, 2012.

[23] S. Shalev-Shwartz and T. Zhang. Accelerated mini-batch stochastic dual coordinate ascent. In *Advances in Neural Information Processing Systems*, pages 378–385, 2013.

[24] M. Takáč, A. Bijral, P. Richtárik, and N. Srebro. Mini-batch primal and dual methods for svms. *arXiv preprint arXiv:1303.2314*, 2013.

[25] H.-F. Yu, C.-J. Hsieh, K.-W. Chang, and C.-J. Lin. Large linear classification when data cannot fit in memory. In B. Rao, B. Krishnapuram, A. Tomkins, and Q. Yang, editors, *Knowledge Discovery and Data Mining*, pages 833–842. ACM, 2010.

[26] M. Zinkevich. Online convex programming and generalised infinitesimal gradient ascent. In *Proceedings of the International Conference on Machine Learning*, pages 928–936, 2003.

[27] M. Zinkevich, A. J. Smola, M. Weimer, and L. Li. Parallelized stochastic gradient descent. In nips23e, editor, *nips23*, pages 2595–2603, 2010.

APPENDIX

PROOF OF LEMMA 1. Since $w_t = \operatorname{argmin}_{w\in\Omega} h_t(w)$, we have from the first order KKT condition at w_t:

$$\nabla h_t(w_t)^\top (w_t - \bar{w}_t) \leq 0$$

In addition, the first order KKT condition of (8) at $\bar{w}_t$ combined with the fact that $h_t(w) = \phi_{I_t}(w) + \frac{\gamma_t}{2}\|w - w_{t-1}\|_2^2$ implies that

$$(\nabla h_t(\bar{w}_t) + \nabla\phi(\bar{w}_t) - \nabla\phi_{I_t}(\bar{w}_t))^\top (w_t - \bar{w}_t) \geq 0.$$

By substracting the first inequality from the second inequality, and rearranging terms, we obtain:

$$\begin{aligned}&\left(\nabla h_t\left(w_t\right) - \nabla h_t\left(\bar{w}_t\right)\right)^\top \left(w_t - \bar{w}_t\right)\\ &\leq \left(\nabla\phi\left(\bar{w}_t\right) - \nabla\phi_{I_t}\left(\bar{w}_t\right)\right)^\top \left(w_t - \bar{w}_t\right). \qquad (12)\end{aligned}$$

By additivity of Bregman divergences we have

$$D_{h_t}(\bar{w}_t; w_t) = D_{\phi_{I_t}}(\bar{w}_t; w_t) + \frac{\gamma_t}{2}\|\bar{w}_t - w_t\|_2^2.$$

hence $D_{h_t}\left(\bar{w}_t; w_t\right) \geq \frac{\gamma_t}{2}\|\bar{w}_t - w_t\|_2^2$.

Similarly $D_{h_t}(w_t;\bar{w}_t) \geq \frac{\gamma_t}{2}\|\bar{w}_t - w_t\|_2^2$. It follows that

$$\begin{aligned}\gamma_t\|w_t-\bar{w}_t\|_2^2 &\leq D_{h_t}(\bar{w}_t;w_t) + D_{h_t}(w_t;\bar{w}_t)\\ &= (\nabla h_t(w_t) - \nabla h_t(\bar{w}_t)^\top(w_t-\bar{w}_t)\\ &\leq (\nabla\phi(\bar{w}_t) - \nabla\phi_{I_t}(\bar{w}_t))^\top(w_t-\bar{w}_t)\\ &\leq \|\nabla\phi(\bar{w}_t) - \nabla\phi_{I_t}(\bar{w}_t)\|_2\|w_t-\bar{w}_t\|_2,\end{aligned}$$

where the second inequality is due to (12). The third inequality is Cauchy-Schwarz inequality. □

Proof of Lemma 2. This bound is essentially a conversion of variances from a minibatch I to the full set when using sampling without replacement. To simplify notation we use the abbreviation of $\psi_i := \nabla\phi_i(w) - \nabla\phi(w)$ and $\psi_I = \nabla\phi_I(w) - \nabla\phi(w)$. Note that by construction

$$\mathbf{E}_i[\psi_i] = \mathbf{E}_I[\psi_I(w)] = \psi \tag{13}$$

and therefore $B^2 = \mathbf{E}_i\left[\|\psi_i\|^2\right]$ and $B^2 \leq A^2$. The latter inequality follows since A^2 is a uniform upper bound on the variance over all $w \in \Omega$. This yields

$$\begin{aligned}\mathbf{E}_I\left[\|\psi_I\|_2^2\right] &= \mathbf{E}_I\left[\left\|\frac{1}{b}\sum_{i\in I}\psi_i\right\|_2^2\right] = \frac{1}{b^2}\mathbf{E}_I\left[\sum_{i,j\in I}{\psi_i}^\top\psi_j\right]\\ &= \frac{1}{b^2}\mathbf{E}_I\left[\sum_{i\neq j\in I}{\psi_i}^\top\psi_j\right] + \frac{B^2}{b}\\ &= \frac{b-1}{bn(n-1)}\sum_{i\neq j}{\psi_i}^\top\psi_j + \frac{B^2}{b}\\ &= \frac{b-1}{bn(n-1)}\sum_{i,j}\psi_i^\top\psi_j + \frac{B^2}{b} - \frac{B^2}{b}\frac{b-1}{n-1}\\ &= 0 + \frac{B^2}{b}\frac{n-b}{n-1} < \frac{A^2}{b}\end{aligned}$$

The last equality used the fact that ψ_i has zero-mean. □

Proof of Lemma 3. We have

$$\begin{aligned}&D_{h_t}(w_*,w_t) - D_{h_t}(w_*,w_{t-1})\\ &= D_{h_t}(w_{t-1},w_t) + (\nabla h_t(w_{t-1}) - \nabla h_t(w_t))^\top(w_*-w_t)\\ &\quad + (\nabla h_t(w_{t-1}) - \nabla h_t(w_t))^\top(w_t-w_{t-1})\\ &\leq D_{h_t}(w_{t-1},w_t) + \nabla\phi_{I_t}(w_{t-1})^\top(w_*-w_t)\\ &\quad + (\nabla h_t(w_{t-1}) - \nabla h_t(w_t))^\top(w_t-w_{t-1})\\ &= \phi(w_*) - \phi(w_t) - D_\phi(w_*;w_{t-1})\\ &\quad - (\nabla\phi(w_{t-1}) - \nabla\phi_{I_t}(w_{t-1}))^\top(w_*-w_t)\\ &\quad + (D_\phi(w_t;w_{t-1}) - D_{h_t}(w_t;w_{t-1})),\end{aligned}$$

where the equalities follow from algebraic manipulations and the definition of Bregman divergence; in the inequality, we used the first order KKT condition of (6) at w_t, implying that

$$\begin{aligned}&(\nabla\phi_{I_t}(w_{t-1}) + \nabla h_t(w_t) - \nabla h_t(w_{t-1}))^\top(w_*-w_t)\\ &= \nabla h_t(w_t)^\top(w_*-w_t) \geq 0.\end{aligned}$$

Taking expectation, we have

$$\begin{aligned}&\mathbf{E}D_{h_t}(w_*,w_t) - \mathbf{E}D_{h_t}(w_*,w_{t-1})\\ &\leq \phi(w_*) - \mathbf{E}\phi(w_t) - \mathbf{E}D_\phi(w_*;w_{t-1})\\ &\quad - \mathbf{E}(\nabla\phi(w_{t-1}) - \nabla\phi_{I_t}(w_{t-1}))^\top(w_*-w_t)\\ &\quad + \mathbf{E}(D_\phi(w_t;w_{t-1}) - D_{h_t}(w_t;w_{t-1}))\\ &\leq \phi(w_*) - \mathbf{E}\phi(w_t) - \mathbf{E}D_\phi(w_*;w_{t-1})\\ &\quad - \mathbf{E}(\nabla\phi(w_{t-1}) - \nabla\phi_{I_t}(w_{t-1}))^\top(w_*-w_t)\\ &= \phi(w_*) - \mathbf{E}\phi(w_t) - \mathbf{E}D_\phi(w_*;w_{t-1})\\ &\quad - \mathbf{E}(\nabla\phi(w_{t-1}) - \nabla\phi_{I_t}(w_{t-1}))^\top(\bar{w}_t-w_t),\end{aligned} \tag{14}$$

where the second inequality follows from $\mathbf{E}(D_\phi(w_t;w_{t-1}) - D_{h_t}(w_t;w_{t-1})) \leq 0$, which is a consequence of Assumption 1. The equality holds because

$$\begin{aligned}&\mathbf{E}(\nabla\phi(w_{t-1}) - \nabla\phi_{I_t}(w_{t-1}))^\top w_*\\ &= \mathbf{E}(\nabla\phi(w_{t-1}) - \nabla\mathbf{E}_{I_t|w_{t-1}}\phi_{I_t}(w_{t-1}))^\top w_*\\ &= 0 = \mathbf{E}(\nabla\phi(w_{t-1}) - \nabla\phi_{I_t}(w_{t-1}))^\top\bar{w}_t.\end{aligned}$$

Note further that

$$\begin{aligned}&-\mathbf{E}(\nabla\phi(w_{t-1}) - \nabla\phi_{I_t}(w_{t-1}))^\top(\bar{w}_t-w_t)\\ &\leq \sqrt{\mathbf{E}\|\nabla\phi(w_{t-1}) - \nabla\phi_{I_t}(w_{t-1}))\|_2^2\,\mathbf{E}\|\bar{w}_t-w_t\|_2^2}\\ &\leq \sqrt{\mathbf{E}\|\nabla\phi(w_{t-1}) - \nabla\phi_{I_t}(w_{t-1}))\|_2^2\,\mathbf{E}\|\nabla\phi(\bar{w}_t) - \nabla\phi_{I_t}(\bar{w}_t)\|_2^2}/\gamma_t\\ &\leq A^2/(\gamma_t b),\end{aligned}$$

where the first inequality follows from Cauchy-Schwarz inequality, the second inequality is due to Lemma 1, and the third inequality is due to Lemma 2. Plugging the above estimate into (14), we obtain the desired bound. □