Accelerated Parallel Optimization Methods for Large Scale Machine Learning

Haipeng Luo
Princeton University
haipengl@cs.princeton.edu

Patrick Haffner and Jean-François Paiement
AT&T Labs - Research
{haffner,jpaiement}@research.att.com

Abstract

The growing amount of high dimensional data in different machine learning applications requires more efficient and scalable optimization algorithms. In this work, we consider combining two techniques, parallelism and Nesterov's acceleration, to design faster algorithms for L_1-regularized loss. We first simplify BOOM [11], a variant of gradient descent, and study it in a unified framework, which allows us to not only propose a refined measurement of sparsity to improve BOOM, but also show that BOOM is provably slower than FISTA [1]. Moving on to parallel coordinate descent methods, we then propose an efficient accelerated version of Shotgun [3], improving the convergence rate from $O(1/t)$ to $O(1/t^2)$. Our algorithm enjoys a concise form and analysis compared to previous work, and also allows one to study several connected work in a unified way.

1 Introduction

Many machine learning problems boil down to optimizing specific objective functions. In this paper, we consider the following generic optimization problem associated with L_1-regularized loss:

$$\min_{\mathbf{w}\in\mathbb{R}^d} F(\mathbf{w}) = \min_{\mathbf{w}\in\mathbb{R}^d} \sum_{i=1}^{n} \ell(\mathbf{x}_i^T\mathbf{w}, y_i) + \lambda\|\mathbf{w}\|_1,$$

where $(\mathbf{x}_i, y_i)_{i=1,\ldots,n}$ represent n training examples of the task, each with feature vector $\mathbf{x}_i \in \mathbb{R}^d$ and label/response y_i, and ℓ is a smooth convex loss function with respect to its first argument. This objective function is the heart of several important machine learning problems, including Lasso [21] where $\ell(\hat{y}, y) = \frac{1}{2}\|\hat{y} - y\|^2$, and sparse logistic regression [14] where $\ell(\hat{y}, y) = \ln(1 + \exp(-y\hat{y}))$.

Much effort has been put into developing optimization methods for this model, ranging from coordinate minimization [5], randomized coordinate descent [18], stochastic gradient descent [2, 17], dual coordinate ascent [19], to higher order methods such as interior point methods [6], L-BFGS [15], to name a few. However, the need for faster and more scalable algorithms is still growing due to the emergence of applications that make use of massive amount of high dimensional data (e.g. [20]).

One direction to design faster algorithms is to utilize parallel computations on shared memory multiprocessors or on clusters. Some methods parallelize over examples [7, 10, 22], while others parallelize over features [3, 16]. As the references of the latter approach argue, it is sometimes more preferable to parallelize over features for L_1-regularized loss, which will thus be the focus of this work.

Another direction to design more efficient algorithms is to make use of the curvature of the objective function to obtain faster theoretical convergence rate. It is well known that for smooth objective functions, vanilla gradient descent only converges at a suboptimal rate $O(1/t)$, while Nesterov's acceleration technique would allow the optimal rate $O(1/t^2)$ [13]. Recent progress along this line includes generalizing Nesterov's acceleration to randomized coordinate descent [12, 8]. Even if

Input: normalized data matrix $\mathbf{X}$, smoothness parameter β, regularization parameter λ.
Set step size $\eta = 1/(c\beta)$, where

- Option 1 (FISTA): $c = \rho$, where ρ is the spectral radius of $\mathbf{X}^T\mathbf{X}$.
- Option 2 (BOOM): $c = \kappa$, where $\kappa = \max_i \kappa_i$ and $\kappa_i = |\{j : X_{ij} \neq 0\}|$.
- Option 3 (Our Method): $c = \bar{\kappa}$, where $\bar{\kappa} = \max_j \sum_{i=1}^n \kappa_i X_{ij}^2$.

Initialize $\mathbf{w}_1 = \mathbf{u}_1$.
for $t = 1, 2, \ldots$ **do** (in parallel for each coordinate)
$\mathbf{w}_{t+1} = \mathcal{P}_{\lambda\eta}\left(\mathbf{u}_t - \eta\nabla f(\mathbf{u}_t)\right)$.
$\mathbf{u}_{t+1} = (1-\gamma_t)\mathbf{w}_{t+1} + \gamma_t \mathbf{w}_t$.

Algorithm 1: Parallel Accelerated Gradient Descent

the objective is not smooth, algorithms that still enjoy the same fast convergence rate have been proposed for functions that are a sum of a smooth part and a simple separable non-smooth part, such as the L_1-regularized loss we consider here (see for instance the FISTA algorithm [1]).

In this work we aim to combine both of the techniques mentioned above, that is, to design parallelizable accelerated optimization algorithms. Similar work includes [11, 4]. We start by revisiting and improving BOOM [11], a parallelizable variant of accelerated gradient descent that tries to utilize the sparsity and elliptical geometry of the data. We first give a simplified form of BOOM which allows one to clearly see the the connection between BOOM and FISTA, and also to study these algorithms in a unified framework. Surprisingly, we show that BOOM is actually provably slower than FISTA when data is normalized, which is an equivalent way of utilizing the elliptical geometry. Moreover, we also propose a refined measurement of sparsity that improves the one used in BOOM.

Moving on to parallel coordinate descent algorithms, we then propose an accelerated version of the Shotgun algorithm [3]. Shotgun converges as fast as vanilla gradient descent while only updating a small subset of coordinates per iteration (in parallel). Our accelerated version even improves the convergence rate from $O(1/t)$ to $O(1/t^2)$, that is, the same convergence rate as FISTA while updating much fewer coordinates per iteration. Our algorithm is a unified framework of accelerated single coordinate descent [12, 8], multiple coordinate descent and full gradient descent. However, instead of directly generalizing [12, 8] or the very recent work [4], we take a different route to present Nesterov's acceleration technique so that our method enjoys a simpler form that makes use of only one auxiliary sequence and our analysis is also much more concise. We discuss how these algorithms are connected and which one is optimal under different circumstances. We finally mention several computational tricks to allow highly efficient implementation of our algorithm.

2 Gradient Descent: Improving BOOM

In this section, we investigate algorithms that make use of a full gradient at each round. Specifically, we revisit, simplify and improve the BOOM algorithm [11].

There are essentially two forms of Nesterov's acceleration technique, one which follows the original presentation of Nesterov and uses two auxiliary sequences of points, and the other which follows the presentation of FISTA and uses only one auxiliary sequence. BOOM falls into the first category. To make the algorithm more clear and the connection to other algorithms more explicit, we will first translate BOOM into the second form. Before doing so, to make things even more concise we assume that the data is *normalized*. Specifically, let $\mathbf{X}$ be an n by d matrix such that the i^{th} row is $\mathbf{x}_i^T$. We assume each column of $\mathbf{X}$ is normalized such that $\sum_{i=1}^n X_{ij}^2 = 1$ for all $j \in \{1, \ldots, d\}$. It is clear that this is without loss of generality (see further discussion at the end of this section).

Now we are ready to present our simplified version of BOOM (see Algorithm 1, Option 2). Here, β is the smoothness parameter of the loss function such that its second derivative $\ell''(\hat{y}, y)$ (with respect to the first argument) is upper bounded by β for all $\hat{y}$ and y. For instance, $\beta = 1$ for square loss used in Lasso and $\beta = 1/4$ for logistic loss. γ_t is the usual coefficient for Nesterov's technique which satisfies: $\gamma_t = (1-\theta_t)/\theta_{t+1}$, with $\theta_0 = 0, \theta_{t+1} = \frac{1}{2}(1 + \sqrt{1 + 4\theta_t^2})$. Finally, the shrinkage function $\mathcal{P}$ is defined as $\mathcal{P}_a(\mathbf{w})_j = \text{sign}(w_j)\max\{|w_j| - a, 0\}$. One can see that BOOM explicitly makes use of the sparsity of the data, and uses κ, a measurement of sparsity, to scale the gradient.

When written in this form, it is clear that BOOM is very close to FISTA (see Algorithm 1, Option 1). The only difference is that BOOM replaces ρ with the sparsity κ. The questions are, whether this slight modification results in a faster algorithm? Does BOOM converges faster by utilizing the sparsity of the data? The following lemma and theorem answer these questions *in the negative*.

Lemma 1. *Let $\rho, \bar{\kappa}$ and κ be defined as in Algorithm 1. Then we have $\rho \leq \bar{\kappa} \leq \kappa$.*

Proof. Let $\mathbf{z}$ be a unit eigenvector of $\mathbf{X}^T\mathbf{X}$ with respect to the eigenvalue ρ. One has $\rho = \mathbf{z}^T(\rho\mathbf{z}) = \mathbf{z}^T(\mathbf{X}^T\mathbf{X}\mathbf{z}) = \|\mathbf{X}\mathbf{z}\|_2^2 = \sum_{i=1}^n \left(\sum_{j:X_{ij}\neq 0} X_{ij}z_j\right)^2$. By Cauchy-Schwarz inequality, we continue

$$\rho \leq \sum_{i=1}^n \left(\sum_{j:X_{ij}\neq 0} 1^2 \sum_{j:X_{ij}\neq 0} X_{ij}^2 z_j^2\right) = \sum_{i=1}^n \left(\kappa_i \sum_{j=1}^d X_{ij}^2 z_j^2\right) = \sum_{j=1}^d \left(z_j^2 \sum_{i=1}^n \kappa_i X_{ij}^2\right) \leq \bar{\kappa} \leq \kappa,$$

where the last two equalities make use of the fact that $\sum_j z_j^2 = 1$ and $\sum_i X_{ij}^2 = 1$ respectively. □

Theorem 1. *Suppose $F(\mathbf{w})$ admits a minimizer $\mathbf{w}^*$. As long as $c \geq \rho$, Algorithm 1 insures $F(\mathbf{w}_t) - F(\mathbf{w}^*) \leq 2c\beta\|\mathbf{w}_1 - \mathbf{w}^*\|_2/t^2$ after t iterations. In other words, to reach ϵ accuracy, Algorithm 1 needs $T_\epsilon = O(\sqrt{c\beta/\epsilon})$ iterations.*

We omit the proof of Theorem 1 since it is a direct generalization of the original proof for FISTA. Together with Lemma 1, this theorem suggests that we should always choose FISTA over BOOM if we know ρ, since it uses a larger step size and converges faster. In practice, computing ρ might be time consuming. However, a useful side product of Lemma 1 is that it proposes a refined measurement of sparsity $\bar{\kappa}$, which is clearly also easy to compute at the same time. We include this improved variant in Option 3 of Algorithm 1. Note that for any fixed j, $X_{1j}^2, \ldots, X_{nj}^2$ forms a distribution, and thus $\bar{\kappa}$ should be interpreted as the maximum weighted average sparsity of the data.

Generality of Feature Normalizing. One might doubt that our results hold merely because of the normalization assumption of the data. Indeed, authors of [11] emphasize that one of the advantages of BOOM is that it utilizes the elliptical geometry of the feature space, which does not exist any more if each feature is normalized. However, one can easily verify that the outputs of the following two methods are completely identical: 1) apply BOOM directly on the original data; 2) scale each feature first so that the data is normalized, apply BOOM, and at the end scale back each coordinate of the output accordingly. Therefore, feature normalization does not affect the behavior of BOOM at all. Put it differently, feature normalization is an equivalent way to utilize the elliptical geometry of the data. Note that the same argument does not hold for FISTA. Indeed, while our results show that FISTA is provably faster than BOOM, experiments in [11] show the opposite on unnormalized data. This suggests that we should always normalize the data before applying FISTA.

3 Coordinate Descent: Accelerated Shotgun

Updating all coordinates in parallel is not realistic, either because of a limited number of cores in multi-processors or communication bottlenecks in clusters. Therefore in this section, we shift gears and consider algorithms that only update a subset of the coordinates at each iterations. To simplify presentation, we assume there is no regularization (i.e. $\lambda = 0$) and again data is normalized. We propose a generalized and accelerated version of the Shotgun algorithm[1] [3] (see Algorithm 2).

What Shotgun does is to perform coordinate descent on several randomly selected coordinates (in parallel), with the *same step size* as in the usual (single) coordinate descent. If the number of updates per iteration P is well tuned, Shotgun converges as fast as doing a full gradient descent, even if it updates much fewer coordinates. The main difference between shotgun and our accelerated version is that as usual accelerated methods, our algorithm maintains an auxiliary sequence $\mathbf{u}_t$ at which we compute gradients. However, $\mathbf{u}_{t+1}$ is not just $(1-\gamma_t)\mathbf{w}_{t+1} + \gamma_t\mathbf{w}_t$ as in Algorithm 1. Instead, a small *step back* has to be taken, which is reflected in the extra term $c_t(\mathbf{u}_t - \mathbf{w}_{t+1})$ where constant c_t will be specified later in Theorem 2. Intuitively, this step back is to reduce the momentum due to the fact that we are not updating all coordinates. Another important generalization of Shotgun is that we

[1] Note that Shotgun has already been successfully accelerated with Coordinate Descent Newton method [3], but no analysis of the convergence rate is provided for this heuristic method.

Input: number of parallel updates P, step size coefficient η, smoothness parameter β
Initialize $\mathbf{w}_1 = \mathbf{u}_1$.
for $t = 1, 2, \ldots,$ **do**
  pick a random subset S_t of $\{1, \ldots, d\}$ such that $|S_t| = P$.
  for $j = 1$ **to** d **do**

$$w_{t+1,j} = \begin{cases} u_{t,j} - \frac{\eta}{\beta}\nabla f(\mathbf{u}_t)_j & \text{if } j \in S_t; \\ u_{t,j} & \text{else.} \end{cases}$$

  $\mathbf{u}_{t+1} = (1-\gamma_t)\mathbf{w}_{t+1} + \gamma_t \mathbf{w}_t + c_t(\mathbf{u}_t - \mathbf{w}_{t+1})$.

Algorithm 2: Accelerated Shotgun

introduce an extra step size coefficient η, which allows us to unify several algorithms (see further discussion below). Note that η is fixed to 1 in the original Shotgun. We now state the convergence rate of our Algorithm in the following theorem.

Theorem 2. *Suppose $F(\mathbf{w})$ admits a minimizer $\mathbf{w}^*$. If P and $\eta > 0$ are such that $\frac{\eta}{2}(1+\sigma) < 1$ where $\sigma = \frac{(P-1)(\rho-1)}{d-1}$ (recall ρ is the spectral radius of $\mathbf{X}^T\mathbf{X}$), constant γ_t is defined as in Section 2, and $c_t = \frac{\theta_t}{\theta_{t+1}}\left(1 - \frac{2P}{d}\left(1 - \frac{\eta}{2}(1+\sigma)\right)\right)$, then the following holds for any $t > 1$,*

$$\mathbb{E}_{S_{1:t-1}}[F(\mathbf{w}_t)] - F(\mathbf{w}^*) \leq \frac{\beta d^2 \|\mathbf{w}_1 - \mathbf{w}^*\|^2}{t^2 P^2 \eta \left(1 - \frac{\eta}{2}(1+\sigma)\right)}, \tag{1}$$

where the expectation is with respect to the random choices of subsets $S_1, \ldots, S_{t-1}$.

The complete proof of Theorem 2 can be found in the long version of this work [9]. Here we focus on explaining how to interpret this result and how to choose parameters P and η. First, note that to reach ϵ accuracy (i.e. $\mathbb{E}[F(\mathbf{w}_t)] - F(\mathbf{w}^*) \leq \epsilon$), Algorithm 2 requires $T_\epsilon = O\left(\frac{d}{P}\sqrt{\beta / \left(\epsilon\eta\left(1 - \frac{\eta}{2}(1+\sigma)\right)\right)}\right)$ iterations. For any fixed P, one can verify that $\eta^* = 1/(1+\sigma)$ is the optimal choice for η to minimize T_ϵ. Plugging η^* back in T_ϵ and minimizing over P, one can check that $P = d$ is the best choice. In this case, since $\sigma = \rho - 1, \eta^* = 1/\rho$ and $c_t = 0$, Algorithm 2 actually degenerates to FISTA and $T_\epsilon = O(\sqrt{\rho\beta/\epsilon})$, recovering the results in Theorem 1 exactly.

Of course, at the end it is not the number of iterations, but the total computational complexity that we care about most. Suppose we implement the algorithm without using any parallel computation. Then as we will discuss later, the time complexity for each iteration would be $O(nP)$ (recall n is the number of examples), and the total complexity $O(T_\epsilon nP)$ is minimized when $\eta = 1$ and $P = 1$, leading to $O(nd\sqrt{\beta/\epsilon})$. In this case, our algorithm degenerates to an accelerated version of randomized (single) coordinate descent. Note that this essentially recovers the algorithms and results in [12, 8], but in a much simpler form (and analysis).

Finally, we consider implementing the algorithm using parallel computation. Note that $\eta^* = 1/(1+\sigma)$ is always at most 1. So we fix η to be 1 (as in the original Shotgun algorithm), leading to the largest possible step size $1/\beta$ and a potentially small P. We assume in this case updating P coordinates in parallel costs approximately the same time no matter what P is. In other words, we are again only interested in minimizing T_ϵ. One can verify that the optimal choice here for P is $\frac{2}{3}(\frac{d-1}{\rho-1} + 1)$ and $T_\epsilon = O(\rho\sqrt{\beta/\epsilon})$. This improves the convergence rate from $O(1/\epsilon)$ to $O(\sqrt{1/\epsilon})$ compared to Shotgun, and is almost as good as FISTA (with much less computation per iteration).

Efficient Implementation. We discuss how to efficiently implement Algorithm 2. 1) At first glance it seems that computing vector $\mathbf{u}$ needs to go over all d coordinates. However, one can easily generalize the trick introduced in [8] to update only P coordinates and compute $\mathbf{w}$ and $\mathbf{u}$ implicitly. 2) Another widely used trick is to maintain inner products between examples and weight vectors (i.e. $\mathbf{Xw}$ and $\mathbf{Xu}$), so that computing a single coordinate of the gradient can be done in $O(n)$, or even faster in the case of sparse data. 3) Instead of choosing S_t uniformly at random, one can also do the following: arbitrarily divide the set $\{1, \ldots, d\}$ into P disjoint subsets of equal size in advanced (assuming d is a multiple of P for simplicity), then at each iteration, select one and only one element from each of the P subsets uniformly at random to form S_t. This would only lead to a minor change of the convergence results (indeed, one only needs to redefine σ to be $(\rho-1)P/d$). The advantage of this approach is that it suggests that we can separate the data matrix $\mathbf{X}$ by columns and store these subsets on P machines separately to naturally allow parallel updates at each iteration.

References

[1] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[2] Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems 20*, 2008.

[3] Joseph K. Bradley, Aapo Kyrola, Danny Bickson, and Carlos Guestrin. Parallel coordinate descent for l_1-regularized loss minimization. In *Proceedings of the 28th International Conference on Machine Learning*, 2011.

[4] Olivier Fercoq and Peter Richtárik. Accelerated, parallel and proximal coordinate descent. *arXiv preprint arXiv:1312.5799*, 2014.

[5] Wenjiang J Fu. Penalized regressions: the bridge versus the lasso. *Journal of computational and graphical statistics*, 7(3):397–416, 1998.

[6] Seung-Jean Kim, Kwangmoo Koh, Michael Lustig, Stephen Boyd, and Dimitry Gorinevsky. An interior-point method for large-scale l 1-regularized least squares. *Selected Topics in Signal Processing, IEEE Journal of*, 1(4):606–617, 2007.

[7] John Langford, Alexander Smola, and Martin Zinkevich. Slow learners are fast. *Advances in Neural Information Processing Systems 22*, 2009.

[8] Yin Tat Lee and Aaron Sidford. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. In *54th Annual Symposium on Foundations of Computer Science*, 2013.

[9] Haipeng Luo, Patrick Haffner, and Jean-François Paiement. Accelerated parallel optimization methods for large scale machine learning. *arXiv preprint arXiv:1411.6725*, 2014.

[10] Gideon Mann, Ryan McDonald, Mehryar Mohri, Nathan Silberman, and Daniel D. Walker. Efficient large-scale distributed training of conditional maximum entropy models. In *Advances in Neural Information Processing Systems 22*, 2009.

[11] Indraneel Mukherjee, Kevin Canini, Rafael Frongillo, and Yoram Singer. Parallel boosting with momentum. *ECML PKDD 2013, Part III, LNAI 8190*, pages 17–32, 2013.

[12] Yu. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

[13] YU. E. Nesterov. A method of solving a convex programming problem with convergence rate $o(1/k^2)$. In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.

[14] Andrew Y. Ng. Feature selection, L_1 vs. L_2 regularization, and rotational invariance. In *Proceedings of the Twenty-First International Conference on Machine Learning*, 2004.

[15] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer Series in Operations Research and Financial Engineering, 2nd edition, 2006.

[16] Peter Richtárik and Martin Takáč. Parallel coordinate descent methods for big data optimization. *arXiv preprint arXiv:1212.0873*, 2012.

[17] Shai Shalev-Shwartz and Nathan Srebro. Svm optimization: inverse dependence on training set size. In *Proceedings of the 25th International Conference on Machine Learning*, 2008.

[18] Shai Shalev-Shwartz and Ambuj Tewari. Stochastic methods for l1-regularized loss minimization. *The Journal of Machine Learning Research*, 12:1865–1892, 2011.

[19] Shai Shalev-Shwartz and Tong Zhang. Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization. In *Proceedings of the 31st International Conference on Machine Learning*, 2014.

[20] Krysta M Svore and CJ Burges. Large-scale learning to rank using boosted decision trees. *Scaling Up Machine Learning: Parallel and Distributed Approaches*, 2, 2011.

[21] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B (Methodological)*, 58(1):267–288, 1996.

[22] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems 23*, 2010.