BLITZ: A Principled Meta-Algorithm for Scaling Sparse Optimization

Tyler B. Johnson  
Carlos Guestrin  
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 $\ell_1$-regularized learning, BLITZ convincingly outperforms existing solvers in sequential, limited-memory, and distributed settings. BLITZ is not specific to $\ell_1$-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, $\ell_1$ 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 $\ell_1$-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 $\ell_1$-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 $\ell_1$-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), Tschantz 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 $\ell_1$-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
Algorithm 1 Common Working Set Algorithm

initialize $x_0 \in \mathbb{R}^n$
for $t = 1, 2, \ldots$ until converged do
    Choose $\tau_t \in \mathbb{R}$
    $C_t \leftarrow \{h_j : h_j(x_{t-1}) \geq \tau_t \lor h_j(x_{t-1}) = 0\}$
    $x_t \leftarrow \text{argmin } f(x) \text{ s.t. } h_j(x) \leq 0 \text{ for all } h_j \in C_t$
end for
return $x$

Algorithm 2 BLITZ

initialize $x_0 \leftarrow \text{argmin } f(x)$ and $y_0 \in \mathcal{D}$
for $t = 1, 2, \ldots$ until converged do
    # Compute extreme feasible point on segment $[x, y]$
    $\alpha_t \leftarrow \max \{\alpha \in [0, 1] : \alpha x_{t-1} + (1 - \alpha) y_{t-1} \in \mathcal{D}\}$
    $y_t \leftarrow \alpha_t x_{t-1} + (1 - \alpha_t) y_{t-1}$
    # Select constraints with boundaries close to $y$
    Choose $\tau_t > 0$
    $C_t \leftarrow \{h_j : \text{dist}(h_j, y_t) \leq \tau_t \lor h_j(x_{t-1}) = 0\}$
    # Solve subproblem subject to selected constraints:
    $x_t \leftarrow \text{argmin } f(x) \text{ s.t. } h_j(x) \leq 0 \text{ for all } h_j \in C_t$
end for
return $x$

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

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

Since $\mathcal{C}^*$ 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 $\mathcal{C}_T \supseteq \mathcal{C}^*$ 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 $x$. (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. 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

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

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

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

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 $x^*$:

$$\mathcal{C}^* = \{h_j : h_j(x^*) = 0\}. \quad (2)$$

In addition to (P1), $x^*$ solves the modified problem

$$\begin{align*}
\text{minimize} \quad & f(x) \\
\text{s.t.} \quad & h_j(x) \leq 0 \quad \text{for all } h_j \in \mathcal{C}^*. \quad (P2)
\end{align*}$$

than $\ell_1$-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 $\ell_1$-regularized learning in a variety of settings.
- Provides a novel proof path for analyzing working set methods for sparse or constrained optimization.
[0, 1] such that \( y \) remains in \( \mathcal{D} \). Constraints are prioritized according to the Euclidean distance

\[
dist(h_j, y) = \inf_{z : h_j(z) = 0} \| z - y \|_2, \tag{3}
\]

where constraints with boundaries closest to \( y \) 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 \( C \) if (i.) \( \text{dist}(h_j, y) \) is less than a threshold \( \tau \), or (ii.) \( h_j(x) = 0 \), meaning \( h_j \) is active at \( x \). \( \tau \) controls the size of each subproblem. Upon determining \( C \), \( x \) is set to the minimizer of \( f \) subject only to constraints in \( C \). BLITZ reaches optimality when \( x \) no longer violates any constraints.

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

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

### 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 \( y_t \in \mathcal{D} \) and \( x_t \) minimizes \( f \) subject to a subset of constraints, we have

\[
f(x_t) \leq f(x^*) \leq f(y_t). \tag{4}
\]

Thus, we may define an optimality gap

\[
\Delta_t = f(y_t) - f(x_t) \geq f(y_t) - f(x^*). \tag{5}
\]

A strength of BLITZ is that both \( f(y_t) \) and \( f(x_t) \) converge monotonically to \( f(x^*) \). At each iteration, substantial improvement must be made in \( f(y_t) \), \( f(x_t) \), or both. This is the intuition of our first theorem:

**Theorem 2.1** (Convergence Progress at Iteration \( t \)). Let \( \Delta_t \) and \( \Delta_{t+1} \) 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

\[
\Delta_{t+1} \leq \Delta_t - \left( \frac{2}{3} \Delta_t^2 \right)^{1/3}. \tag{6}
\]

Another consequence of Theorem 2.1 is a method for identifying constraints guaranteed to be inactive at \( x^* \). 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 \( \tau \) such that \( \Delta_t \leq 0 \) in (6), we arrive at the following corollary:

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

\[
\Delta_t' = f(y_t) - f(x_{t-1}), \tag{7}
\]

and suppose we run Algorithm 2 choosing \( \tau_t \) as

\[
\tau_t = \sqrt{\frac{2}{3} (1 - r)^3 \Delta_t'}, \tag{8}
\]

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

\[
f(y_t) - f(x^*) \leq r^{t-1} \Delta_0. \tag{9}
\]

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

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

\[
\text{dist}(h_j, y_t) > \sqrt{\frac{2}{3} \Delta_t'}, \tag{10}
\]

then \( h_j(x^*) < 0 \), 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

\[
\minimize_{x \in \mathbb{R}^n} \| x - b \|_2 \quad \text{s.t.} \quad A_j^T x \leq \lambda \quad j = 1, \ldots, m. \tag{P3}
\]

![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 \( r \) and \( \Delta \) (plotted appropriately, trends are approximately linear).](image)
We focus on two popular forms of (P4): the lasso (Tibshirani, 2005) and sparse logistic regression (Ng, 2004), for which (P3) is dual to the lasso. We let $B = [-1, 1]$ and
\[ g(w) = -\sum_{i=1}^{n} \log (1 + \exp(-b_i a_i^T w)) - \lambda \|w\|_1. \] (12)

For arbitrary loss $\phi_i$, we require a single assumption:

**Assumption 3.1 (Smooth Loss).** The derivative $\phi_i'$ exists and is Lipschitz continuous with constant $L$:
\[ |\phi_i'(x) - \phi_i'(y)| \leq L|x - y| \quad \text{for all } x, y \in \mathbb{R}. \] (13)

### 3.2. $\ell_1$ Duality
To solve (P4) with BLITZ, we transform (P4) into its dual:

\[
\begin{align*}
\text{minimize} & \quad \sum_{i=1}^{n} \phi_i^*(x_i) \\
\text{s.t.} & \quad |A_i^T x| \leq \lambda \\
& \quad j = 1, \ldots, m. \\
\end{align*}
\] (P5)

Here $\phi_i^*$ is the convex conjugate of $\phi_i$, $f(x) = \sum_i \phi_i^*(x_i)$ is strongly convex due to the following proposition:

**Proposition 3.2 (Strong Convexity of $\ell_1$ Dual).** Given Assumption 3.1, $f(x)$ is strongly convex with parameter $\frac{1}{L}$.

Strong duality holds for this problem ($f(x^*) = g(w^*)$), and there exists a mapping $p$ between optimal variables:
\[ x^* = p(Aw^*, b). \] (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 $(x_t, w_t)$, where
\[ x_t = \xi_t \cdot p(Aw_t, b), \] (15)
and $\xi_t$ is the largest scaler in $(0, 1]$ such that $|A_i^T x_t| \leq \lambda$ for all constraints in $C_t$. Here we must redefine $\Delta_t$ as
\[ \Delta_t = f(y_t) - g(w_t), \] (16)
so that $\Delta_t$ upper bounds $f(y_t) - f(x^*)$ and $g(w^*) - g(w_t)$ for all $t$. We avoid spending excessive time on subproblem $t$ by monitoring its duality gap, terminating when 

$$f(x_t) - g(w_t) \leq \epsilon_t(f(y_t) - g(w_t))$$  \hfill (17)

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

**Theorem 3.3** (Progress for $\epsilon_t$ with Approximate Solver). For (P5), define $\Delta_t$ as in (16), and assume $x_t$ and $w_t$ satisfy (17). If $\alpha_{t+1} = 1$, assume $g(w_{t+1}) \geq g(w_t)$. If $\alpha_{t+1} < 1$, let $h_j$ be the (possibly non-unique) constraint such that $h_j(x_t) > 0$ and $h_j(y_{t+1}) = 0$ and assume $g(w_{t+1}) \geq \max_{\delta} g(w_t + \delta e_j)$. Then for $t \geq 1$, we have

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

Note that when $\epsilon_t = 0$, we recover Theorem 2.1. The technical condition $g(w_{t+1}) \geq \max\{g(w_t + \delta e_j)\}$ 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 $\tau$ and $\epsilon$ 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 $T_0 + T_{\text{solve-}}(\tau, \epsilon)$, where

$$T_0 = C_0, \quad T_{\text{solve-}}(\tau, \epsilon) = C_{\text{solve}} \frac{\text{NNZ}(\tau, t)}{\epsilon}.$$  \hfill (19)

Above $T_0$ is the time to compute $\alpha$. $T_{\text{solve-}}(\tau, \epsilon)$ estimates the time to solve the subproblem, increasing proportional to the number of nonzero elements in columns $A_j$ for which $h_j \in C$, and inversely proportional to $\epsilon$. $C_0$ and $C_{\text{solve}}$ are constants, which are computed using runtime data by solving for $C_0$ and $C_{\text{solve}}$ in (19) after each iteration and taking median values over this history. Applying Theorem 3.3, we model convergence progress as

$$\Delta_{t+1}(\tau, \epsilon) = \max\left\{\Delta_t' - C_{P}((1 - \epsilon)\tau \Delta_t')^{2/3}, \epsilon \Delta_t'\right\}.$$  \hfill (20)

Above, $\Delta_t' = f(y_t) - g(w_{t-1})$, which is used as an approximation to $\Delta_t$ since $\Delta_t$ cannot be computed before choosing $\tau_t$. The constant $C_P$ accounts for bound looseness (see Figure 2), estimated using an analogous procedure to that for $C_0$ and $C_{\text{solve}}$. Finally, we choose $\tau_t$ and $\epsilon_t$ by solving

$$\tau_t, \epsilon_t = \arg\min_{\tau, \epsilon} \frac{\Delta_{t+1}(\tau, \epsilon)}{\exp\left\{-C_{\text{TC}}[T_0 + T_{\text{solve-}}(\tau, \epsilon)]\right\}}.$$  \hfill (21)

approximately with grid search. The time constant $C_{\text{TC}}$ accounts for empirical evidence that BLITZ’s overall convergence rate should be closer to linear than sublinear (see Figure 4). We set $C_{\text{TC}}$ to the ratio of elapsed time to $\log(\Delta_0/\Delta_t)$. Since $C_0, C_{\text{solve}}, C_P$, and $C_{\text{TC}}$ cannot be computed before the first iteration, we initialize BLITZ with a relatively small, easy subproblem (100 features in sequential setting and $\epsilon_1 = 0.5$).

We experiment with this approach using two synthetic datasets, each containing $5 \times 10^4$ examples, $1 \times 10^5$ features and elements drawn i.i.d. from $N(0, 1)$. We solve lasso on the first dataset using labels drawn from $N(0, 1)$, and we solve logistic regression on the second dataset assigning labels $\pm 1$ with equal probability. We solve for 15 seconds using regularization $\lambda = 0.05\lambda_{\text{MAX}}$ and a variety of fixed $r$ (from $(8)$) and $\epsilon$ values, comparing to the proposed auto-adjustment method. As Figure 3 illustrates, performance varies for choice of $r$ and $\epsilon$, 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 $(A, b)$ 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 $\lambda_{\text{MAX}}$ is the smallest $\lambda$ for which $w^* = 0$.)}
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.
- LIBLINEAR 1.94 (Yuan et al., 2012): Widely-used C++ solver for sparse logistic regression (lasso not implemented); uses working set heuristics.
- L1_LS (Kim 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.

---

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

<table>
<thead>
<tr>
<th>DATASET</th>
<th>LOSS</th>
<th>n</th>
<th>m</th>
<th>NNZ</th>
<th>$|w^\ast|_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>FINANCE</td>
<td>SQUARED</td>
<td>1.6×10^4</td>
<td>1.6×10^6</td>
<td>9.2×10^7</td>
<td>1419</td>
</tr>
<tr>
<td>RCV1</td>
<td>LOGISTIC</td>
<td>2.0×10^4</td>
<td>2.4×10^6</td>
<td>6.2×10^7</td>
<td>537</td>
</tr>
</tbody>
</table>

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 $\ell_2$-norm for lasso and unit variance for logistic regression. For lasso, we standardize $b$ to have zero mean and unit variance.

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

---

\(^2\)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.

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 $\ell_1$-regularized learning (Xiao, 2010).
- **STRONG** (Tibshirani et al., 2012): Like BLITZ but features are prioritized according to the “Strong Rule.” Regularization is initialized to $\lambda_{\text{MAX}}$ and decreased at each iteration until reaching the target $\lambda$. **STRONG** uses (in-memory) BLITZ to solve subproblems.
- **CD**: A memory-limited coordinate descent implementation. $A_j$ is loaded, $(P4)$ is maximized with respect to $w_j$, then memory for $A_j$ 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 $3.5 \times 10^5$ examples, $6.8 \times 10^5$ features, and $1.3 \times 10^9$ 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.

---

**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 $\{A_j : w_j^* \neq 0\}$ fits comfortably in memory. At each iteration, $\tau$ is chosen such that the resulting subproblem includes as many features as memory limitations allow. Computing this $\tau$ requires a single pass over the data. Each subproblem is then solved with (in-memory) BLITZ.

We set $\lambda$ to exactly 0, $j$ fits comfortably in memory. At each iteration until reaching the target $\lambda$. **STRONG** requires a single pass over the data. Each subproblem is then solved with (in-memory) BLITZ.

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 $3.5 \times 10^5$ examples, $6.8 \times 10^5$ features, and $1.3 \times 10^9$ 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 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.
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 $O(m)$ reduce operation aggregates these contributions to determine the global gradient. By solving subproblems with only $|C|$ features, BLITZ reduces the time complexity of this reduce operation to $O(|C|)$ 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 Rabbit\(^5\), 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 $4.6 \times 10^7$ examples, $3.3 \times 10^7$ features, and $1.5 \times 10^9$ 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

$\ell_1$-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 $\ell_1$-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 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 (Tschantaridis 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


