BLITZ: A Principled Meta-Algorithm for Scaling Sparse Optimization

Tyler B. Johnson TBJOHNS@WASHINGTON.EDU
Carlos Guestrin GUESTRIN@CS.WASHINGTON.EDU
University of Washington, Seattle, WA 98195, USA

Abstract

By reducing optimization to a sequence of small subproblems, working set methods achieve fast convergence times for many challenging problems. Despite excellent performance, theoretical understanding of working sets is limited, and implementations often resort to heuristics to determine subproblem size, makeup, and stopping criteria. We propose BLITZ, a fast working set algorithm accompanied by useful guarantees. Making no assumptions on data, our theory relates subproblem size to progress toward convergence. This result motivates methods for optimizing algorithmic parameters and discarding irrelevant variables as iterations progress. Applied to ℓ_1-regularized learning, BLITZ convincingly outperforms existing solvers in sequential, limited-memory, and distributed settings. BLITZ is not specific to ℓ_1-regularized learning, making the algorithm relevant to many applications involving sparsity or constraints.

1. Introduction

With user-specific features for recommendation, n-gram phrases in text, or high-order transformations for feature engineering, many learning problems involve large numbers of features. In these cases, ℓ_1 regularization is a popular tool, as it biases learning toward sparse solutions. Sparsity offers many advantages, including reduced resources needed at test time, more interpretable models, and statistical efficiency, as the feature space may increase exponentially with sample size (Ng, 2004; Wainwright, 2009).

Unfortunately, convergence times for ℓ_1-regularized loss minimization tend to grow linearly with the number of features. For faster solutions, recent works have considered parallel algorithms (Boyd et al., 2011; Bradley et al., 2011; Fercoq & Richtárik, 2013). Despite parallel speedups, these algorithms in their basic form share a significant inefficiency: equal priority is assigned to all features. Due to sparsity, most features are instead irrelevant to the solution!

We propose BLITZ, a general optimization algorithm that prioritizes resources on important parts of the problem. For ℓ_1-regularized learning, BLITZ solves a sequence of subproblems restricted to small subsets of features using an existing solver, converging quickly to the original problem's solution. Known as a *working set* method, this concept is not new. GLMNET (Friedman et al., 2010) and LIBLINEAR (Yuan et al., 2012), two libraries for ℓ_1-regularized learning, prioritize computation with working set heuristics. More broadly, working sets have been applied successfully to a diverse set of optimization problems involving sparsity or constraints; see Fan et al. (2005), Tsochantaridis et al. (2005), and Kim & Park (2008) as examples.

Given the practical success of working set methods, theoretical understanding of these algorithms is surprisingly limited. How to choose a subproblem, how large it should be, and when it should terminate are questions inadequately answered by existing theory. We present novel analysis to offer such perspective. Without assumptions on data, our theory explains how to choose working sets to guarantee a desired amount of progress toward convergence. This motivates methods for eliminating irrelevant variables and optimizing algorithmic parameters, making BLITZ's choices of subproblem size, variables, and stopping criteria more principled and robust than previous approaches allow.

In practice, our theoretical insights lead to very fast convergence times for ℓ_1-regularized learning. In the sequential setting, BLITZ outperforms solvers such as GLMNET and LIBLINEAR, making BLITZ one of the fastest algorithms for high dimensional lasso and sparse logistic regression on a single machine. We then show additional gains for BLITZ in limited-memory and distributed regimes. By considering data in subsets, BLITZ prioritizes not only computation but also memory and bandwidth usage, directly targeting I/O and communication bottlenecks for problems at scale.

Importantly, BLITZ directly extends to objectives other

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

Algorithm 1 Common Working Set Algorithm

initialize $\mathbf{x}_0 \in \mathbb{R}^n$
for $t = 1, 2, \ldots$ **until** converged **do**
  Choose $\tau_t \in \mathbb{R}$
  $\mathcal{C}_t \leftarrow \{h_j : h_j(\mathbf{x}_{t-1}) \geq \tau_t \vee h_j(\mathbf{x}_{t-1}) = 0\}$
  $\mathbf{x}_t \leftarrow \operatorname{argmin} f(\mathbf{x})$ s.t. $h_j(\mathbf{x}) \leq 0$ for all $h_j \in \mathcal{C}_t$
end for
return $\mathbf{x}$

than ℓ_1-regularized loss minimization. Given the performance of BLITZ for this well-studied application, an intriguing open question is whether similar performance is achievable for additional objectives.

In summary of our contributions, we propose BLITZ, a working set algorithm that:

- Selects theoretically justified subproblems to maximize guaranteed progress toward convergence.
- Applies theoretical analysis to automatically tune algorithmic parameters and discard irrelevant constraints as the algorithm runs.
- Achieves very fast convergence times when applied to ℓ_1-regularized learning in a variety of settings.
- Provides a novel proof path for analyzing working set methods for sparse or constrained optimization.

2. The BLITZ Algorithm

In this section, we introduce BLITZ, including convergence analysis and numerical experiments examining our bounds.

2.1. Problem Formulation

We consider the convex problem

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}) \\ \text{s.t.} & h_j(\mathbf{x}) \leq 0 \quad j = 1, \ldots, m\,, \end{array} \tag{P1}$$

where $\mathbf{x} \in \mathbb{R}^n$, and h_j is convex for all j. We assume f is γ-strongly convex, and we denote (P1)'s solution by $\mathbf{x}^\star$. We define the feasible region

$$\mathcal{D} = \{\mathbf{x} : h_j(\mathbf{x}) \leq 0 \text{ for all } j = 1, \ldots, m\}\,. \tag{1}$$

We focus on instances of (P1) with large m. While not obvious, many unconstrained problems involving sparsity are instances of (P1), as sparsity often appears as constraints in a problem's dual (see Section 3 or Bach et al. (2012)).

Define the set of active constraints at $\mathbf{x}^\star$:

$$\mathcal{C}^\star = \{h_j : h_j(\mathbf{x}^\star) = 0\}\,. \tag{2}$$

In addition to (P1), $\mathbf{x}^\star$ solves the modified problem

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}) \\ \text{s.t.} & h_j(\mathbf{x}) \leq 0 \quad \text{for all } h_j \in \mathcal{C}^\star\,. \end{array} \tag{P2}$$

Algorithm 2 BLITZ

initialize $\mathbf{x}_0 \leftarrow \operatorname{argmin} f(\mathbf{x})$ **and** $\mathbf{y}_0 \in \mathcal{D}$
for $t = 1, 2, \ldots$ **until** converged **do**
  # Compute extreme feasible point on segment $[\mathbf{x}, \mathbf{y}]$:
  $\alpha_t \leftarrow \max\{\alpha \in [0, 1] : \alpha\mathbf{x}_{t-1} + (1-\alpha)\mathbf{y}_{t-1} \in \mathcal{D}\}$
  $\mathbf{y}_t \leftarrow \alpha_t\mathbf{x}_{t-1} + (1-\alpha_t)\mathbf{y}_{t-1}$
  # Select constraints with boundaries close to $\mathbf{y}$:
  Choose $\tau_t > 0$
  $\mathcal{C}_t \leftarrow \{h_j : \operatorname{dist}(h_j, \mathbf{y}_t) \leq \tau_t \vee h_j(\mathbf{x}_{t-1}) = 0\}$
  # Solve subproblem subject to selected constraints:
  $\mathbf{x}_t \leftarrow \operatorname{argmin} f(\mathbf{x})$ s.t. $h_j(\mathbf{x}) \leq 0$ for all $h_j \in \mathcal{C}_t$
end for
return $\mathbf{x}$

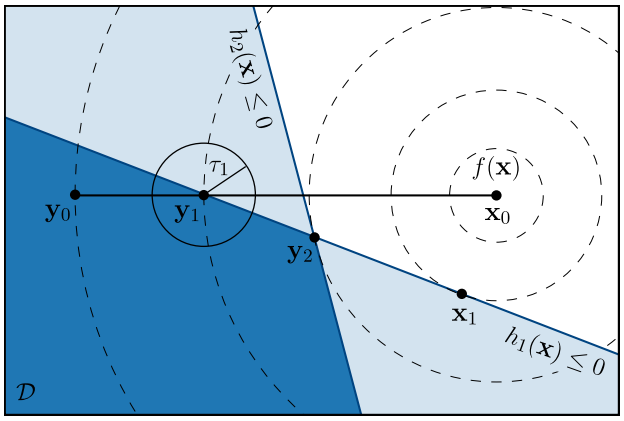


Figure 1. **BLITZ Illustration.** At iteration 1, $\mathcal{C} = \{h_1\}$. At iteration 2, $\mathcal{C}$ will include both h_1 and h_2. $\mathbf{y} \leftarrow \alpha\mathbf{x} + (1-\alpha)\mathbf{y}$ updates $\mathbf{y}$ to be the extreme feasible point on segment $[\mathbf{x}, \mathbf{y}]$.

In other words, constraints h_j for which $h_j \notin \mathcal{C}^\star$ have *no effect* on $\mathbf{x}^\star$. Often when m is large, $|\mathcal{C}^\star| \ll m$. Given $\mathcal{C}^\star$, (P1) could be solved extremely efficiently by solving (P2).

Since $\mathcal{C}^\star$ is unknown, algorithms known as *working set* algorithms instead solve (P1) by minimizing f subject to a sequence of small constraint sets $\mathcal{C}_1, \mathcal{C}_2, \ldots$ until $\mathcal{C}_T \supseteq \mathcal{C}^\star$ at which point the algorithm converges. Algorithm 1 is a simple working set method. At each iteration, $\mathcal{C}$ includes constraints active or most violated at the previous subproblem solution $\mathbf{x}$. (Note constraints may later exit $\mathcal{C}$.) While effective in practice, except for guaranteed convergence, we know of no theoretical guarantees for Algorithm 1. Improving upon Algorithm 1 in both theory and practice is an important problem this work begins to address.

2.2. BLITZ Algorithm Overview

BLITZ is defined in Algorithm 2. $\mathbf{x}$ is initialized as the unconstrained minimizer of f (unique due to strong convexity), while $\mathbf{y}$ is a point in $\mathcal{D}$. We update $\mathbf{y}$ via $\mathbf{y} \leftarrow \alpha\mathbf{x} + (1-\alpha)\mathbf{y}$, where α is the largest coefficient in

$[0, 1]$ such that $\mathbf{y}$ remains in $\mathcal{D}$. Constraints are prioritized according to the Euclidean distance

$$\text{dist}(h_j, \mathbf{y}) = \inf_{\mathbf{z}\,:\,h_j(\mathbf{z})=0} \|\mathbf{z} - \mathbf{y}\|_2 \,, \tag{3}$$

where constraints with boundaries closest to $\mathbf{y}$ receive highest priority. Often (3) can be computed in closed form (and often lower bounded for more complex h_j), and we include examples for doing so in supplementary material. A constraint h_j is included in the working set $\mathcal{C}$ if (i.) $\text{dist}(h_j, \mathbf{y})$ is less than a threshold τ, or (ii.) $h_j(\mathbf{x}) = 0$, meaning h_j is active at $\mathbf{x}$. τ controls the size of each subproblem. Upon determining $\mathcal{C}$, $\mathbf{x}$ is set to the minimizer of f subject only to constraints in $\mathcal{C}$. BLITZ reaches optimality when $\mathbf{x}$ no longer violates any constraints.

Before considering analysis, we can observe two intuitive advantages Algorithm 2 has over Algorithm 1:

- *Scale invariance:* Consider $h_j(\mathbf{x}) = \sum_i x_i$. In this case, h_j and $h_k = 100h_j$ are effectively the same constraint. However, in Algorithm 1, h_k may be included in $\mathcal{C}$ when h_j is not. BLITZ is invariant to this scaling.
- *Feasibility regularization:* BLITZ chooses constraints $\mathcal{C}$ that are close to a feasible point $\mathbf{y}$ or tight at $\mathbf{x}$. This ensures both $f(\mathbf{y})$ decreases and $f(\mathbf{x})$ increases during an iteration. Algorithm 1 chooses constraints that are active or most violated by $\mathbf{x}$, which only ensures $f(\mathbf{x})$ increases. By using $\mathbf{y}$ to choose $\mathcal{C}$, BLITZ compensates for constraints that are greatly violated by $\mathbf{x}$.

2.3. Convergence Analysis

We now analyze the convergence of BLITZ. For now, we assume each iteration's subproblem is solved exactly. All proofs are provided in supplementary material.

For all iterations t, since $\mathbf{y}_t \in \mathcal{D}$ and $\mathbf{x}_t$ minimizes f subject to a subset of constraints, we have

$$f(\mathbf{x}_t) \leq f(\mathbf{x}^\star) \leq f(\mathbf{y}_t) \,. \tag{4}$$

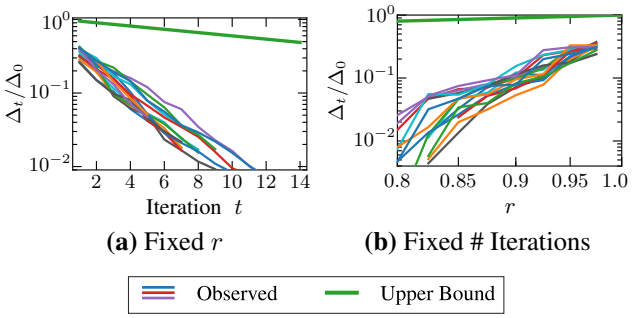
Thus, we may define an optimality gap

$$\Delta_t = f(\mathbf{y}_t) - f(\mathbf{x}_t) \geq f(\mathbf{y}_t) - f(\mathbf{x}^\star) \,. \tag{5}$$

A strength of BLITZ is that both $f(\mathbf{y}_t)$ and $f(\mathbf{x}_t)$ converge monotonically to $f(\mathbf{x}^\star)$. At each iteration, substantial improvement must be made in $f(\mathbf{y}_t)$, $f(\mathbf{x}_t)$, or both. This is the intuition of our first theorem:

Theorem 2.1 (Convergence Progress at Iteration t)**.** *Let Δ_t and Δ_{t+1} be the optimality gaps after iterations t and $t+1$ of Algorithm 2. Then for all $t \geq 1$ if the algorithm does not converge at iteration $t+1$, we have*

$$\Delta_{t+1} \leq \Delta_t - \left(\tfrac{\gamma}{2}\tau_t^2\Delta_t^2\right)^{1/3} \,. \tag{6}$$



(a) Fixed r **(b)** Fixed # Iterations

Figure 2. **Theory vs. Practice. (a)** For $r = 0.95$, 15 trials of observed optimality gap and bound (Corollary 2.2) vs. iteration. **(b)** After 2 iterations, optimality gap and bound (Corollary 2.2) vs. decrease ratio r. Convergence is faster than theory guarantees, but theory and experiments agree on the scaling of τ and Δ (plotted appropriately, trends are approximately linear).

If τ is held constant for all t, Algorithm 2 converges in a fixed number of subproblems. In practice, τ should decrease over time to ensure $|\mathcal{C}|$ remains small. The following corollary suggests a scaling of τ for fast convergence:

Corollary 2.2 (Linear Convergence)**.** *For $t \geq 1$, define*

$$\Delta_t' = f(\mathbf{y}_t) - f(\mathbf{x}_{t-1}) \,, \tag{7}$$

and suppose we run Algorithm 2 choosing τ_t as

$$\tau_t = \sqrt{\tfrac{2}{\gamma}(1-r)^3\Delta_t'} \tag{8}$$

for some $r \in [0, 1)$. Then for $t \geq 1$, we have

$$f(\mathbf{y}_t) - f(\mathbf{x}^\star) \leq r^{t-1}\Delta_0 \,. \tag{9}$$

Another consequence of Theorem 2.1 is a method for identifying constraints guaranteed to be inactive at $\mathbf{x}^\star$. This is similar to *prescreening*, a useful preprocessing step that eliminates irrelevant constraints for particular instances of (P1) (Ghaoui et al., 2012; Liu et al., 2014). Finding τ_t such that $\Delta_t \leq 0$ in (6), we arrive at the following corollary:

Corollary 2.3 (Constraint Elimination)**.** *For $t \geq 1$, define Δ_t' as in (7). If*

$$\text{dist}(h_j, \mathbf{y}_t) > \sqrt{\tfrac{2}{\gamma}\Delta_t'} \,, \tag{10}$$

then $h_j(\mathbf{x}^\star) < 0$, and h_j may be eliminated from (P1).

Compared to prescreening, Corollary 2.3 is more general and can be applied at *any* iteration of BLITZ; however, fewer constraints may be discarded initially. Elaboration on this topic is included in supplementary material.

2.4. Experiments with Bounds

To examine our bounds numerically, we instantiate (P1) as

$$\begin{array}{ll} \underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} & \|\mathbf{x} - \mathbf{b}\|_2^2 \\ \text{s.t.} & \left|\mathbf{A}_j^T\mathbf{x}\right| \leq \lambda \quad j = 1, \ldots, m \,. \end{array} \tag{P3}$$

Table 1. **Summary of Quantities for ℓ_1-Regularized Learning.** Table includes loss ϕ_i, convex conjugate ϕ_i^*, primal-dual mapping p, and smoothness constant L for lasso and logistic regression.

LOSS	$\phi_i(\mathbf{a}_i^T\mathbf{w})$	$\phi_i^*(x_i)$	$[p(\mathbf{A}\mathbf{w}^\star, \mathbf{b})]_i$	L
SQUARED	$\frac{1}{2}(\mathbf{a}_i^T\mathbf{w} - b_i)^2$	$\frac{1}{2}(b_i + x_i)^2 - \frac{1}{2}b_i^2$	$\mathbf{a}_i^T\mathbf{w}^\star - b_i$	1
LOGISTIC	$\log\left(1 + \exp(-b_i\mathbf{a}_i^T\mathbf{w})\right)$	$-\frac{x_i}{b_i}\log(-\frac{x_i}{b_i}) + (1 + \frac{x_i}{b_i})\log(1 + \frac{x_i}{b_i})$	$\frac{-b_i \exp(-b_i\mathbf{a}_i^T\mathbf{w}^\star)}{1+\exp(-b_i\mathbf{a}_i^T\mathbf{w}^\star)}$	$\frac{1}{4}$

(Later we will see (P3) is dual to the lasso.) We let $m = 10{,}000$ and $n = 100$. Elements of $\mathbf{b}$ and $\mathbf{A}_j$ are drawn i.i.d. from $\mathcal{N}(0, 1)$. We set $\lambda = \frac{3}{10}\max_j \left|\mathbf{A}_j^T\mathbf{b}\right|$, resulting in approximately 30 active constraints at $\mathbf{x}^\star$.

In Figure 2, we compare results solving (P3) with BLITZ to our worst-case bounds. Figure 2(a) plots convergence vs. iteration choosing τ with (8) and $r = 0.95$. Figure 2(b) plots optimality gaps after 2 iterations using a range of r values. Each plot aggregates 15 problem instances. The solid green line is our analytical bound. Axes are scaled so that the bound displays as a line.

From Figure 2, we see that while convergence is faster in practice than our bounds guarantee, theory and practice agree well on the scaling of τ and Δ.

3. Application: ℓ_1-Regularized Learning

We now apply BLITZ to ℓ_1-regularized optimization. This class of problems is widely used for supervised learning, compressed sensing, and algorithms for more complex problems in which ℓ_1 penalties appear in subproblems.

3.1. ℓ_1-Regularized Loss Minimization

We consider problems for which a feature vector $\mathbf{a}_i \in \mathbb{R}^m$ is used to predict a label $b_i \in \mathcal{B}$. Our prediction function is parameterized by a vector $\mathbf{w}^\star \in \mathbb{R}^m$, which is computed by maximizing an ℓ_1-regularized likelihood function over a set of n training examples $\{(\mathbf{a}_1, b_1), \ldots, (\mathbf{a}_n, b_n)\}$:

$$\underset{\mathbf{w}\in\mathbb{R}^m}{\text{maximize}}\; g(\mathbf{w}) = -\sum_{i=1}^{n} \phi_i(\mathbf{a}_i^T\mathbf{w}) - \lambda \left\|\mathbf{w}\right\|_1 . \qquad \text{(P4)}$$

Above $\phi_i : \mathbb{R} \to \mathbb{R}_{\geq 0}$ is a convex loss function parameterized by b_i. $\lambda > 0$ is a tuning parameter. For large enough λ, many values of $\mathbf{w}^\star$ are exactly zero. We let $\mathbf{A} \in \mathbb{R}^{n\times m}$ denote the design matrix, its ith row $\mathbf{a}_i$ and jth column $\mathbf{A}_j$, while $\mathbf{b} \in \mathcal{B}^n$ denotes a labels vector with ith element b_i.

We focus on two popular forms of (P4): the lasso (Tibshirani, 1996), for which $\mathcal{B} = \mathbb{R}$ and

$$g(\mathbf{w}) = -\tfrac{1}{2}\left\|\mathbf{A}\mathbf{w} - \mathbf{b}\right\|_2^2 - \lambda\left\|\mathbf{w}\right\|_1 , \qquad (11)$$

as well as sparse logistic regression (Ng, 2004), for which $\mathcal{B} = [-1, 1]$ and

$$g(\mathbf{w}) = -\sum_{i=1}^{n} \log\left(1 + \exp(-b_i\mathbf{a}_i^T\mathbf{w})\right) - \lambda\left\|\mathbf{w}\right\|_1 . \qquad (12)$$

For arbitrary loss ϕ_i, we require a single assumption:

Assumption 3.1 (Smooth Loss)**.** *The derivative ϕ_i' exists and is Lipschitz continuous with constant L:*

$$|\phi_i'(x) - \phi_i'(y)| \leq L|x - y| \quad \text{for all} \quad x, y \in \mathbb{R} . \qquad (13)$$

3.2. ℓ_1 Duality

To solve (P4) with BLITZ, we transform (P4) into its dual:

$$\begin{array}{ll} \underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} & \sum_{i=1}^n \phi_i^*(x_i) \\ \text{s.t.} & \left|\mathbf{A}_j^T\mathbf{x}\right| \leq \lambda \qquad j = 1, \ldots, m . \end{array} \qquad \text{(P5)}$$

Here ϕ_i^* is the convex conjugate of ϕ_i. $f(\mathbf{x}) = \sum_i \phi^*(x_i)$ is strongly convex due to the following proposition:

Proposition 3.2 (Strong Convexity of ℓ_1 Dual)**.** *Given Assumption 3.1, $f(\mathbf{x})$ is strongly convex with parameter $\frac{1}{L}$.*

Strong duality holds for this problem ($f(\mathbf{x}^\star) = g(\mathbf{w}^\star)$), and there exists a mapping p between optimal variables:

$$\mathbf{x}^\star = p(\mathbf{A}\mathbf{w}^\star, \mathbf{b}) . \qquad (14)$$

Table 1 summarizes relevant quantities for (P4) and (P5). Derivations are included in supplementary material.

3.3. Partial Subproblem Convergence

(P5) can be solved naturally with BLITZ. Minimizing (P5) subject to a subset of constraints corresponds to maximizing (P4) over a subset of variables, prioritizing resources on important features. However, Algorithm 2 requires exact subproblem solutions, which is impractical. To accommodate partial solutions in our analysis, we require the subproblem solver returns a primal-dual pair $(\mathbf{x}_t, \mathbf{w}_t)$, where

$$\mathbf{x}_t = \xi_t \cdot p(\mathbf{A}\mathbf{w}_t, \mathbf{b}) , \qquad (15)$$

and ξ_t is the largest scaler in $(0, 1]$ such that $\left|\mathbf{A}_j^T\mathbf{x}_t\right| \leq \lambda$ for all constraints in $\mathcal{C}_t$. Here we must redefine Δ_t as

$$\Delta_t = f(\mathbf{y}_t) - g(\mathbf{w}_t) , \qquad (16)$$

so that Δ_t upper bounds $f(\mathbf{y}_t) - f(\mathbf{x}^\star)$ and $g(\mathbf{w}^\star) - g(\mathbf{w}_t)$ for all t. We avoid spending excessive time on subproblem t by monitoring its duality gap, terminating when

$$f(\mathbf{x}_t) - g(\mathbf{w}_t) \leq \epsilon_t \left(f(\mathbf{y}_t) - g(\mathbf{w}_t)\right) \tag{17}$$

for a tolerance $\epsilon_t \in [0, 1)$. This enables our next theorem:

Theorem 3.3 (Progress for ℓ_1 with Approximate Solver)**.** *For (P5), define Δ_t as in (16), and assume $\mathbf{x}_t$ and $\mathbf{w}_t$ satisfy (17). If $\alpha_{t+1} = 1$, assume $g(\mathbf{w}_{t+1}) \geq g(\mathbf{w}_t)$. If $\alpha_{t+1} < 1$, let h_j be the (possibly non-unique) constraint such that $h_j(\mathbf{x}_t) > 0$ and $h_j(\mathbf{y}_{t+1}) = 0$ and assume $g(\mathbf{w}_{t+1}) \geq \max_\delta g(\mathbf{w}_t + \delta \mathbf{e}_j)$. Then for $t \geq 1$, we have*

$$\Delta_{t+1} \leq \max\left\{\Delta_t - \left(\tfrac{1}{2L}(1-\epsilon_t)^2\tau_t^2\Delta_t^2\right)^{1/3}, \epsilon_t\Delta_t\right\}. \tag{18}$$

Note that when $\epsilon_t = 0$, we recover Theorem 2.1. The technical condition $g(\mathbf{w}_{t+1}) \geq \max_\delta g(\mathbf{w}_t + \delta\mathbf{e}_j)$ can easily be satisfied with one coordinate descent update of w_j.

3.4. Optimizing Algorithmic Parameters

The performance of working set algorithms is sensitive to subproblem size and stopping criteria. We apply Theorem 3.3 to optimize τ and ϵ at runtime. This procedure is not meant to be exact, rather to provide BLITZ with a basic mechanism for adjusting these parameters. We model the duration of iteration t as $T_\alpha + T_{\text{solve-}t}(\tau, \epsilon)$, where

$$T_\alpha = C_\alpha\,, \quad T_{\text{solve-}t}(\tau, \epsilon) = C_{\text{solve}}\frac{\text{NNZ}(\tau, t)}{\epsilon}\,. \tag{19}$$

Above T_α is the time to compute α. $T_{\text{solve-}t}(\tau, \epsilon)$ estimates the time to solve the subproblem, increasing proportional to the number of nonzero elements in columns $\mathbf{A}_j$ for which $h_j \in \mathcal{C}_t$ and inversely proportional to ϵ. C_α and C_{solve} are constants, which are computed using runtime data by solving for C_α and C_{solve} in (19) after each iteration and taking median values over this history. Applying Theorem 3.3, we model convergence progress as

$$\hat{\Delta}_{t+1}(\tau, \epsilon) = \max\left\{\Delta_t' - C_{\text{P}}((1-\epsilon)\tau\Delta_t')^{2/3}, \epsilon\Delta_t'\right\}. \tag{20}$$

Above, $\Delta_t' = f(\mathbf{y}_t) - g(\mathbf{w}_{t-1})$, which is used as an approximation to Δ_t since Δ_t cannot be computed before choosing τ_t. The constant C_{P} accounts for bound looseness (see Figure 2), estimated using an analogous procedure to that for C_α and C_{solve}. Finally, we choose τ_t and ϵ_t by solving

$$\tau_t, \epsilon_t = \underset{\tau,\epsilon}{\text{argmin}} \frac{\hat{\Delta}_{t+1}(\tau, \epsilon)}{\exp\left\{-C_{\text{TC}}\left[T_\alpha + T_{\text{solve-}t}(\tau, \epsilon)\right]\right\}} \tag{21}$$

approximately with grid search. The time constant C_{TC} accounts for empirical evidence that BLITZ's overall convergence rate should be closer to linear than sublinear (see Figure 4). We set C_{TC} to the ratio of elapsed time to $\log(\Delta_0/\Delta_t')$. Since $C_\alpha, C_{\text{solve}}, C_{\text{P}}$, and C_{TC} cannot be computed before the first iteration, we initialize BLITZ with a relatively small, easy subproblem (100 features in sequential setting and $\epsilon_1 = 0.5$).

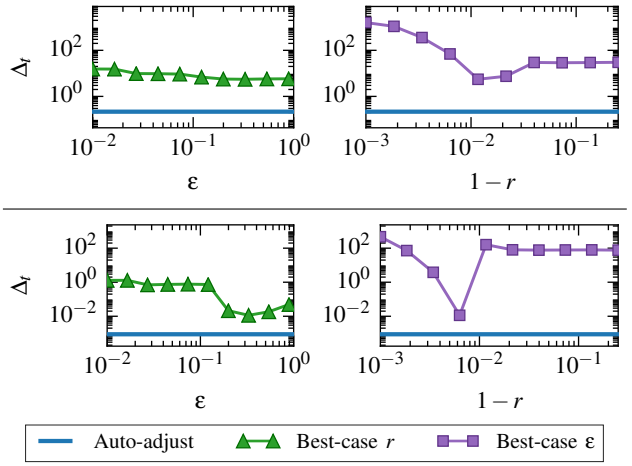


Figure 3. **Optimizing Parameters. (above)** Squared loss. **(below)** Logistic loss. For synthetic problem, BLITZ is run multiple times for 15 seconds using different ϵ and r which are fixed as Blitz runs. Plotted is resulting optimality gap. Green curve fixes best-case r and varies ϵ. Purple curve fixes best-case ϵ and varies r. Blue line is result of automatically tuning via (21). In these cases, parameter adaption is better than any fixed (r, ϵ) pair.

We experiment with this approach using two synthetic datasets, each containing 5×10^3 examples, 1×10^5 features and elements drawn i.i.d. from $\mathcal{N}(0, 1)$. We solve lasso on the first dataset using labels drawn from $\mathcal{N}(0, 1)$, and we solve logistic regression on the second dataset assigning labels ±1 with equal probability. We solve for 15 seconds using regularization $\lambda = 0.05\lambda_{\text{MAX}}$[1] and a variety of fixed r (from (8)) and ϵ values, comparing to the proposed auto-adjustment method. As Figure 3 illustrates, performance varies for choice of r and ϵ, but our tuning method makes BLITZ robust to this effect and improves upon any single choice of parameters by an order of magnitude in this case.

3.5. Sequential Comparisons

We now demonstrate the performance of BLITZ in practice. Our comparisons begin with the case that the dataset $(\mathbf{A}, \mathbf{b})$ fits in memory of a single machine. For this setting, we implement BLITZ in C++ using a coordinate descent-based proximal Newton method to solve each subproblem.

In this setting, we compare BLITZ to seven alternatives:

- PROXNEWT: Our subproblem solver for BLITZ (no

[1] λ_{MAX} is the smallest λ for which $\mathbf{w}^\star = \mathbf{0}$.

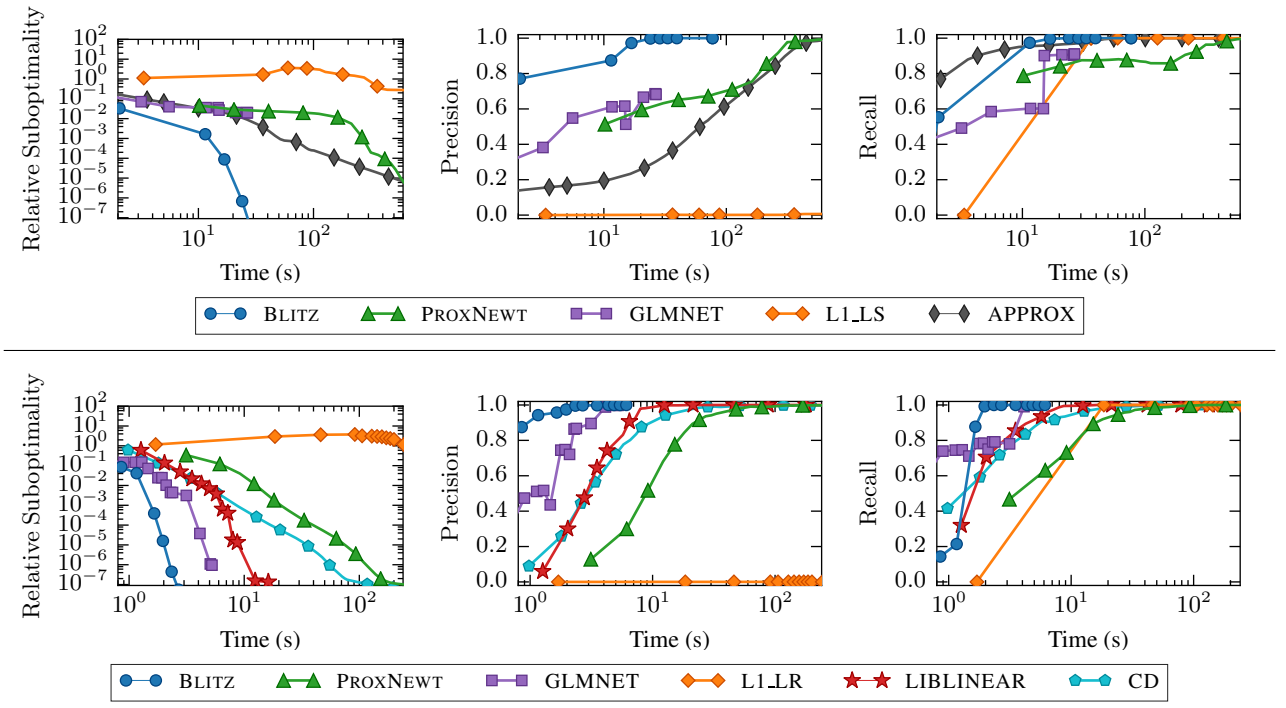


Figure 4. **Sequential Comparisons. (above)** Results from lasso problem on finance dataset. **(below)** Results from logistic regression problem on RCV1 dataset. BLITZ quickly determines the sparsity pattern of $\mathbf{w}^\star$, converging faster than alternative solvers.

prioritization of features).

- GLMNET 1.9-8 (Friedman et al., 2010): Popular R package for lasso and sparse logistic regression; implemented in Fortran; uses working set heuristics[2].
- LIBLINEAR 1.94 (Yuan et al., 2012): Widely-used C++ solver for sparse logistic regression (lasso not implemented); uses working set heuristics[3].
- L1_LS (Kim et al., 2007): Interior point method for lasso implemented in MATLAB®.
- L1_LOGREG 0.8.2 (Koh et al., 2007): Interior point method for sparse logistic regression written in C.
- APPROX (Fercoq & Richtárik, 2013): Parallel, accelerated coordinate descent for lasso; pre-computed step sizes ensure convergence; C++ implementation.
- CD: C++ implementation of coordinate descent for sparse logistic regression.

With the exception of L1_LS, each solver is compiled with version 4.8.2 of the applicable GNU C/C++/Fortran compiler and `-O3` optimization flag. Our hardware is a 64-bit machine with 2.0 GHz Intel i7-2630QM processors, 8 GB memory, and 6 MB cache. Solvers that utilize parallelism (APPROX, L1_LS, and L1_LR) use up to 8 threads.

[2]We found the performance of GLMNET depends significantly on its termination threshold—even during early iterations. We run GLMNET using only its default stopping condition.

[3] To achieve consistent solutions, we slightly modify this implementation to use an unregularized bias term.

Table 2. **Problem Instances for Sequential Comparisons.** We choose $\lambda = 0.05\lambda_{\text{MAX}}$ to select a desirable number of features ($\|\mathbf{w}^\star\|_0$ significantly smaller than $\min(n, m)$ while still resulting in a difficult problem).

DATASET	LOSS	n	m	NNZ	$\|\mathbf{w}^\star\|_0$
FINANCE	SQUARED	1.6×10^4	1.6×10^6	9.2×10^7	1419
RCV1	LOGISTIC	2.0×10^4	2.4×10^6	6.2×10^7	537

We include results for two problem instances listed in Table 2. Datasets are publicly available from LIBSVM[4]. To emphasize the high dimensional setting, we expand RCV1, including features formed by taking the element-wise product of each pair of original features, disregarding new features that contain five or fewer nonzeros. Since L1_LS and APPROX do not support an unregularized intercept term, we include this variable for logistic regression but not lasso. We standardize columns to have unit ℓ_2-norm for lasso and unit variance for logistic regression. For lasso, we standardize $\mathbf{b}$ to have zero mean and unit variance.

We quantify the performance of each solver using three metrics. The first metric measures convergence progress

[4]URL: http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/.

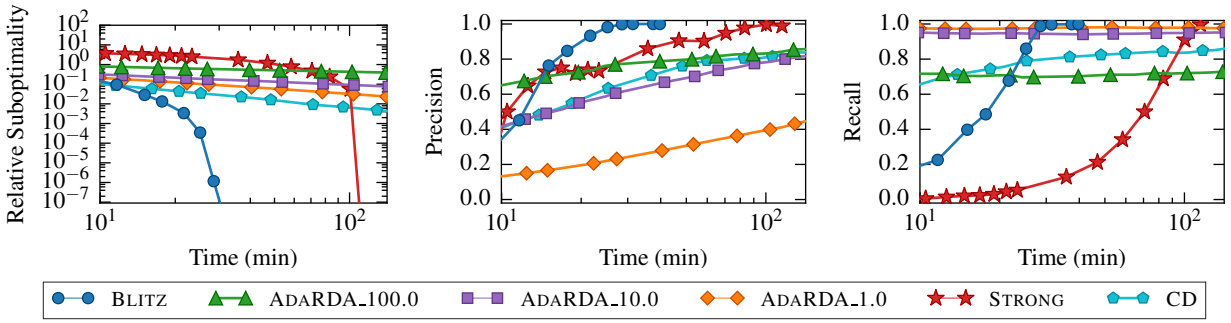


Figure 5. **Limited Memory Comparison.** Results for Webspam dataset and logistic loss. ADARDA's numeral suffix refers to the value of its step-size parameter. By efficiently prioritizing available memory, BLITZ quickly obtains an accurate solution.

vs. time in terms of relative suboptimality:

$$|g(\mathbf{w}^\star) - g(\mathbf{w}_t)| \,/\, |g(\mathbf{w}^\star)| \,. \tag{22}$$

$\mathbf{w}^\star$ is approximated as the solution returned by BLITZ after solving to machine precision. We also plot precision and recall for nonzero weight variables w_j. Define $S^\star = \{j : w_j^\star \neq 0\}$ and S_t as the analogous support set for $\mathbf{w}_t$. (For solvers that do not set values w_j to exactly 0, we take w_j to be nonzero i.f.f. $|w_j| \geq 10^{-3}$.) We measure

$$\text{Precision} = \frac{|S_t \cap S^\star|}{|S_t|}, \quad \text{and} \quad \text{Recall} = \frac{|S_t \cap S^\star|}{|S^\star|}. \tag{23}$$

Precision and recall are suitable metrics for ℓ_1-regularized learning, since ℓ_1 regularization is most prominently used for feature selection, while generalization performance can be suppressed by coefficients overly biased toward zero.

Results of our comparison are included in Figure 4. Comparing BLITZ to its subproblem solver, PROXNEWT, as well as other methods without working sets, we see prioritizing computation provides extreme gains. With 8 threads, APPROX requires at least 6 minutes to solve a lasso problem that our sequential implementation of BLITZ completes in fewer than 30 seconds. Compared to other working set algorithms (GLMNET and LIBLINEAR), we see BLITZ still can be faster. While GLMNET and LIBLINEAR are highly optimized implementations, we see precision and recall results are superior for BLITZ, suggesting computation is better-focused on relevant features.

3.6. Limited Memory Comparison

Often datasets are too large to fit in the memory of a single machine. To solve (P4), one option is to load data multiple times from disk. While disk I/O becomes a bottleneck, BLITZ can be used to prioritize memory usage.

Applying BLITZ is straightforward in this setting if the set $\{\mathbf{A}_j : w_j^\star \neq 0\}$ fits comfortably in memory. At each iteration, τ is chosen such that the resulting subproblem includes as many features as memory limitations allow. Computing this τ requires a single pass over the data. Each subproblem is then solved with (in-memory) BLITZ.

We compare this approach to three alternatives:

- ADARDA (Duchi et al., 2011): Stochastic gradient descent method with adaptive step-sizes. RDA is well-suited for ℓ_1-regularized learning (Xiao, 2010).
- STRONG (Tibshirani et al., 2012): Like BLITZ but features are prioritized according to the "Strong Rule." Regularization is initialized to λ_{MAX} and decreased at each iteration until reaching the target λ. STRONG uses (in-memory) BLITZ to solve subproblems.
- CD: A memory-limited coordinate descent implementation. $\mathbf{A}_j$ is loaded, (P4) is maximized with respect to w_j, then memory for $\mathbf{A}_j$ is deallocated.

We implement each method in C++. To enable sequential loads, training data is stored on disk in compressed row format for ADARDA and compressed column format for all other methods. Data is stored in binary format and compressed with gzip. Our hardware is a 64-bit machine with 2.60 GHz Intel i5-4278U processors and a SATA HDD that achieves read rates of 100 MB/s.

We compare algorithms using the Webspam dataset from LIBSVM and logistic loss. This dataset contains 3.5×10^5 examples, 6.8×10^5 features, and 1.3×10^9 nonzero entries. We set $\lambda = 0.01\lambda_{\text{MAX}}$, resulting in 762 selected features. We normalize features to have unit variance. Under default compression, the dataset occupies approximately 12 GB. To emphasize the limited memory setting, we allow each algorithm use of just 1 GB memory.

Results of this experiment are included in Figure 5. BLITZ and STRONG greatly outperform alternative solvers that do not use more of the available memory. Clearly for some large problems, one need not settle for approximate solutions when the solution is sparse.

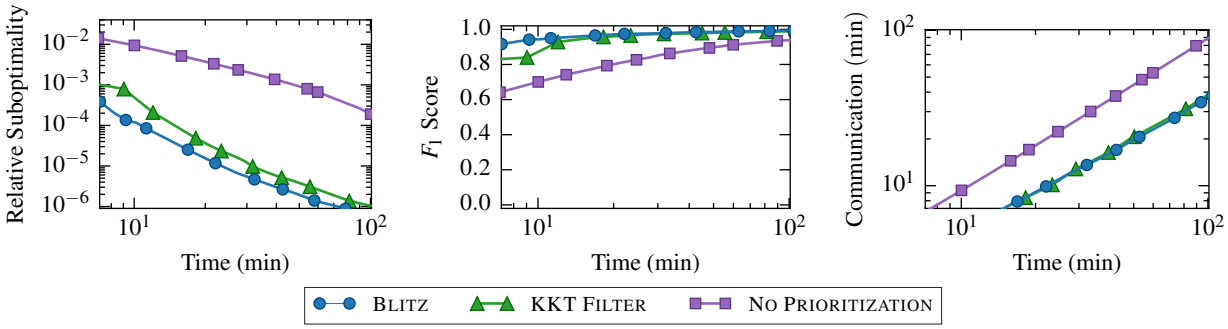


Figure 6. **Distributed Comparison.** Results for CTR dataset with logistic loss. BLITZ and the KKT filter approach prioritize communication, greatly improving convergence times. $F_1 = 2 \cdot \text{Precision} \cdot \text{Recall} \,/\, (\text{Precision} + \text{Recall})$ for selected features.

3.7. Distributed Comparison

For the largest problems, it is necessary to distribute data among many machines. Often distributed solvers for (P4) partition data by training examples and communicate gradient vectors of length m, the number of features, at each iteration. With m exceeding one billion in some industrial applications (Chen et al., 2014), communication becomes a bottleneck to optimization. In this setting, BLITZ can be used to drastically decrease the communication needed.

As a concrete example, consider a bulk synchronous proximal gradient descent implementation with data partitioned by examples. During an iteration, each node computes the gradient contribution of its local partition, and an $\mathcal{O}(m)$ reduce operation aggregates these contributions to determine the global gradient. By solving subproblems with only $|\mathcal{C}|$ features, BLITZ reduces the time complexity of this reduce operation to $\mathcal{O}(|\mathcal{C}|)$ per subproblem iteration. A "KKT filter" heuristic with similar motivation was recently proposed by Li et al. (2014). Communication of gradient values that are small in magnitude is prolonged until later iterations, which greatly improves convergence times.

We compare BLITZ with the KKT filter approach and a proximal gradient method with no prioritization of communication. The underlying solver for each method is an identical proximal gradient descent implementation which uses backtracking as detailed by Beck & Teboulle (2009) to ensure convergence. We implement this method in C++ using Rabit[5], a reliable all-reduce communication library. The KKT filtering step is directly translated from the implementation of Li et al. (2014).

We compare methods using sparse logistic regression and the Criteo click-through rate dataset[6]. This dataset has 4.6×10^7 examples, 3.3×10^7 features, and 1.5×10^9 nonzero entries. We normalize features to have unit variance. Using $\lambda = 0.01\lambda_{\text{MAX}}$, the solution contains 5717 nonzero elements. We use 64 workers on 16 servers connected with 1 Gb/s networking. We approximate the optimal solution by running BLITZ for 200 minutes.

Results of this experiment are provided in Figure 6. By prioritizing communication, BLITZ and the KTT filtering method converge an order of magnitude faster than the naïve proximal gradient algorithm.

4. Discussion

ℓ_1-regularized learning owes its popularity to the practical and statistical benefits of sparsity. In this work, we propose BLITZ, a method for exploiting sparsity during *optimization*. Unlike previous working set heuristics, BLITZ enables theoretically justified methods for choosing the contents, size, and stopping criteria of subproblems.

In several settings, BLITZ converges extremely quickly for ℓ_1-regularized learning. Given such performance, it is important to consider additional problems for which BLITZ can work well. As a beginning, the analogy between constraint elimination (Corollary 2.3) and screening methods suggest BLITZ may work well for other applications for which screening has found traction (for example Wang et al. (2014)). It would also be interesting to consider more challenging objectives, including graphical lasso and problems with trace or total variation norms.

Another remaining challenge is to apply BLITZ to problems for which the constraint space is intractably large and cannot be enumerated. This includes structured prediction (Tsochantaridis et al., 2005) and submodular minimization (Fujishige & Isotani, 2011). We view BLITZ as a very promising starting point for future work on these problems and large-scale machine learning in general.

[5]URL: https://github.com/tqchen/rabit.
[6]URL: http://labs.criteo.com/downloads.

Acknowledgments

The authors would like to thank the reviewers for their thoughtful suggestions as well as Joseph Bradley for his feedback on early versions of BLITZ. This work is supported in part by PECASE N00014-13-1-0023, NSF IIS-1258741, and the TerraSwarm Research Center 00008169.

References

Bach, F., Jenatton, R., Mairal, J., and Obozinski, G. Optimization with sparsity-inducing penalties. *Foundations and Trends in Machine Learning*, 4(1):1–106, 2012.

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

Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

Bradley, J. K., Kyrola, A., Bickson, D., and Guestrin, C. Parallel coordinate descent for L_1-regularized loss minimization. In *International Conference on Machine Learning*, 2011.

Chen, W., Wang, Z., and Zhou, J. Large-scale L-BFGS using MapReduce. In *Advances in Neural Information Processing Systems 27*, 2014.

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

Fan, R. E., Chen, P. H., and Lin, C. J. Working set selection using second order information for training support vector machines. *Journal of Machine Learning Research*, 6: 1889–1918, 2005.

Fercoq, O. and Richtárik, P. Accelerated, parallel and proximal coordinate descent. Technical Report arXiv:1312.5799, 2013.

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.

Fujishige, S. and Isotani, S. A submodular function minimization algorithm based on the minimum-norm base. *Pacific Journal of Optimization*, 7:3–17, 2011.

Ghaoui, L. E., Viallon, V., and Rabbani, T. Safe feature elimination for the lasso and sparse supervised learning problems. *Pacific Journal of Optimization*, 8(4):667–698, 2012.

Kim, H. and Park, H. Nonnegative matrix factorization based on alternating nonnegativity constrained least squares and active set method. *SIAM Journal on Matrix Analysis and Applications*, 30(2):713–730, 2008.

Kim, S. J., Koh, K., Lustig, M., Boyd, S., and Gorinevsky, D. An interior-point method for large-scale ℓ_1-regularized least squares. *IEEE Journal on Selected Topics in Signal Processing*, 1(4):606–617, 2007.

Koh, K., Kim, S. J., and Boyd, S. An interior-point method for large-scale ℓ_1-regularized logistic regression. *Journal of Machine Learning Research*, 8:519–1555, 2007.

Li, M., Smola, A., and Andersen, D. G. Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems 27*, 2014.

Liu, J., Zhao, Z., Wang, J., and Ye, J. Safe screening with variational inequalities and its application to lasso. In *International Conference on Machine Learning*, 2014.

Ng, A. Y. Feature selection, L_1 vs. L_2 regularization, and rotational invariance. In *International Conference on Machine Learning*, 2004.

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

Tibshirani, R., Bien, J., Friedman, J., Hastie, T., Simon, N., Taylor, J., and Tibshirani, R. J. Strong rules for discarding predictors in lasso-type problems. *Journal of the Royal Statistical Society, Series B*, 74(2):245–266, 2012.

Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005.

Wainwright, M. J. Sharp thresholds for high-dimensional and noisy sparsity recovery using ℓ_1-constrained quadratic programming (Lasso). *IEEE Transactions on Information Theory*, 55(5):2183–2202, 2009.

Wang, J., Wonka, P., and Ye, J. Scaling SVM and least absolute deviations via exact data reduction. In *International Conference on Machine Learning*, 2014.

Xiao, L. Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research*, 11:2543–2596, 2010.

Yuan, G. X., Ho, C. H., and Lin, C. J. An improved GLMNET for L1-regularized logistic regression. *Journal of Machine Learning Research*, 13:1999–2030, 2012.