Parallel Coordinate Descent for L_1-Regularized Loss Minimization

Joseph K. Bradley † JKBRADLE@CS.CMU.EDU
Aapo Kyrola † AKYROLA@CS.CMU.EDU
Danny Bickson BICKSON@CS.CMU.EDU
Carlos Guestrin GUESTRIN@CS.CMU.EDU
Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213 USA

Abstract

We propose Shotgun, a parallel coordinate descent algorithm for minimizing L_1-regularized losses. Though coordinate descent seems inherently sequential, we prove convergence bounds for Shotgun which predict linear speedups, up to a problem-dependent limit. We present a comprehensive empirical study of Shotgun for Lasso and sparse logistic regression. Our theoretical predictions on the potential for parallelism closely match behavior on real data. Shotgun outperforms other published solvers on a range of large problems, proving to be one of the most scalable algorithms for L_1.

1. Introduction

Many applications use L_1-regularized models such as the Lasso (Tibshirani, 1996) and sparse logistic regression (Ng, 2004). L_1 regularization biases learning towards sparse solutions, and it is especially useful for *high-dimensional* problems with large numbers of features. For example, in logistic regression, it allows sample complexity to scale logarithmically w.r.t. the number of irrelevant features (Ng, 2004).

Much effort has been put into developing optimization algorithms for L_1 models. These algorithms range from coordinate minimization (Fu, 1998) and stochastic gradient (Shalev-Shwartz & Tewari, 2009) to more complex interior point methods (Kim et al., 2007).

Coordinate descent, which we call *Shooting* after Fu (1998), is a simple but very effective algorithm which updates one coordinate per iteration. It often requires no tuning of parameters, unlike, e.g., stochastic gradient. As we discuss in Sec. 2, theory (Shalev-Shwartz & Tewari, 2009) and extensive empirical results (Yuan et al., 2010) have shown that variants of Shooting are particularly competitive for high-dimensional data.

The need for scalable optimization is growing as more applications use high-dimensional data, but processor core speeds have stopped increasing in recent years. Instead, computers come with more cores, and the new challenge is utilizing them efficiently. Yet despite the many sequential optimization algorithms for L_1-regularized losses, few parallel algorithms exist.

Some algorithms, such as interior point methods, can benefit from parallel matrix-vector operations. However, we found empirically that such algorithms were often outperformed by Shooting.

Recent work analyzes parallel stochastic gradient descent for multicore (Langford et al., 2009b) and distributed settings (Mann et al., 2009; Zinkevich et al., 2010). These methods parallelize over samples. In applications using L_1 regularization, though, there are often many more features than samples, so parallelizing over samples may be of limited utility.

We therefore take an orthogonal approach and parallelize over features, with a remarkable result: we can parallelize coordinate descent—an algorithm which seems inherently sequential—for L_1-regularized losses. In Sec. 3, we propose *Shotgun*, a simple multicore algorithm which makes $\mathtt{P}$ coordinate updates in parallel. We prove strong convergence bounds for Shotgun which predict speedups over Shooting which are linear in $\mathtt{P}$, up to a problem-dependent maximum $\mathtt{P}^*$. Moreover, our theory provides an estimate for this ideal $\mathtt{P}^*$ which may be easily computed from the data.

Parallel coordinate descent was also considered by Tsitsiklis et al. (1986), but for differentiable objectives in the asynchronous setting. They give a very

Appearing in *Proceedings of the 28th International Conference on Machine Learning*, Bellevue, WA, USA, 2011.

† These authors contributed equally to this work.

general analysis, proving asymptotic convergence but not convergence rates. We are able to prove rates and theoretical speedups for our class of objectives.

In Sec. 4, we compare multicore Shotgun with five state-of-the-art algorithms on 35 real and synthetic datasets. The results show that in large problems Shotgun outperforms the other algorithms. Our experiments also validate the theoretical predictions by showing that Shotgun requires only about $1/\mathtt{P}$ as many iterations as Shooting. We measure the parallel speedup in running time and analyze the limitations imposed by the multicore hardware.

2. L_1-Regularized Loss Minimization

We consider optimization problems of the form

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

where $L(\cdot)$ is a non-negative convex loss. Each of n *samples* has a *feature vector* $\mathbf{a}_i \in \mathbb{R}^d$ and observation y_i (where $\mathbf{y} \in \mathcal{Y}^n$). $\mathbf{x} \in \mathbb{R}^d$ is an *unknown* vector of *weights* for features. $\lambda \geq 0$ is a regularization parameter. Let $\mathbf{A} \in \mathbb{R}^{n\times d}$ be the design matrix, whose i^{th} row is $\mathbf{a}_i$. Assume w.l.o.g. that columns of $\mathbf{A}$ are normalized s.t. $diag(\mathbf{A}^T\mathbf{A}) = \mathbf{1}$.[1]

An instance of (1) is the Lasso (Tibshirani, 1996) (in penalty form), for which $\mathcal{Y} \equiv \mathbb{R}$ and

$$F(\mathbf{x}) = \tfrac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{x}\|_1\,, \tag{2}$$

as well as sparse logistic regression (Ng, 2004), for which $\mathcal{Y} \equiv \{-1, +1\}$ and

$$F(\mathbf{x}) = \sum_{i=1}^{n} \log\left(1 + \exp\left(-y_i\mathbf{a}_i^T\mathbf{x}\right)\right) + \lambda\|\mathbf{x}\|_1\,. \tag{3}$$

For analysis, we follow Shalev-Shwartz and Tewari (2009) and transform (1) into an equivalent problem with a twice-differentiable regularizer. We let $\hat{\mathbf{x}} \in \mathbb{R}^{2d}_+$, use *duplicated features* $\hat{\mathbf{a}}_i = [\mathbf{a}_i; -\mathbf{a}_i] \in \mathbb{R}^{2d}$, and solve

$$\min_{\hat{\mathbf{x}}\in\mathbb{R}^{2d}_+} \sum_{i=1}^{n} L(\hat{\mathbf{a}}_i^T\hat{\mathbf{x}}, y_i) + \lambda\sum_{j=1}^{2d}\hat{x}_j\,. \tag{4}$$

If $\hat{\mathbf{x}} \in \mathbb{R}^{2d}_+$ minimizes (4), then $\mathbf{x}: x_i = \hat{x}_{d+i} - \hat{x}_i$ minimizes (1). Though our analysis uses duplicate features, they are not needed for an implementation.

2.1. Sequential Coordinate Descent

Shalev-Shwartz and Tewari (2009) analyze Stochastic Coordinate Descent (SCD), a stochastic version of Shooting for solving (1). SCD (Alg. 1) randomly chooses one weight x_j to update per iteration. It computes the update $x_j \leftarrow x_j + \delta x_j$ via

$$\delta x_j = \max\{-x_j, -(\nabla F(\mathbf{x}))_j/\beta\}\,, \tag{5}$$

where $\beta > 0$ is a loss-dependent constant.

Algorithm 1 Shooting: Sequential SCD

Set $\mathbf{x} = \mathbf{0} \in \mathbb{R}^{2d}_+$.
while not converged **do**
  Choose $j \in \{1, \ldots, 2d\}$ uniformly at random.
  Set $\delta x_j \longleftarrow \max\{-x_j, -(\nabla F(\mathbf{x}))_j/\beta\}$.
  Update $x_j \longleftarrow x_j + \delta x_j$.
end while

To our knowledge, Shalev-Shwartz and Tewari (2009) provide the best known convergence bounds for SCD. Their analysis requires a uniform upper bound on the change in the loss $F(\mathbf{x})$ from updating a single weight:

Assumption 2.1. *Let $F(\mathbf{x}) : \mathbb{R}^{2d}_+ \longrightarrow \mathbb{R}$ be a convex function. Assume there exists $\beta > 0$ s.t., for all $\mathbf{x}$ and single-weight updates δx_j, we have:*

$$F(\mathbf{x} + (\delta x_j)\mathbf{e}^j) \leq F(\mathbf{x}) + \delta x_j(\nabla F(\mathbf{x}))_j + \tfrac{\beta(\delta x_j)^2}{2}\,,$$

where $\mathbf{e}^j$ is a unit vector with 1 in its j^{th} entry. For the losses in (2) and (3), Taylor expansions give

$$\beta = 1 \text{ (squared loss) and } \beta = \tfrac{1}{4} \text{ (logistic loss).} \tag{6}$$

Using this bound, they prove the following theorem.

Theorem 2.1. *(Shalev-Shwartz & Tewari, 2009) Let $\mathbf{x}^*$ minimize (4) and $\mathbf{x}^{(T)}$ be the output of Alg. 1 after T iterations. If $F(\mathbf{x})$ satisfies Assumption 2.1, then*

$$\mathbf{E}\left[F(\mathbf{x}^{(T)})\right] - F(\mathbf{x}^*) \leq \frac{d(\beta\|\mathbf{x}^*\|_2^2 + 2F(\mathbf{x}^{(0)}))}{T+1}\,, \tag{7}$$

where $\mathbf{E}[\cdot]$ is w.r.t. the random choices of weights j.

As Shalev-Shwartz and Tewari (2009) argue, Theorem 2.1 indicates that SCD scales well in the dimensionality d of the data. For example, it achieves better runtime bounds w.r.t. d than stochastic gradient methods such as SMIDAS (Shalev-Shwartz & Tewari, 2009) and truncated gradient (Langford et al., 2009a).

3. Parallel Coordinate Descent

As the dimensionality d or sample size n increase, even fast sequential algorithms become expensive. To scale to larger problems, we turn to parallel computation. In this section, we present our main theoretical contribution: we show coordinate descent can be parallelized by proving strong convergence bounds.

[1] Normalizing $\mathbf{A}$ does not change the objective if a separate, normalized λ_j is used for each x_j.

Algorithm 2 Shotgun: Parallel SCD

Choose number of parallel updates $\mathtt{P} \geq 1$.
Set $\mathbf{x} = \mathbf{0} \in \mathbb{R}^{2d}_+$
while not converged **do**
  In parallel on P processors
    Choose $j \in \{1, \ldots, 2d\}$ uniformly at random.
    Set $\delta x_j \longleftarrow \max\{-x_j, -(\nabla F(\mathbf{x}))_j/\beta\}$.
    Update $x_j \longleftarrow x_j + \delta x_j$.
end while

We parallelize stochastic Shooting and call our algorithm Shotgun (Alg. 2). Shotgun initially chooses P, the number of weights to update in parallel. On each iteration, it chooses P weights independently and uniformly at random from $\{1, \ldots, 2d\}$; these form a multiset $\mathcal{P}_t$. It updates each $x_{i_j} : i_j \in \mathcal{P}_t$, in parallel using the same update as Shooting (5). Let $\Delta\mathbf{x}$ be the collective update to $\mathbf{x}$, i.e., $(\Delta\mathbf{x})_k = \sum_{i_j \in \mathcal{P}_t : k = i_j} \delta x_{i_j}$.

Intuitively, parallel updates might increase the risk of divergence. In Fig. 1, in the left subplot, parallel updates speed up convergence since features are uncorrelated; in the right subplot, parallel updates of correlated features risk increasing the objective. We can avoid divergence by imposing a step size, but our experiments showed that approach to be impractical.[2]

We formalize this intuition for the Lasso in Theorem 3.1. We can separate a sequential progress term (summing the improvement from separate updates) from a term measuring interference between parallel updates. If $\mathbf{A}^T\mathbf{A}$ were normalized and centered to be a covariance matrix, the elements in the interference term's sum would be non-zero only for correlated variables, matching our intuition from Fig. 1. Harmful interference could occur when, e.g., $\delta x_i, \delta x_j > 0$ and features i, j were positively correlated.

Theorem 3.1. *Fix* $\mathbf{x}$. *If* $\Delta\mathbf{x}$ *is the collective update to* $\mathbf{x}$ *in one iteration of Alg. 2 for the Lasso, then*

$$F(\mathbf{x} + \Delta\mathbf{x}) - F(\mathbf{x}) \leq \underbrace{-\tfrac{1}{2} \sum_{i_j \in \mathcal{P}_t} (\delta x_{i_j})^2}_{\text{sequential progress}} + \underbrace{\tfrac{1}{2} \sum_{\substack{i_j, i_k \in \mathcal{P}_t, \\ j \neq k}} (\mathbf{A}^T\mathbf{A})_{i_j, i_k} \delta x_{i_j} \delta x_{i_k}}_{\text{interference}} .$$

Proof Sketch[3]: Write the Taylor expansion of F around $\mathbf{x}$. Bound the first-order term using (5). ■
In the next section, we show that this intuition holds for the more general optimization problem in (1).

[2] A step size of $\frac{1}{\mathtt{P}}$ ensures convergence since F is convex in $\mathbf{x}$, but it results in very small steps and long runtimes.

[3] We include detailed proofs of all theorems and lemmas in the supplementary material.

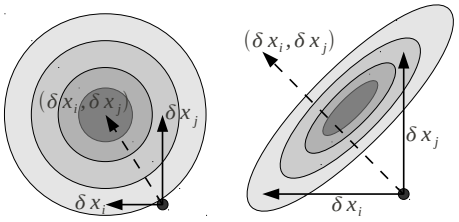


Figure 1. Intuition for parallel coordinate descent. Contour plots of two objectives, with darker meaning better. Left: Features are uncorrelated; parallel updates are useful. Right: Features are correlated; parallel updates conflict.

3.1. Shotgun Convergence Analysis

In this section, we present our convergence result for Shotgun. The result provides a problem-specific measure of the potential for parallelization: the spectral radius ρ of $\mathbf{A}^T\mathbf{A}$ (i.e., the maximum of the magnitudes of eigenvalues of $\mathbf{A}^T\mathbf{A}$). Moreover, this measure is prescriptive: ρ may be estimated via, e.g., power iteration[4] (Strang, 1988), and it provides a plug-in estimate of the ideal number of parallel updates.

We begin by generalizing Assumption 2.1 to our parallel setting. The scalars β for Lasso and logistic regression remain the same as in (6).

Assumption 3.1. *Let* $F(\mathbf{x}) : \mathbb{R}^{2d}_+ \longrightarrow \mathbb{R}$ *be a convex function. Assume that there exists* $\beta > 0$ *such that, for all* $\mathbf{x}$ *and parallel updates* $\Delta\mathbf{x}$, *we have*

$$F(\mathbf{x} + \Delta\mathbf{x}) \leq F(\mathbf{x}) + \Delta\mathbf{x}^T \nabla F(\mathbf{x}) + \tfrac{\beta}{2} \Delta\mathbf{x}^T \mathbf{A}^T \mathbf{A} \Delta\mathbf{x} .$$

We now state our main result, generalizing the convergence bound in Theorem 2.1 to the Shotgun algorithm.

Theorem 3.2. *Let* $\mathbf{x}^*$ *minimize* (4) *and* $\mathbf{x}^{(T)}$ *be the output of Alg. 2 after* T *iterations with* P *parallel updates/iteration. Let* ρ *be the spectral radius of* $\mathbf{A}^T\mathbf{A}$. *If* $F(\mathbf{x})$ *satisfies Assumption 3.1 and* $\mathtt{P} < \frac{2d}{\rho} + 1$, *then*

$$\mathbf{E}\left[F(\mathbf{x}^{(T)})\right] - F(\mathbf{x}^*) \leq \frac{d\left(\beta \|\mathbf{x}^*\|_2^2 + 2F(\mathbf{x}^{(0)})\right)}{(T+1)\mathtt{P}} ,$$

where the expectation is w.r.t. the random choices of weights to update. Choosing a maximal $\mathtt{P}^* \approx \frac{2d}{\rho}$ *gives*

$$\mathbf{E}\left[F(\mathbf{x}^{(T)})\right] - F(\mathbf{x}^*) \lesssim \frac{\rho\left(\frac{\beta}{2}\|\mathbf{x}^*\|_2^2 + F(\mathbf{x}^{(0)})\right)}{T+1} .$$

Without duplicated features, Theorem 3.2 predicts that we can do up to $\mathtt{P} < \frac{d}{\rho} + 1$ parallel updates and achieve speedups linear in P. We denote the predicted

[4] For our datasets, power iteration gave reasonable estimates within a small fraction of the total runtime.

maximum $\mathtt{P}$ as $\mathtt{P}^* \equiv \textit{ceiling}(\frac{d}{\rho})$. For an ideal problem with uncorrelated features, $\rho = 1$, so we could do up to $\mathtt{P}^* = d$ parallel updates. For a pathological problem with exactly correlated features, $\rho = d$, so our theorem tells us that we could not do parallel updates. With $\mathtt{P} = 1$, we recover the result for Shooting in Theorem 2.1.

To prove Theorem 3.2, we first bound the negative impact of interference between parallel updates.

Lemma 3.3. *Fix* $\mathbf{x}$. *Under the assumptions and definitions from Theorem 3.2, if* $\Delta\mathbf{x}$ *is the collective update to* $\mathbf{x}$ *in one iteration of Alg. 2, then*

$$\mathbf{E}_{\mathcal{P}_t}\left[F(\mathbf{x}+\Delta\mathbf{x}) - F(\mathbf{x})\right] \leq \mathtt{P}\mathbf{E}_j\left[\delta x_j(\nabla F(\mathbf{x}))_j + \tfrac{\beta}{2}\left(1 - \tfrac{(\mathtt{P}-1)\rho}{2d}\right)(\delta x_j)^2\right],$$

where $\mathbf{E}_{\mathcal{P}_t}$ *is w.r.t. a random choice of* $\mathcal{P}_t$ *and* $\mathbf{E}_j$ *is w.r.t. choosing* $j \in \{1,\ldots,2d\}$ *uniformly at random.*

Proof Sketch: Take the expectation w.r.t. $\mathcal{P}_t$ of the inequality in Assumption 3.1.

$$\begin{aligned}&\mathbf{E}_{\mathcal{P}_t}\left[F(\mathbf{x}+\Delta\mathbf{x}) - F(\mathbf{x})\right] \\ &\quad\leq \mathbf{E}_{\mathcal{P}_t}\left[\Delta\mathbf{x}^T\nabla F(\mathbf{x}) + \tfrac{\beta}{2}\Delta\mathbf{x}^T\mathbf{A}^T\mathbf{A}\Delta\mathbf{x}\right]\end{aligned} \tag{8}$$

Separate the diagonal elements from the second order term, and rewrite the expectation using our independent choices of $i_j \in \mathcal{P}_t$. (Here, δx_j is the update given by (5), regardless of whether $j \in \mathcal{P}_t$.)

$$\begin{aligned}&= \mathtt{P}\mathbf{E}_j\left[\delta x_j(\nabla F(\mathbf{x}))_j + \tfrac{\beta}{2}(\delta x_j)^2\right] \\ &\quad + \tfrac{\beta}{2}\mathtt{P}(\mathtt{P}-1)\mathbf{E}_i\left[\mathbf{E}_j\left[\delta x_i(\mathbf{A}^T\mathbf{A})_{i,j}\delta x_j\right]\right]\end{aligned} \tag{9}$$

Upper bound the double expectation in terms of $\mathbf{E}_j\left[(\delta x_j)^2\right]$ by expressing the spectral radius ρ of $\mathbf{A}^T\mathbf{A}$ as $\rho = \max_{\mathbf{z}:\,\mathbf{z}^T\mathbf{z}=1}\mathbf{z}^T(\mathbf{A}^T\mathbf{A})\mathbf{z}$.

$$\mathbf{E}_i\left[\mathbf{E}_j\left[\delta x_i(\mathbf{A}^T\mathbf{A})_{i,j}\delta x_j\right]\right] \leq \tfrac{\rho}{2d}\mathbf{E}_j\left[(\delta x_j)^2\right] \tag{10}$$

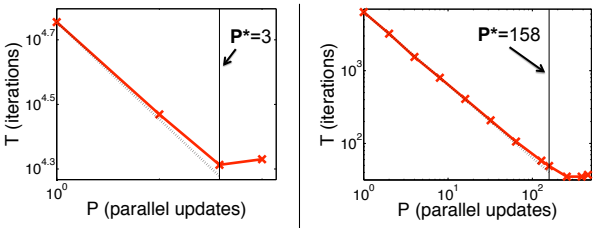
Combine (10) back into (9), and rearrange terms to get the lemma's result. ■

Proof Sketch (Theorem 3.2): Our proof resembles Shalev-Shwartz and Tewari (2009)'s proof of Theorem 2.1. The result from Lemma 3.3 replaces Assumption 2.1. One bound requires $\frac{(\mathtt{P}-1)\rho}{2d} < 1$. ■

Our analysis implicitly assumes that parallel updates of the same weight x_j will not make x_j negative. Proper write-conflict resolution can ensure this assumption holds and is viable in our multicore setting.

3.2. Theory vs. Empirical Performance

We end this section by comparing the predictions of Theorem 3.2 about the number of parallel updates $\mathtt{P}$ with empirical performance for Lasso. We exactly simulated Shotgun as in Alg. 2 to eliminate effects from the practical implementation choices made in Sec. 4. We tested two single-pixel camera datasets from Duarte et al. (2008) with very different ρ, estimating $\mathbf{E}_{P_t}\left[F(\mathbf{x}^{(T)})\right]$ by averaging 10 runs of Shotgun. We used $\lambda = 0.5$ for `Ball64_singlepixcam` to get $\mathbf{x}^*$ with about 27% non-zeros; we used $\lambda = 0.05$ for `Mug32_singlepixcam` to get about 20% non-zeros.



Data: Ball64_singlepixcam
$d = 4096$, $\rho = 2047.8$

Data: Mug32_singlepixcam
$d = 1024$, $\rho = 6.4967$

Figure 2. Theory for Shotgun's $\mathtt{P}$ (Theorem 3.2) vs. empirical performance for Lasso on two datasets. Y-axis has iterations T until $\mathbf{E}_{P_t}[F(\mathbf{x}^{(T)})]$ came within 0.5% of $F(\mathbf{x}^*)$. Thick red lines trace T for increasing $\mathtt{P}$ (until too large $\mathtt{P}$ caused divergence). Vertical lines mark $\mathtt{P}^*$. Dotted diagonal lines show optimal (linear) speedups (partly hidden by solid line in right-hand plot).

Fig. 2 plots $\mathtt{P}$ versus the iterations T required for $\mathbf{E}_{P_t}\left[F(\mathbf{x}^{(T)})\right]$ to come within 0.5% of the optimum $F(\mathbf{x}^*)$. Theorem 3.2 predicts that T should decrease as $\frac{1}{\mathtt{P}}$ as long as $\mathtt{P} < \mathtt{P}^* \approx \frac{d}{\rho}+1$. The empirical behavior follows this theory: using the predicted $\mathtt{P}^*$ gives almost optimal speedups, and speedups are almost linear in $\mathtt{P}$. As $\mathtt{P}$ exceeds $\mathtt{P}^*$, Shotgun soon diverges.

Fig. 2 confirms Theorem 3.2's result: Shooting, a seemingly sequential algorithm, can be parallelized and achieve near-linear speedups, and the spectral radius of $\mathbf{A}^T\mathbf{A}$ succinctly captures the potential for parallelism in a problem. To our knowledge, our convergence results are the first for parallel coordinate descent for L_1-regularized losses, and they apply to any convex loss satisfying Assumption 3.1. Though Fig. 2 ignores certain implementation issues, we show in the next section that Shotgun performs well in practice.

3.3. Beyond L_1

Theorems 2.1 and 3.2 generalize beyond L_1, for their main requirements (Assumptions 2.1, 3.1) apply to a more general class of problems: $\min F(\mathbf{x})$ s.t. $\mathbf{x} \geq 0$, where $F(\mathbf{x})$ is smooth. We discuss Shooting and Shotgun for sparse regression since both the method (coordinate descent) and problem (sparse regression) are arguably most useful for high-dimensional settings.

4. Experimental Results

We present an extensive study of Shotgun for the Lasso and sparse logistic regression. On a wide variety of datasets, we compare Shotgun with published state-of-the-art solvers. We also analyze self-speedup in detail in terms of Theorem 3.2 and hardware issues.

4.1. Lasso

We tested Shooting and Shotgun for the Lasso against five published Lasso solvers on 35 datasets. We summarize the results here; details are in the supplement.

4.1.1. Implementation: `Shotgun`

Our implementation made several practical improvements to the basic Shooting and Shotgun algorithms.

Following Friedman et al. (2010), we maintained a vector $\mathbf{Ax}$ to avoid repeated computation. We also used their pathwise optimization scheme: rather than directly solving with the given λ, we solved with an exponentially decreasing sequence $\lambda_1, \lambda_2, \ldots, \lambda$. The solution $\mathbf{x}$ for λ_k is used to warm-start optimization for λ_{k+1}. This scheme can give significant speedups.

Though our analysis is for the synchronous setting, our implementation was asynchronous because of the high cost of synchronization. We used atomic compare-and-swap operations for updating the $\mathbf{Ax}$ vector.

We used C++ and the CILK++ library (Leiserson, 2009) for parallelism. All tests ran on an AMD processor using up to eight Opteron 8384 cores (2.69 GHz).

4.1.2. Other Algorithms

`L1_LS` (Kim et al., 2007) is a log-barrier interior point method. It uses Preconditioned Conjugate Gradient (PCG) to solve Newton steps iteratively and avoid explicitly inverting the Hessian. The implementation is in Matlab®, but the expensive step (PCG) uses very efficient native Matlab calls. In our tests, matrix-vector operations were parallelized on up to 8 cores.

`FPC_AS` (Wen et al., 2010) uses iterative shrinkage to estimate which elements of $\mathbf{x}$ should be non-zero, as well as their signs. This reduces the objective to a smooth, quadratic function which is then minimized.

`GPSR_BB` (Figueiredo et al., 2008) is a gradient projection method which uses line search and termination techniques tailored for the Lasso.

`Hard_l0` (Blumensath & Davies, 2009) uses iterative hard thresholding for compressed sensing. It sets all but the s largest weights to zero on each iteration. We set s as the sparsity obtained by `Shooting`.

`SpaRSA` (Wright et al., 2009) is an accelerated iterative shrinkage/thresholding algorithm which solves a sequence of quadratic approximations of the objective.

As with `Shotgun`, all of `Shooting`, `FPC_AS`, `GPSR_BB`, and `SpaRSA` use pathwise optimization schemes.

We also tested published implementations of the classic algorithms `GLMNET` (Friedman et al., 2010) and `LARS` (Efron et al., 2004). Since we were unable to get them to run on our larger datasets, we exclude their results.

4.1.3. Results

We divide our comparisons into four categories of datasets; the supplementary material has descriptions.

Sparco: Real-valued datasets of varying sparsity from the Sparco testbed (van den Berg et al., 2009). $n \in [12829166]$, $d \in [128, 29166]$.

Single-Pixel Camera: Dense compressed sensing problems from Duarte et al. (2008). $n \in [410, 4770]$, $d \in [1024, 16384]$.

Sparse Compressed Imaging: Similar to Single-Pixel Camera datasets, but with very sparse random $-1/+1$ measurement matrices. Created by us. $n \in [477, 32768]$, $d \in [954, 65536]$.

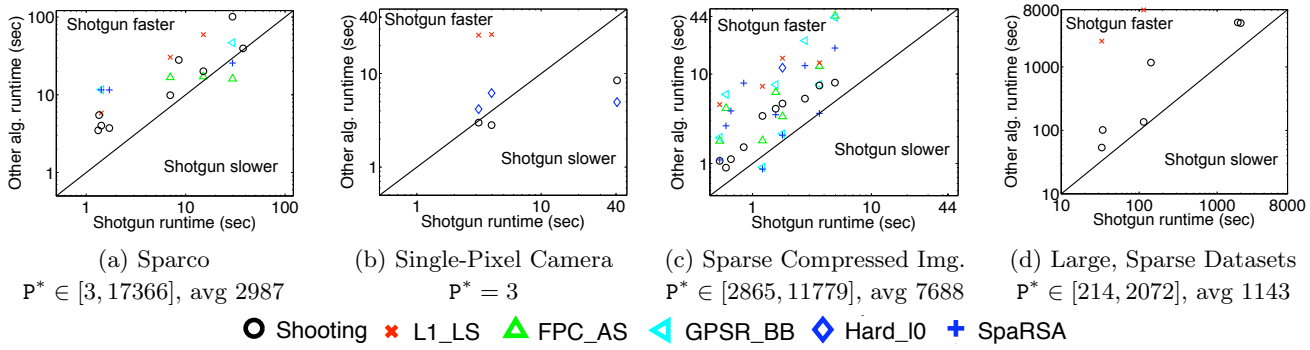
Large, Sparse Datasets: Very large and sparse problems, including predicting stock volatility from text in financial reports (Kogan et al., 2009). $n \in [30465, 209432]$, $d \in [209432, 5845762]$.

We ran each algorithm on each dataset with regularization $\lambda = 0.5$ and 10. Fig. 3 shows runtime results, divided by dataset category. We omit runs which failed to converge within a reasonable time period.

`Shotgun` (with `P` $= 8$) consistently performs well, converging faster than other algorithms on most dataset categories. `Shotgun` does particularly well on the Large, Sparse Datasets category, for which most algorithms failed to converge anywhere near the ranges plotted in Fig. 3. The largest dataset, whose features are occurrences of bigrams in financial reports (Kogan et al., 2009), has 5 million features and 30K samples. On this dataset, `Shooting` converges but requires ~ 4900 seconds, while `Shotgun` takes < 2000 seconds.

On the Single-Pixel Camera datasets, `Shotgun` (`P` $= 8$) is slower than `Shooting`. In fact, it is surprising that `Shotgun` converges at all with `P` $= 8$, for the plotted datasets all have `P`$^* = 3$. Fig. 2 shows `Shotgun` with `P` > 4 diverging for the `Ball64_singlepixcam` dataset; however, after the practical adjustments to `Shotgun` used to produce Fig. 3, `Shotgun` converges with `P` $= 8$.

Among the other solvers, `L1_LS` is the most robust and even solves some of the Large, Sparse Datasets.



(a) Sparco
$P^* \in [3, 17366]$, avg 2987

(b) Single-Pixel Camera
$P^* = 3$

(c) Sparse Compressed Img.
$P^* \in [2865, 11779]$, avg 7688

(d) Large, Sparse Datasets
$P^* \in [214, 2072]$, avg 1143

Figure 3. Runtime comparison of algorithms for the Lasso on 4 dataset categories. Each marker compares an algorithm with `Shotgun` (with `P` = 8) on one dataset (and one $\lambda \in \{0.5, 10\}$). Y-axis is that algorithm's running time; X-axis is `Shotgun`'s (P=8) running time on the same problem. Markers above the diagonal line indicate that `Shotgun` was faster; markers below the line indicate `Shotgun` was slower.

It is difficult to compare optimization algorithms and their implementations. Algorithms' termination criteria differ; e.g., primal-dual methods such as `L1_LS` monitor the duality gap, while `Shotgun` monitors the change in $\mathbf{x}$. `Shooting` and `Shotgun` were written in C++, which is generally fast; the other algorithms were in Matlab, which handles loops slowly but linear algebra quickly. Therefore, we emphasize major trends: `Shotgun` robustly handles a range of problems; Theorem 3.2 helps explain its speedups; and `Shotgun` generally outperforms published solvers for the Lasso.

4.2. Sparse Logistic Regression

For logistic regression, we focus on comparing Shotgun with Stochastic Gradient Descent (SGD) variants. SGD methods are of particular interest to us since they are often considered to be very efficient, especially for learning with many samples; they often have convergence bounds independent of the number of samples.

For a large-scale comparison of various algorithms for sparse logistic regression, we refer the reader to the recent survey by Yuan et al. (2010). On `L1_logreg` (Koh et al., 2007) and CDN (Yuan et al., 2010), our results qualitatively matched their survey. Yuan et al. (2010) do not explore SGD empirically.

4.2.1. Implementation: `Shotgun CDN`

As Yuan et al. (2010) show empirically, their Coordinate Descent Newton (CDN) method is often orders of magnitude faster than the basic Shooting algorithm (Alg. 1) for sparse logistic regression. Like Shooting, CDN does coordinate descent, but instead of using a fixed step size, it uses a backtracking line search starting at a quadratic approximation of the objective.

Although our analysis uses the fixed step size in (5), we modified Shooting and Shotgun to use line searches as in CDN. We refer to CDN as `Shooting CDN`, and we refer to parallel CDN as `Shotgun CDN`.

`Shooting CDN` and `Shotgun CDN` maintain an *active set* of weights which are allowed to become non-zero; this scheme speeds up optimization, though it can limit parallelism by shrinking d.

4.2.2. Other Algorithms

`SGD` iteratively updates $\mathbf{x}$ in a gradient direction estimated with one sample and scaled by a learning rate. We implemented `SGD` in C++ following, e.g., Zinkevich et al. (2010). We used lazy shrinkage updates (Langford et al., 2009a) to make use of sparsity in $\mathbf{A}$. Choosing learning rates for SGD can be challenging. In our tests, constant rates led to faster convergence than decaying rates (decaying as $1/\sqrt{T}$). For each test, we tried 14 exponentially increasing rates in $[10^{-4}, 1]$ (in parallel) and chose the rate giving the best training objective. We did not use a sparsifying step for `SGD`.

`SMIDAS` (Shalev-Shwartz & Tewari, 2009) uses stochastic mirror descent but truncates gradients to sparsify $\mathbf{x}$. We tested their published C++ implementation.

`Parallel SGD` refers to Zinkevich et al. (2010)'s work, which runs `SGD` in parallel on different subsamples of the data and averages the solutions $\mathbf{x}$. We tested this method since it is one of the few existing methods for parallel regression, but we note that Zinkevich et al. (2010) did not address L_1 regularization in their analysis. We averaged over 8 instances of `SGD`.

4.2.3. Results

Fig. 4 plots training objectives and test accuracy (on a held-out 10% of the data) for two large datasets.

The `zeta` dataset [5] illustrates the regime with $n \gg d$. It contains 500K samples with 2000 features and is fully dense (in $\mathbf{A}$). SGD performs well and is fairly competitive with `Shotgun CDN` (with `P` $= 8$).

The `rcv1` dataset [6] (Lewis et al., 2004) illustrates the high-dimensional regime ($d > n$). It has about twice as many features (44504) as samples (18217), with 17% non-zeros in $\mathbf{A}$. `Shotgun CDN` (`P` $= 8$) was much faster than `SGD`, especially in terms of the objective. `Parallel SGD` performed almost identically to `SGD`.

Though convergence bounds for `SMIDAS` are comparable to those for `SGD`, `SMIDAS` iterations take much longer due to the mirror descent updates. To execute 10M updates on the `zeta` dataset, `SGD` took 728 seconds, while `SMIDAS` took over 8500 seconds.

These results highlight how SGD is orthogonal to Shotgun: SGD can cope with large n, and Shotgun can cope with large d. A hybrid algorithm might be scalable in both n and d and, perhaps, be parallelized over both samples and features.

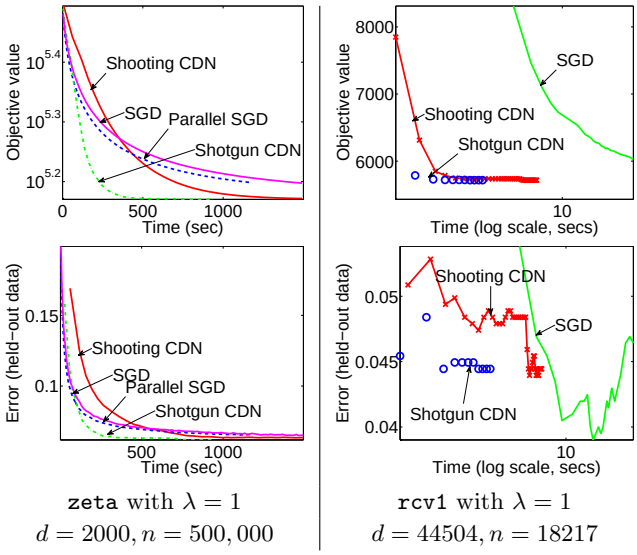


Figure 4. Sparse logistic regression on 2 datasets. Top plots trace training objectives over time; bottom plots trace classification error rates on held-out data (10%). On `zeta` ($n \gg d$), `SGD` converges faster initially, but `Shotgun CDN` (P=8) overtakes it. On `rcv1` ($d > n$), `Shotgun CDN` converges much faster than `SGD` (note the log scale); `Parallel SGD` (P=8) is hidden by `SGD`.

4.3. Self-Speedup of Shotgun

To study the self-speedup of `Shotgun Lasso` and `Shotgun CDN`, we ran both solvers on our datasets with varying λ, using varying `P` (number of parallel updates = number of cores). We recorded the running time as the first time when an algorithm came within 0.5% of the optimal objective, as computed by `Shooting`.

Fig. 5 shows results for both speedup (in time) and speedup in iterations until convergence. The speedups in iterations match Theorem 3.2 quite closely. However, relative speedups in iterations (about 8×) are not matched by speedups in runtime (about 2× to 4×).

We thus discovered that speedups in time were limited by low-level technical issues. To understand the limiting factors, we analyzed various Shotgun-like algorithms to find bottlenecks.[7] We found we were hitting the *memory wall* (Wulf & McKee, 1995); memory bus bandwidth and latency proved to be the most limiting factors. Each weight update requires an atomic update to the shared $\mathbf{Ax}$ vector, and the ratio of memory accesses to floating point operations is only $O(1)$. Data accesses have no temporal locality since each weight update uses a different column of $\mathbf{A}$. We further validated these conclusions by monitoring CPU counters.

[5] The `zeta` dataset is from the Pascal Large Scale Learning Challenge: `http://www.mlbench.org/instructions/`

[6] Our version of the `rcv1` dataset is from the LIBSVM repository (Chang & Lin, 2001).

[7] See the supplement for the scalability analysis details.

5. Discussion

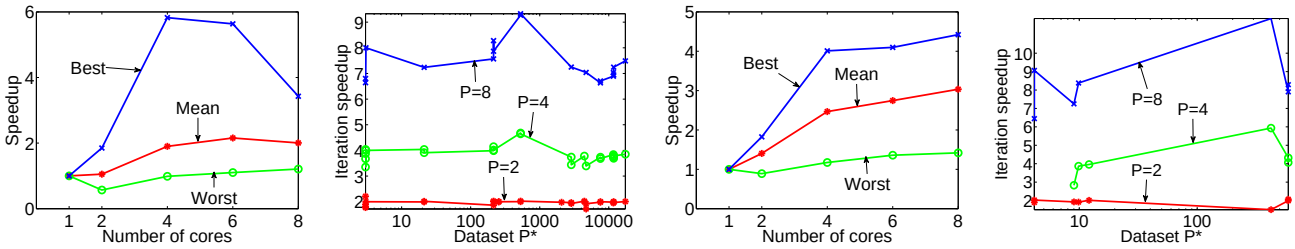
We introduced the Shotgun, a simple parallel algorithm for L_1-regularized optimization. Our convergence results for Shotgun are the first such results for parallel coordinate descent with L_1 regularization. Our bounds predict linear speedups, up to an interpretable, problem-dependent limit. In experiments, these predictions matched empirical behavior.

Extensive comparisons showed that Shotgun outperforms state-of-the-art L_1 solvers on many datasets. We believe that, currently, Shotgun is one of the most efficient and scalable solvers for L_1-regularized problems.

The most exciting extension to this work might be the hybrid of SGD and Shotgun discussed in Sec. 4.3.

Code, Data, and Benchmark Results: Available at `http://www.select.cs.cmu.edu/projects`

Acknowledgments
Thanks to John Langford, Guy Blelloch, Joseph Gonzalez, Yucheng Low and our reviewers for feedback. Funded by NSF IIS-0803333, NSF CNS-0721591, ARO MURI W911NF0710287, ARO MURI W911NF0810242.



(a) `Shotgun Lasso` runtime speedup (b) `Shotgun Lasso` iterations (c) `Shotgun CDN` runtime speedup (d) `Shotgun CDN` iterations

Figure 5. **(a,c)** Runtime speedup in time for `Shotgun Lasso` and `Shotgun CDN` (sparse logistic regression). **(b,d)** Speedup in iterations until convergence as a function of $\mathtt{P}^*$. Both Shotgun instances exhibit almost linear speedups w.r.t. iterations.

References

Blumensath, T. and Davies, M.E. Iterative hard thresholding for compressed sensing. *Applied and Computational Harmonic Analysis*, 27(3):265–274, 2009.

Chang, C.-C. and Lin, C.-J. *LIBSVM: a library for support vector machines*, 2001. `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Duarte, M.F., Davenport, M.A., Takhar, D., Laska, J.N., Sun, T., Kelly, K.F., and Baraniuk, R.G. Single-pixel imaging via compressive sampling. *Signal Processing Magazine, IEEE*, 25(2):83–91, 2008.

Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. Least angle regression. *Annals of Statistics*, 32(2):407–499, 2004.

Figueiredo, M.A.T, Nowak, R.D., and Wright, S.J. Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems. *IEEE J. of Sel. Top. in Signal Processing*, 1(4):586–597, 2008.

Friedman, J., Hastie, T., and Tibshirani, R. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.

Fu, W.J. Penalized regressions: The bridge versus the lasso. *J. of Comp. and Graphical Statistics*, 7(3):397–416, 1998.

Kim, S. J., Koh, K., Lustig, M., Boyd, S., and Gorinevsky, D. An interior-point method for large-scale ℓ1-regularized least squares. *IEEE Journal of Sel. Top. in Signal Processing*, 1(4):606–617, 2007.

Kogan, S., Levin, D., Routledge, B.R., Sagi, J.S., and Smith, N.A. Predicting risk from financial reports with regression. In *Human Language Tech.-NAACL*, 2009.

Koh, K., Kim, S.-J., and Boyd, S. An interior-point method for large-scale l1-regularized logistic regression. *JMLR*, 8:1519–1555, 2007.

Langford, J., Li, L., and Zhang, T. Sparse online learning via truncated gradient. In *NIPS*, 2009a.

Langford, J., Smola, A.J., and Zinkevich, M. Slow learners are fast. In *NIPS*, 2009b.

Leiserson, C. E. The Cilk++ concurrency platform. In *46th Annual Design Automation Conference*. ACM, 2009.

Lewis, D.D., Yang, Y., Rose, T.G., and Li, F. RCV1: A new benchmark collection for text categorization research. *JMLR*, 5:361–397, 2004.

Mann, G., McDonald, R., Mohri, M., Silberman, N., and Walker, D. Efficient large-scale distributed training of conditional maximum entropy models. In *NIPS*, 2009.

Ng, A.Y. Feature selection, l_1 vs. l_2 regularization and rotational invariance. In *ICML*, 2004.

Shalev-Shwartz, S. and Tewari, A. Stochastic methods for ℓ_1 regularized loss minimization. In *ICML*, 2009.

Strang, G. *Linear Algebra and Its Applications*. Harcourt Brace Jovanovich, 3rd edition, 1988.

Tibshirani, R. Regression shrinkage and selection via the lasso. *J. Royal Statistical Society*, 58(1):267–288, 1996.

Tsitsiklis, J. N., Bertsekas, D. P., and Athans, M. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. 31(9):803–812, 1986.

van den Berg, E., Friedlander, M.P., Hennenfent, G., Herrmann, F., Saab, R., and Yılmaz, O. Sparco: A testing framework for sparse reconstruction. *ACM Transactions on Mathematical Software*, 35(4):1–16, 2009.

Wen, Z., Yin, W. Goldfarb, D., and Zhang, Y. A fast algorithm for sparse reconstruction based on shrinkage, subspace optimization and continuation. *SIAM Journal on Scientific Computing*, 32(4):1832–1857, 2010.

Wright, S.J., Nowak, D.R., and Figueiredo, M.A.T. Sparse reconstruction by separable approximation. *IEEE Trans. on Signal Processing*, 57(7):2479–2493, 2009.

Wulf, W.A. and McKee, S.A. Hitting the memory wall: Implications of the obvious. *ACM SIGARCH Computer Architecture News*, 23(1):20–24, 1995.

Yuan, G. X., Chang, K. W., Hsieh, C. J., and Lin, C. J. A comparison of optimization methods and software for large-scale l1-reg. linear classification. *JMLR*, 11:3183–3234, 2010.

Zinkevich, M., Weimer, M., Smola, A.J., and Li, L. Parallelized stochastic gradient descent. In *NIPS*, 2010.