Journal of Machine Learning Research 10 (2009) 719-743 Submitted 6/08; Revised 11/08; Published 3/09

Sparse Online Learning via Truncated Gradient

John Langford JL@YAHOO-INC.COM
Yahoo! Research
New York, NY, USA

Lihong Li LIHONG@CS.RUTGERS.EDU
Department of Computer Science
Rutgers University
Piscataway, NJ, USA

Tong Zhang[*] TONGZ@RCI.RUTGERS.EDU
Department of Statistics
Rutgers University
Piscataway, NJ, USA

Editor: Manfred Warmuth

Abstract

We propose a general method called *truncated gradient* to induce sparsity in the weights of online-learning algorithms with convex loss functions. This method has several essential properties:

1. The degree of sparsity is continuous—a parameter controls the rate of sparsification from no sparsification to total sparsification.
2. The approach is theoretically motivated, and an instance of it can be regarded as an online counterpart of the popular L_1-regularization method in the batch setting. We prove that small rates of sparsification result in only small additional regret with respect to typical online-learning guarantees.
3. The approach works well empirically.

We apply the approach to several data sets and find for data sets with large numbers of features, substantial sparsity is discoverable.

Keywords: truncated gradient, stochastic gradient descent, online learning, sparsity, regularization, Lasso

1. Introduction

We are concerned with machine learning over large data sets. As an example, the largest data set we use here has over 10^7 sparse examples and 10^9 features using about 10^{11} bytes. In this setting, many common approaches fail, simply because they cannot load the data set into memory or they are not sufficiently efficient. There are roughly two classes of approaches which can work:

1. Parallelize a batch-learning algorithm over many machines (e.g., Chu et al., 2008).

2. Stream the examples to an online-learning algorithm (e.g., Littlestone, 1988; Littlestone et al., 1995; Cesa-Bianchi et al., 1996; Kivinen and Warmuth, 1997).

∗. Partially supported by NSF grant DMS-0706805.

This paper focuses on the second approach.

Typical online-learning algorithms have at least one weight for every feature, which is too much in some applications for a couple reasons:

1. Space constraints. If the state of the online-learning algorithm overflows RAM it can not efficiently run. A similar problem occurs if the state overflows the L2 cache.

2. Test-time constraints on computation. Substantially reducing the number of features can yield substantial improvements in the computational time required to evaluate a new sample.

This paper addresses the problem of inducing sparsity in learned weights while using an online-learning algorithm. There are several ways to do this wrong for our problem. For example:

1. Simply adding L_1-regularization to the gradient of an online weight update doesn't work because gradients don't induce sparsity. The essential difficulty is that a gradient update has the form $a+b$ where a and b are two floats. Very few float pairs add to 0 (or any other default value) so there is little reason to expect a gradient update to accidentally produce sparsity.

2. Simply rounding weights to 0 is problematic because a weight may be small due to being useless or small because it has been updated only once (either at the beginning of training or because the set of features appearing is also sparse). Rounding techniques can also play havoc with standard online-learning guarantees.

3. Black-box wrapper approaches which eliminate features and test the impact of the elimination are not efficient enough. These approaches typically run an algorithm many times which is particularly undesirable with large data sets.

1.1 What Others Do

In the literature, the Lasso algorithm (Tibshirani, 1996) is commonly used to achieve sparsity for linear regression using L_1-regularization. This algorithm does not work automatically in an online fashion. There are two formulations of L_1-regularization. Consider a loss function $L(w,z_i)$ which is convex in w, where $z_i = (x_i, y_i)$ is an input/output pair. One is the *convex constraint formulation*

$$\hat{w} = \arg\min_w \sum_{i=1}^{n} L(w,z_i) \quad \text{subject to } \|w\|_1 \leq s, \tag{1}$$

where s is a tunable parameter. The other is the *soft regularization formulation*, where

$$\hat{w} = \arg\min_w \sum_{i=1}^{n} L(w,z_i) + g\|w\|_1. \tag{2}$$

With appropriately chosen g, the two formulations are equivalent. The convex constraint formulation has a simple online version using the projection idea of Zinkevich (2003), which requires the projection of weight w into an L_1-ball at every online step. This operation is difficult to implement efficiently for large-scale data with many features even if all examples have sparse features although recent progress was made (Duchi et al., 2008) to reduce the *amortized* time complexity to $O(k \log d)$, where k is the number of nonzero entries in x_i, and d is the total number of features (i.e.,

the dimension of x_i). In contrast, the soft-regularization method is efficient for a batch setting (Lee et al., 2007) so we pursue it here in an online setting where we develop an algorithm whose complexity is linear in k but independent of d; these algorithms are therefore more efficient in problems where d is prohibitively large.

More recently, Duchi and Singer (2008) propose a framework for empirical risk minimization with regularization called *Forward Looking Subgradients*, or FOLOS in short. The basic idea is to solve a regularized optimization problem after every gradient-descent step. This family of algorithms allow general convex regularization function, and reproduce a special case of the truncated gradient algorithm we will introduce in Section 3.3 (with θ set to ∞) when L_1-regularization is used.

The Forgetron algorithm (Dekel et al., 2006) is an online-learning algorithm that manages memory use. It operates by decaying the weights on previous examples and then rounding these weights to zero when they become small. The Forgetron is stated for kernelized online algorithms, while we are concerned with the simpler linear setting. When applied to a linear kernel, the Forgetron is not computationally or space competitive with approaches operating directly on feature weights.

A different, Bayesian approach to learning sparse linear classifiers is taken by Balakrishnan and Madigan (2008). Specifically, their algorithms approximate the posterior by a Gaussian distribution, and hence need to store second-order covariance statistics which require $O(d^2)$ space and time per online step. In contrast, our approach is much more efficient, requiring only $O(d)$ space and $O(k)$ time at every online step.

After completing the paper, we learned that Carpenter (2008) independently developed an algorithm similar to ours.

1.2 What We Do

We pursue an algorithmic strategy which can be understood as an online version of an efficient L_1 loss optimization approach (Lee et al., 2007). At a high level, our approach works with the soft-regularization formulation (2) and decays the weight to a default value after every online stochastic gradient step. This simple approach enjoys minimal time complexity (which is linear in k and independent of d) as well as strong performance guarantee, as discussed in Sections 3 and 5. For instance, the algorithm never performs much worse than a standard online-learning algorithm, and the additional loss due to sparsification is controlled continuously with a single real-valued parameter. The theory gives a family of algorithms with convex loss functions for inducing sparsity—one per online-learning algorithm. We instantiate this for square loss and show how an efficient implementation can take advantage of sparse examples in Section 4. In addition to the L_1-regularization formulation (2), the family of algorithms we consider also include some non-convex sparsification techniques.

As mentioned in the introduction, we are mainly interested in sparse online methods for large scale problems with sparse features. For such problems, our algorithm should satisfy the following requirements:

- The algorithm should be computationally efficient: the number of operations per online step should be linear in the number of nonzero features, and independent of the total number of features.
- The algorithm should be memory efficient: it needs to maintain a list of active features, and can insert (when the corresponding weight becomes nonzero) and delete (when the corresponding weight becomes zero) features dynamically.

Our solution, referred to as *truncated gradient*, is a simple modification of the standard stochastic gradient rule. It is defined in (6) as an improvement over simpler ideas such as rounding and sub-gradient method with L_1 -regularization. The implementation details, showing our methods satisfy the above requirements, are provided in Section 5.

Theoretical results stating how much sparsity is achieved using this method generally require additional assumptions which may or may not be met in practice. Consequently, we rely on experiments in Section 6 to show our method achieves good sparsity practice. We compare our approach to a few others, including L_1 -regularization on small data, as well as online rounding of coefficients to zero.

2. Online Learning with Stochastic Gradient Descent

In the setting of standard online learning, we are interested in sequential prediction problems where repeatedly from $i = 1, 2, \ldots$:

1. An unlabeled example x_i arrives.
2. We make a prediction based on existing weights $w_i \in R^d$.
3. We observe y_i, let $z_i = (x_i, y_i)$, and incur some known loss $L(w_i, z_i)$ that is convex in parameter w_i.
4. We update weights according to some rule: $w_{i+1} \leftarrow f(w_i)$.

We want to come up with an update rule f, which allows us to bound the sum of losses

$$\sum_{i=1}^{t} L(w_i, z_i)$$

as well as achieving sparsity. For this purpose, we start with the standard stochastic gradient descent (SGD) rule, which is of the form:

$$f(w_i) = w_i - \eta \nabla_1 L(w_i, z_i), \tag{3}$$

where $\nabla_1 L(a, b)$ is a sub-gradient of $L(a, b)$ with respect to the first variable a. The parameter $\eta > 0$ is often referred to as the learning rate. In our analysis, we only consider constant learning rate with fixed $\eta > 0$ for simplicity. In theory, it might be desirable to have a decaying learning rate η_i which becomes smaller when i increases to get the so called *no-regret bound* without knowing T in advance. However, if T is known in advance, one can select a constant η accordingly so the regret vanishes as $T \to \infty$. Since our focus is on sparsity, not how to adapt learning rate, for clarity, we use a constant learning rate in the analysis because it leads to simpler bounds.

The above method has been widely used in online learning (Littlestone et al., 1995; Cesa-Bianchi et al., 1996). Moreover, it is argued to be efficient even for solving batch problems where we repeatedly run the online algorithm over training data multiple times. For example, the idea has been successfully applied to solve large-scale standard SVM formulations (Shalev-Shwartz et al., 2007; Zhang, 2004). In the scenario outlined in the introduction, online-learning methods are more suitable than some traditional batch-learning methods.

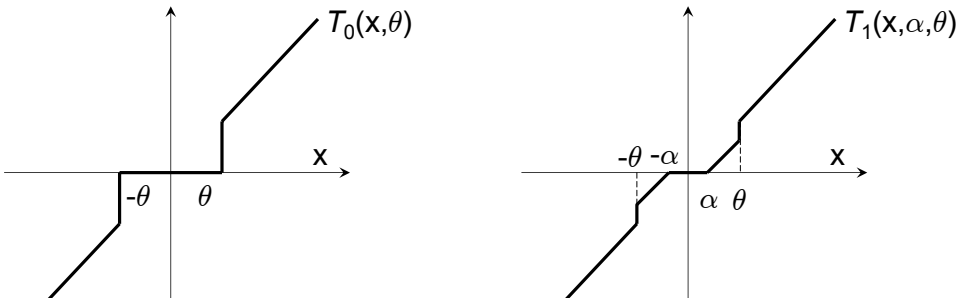

Figure 1: Plots for the truncation functions, T_0 and T_1, which are defined in the text.

However, a main drawback of (3) is that it does not achieve sparsity, which we address in this paper. In the literature, the stochastic-gradient descent rule is often referred to as gradient descent (GD). There are other variants, such as exponentiated gradient descent (EG). Since our focus in this paper is sparsity, not GD versus EG, we shall only consider modifications of (3) for simplicity.

3. Sparse Online Learning

In this section, we examine several methods for achieving sparsity in online learning. The first idea is simple coefficient rounding, which is the most natural method. We will then consider another method which is the online counterpart of L_1-regularization in batch learning. Finally, we combine such two ideas and introduce truncated gradient. As we shall see, all these ideas are closely related.

3.1 Simple Coefficient Rounding

In order to achieve sparsity, the most natural method is to round small coefficients (that are no larger than a threshold $\theta > 0$) to zero after every K online steps. That is, if i/K is not an integer, we use the standard GD rule in (3); if i/K is an integer, we modify the rule as:

$$f(w_i) = T_0(w_i - \eta \nabla_1 L(w_i, z_i), \theta), \tag{4}$$

where for a vector $v = [v_1, \ldots, v_d] \in R^d$, and a scalar $\theta \geq 0$, $T_0(v, \theta) = [T_0(v_1, \theta), \ldots, T_0(v_d, \theta)]$, with T_0 defined by (cf., Figure 1)

$$T_0(v_j, \theta) = \begin{cases} 0 & \text{if } |v_j| \leq \theta \\ v_j & \text{otherwise} \end{cases}.$$

That is, we first apply the standard stochastic gradient descent rule, and then round small coefficients to zero.

In general, we should not take $K = 1$, especially when η is small, since each step modifies w_i by only a small amount. If a coefficient is zero, it remains small after one online update, and the rounding operation pulls it back to zero. Consequently, rounding can be done only after every K steps (with a reasonably large K); in this case, nonzero coefficients have sufficient time to go above the threshold θ. However, if K is too large, then in the training stage, we will need to keep many more nonzero features in the intermediate steps before they are rounded to zero. In the extreme case, we may simply round the coefficients in the end, which does not solve the storage problem in

the training phase. The sensitivity in choosing appropriate K is a main drawback of this method; another drawback is the lack of theoretical guarantee for its online performance.

3.2 A Sub-gradient Algorithm for L_1-Regularization

In our experiments, we combine rounding-in-the-end-of-training with a simple online sub-gradient method for L_1-regularization with a regularization parameter $g > 0$:

$$f(w_i) = w_i - \eta \nabla_1 L(w_i, z_i) - \eta g \operatorname{sgn}(w_i), \tag{5}$$

where for a vector $v = [v_1, \ldots, v_d]$, $\operatorname{sgn}(v) = [\operatorname{sgn}(v_1), \ldots, \operatorname{sgn}(v_d)]$, and $\operatorname{sgn}(v_j) = 1$ when $v_j > 0$, $\operatorname{sgn}(v_j) = -1$ when $v_j < 0$, and $\operatorname{sgn}(v_j) = 0$ when $v_j = 0$. In the experiments, the online method (5) plus rounding in the end is used as a simple baseline. This method does not produce sparse weights online. Therefore it does not handle large-scale problems for which we cannot keep all features in memory.

3.3 Truncated Gradient

In order to obtain an online version of the simple rounding rule in (4), we observe that the direct rounding to zero is too aggressive. A less aggressive version is to shrink the coefficient to zero by a smaller amount. We call this idea truncated gradient.

The amount of shrinkage is measured by a *gravity* parameter $g_i > 0$:

$$f(w_i) = T_1(w_i - \eta \nabla_1 L(w_i, z_i), \eta g_i, \theta), \tag{6}$$

where for a vector $v = [v_1, \ldots, v_d] \in R^d$, and a scalar $g \geq 0$, $T_1(v, \alpha, \theta) = [T_1(v_1, \alpha, \theta), \ldots, T_1(v_d, \alpha, \theta)]$, with T_1 defined by (cf., Figure 1)

$$T_1(v_j, \alpha, \theta) = \begin{cases} \max(0, v_j - \alpha) & \text{if } v_j \in [0, \theta] \\ \min(0, v_j + \alpha) & \text{if } v_j \in [-\theta, 0] \\ v_j & \text{otherwise} \end{cases}.$$

Again, the truncation can be performed every K online steps. That is, if i/K is not an integer, we let $g_i = 0$; if i/K is an integer, we let $g_i = Kg$ for a gravity parameter $g > 0$. This particular choice is equivalent to (4) when we set g such that $\eta K g \geq \theta$. This requires a large g when η is small. In practice, one should set a small, fixed g, as implied by our regret bound developed later.

In general, the larger the parameters g and θ are, the more sparsity is incurred. Due to the extra truncation T_1, this method can lead to sparse solutions, which is confirmed in our experiments described later. In those experiments, the degree of sparsity discovered varies with the problem.

A special case, which we will try in the experiment, is to let $g = \theta$ in (6). In this case, we can use only one parameter g to control sparsity. Since $\eta K g \ll \theta$ when ηK is small, the truncation operation is less aggressive than the rounding in (4). At first sight, the procedure appears to be an ad-hoc way to fix (4). However, we can establish a regret bound for this method, showing it is theoretically sound.

Setting $\theta = \infty$ yields another important special case of (6), which becomes

$$f(w_i) = T(w_i - \eta \nabla_1 L(w_i, z_i), g_i \eta), \tag{7}$$

where for a vector $v = [v_1, \ldots, v_d] \in R^d$, and a scalar $g \geq 0$, $T(v, \alpha) = [T(v_1, \alpha), \ldots, T(v_d, \alpha)]$, with

$$T(v_j, \alpha) = \begin{cases} \max(0, v_j - \alpha) & \text{if } v_j > 0 \\ \min(0, v_j + \alpha) & \text{otherwise} \end{cases}.$$

The method is a modification of the standard sub-gradient descent method with L_1-regularization given in (5). The parameter $g_i \geq 0$ controls the sparsity that can be achieved with the algorithm. Note when $g_i = 0$, the update rule is identical to the standard stochastic gradient descent rule. In general, we may perform a truncation every K steps. That is, if i/K is not an integer, we let $g_i = 0$; if i/K is an integer, we let $g_i = Kg$ for a gravity parameter $g > 0$. The reason for doing so (instead of a constant g) is that we can perform a more aggressive truncation with gravity parameter Kg after each K steps. This may lead to better sparsity. An alternative way to derive a procedure similar to (7) is through an application of convex hull projection idea of Zinkevich (2003) to the L_1-regularized loss, as in (5). However, instead of working with the original feature set, we need to consider a $2d$-dimensional duplicated feature set $[x_i, -x_i]$, with the non-negativity constraint $w^j \geq 0$ for each component of j (w will also have dimension $2d$ in this case). The resulting method is similar to ours, with a similar theoretical guarantee as in Theorem 3.1. The proof presented in this paper is more specialized to truncated gradient, and directly works with x_i instead of augmented data $[x_i, -x_i]$. Moreover, our analysis does not require the loss function to have bounded gradient, and thus can directly handle the least squares loss.

The procedure in (7) can be regarded as an online counterpart of L_1-regularization in the sense that it approximately solves an L_1-regularization problem in the limit of $\eta \to 0$. Truncated gradient for L_1-regularization is different from (5), which is a naïve application of stochastic gradient descent rule with an added L_1-regularization term. As pointed out in the introduction, the latter fails because it rarely leads to sparsity. Our theory shows even with sparsification, the prediction performance is still comparable to standard online-learning algorithms. In the following, we develop a general regret bound for this general method, which also shows how the regret may depend on the sparsification parameter g.

3.4 Regret Analysis

Throughout the paper, we use $\|\cdot\|_1$ for 1-norm, and $\|\cdot\|$ for 2-norm. For reference, we make the following assumption regarding the loss function:

Assumption 3.1 *We assume $L(w,z)$ is convex in w, and there exist non-negative constants A and B such that $\|\nabla_1 L(w,z)\|^2 \leq AL(w,z) + B$ for all $w \in R^d$ and $z \in R^{d+1}$.*

For linear prediction problems, we have a general loss function of the form $L(w,z) = \phi(w^T x, y)$. The following are some common loss functions $\phi(\cdot,\cdot)$ with corresponding choices of parameters A and B (which are not unique), under the assumption $\sup_x \|x\| \leq C$.

- Logistic: $\phi(p,y) = \ln(1 + \exp(-py))$; $A = 0$ and $B = C^2$. This loss is for binary classification problems with $y \in \{\pm 1\}$.
- SVM (hinge loss): $\phi(p,y) = \max(0, 1 - py)$; $A = 0$ and $B = C^2$. This loss is for binary classification problems with $y \in \{\pm 1\}$.
- Least squares (square loss): $\phi(p,y) = (p-y)^2$; $A = 4C^2$ and $B = 0$. This loss is for regression problems.

Our main result is Theorem 3.1 which is parameterized by A and B. The proof is left to the appendix. Specializing it to particular losses yields several corollaries. A corollary applicable to the least square loss is given later in Corollary 4.1.

Theorem 3.1 *(Sparse Online Regret) Consider sparse online update rule (6) with $w_1 = 0$ and $\eta > 0$. If Assumption 3.1 holds, then for all $\bar{w} \in R^d$ we have*

$$\frac{1-0.5A\eta}{T}\sum_{i=1}^{T}\left[L(w_i,z_i)+\frac{g_i}{1-0.5A\eta}\|w_{i+1}\cdot I(w_{i+1}\le\theta)\|_1\right]$$
$$\le\frac{\eta}{2}B+\frac{\|\bar{w}\|^2}{2\eta T}+\frac{1}{T}\sum_{i=1}^{T}[L(\bar{w},z_i)+g_i\|\bar{w}\cdot I(w_{i+1}\le\theta)\|_1],$$

where for vectors $v=[v_1,\ldots,v_d]$ and $v'=[v'_1,\ldots,v'_d]$, we let

$$\|v\cdot I(|v'|\le\theta)\|_1=\sum_{j=1}^{d}|v_j|I(|v'_j|\le\theta),$$

where $I(\cdot)$ is the set indicator function.

We state the theorem with a constant learning rate η. As mentioned earlier, it is possible to obtain a result with variable learning rate where $\eta=\eta_i$ decays as i increases. Although this may lead to a no-regret bound without knowing T in advance, it introduces extra complexity to the presentation of the main idea. Since our focus is on sparsity rather than adapting learning rate, we do not include such a result for clarity. If T is known in advance, then in the above bound, one can simply take $\eta=O(1/\sqrt{T})$ and the L_1-regularized regret is of order $O(1/\sqrt{T})$.

In the above theorem, the right-hand side involves a term $g_i\|\bar{w}\cdot I(w_{i+1}\le\theta)\|_1$ depending on w_{i+1} which is not easily estimated. To remove this dependency, a trivial upper bound of $\theta=\infty$ can be used, leading to L_1 penalty $g_i\|\bar{w}\|_1$. In the general case of $\theta<\infty$, we cannot replace w_{i+1} by $\bar{w}$ because the effective regularization condition (as shown on the left-hand side) is the non-convex penalty $g_i\|w\cdot I(|w|\le\theta)\|_1$. Solving such a non-convex formulation is hard both in the online and batch settings. In general, we only know how to efficiently discover a local minimum which is difficult to characterize. Without a good characterization of the local minimum, it is not possible for us to replace $g_i\|\bar{w}\cdot I(w_{i+1}\le\theta)\|_1$ on the right-hand side by $g_i\|\bar{w}\cdot I(\bar{w}\le\theta)\|_1$ because such a formulation implies we can efficiently solve a non-convex problem with a simple online update rule. Still, when $\theta<\infty$, one naturally expects the right-hand side penalty $g_i\|\bar{w}\cdot I(w_{i+1}\le\theta)\|_1$ is much smaller than the corresponding L_1 penalty $g_i\|\bar{w}\|_1$, especially when w_j has many components close to 0. Therefore the situation with $\theta<\infty$ can potentially yield better performance on some data. This is confirmed in our experiments.

Theorem 3.1 also implies a trade-off between sparsity and regret performance. We may simply consider the case where $g_i=g$ is a constant. When g is small, we have less sparsity but the regret term $g\|\bar{w}\cdot I(w_{i+1}\le\theta)\|_1\le g\|\bar{w}\|_1$ on the right-hand side is also small. When g is large, we are able to achieve more sparsity but the regret $g\|\bar{w}\cdot I(w_{i+1}\le\theta)\|_1$ on the right-hand side also becomes large. Such a trade-off (sparsity versus prediction accuracy) is empirically studied in Section 6. Our observation suggests we can gain significant sparsity with only a small decrease of accuracy (that is, using a small g).

Now consider the case $\theta = \infty$ and $g_i = g$. When $T \to \infty$, if we let $\eta \to 0$ and $\eta T \to \infty$, then Theorem 3.1 implies

$$\frac{1}{T}\sum_{i=1}^{T}[L(w_i,z_i)+g\|w_i\|_1] \le \inf_{\bar{w}\in R^d}\left[\frac{1}{T}\sum_{i=1}^{T}L(\bar{w},z_i)+g\|\bar{w}\|_1\right]+o(1).$$

In other words, if we let $L'(w,z) = L(w,z)+g\|w\|_1$ be the L_1-regularized loss, then the L_1-regularized regret is small when $\eta \to 0$ and $T \to \infty$. In particular, if we let $\eta = 1/\sqrt{T}$, then the theorem implies the L_1-regularized regret is

$$\begin{aligned}&\sum_{i=1}^{T}(L(w_i,z_i)+g\|w_i\|_1) - \sum_{i=1}^{T}(L(\bar{w},z_i)+g\|\bar{w}\|_1)\\ &\le \frac{\sqrt{T}}{2}(B+\|\bar{w}\|^2)\left(1+\frac{A}{2\sqrt{T}}\right)+\frac{A}{2\sqrt{T}}\left(\sum_{i=1}^{T}L(\bar{w},z_i)+g\sum_{i=1}^{T}(\|\bar{w}\|_1-\|w_{i+1}\|_1)\right)+o(\sqrt{T}),\end{aligned}$$

which is $O(\sqrt{T})$ for bounded loss function L and weights w_i. These observations imply our procedure can be regarded as the online counterpart of L_1-regularization methods. In the stochastic setting where the examples are drawn iid from some underlying distribution, the sparse online gradient method proposed in this paper solves the L_1-regularization problem.

3.5 Stochastic Setting

SGD-based online-learning methods can be used to solve large-scale batch optimization problems, often quite successfully (Shalev-Shwartz et al., 2007; Zhang, 2004). In this setting, we can go through training examples one-by-one in an online fashion, and repeat multiple times over the training data. In this section, we analyze the performance of such a procedure using Theorem 3.1.

To simplify the analysis, instead of assuming we go through the data one by one, we assume each additional data point is drawn from the training data randomly with equal probability. This corresponds to the standard stochastic optimization setting, in which observed samples are iid from some underlying distributions. The following result is a simple consequence of Theorem 3.1. For simplicity, we only consider the case with $\theta = \infty$ and constant gravity $g_i = g$.

Theorem 3.2 *Consider a set of training data $z_i = (x_i, y_i)$ for $i = 1,\ldots,n$, and let*

$$R(w,g) = \frac{1}{n}\sum_{i=1}^{n}L(w,z_i)+g\|w\|_1$$

be the L_1-regularized loss over training data. Let $\hat{w}_1 = w_1 = 0$, and define recursively for $t = 1,2,\ldots$

$$w_{t+1} = T(w_t - \eta\nabla_1(w_t,z_{i_t}),g\eta), \qquad \hat{w}_{t+1} = \hat{w}_t + \frac{w_{t+1}-\hat{w}_t}{t+1},$$

where each i_t is drawn from $\{1,\ldots,n\}$ uniformly at random. If Assumption 3.1 holds, then at any time T, the following inequalities are valid for all $\bar{w} \in R^d$:

$$\begin{aligned}
&\mathbf{E}_{i_1,\ldots,i_T}\left[(1-0.5A\eta)R\left(\hat{w}_T, \frac{g}{1-0.5A\eta}\right)\right] \\
\leq &\mathbf{E}_{i_1,\ldots,i_T}\left[\frac{1-0.5A\eta}{T}\sum_{i=1}^{T} R\left(w_i, \frac{g}{1-0.5A\eta}\right)\right] \\
\leq &\frac{\eta}{2}B + \frac{\|\bar{w}\|^2}{2\eta T} + R(\bar{w}, g).
\end{aligned}$$

Proof Note the recursion of $\hat{w}_t$ implies

$$\hat{w}_T = \frac{1}{T}\sum_{t=1}^{T} w_t$$

from telescoping the update rule. Because $R(w,g)$ is convex in w, the first inequality follows directly from Jensen's inequality. It remains to prove the second inequality. Theorem 3.1 implies the following:

$$\frac{1-0.5A\eta}{T}\sum_{t=1}^{T}\left[L(w_t, z_{i_t}) + \frac{g}{1-0.5A\eta}\|w_t\|_1\right] \leq g\|\bar{w}\|_1 + \frac{\eta}{2}B + \frac{\|\bar{w}\|^2}{2\eta T} + \frac{1}{T}\sum_{t=1}^{T} L(\bar{w}, z_{i_t}). \quad (8)$$

Observe that

$$\mathbf{E}_{i_t}\left[L(w_t, z_{i_t}) + \frac{g}{1-0.5A\eta}\|w_t\|_1\right] = R\left(w_t, \frac{g}{1-0.5A\eta}\right)$$

and

$$g\|\bar{w}\|_1 + \mathbf{E}_{i_1,\ldots,i_T}\left[\frac{1}{T}\sum_{t=1}^{T} L(\bar{w}, z_{i_t})\right] = R(\bar{w}, g).$$

The second inequality is obtained by taking the expectation with respect to $\mathbf{E}_{i_1,\ldots,i_T}$ in (8). ■

If we let $\eta \to 0$ and $\eta T \to \infty$, the bound in Theorem 3.2 becomes

$$\mathbf{E}\left[R(\hat{w}_T, g)\right] \leq \mathbf{E}\left[\frac{1}{T}\sum_{t=1}^{T} R(w_t, g)\right] \leq \inf_{\bar{w}} R(\bar{w}, g) + o(1).$$

That is, on average, $\hat{w}_T$ approximately solves the L_1-regularization problem

$$\inf_{w}\left[\frac{1}{n}\sum_{i=1}^{n} L(w, z_i) + g\|w\|_1\right].$$

If we choose a random stopping time T, then the above inequalities says that on average w_T also solves this L_1-regularization problem approximately. Therefore in our experiment, we use the last solution w_T instead of the aggregated solution $\hat{w}_T$. For practice purposes, this is adequate even though we do not intentionally choose a random stopping time.

Since L_1-regularization is frequently used to achieve sparsity in the batch learning setting, the connection to L_1-regularization can be regarded as an alternative justification for the sparse-online algorithm developed in this paper.

Algorithm 1 Truncated Gradient for Least Squares

Inputs:

- threshold $\theta \geq 0$
- gravity sequence $g_i \geq 0$
- learning rate $\eta \in (0,1)$
- example oracle O

initialize weights $w^j \leftarrow 0$ $(j = 1,\ldots,d)$
for trial $i = 1,2,\ldots$

1. Acquire an unlabeled example $x = [x^1, x^2, \ldots, x^d]$ from oracle O
2. **forall** weights w^j $(j = 1,\ldots,d)$
   (a) **if** $w^j > 0$ and $w^j \leq \theta$ **then** $w^j \leftarrow \max\{w^j - g_i\eta, 0\}$
   (b) **elseif** $w^j < 0$ and $w^j \geq -\theta$ **then** $w^j \leftarrow \min\{w^j + g_i\eta, 0\}$
3. Compute prediction: $\hat{y} = \sum_j w^j x^j$
4. Acquire the label y from oracle O
5. Update weights for all features j: $w^j \leftarrow w^j + 2\eta(y - \hat{y})x^j$

4. Truncated Gradient for Least Squares

The method in Section 3 can be directly applied to least squares regression. This leads to Algorithm 1 which implements sparsification for square loss according to Equation (6). In the description, we use superscripted symbol w^j to denote the j-th component of vector w (in order to differentiate from w_i, which we have used to denote the i-th weight vector). For clarity, we also drop the index i from w_i. Although we keep the choice of gravity parameters g_i open in the algorithm description, in practice, we only consider the following choice:

$$g_i = \begin{cases} Kg & \text{if } i/K \text{ is an integer} \\ 0 & \text{otherwise} \end{cases}.$$

This may give a more aggressive truncation (thus sparsity) after every K-th iteration. Since we do not have a theorem formalizing how much more sparsity one can gain from this idea, its effect will only be examined empirically in Section 6.

In many online-learning situations (such as web applications), only a small subset of the features have nonzero values for any example x. It is thus desirable to deal with sparsity only in this small subset rather than in all features, while simultaneously inducing sparsity on all feature weights. Moreover, it is important to store only features with non-zero coefficients (if the number of features is too large to be stored in memory, this approach allows us to use a hash table to track only the nonzero coefficients). We describe how this can be implemented efficiently in the next section.

For reference, we present a specialization of Theorem 3.1 in the following corollary which is directly applicable to Algorithm 1.

Corollary 4.1 *(Sparse Online Square Loss Regret) If there exists $C > 0$ such that for all x, $\|x\| \leq C$, then for all $\bar{w} \in R^d$, we have*

$$\frac{1-2C^2\eta}{T}\sum_{i=1}^{T}\left[(w_i^T x_i - y_i)^2 + \frac{g_i}{1-2C^2\eta}\|w_i \cdot I(|w_i| \leq \theta)\|_1\right]$$
$$\leq \frac{\|\bar{w}\|^2}{2\eta T} + \frac{1}{T}\sum_{i=1}^{T}\left[(\bar{w}^T x_i - y_i)^2 + g_{i+1}\|\bar{w} \cdot I(|w_{i+1}| \leq \theta)\|_1\right],$$

where $w_i = [w^1, \ldots, w^d] \in R^d$ is the weight vector used for prediction at the i-th step of Algorithm 1; (x_i, y_i) is the data point observed at the i-step.

This corollary explicitly states that average square loss incurred by the learner (the left-hand side) is bounded by the average square loss of the best weight vector $\bar{w}$, plus a term related to the size of $\bar{w}$ which decays as $1/T$ and an additive offset controlled by the sparsity threshold θ and the gravity parameter g_i.

5. Efficient Implementation

We altered a standard gradient-descent implementation, VOWPAL WABBIT(Langford et al., 2007), according to algorithm 1. VOWPAL WABBIT optimizes square loss on a linear representation $w^T x$ via gradient descent (3) with a couple caveats:

1. The prediction is normalized by the square root of the number of nonzero entries in a sparse vector, $w^T x/\sqrt{\|x\|_0}$. This alteration is just a constant rescaling on dense vectors which is effectively removable by an appropriate rescaling of the learning rate.

2. The prediction is clipped to the interval $[0,1]$, implying the loss function is not square loss for unclipped predictions outside of this dynamic range. Instead the update is a constant value, equivalent to the gradient of a linear loss function.

The learning rate in VOWPAL WABBIT is controllable, supporting $1/i$ decay as well as a constant learning rate (and rates in-between). The program operates in an entirely online fashion, so the memory footprint is essentially just the weight vector, even when the amount of data is very large.

As mentioned earlier, we would like the algorithm's computational complexity to depend linearly on the number of nonzero features of an example, rather than the total number of features. The approach we took was to store a time-stamp τ_j for each feature j. The time-stamp was initialized to the index of the example where feature j was nonzero for the first time. During online learning, we simply went through all nonzero features j of example i, and could "simulate" the shrinkage of w^j after τ_j in a batch mode. These weights are then updated, and their time stamps are set to i. This lazy-update idea of delaying the shrinkage calculation until needed is the key to efficient implementation of truncated gradient. Specifically, instead of using update rule (6) for weight w^j, we shrunk the weights of all nonzero feature j differently by the following:

$$f(w^j) = T_1\left(w^j + 2\eta(y-\hat{y})x^j, \left\lfloor\frac{i-\tau_j}{K}\right\rfloor K\eta g, \theta\right),$$

and τ_j is updated by

$$\tau_j \leftarrow \tau_j + \left\lfloor\frac{i-\tau_j}{K}\right\rfloor K.$$

This lazy-update trick can be applied to the other two algorithms given in Section 3. In the coefficient rounding algorithm (4), for instance, for each nonzero feature j of example i, we can first perform a regular gradient descent on the square loss, and then do the following: if $|w_j|$ is below the threshold θ and $i \geq \tau_j + K$, we round w_j to 0 and set τ_j to i.

This implementation shows the truncated gradient method satisfies the following requirements needed for solving large scale problems with sparse features.

- The algorithm is computationally efficient: the number of operations per online step is linear in the number of nonzero features, and independent of the total number of features.

- The algorithm is memory efficient: it maintains a list of active features, and a feature can be inserted when observed, and deleted when the corresponding weight becomes zero.

If we directly apply the online projection idea of Zinkevich (2003) to solve (1), then in the update rule (7), one has to pick the smallest $g_i \geq 0$ such that $\|w_{i+1}\|_1 \leq s$. We do not know an efficient method to find this specific g_i using operations independent of the total number of features. A standard implementation relies on sorting all weights, which requires $O(d \log d)$ operations, where d is the total number of (nonzero) features. This complexity is unacceptable for our purpose. However, in an important recent work, Duchi et al. (2008) proposed an efficient online ℓ_1-projection method. The idea is to use a balanced tree to keep track of weights, which allows efficient threshold finding and tree updates in $O(k \ln d)$ operations on average, where k denotes the number of nonzero coefficients in the current training example. Although the algorithm still has weak dependency on d, it is applicable to large-scale practical applications. The theoretical analysis presented in this paper shows we can obtain a meaningful regret bound by picking an arbitrary g_i. This is useful because the resulting method is much simpler to implement and is computationally more efficient per online step. Moreover, our method allows non-convex updates closely related to the simple coefficient rounding idea. Due to the complexity of implementing the balanced tree strategy in Duchi et al. (2008), we shall not compare to it in this paper and leave it as a future direction. However, we believe the sparsity achieved with their approach should be comparable to the sparsity achieved with our method.

6. Empirical Results

We applied VOWPAL WABBIT with the efficiently implemented sparsify option, as described in the previous section, to a selection of data sets, including eleven data sets from the UCI repository (Asuncion and Newman, 2007), the much larger data set rcv1 (Lewis et al., 2004), and a private large-scale data set Big_Ads related to ad interest prediction. While UCI data sets are useful for benchmark purposes, rcv1 and Big_Ads are more interesting since they embody real-world data sets with large numbers of features, many of which are less informative for making predictions than others. The data sets are summarized in Table 1.

The UCI data sets used do not have many features so we expect that a large fraction of these features are useful for making predictions. For comparison purposes as well as to better demonstrate the behavior of our algorithm, we also added 1000 random binary features to those data sets. Each feature has value 1 with probability 0.05 and 0 otherwise.

Data Set	#features	#train data	#test data	task
ad	1411	2455	824	classification
crx	47	526	164	classification
housing	14	381	125	regression
krvskp	74	2413	783	classification
magic04	11	14226	4794	classification
mushroom	117	6079	2045	classification
spambase	58	3445	1156	classification
wbc	10	520	179	classification
wdbc	31	421	148	classification
wpbc	33	153	45	classification
zoo	17	77	24	regression
rcv1	38853	781265	23149	classification
Big_Ads	3×10^9	26×10^6	2.7×10^6	classification

Table 1: Data Set Summary.

6.1 Feature Sparsification of Truncated Gradient

In the first set of experiments, we are interested in how much reduction in the number of features is possible without affecting learning performance significantly; specifically, we require the accuracy be reduced by no more than 1% for classification tasks, and the total square loss be increased by no more than 1% for regression tasks. As common practice, we allowed the algorithm to run on the training data set for multiple passes with decaying learning rate. For each data set, we performed 10-fold cross validation over the training set to identify the best set of parameters, including the learning rate η (ranging from 0.1 to 0.5), the sparsification rate g (ranging from 0 to 0.3), number of passes of the training set (ranging from 5 to 30), and the decay of learning rate across these passes (ranging from 0.5 to 0.9). The optimized parameters were used to train VOWPAL WABBIT on the whole training set. Finally, the learned classifier/regressor was evaluated on the test set. We fixed $K = 1$ and $\theta = \infty$, and will study the effects of K and θ in later subsections.

Figure 2 shows the fraction of reduced features after sparsification is applied to each data set. For UCI data sets, we also include experiments with 1000 random features added to the original feature set. We do not add random features to rcv1 and Big_Ads since the experiment is not as interesting.

For UCI data sets, with randomly added features, VOWPAL WABBIT is able to reduce the number of features by a fraction of more than 90%, except for the ad data set in which only 71% reduction is observed. This less satisfying result might be improved by a more extensive parameter search in cross validation. However, if we can tolerate 1.3% decrease in accuracy (instead of 1% as for other data sets) during cross validation, VOWPAL WABBIT is able to achieve 91.4% reduction, indicating that a large reduction is still possible at the tiny additional cost of 0.3% accuracy loss. With this slightly more aggressive sparsification, the test-set accuracy drops from 95.9% (when only 1% loss in accuracy is allowed in cross validation) to 95.4%, while the accuracy without sparsification is 96.5%.

Even for the original UCI data sets without artificially added features, VOWPAL WABBIT manages to filter out some of the less useful features while maintaining the same level of performance. For example, for the ad data set, a reduction of 83.4% is achieved. Compared to the results above, it seems the most effective feature reductions occur on data sets with a large number of less useful features, exactly where sparsification is needed.

For rcv1, more than 75% of features are removed after the sparsification process, indicating the effectiveness of our algorithm in real-life problems. We were not able to try many parameters in cross validation because of the size of rcv1. It is expected that more reduction is possible when a more thorough parameter search is performed.

The previous results do not exercise the full power of the approach presented here because the standard Lasso (Tibshirani, 1996) is or may be computationally viable in these data sets. We have also applied this approach to a large non-public data set Big_Ads where the goal is predicting which of two ads was clicked on given context information (the content of ads and query information). Here, accepting a 0.009 increase in classification error (from error rate 0.329 to error rate 0.338) allows us to reduce the number of features from about 3×10^9 to about 24×10^6, a factor of 125 decrease in the number of features.

For classification tasks, we also study how our sparsification solution affects AUC (Area Under the ROC Curve), which is a standard metric for classification.[1] Using the same sets of parameters from 10-fold cross validation described above, we find the criterion is not affected significantly by sparsification and in some cases, they are actually slightly improved. The reason may be that our sparsification method removed some of the features that could have confused VOWPAL WABBIT. The ratios of the AUC with and without sparsification for all classification tasks are plotted in Figures 3. Often these ratios are above 98%.

6.2 The Effects of K

As we argued before, using a K value larger than 1 may be desired in truncated gradient and the rounding algorithms. This advantage is empirically demonstrated here. In particular, we try $K = 1$, $K = 10$, and $K = 20$ in both algorithms. As before, cross validation is used to select parameters in the rounding algorithm, including learning rate η, number of passes of data during training, and learning rate decay over training passes.

Figures 4 and 5 give the AUC vs. number-of-feature plots, where each data point is generated by running respective algorithm using a different value of g (for truncated gradient) and θ (for the rounding algorithm). We used $\theta = \infty$ in truncated gradient.

The effect of K is large in the rounding algorithm. For instance, in the ad data set the algorithm using $K = 1$ achieves an AUC of 0.94 with 322 features, while 13 and 7 features are needed using $K = 10$ and $K = 20$, respectively. However, the same benefits of using a larger K is not observed in truncated gradient, although the performances with $K = 10$ or 20 are at least as good as those with $K = 1$ and for the spambase data set further feature reduction is achieved at the same level of performance, reducing the number of features from 76 (when $K = 1$) to 25 (when $K = 10$ or 20) with of an AUC of about 0.89.

1. We use AUC here and in later subsections because it is insensitive to threshold, which is unlike accuracy.

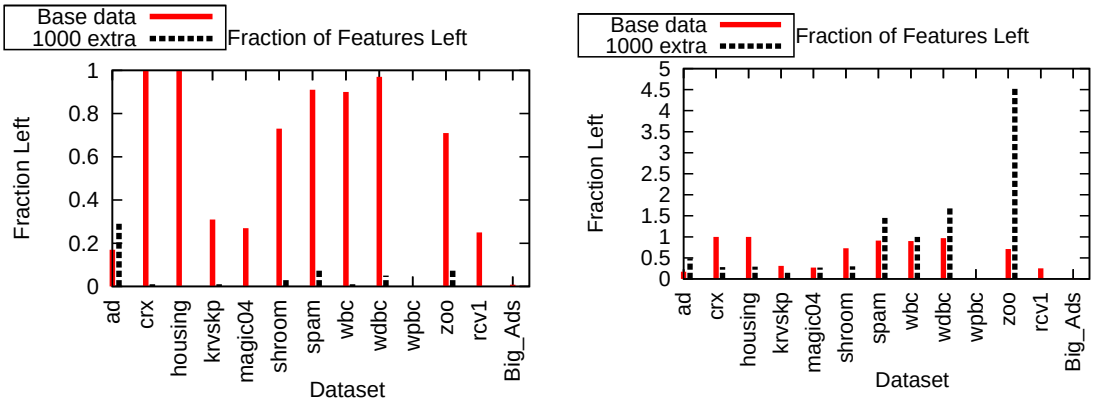

Figure 2: Plots showing the amount of features left after sparsification using truncated gradient for each data set, when the performance is changed by at most 1% due to sparsification. The solid bar: with the original feature set; the dashed bar: with 1000 random features added to each example. Plot on left: fraction left with respect to the total number of features (original with 1000 artificial features for the dashed bar). Plot on right: fraction left with respect to the original features (not counting the 1000 artificial features in the denominator for the dashed bar).

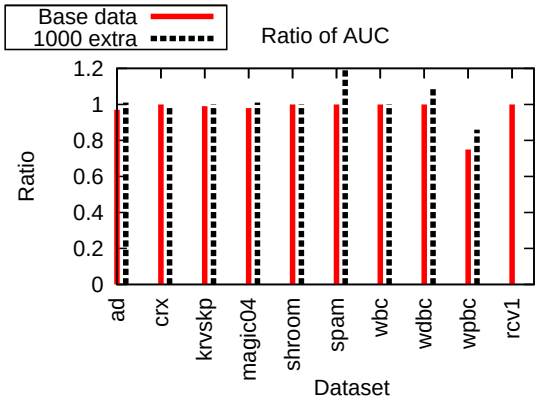

Figure 3: A plot showing the ratio of the AUC when sparsification is used over the AUC when no sparsification is used. The same process as in Figure 2 is used to determine empirically good parameters. The first result is for the original data set, while the second result is for the modified data set where 1000 random features are added to each example.

6.3 The Effects of θ in Truncated Gradient

In this subsection, we empirically study the effect of θ in truncated gradient. The rounding algorithm is also included for comparison due to its similarity to truncated gradient when $\theta = g$. Again, we used cross validation to choose parameters for each θ value tried, and focused on the AUC metric in the eight UCI classification tasks, except the degenerate one of wpbc. We fixed $K = 10$ in both algorithm.

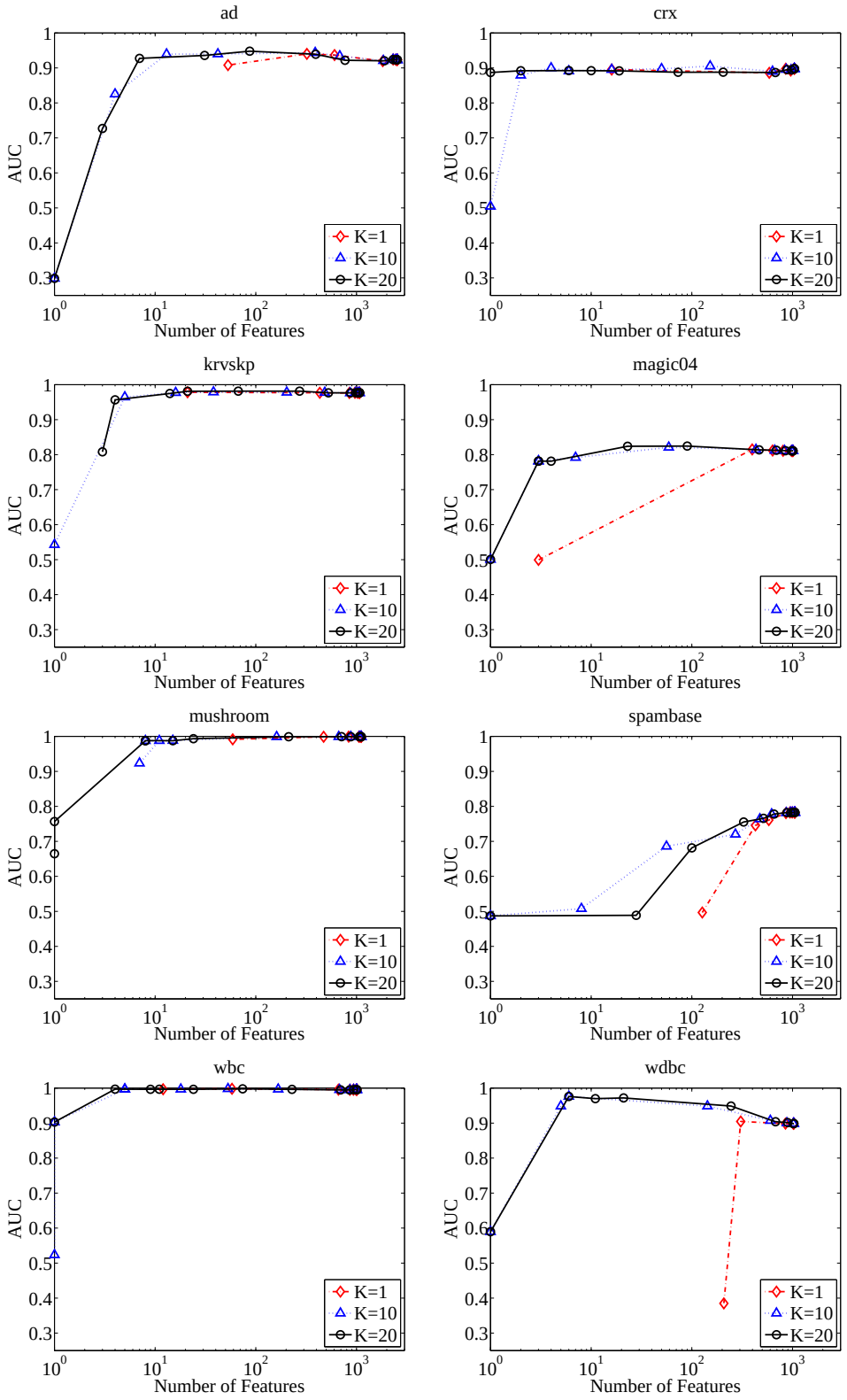

Figure 4: Effect of K on AUC in the rounding algorithm.

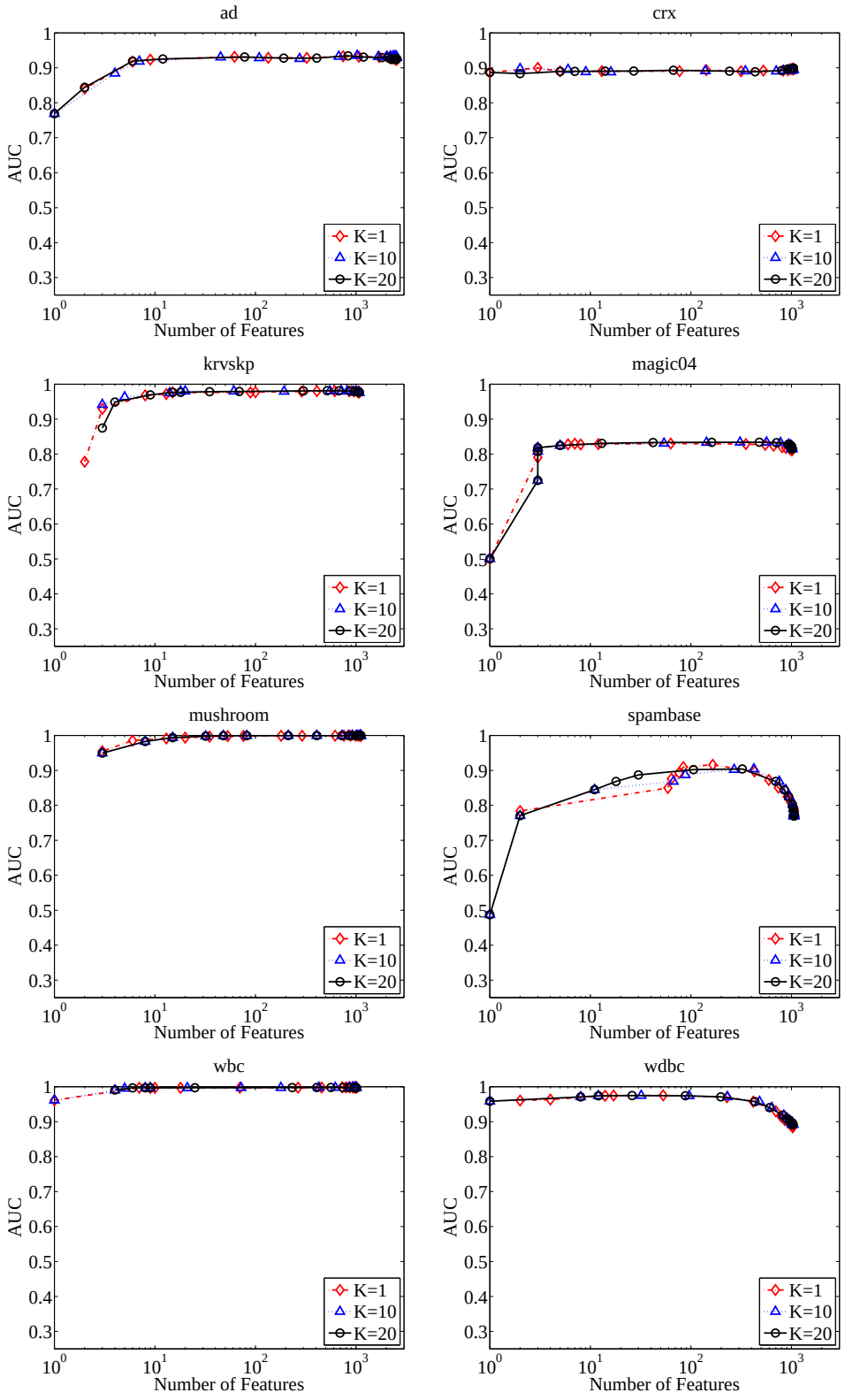

Figure 5: Effect of K on AUC in truncated gradient.

Figure 6 gives the AUC vs. number-of-feature plots, where each data point is generated by running respective algorithms using a different value of g (for truncated gradient) and θ (for the rounding algorithm). A few observations are in place. First, the results verify the observation that the behavior of truncated gradient with $\theta = g$ is similar to the rounding algorithm. Second, these results suggest that, in practice, it may be desirable to use $\theta = \infty$ in truncated gradient because it avoids the local-minimum problem.

6.4 Comparison to Other Algorithms

The next set of experiments compares truncated gradient to other algorithms regarding their abilities to balance feature sparsification and performance. Again, we focus on the AUC metric in UCI classification tasks except wpdc. The algorithms for comparison include:

- The truncated gradient algorithm: We fixed $K = 10$ and $\theta = \infty$, used crossed-validated parameters, and altered the gravity parameter g.
- The rounding algorithm described in Section 3.1: We fixed $K = 10$, used cross-validated parameters, and altered the rounding threshold θ.
- The subgradient algorithm described in Section 3.2: We fixed $K = 10$, used cross-validated parameters, and altered the regularization parameter g.
- The Lasso (Tibshirani, 1996) for batch L_1-regularization: We used a publicly available implementation (Sjöstrand, 2005).

Note that we do not attempt to compare these algorithms on rcv1 and Big_Ads simply because their sizes are too large for the Lasso.

Figure 7 gives the results. Truncated gradient is consistently competitive with the other two online algorithms and significantly outperformed them in some problems. This suggests the effectiveness of truncated gradient.

Second, it is interesting to observe that the qualitative behavior of truncated gradient is often similar to LASSO, especially when very sparse weight vectors are allowed (the left side in the graphs). This is consistent with theorem 3.2 showing the relation between them. However, LASSO usually has worse performance when the allowed number of nonzero weights is set too large (the right side of the graphs). In this case, LASSO seems to overfit, while truncated gradient is more robust to overfitting. The robustness of online learning is often attributed to early stopping, which has been extensively discussed in the literature (e.g., Zhang, 2004).

Finally, it is worth emphasizing that the experiments in this subsection try to shed some light on the relative strengths of these algorithms in terms of feature sparsification. For large data sets such as Big_Ads only truncated gradient, coefficient rounding, and the sub-gradient algorithms are applicable. As we have shown and argued, the rounding algorithm is quite ad hoc and may not work robustly in some problems, and the sub-gradient algorithm does not lead to sparsity in general during training.

7. Conclusion

This paper covers the first sparsification technique for large-scale online learning with strong theoretical guarantees. The algorithm, truncated gradient, is the natural extension of Lasso-style re-

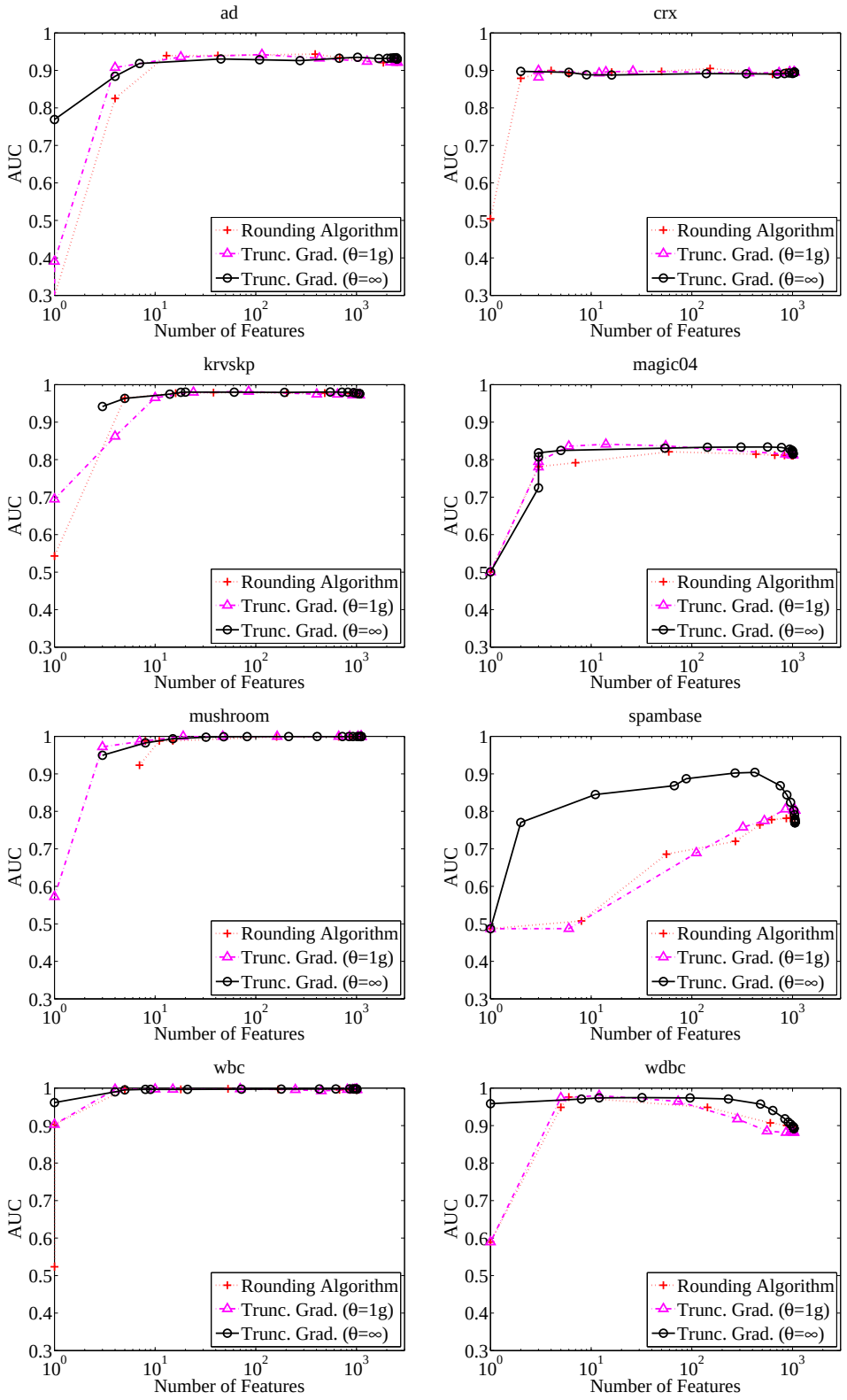

Figure 6: Effect of θ on AUC in truncated gradient.

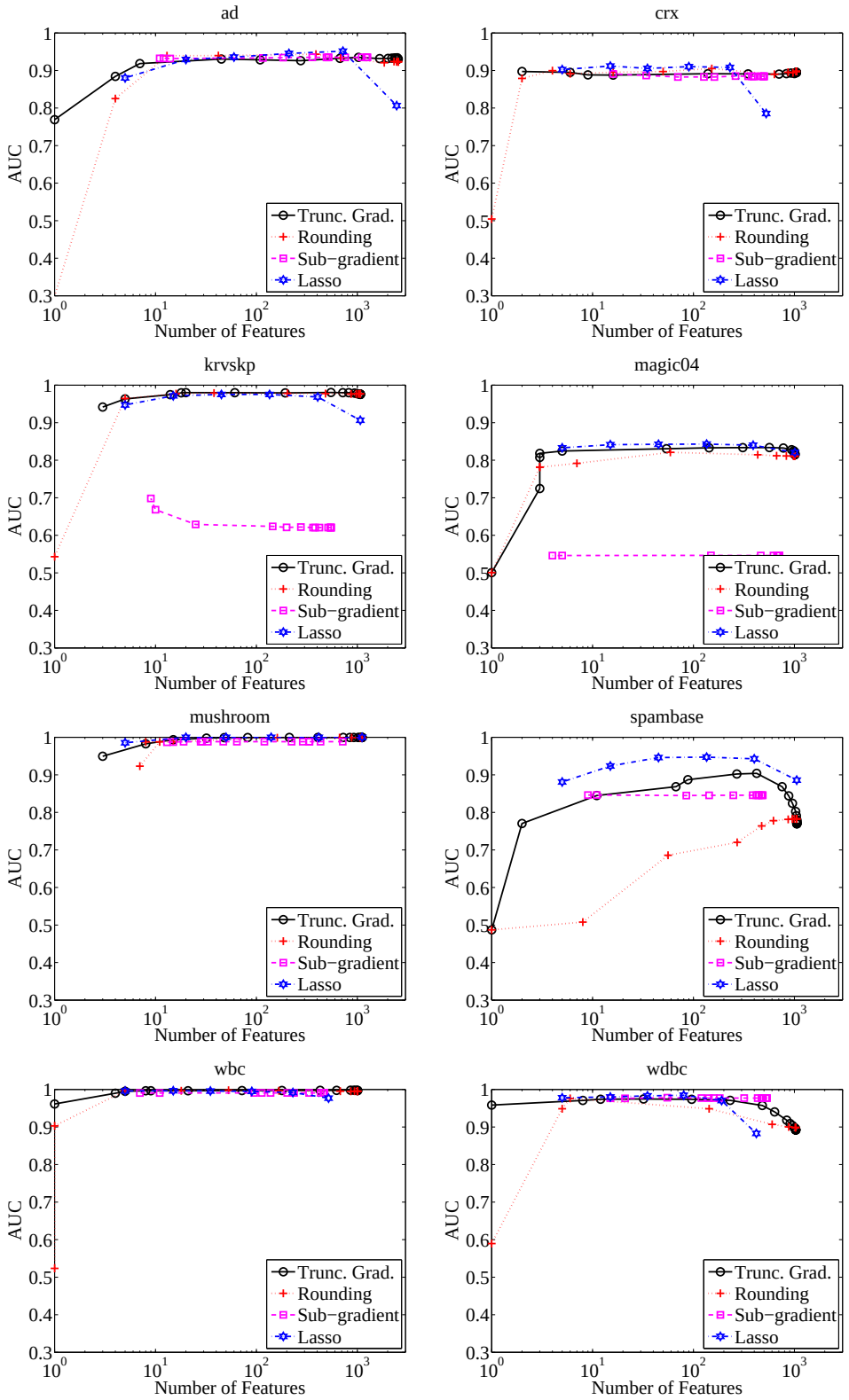

Figure 7: Comparison of four algorithms.

gression to the online-learning setting. Theorem 3.1 proves the technique is sound: it never harms performance much compared to standard stochastic gradient descent in adversarial situations. Furthermore, we show the asymptotic solution of one instance of the algorithm is essentially equivalent to the Lasso regression, thus justifying the algorithm's ability to produce sparse weight vectors when the number of features is intractably large.

The theorem is verified experimentally in a number of problems. In some cases, especially for problems with many irrelevant features, this approach achieves a one or two order of magnitude reduction in the number of features.

Acknowledgments

We thank Alex Strehl for discussions and help in developing VOWPAL WABBIT. Part of this work was done when Lihong Li and Tong Zhang were at Yahoo! Research in 2007.

Appendix A. Proof of Theorem 3.1

The following lemma is the essential step in our analysis.

Lemma 1 *Suppose update rule (6) is applied to weight vector w on example $z = (x,y)$ with gravity parameter $g_i = g$, and results in a weight vector w'. If Assumption 3.1 holds, then for all $\bar{w} \in R^d$, we have*

$$\begin{aligned}&(1-0.5A\eta)L(w,z) + g\|w' \cdot I(|w'| \le \theta)\|_1 \\ \le& L(\bar{w},z) + g\|\bar{w} \cdot I(|w'| \le \theta)\|_1 + \frac{\eta}{2}B + \frac{\|\bar{w}-w\|^2 - \|\bar{w}-w'\|^2}{2\eta}.\end{aligned}$$

Proof Consider any target vector $\bar{w} \in R^d$ and let $\tilde{w} = w - \eta\nabla_1 L(w,z)$. We have $w' = T_1(\tilde{w}, g\eta, \theta)$. Let

$$u(\bar{w},w') = g\|\bar{w} \cdot I(|w'| \le \theta)\|_1 - g\|w' \cdot I(|w'| \le \theta)\|_1.$$

Then the update equation implies the following:

$$\begin{aligned}&\|\bar{w}-w'\|^2 \\ \le& \|\bar{w}-w'\|^2 + \|w'-\tilde{w}\|^2 \\ =& \|\bar{w}-\tilde{w}\|^2 - 2(\bar{w}-w')^T(w'-\tilde{w}) \\ \le& \|\bar{w}-\tilde{w}\|^2 + 2\eta u(\bar{w},w') \\ =& \|\bar{w}-w\|^2 + \|w-\tilde{w}\|^2 + 2(\bar{w}-w)^T(w-\tilde{w}) + 2\eta u(\bar{w},w') \\ =& \|\bar{w}-w\|^2 + \eta^2\|\nabla_1 L(w,z)\|^2 + 2\eta(\bar{w}-w)^T\nabla_1 L(w,z) + 2\eta u(\bar{w},w') \\ \le& \|\bar{w}-w\|^2 + \eta^2\|\nabla_1 L(w,z)\|^2 + 2\eta(L(\bar{w},z) - L(w,z)) + 2\eta u(\bar{w},w') \\ \le& \|\bar{w}-w\|^2 + \eta^2(AL(w,z)+B) + 2\eta(L(\bar{w},z) - L(w,z)) + 2\eta u(\bar{w},w').\end{aligned}$$

Here, the first and second equalities follow from algebra, and the third from the definition of $\tilde{w}$. The first inequality follows because a square is always non-negative. The second inequality follows

because $w' = T_1(\tilde{w}, g\eta, \theta)$, which implies $(w' - \tilde{w})^T w' = -g\eta \|w' \cdot I(|\tilde{w}| \leq \theta)\|_1 = -g\eta \|w' \cdot I(|w'| \leq \theta)\|_1$ and $|w'_j - \tilde{w}_j| \leq g\eta I(|w'_j| \leq \theta)$. Therefore,

$$\begin{aligned}
-(\bar{w} - w')^T (w' - \tilde{w}) =& -\bar{w}^T (w' - \tilde{w}) + w'^T (w' - \tilde{w}) \\
\leq& \sum_{j=1}^{d} |\bar{w}_j| |w'_j - \tilde{w}_j| + (w' - \tilde{w})^T w' \\
\leq& g\eta \sum_{j=1}^{d} |\bar{w}_j| I(|w'_j| \leq \theta) + (w' - \tilde{w})^T w' = \eta u(\bar{w}, w'),
\end{aligned}$$

where the third inequality follows from the definition of sub-gradient of a convex function, implying

$$(\bar{w} - w)^T \nabla_1 L(w, z) \leq L(\bar{w}, z) - L(w, z)$$

for all w and $\bar{w}$; the fourth inequality follows from Assumption 3.1. Rearranging the above inequality leads to the desired bound. ■

Proof (of Theorem 3.1) Applying Lemma 1 to the update on trial i gives

$$\begin{aligned}
&(1 - 0.5A\eta) L(w_i, z_i) + g_i \|w_{i+1} \cdot I(|w_{i+1}| \leq \theta)\|_1 \\
\leq& L(\bar{w}, z_i) + \frac{\|\bar{w} - w_i\|^2 - \|\bar{w} - w_{i+1}\|^2}{2\eta} + g_i \|\bar{w} \cdot I(|w_{i+1}| \leq \theta)\|_1 + \frac{\eta}{2} B.
\end{aligned}$$

Now summing over $i = 1, 2, \ldots, T$, we obtain

$$\begin{aligned}
& \sum_{i=1}^{T} [(1 - 0.5A\eta) L(w_i, z_i) + g_i \|w_{i+1} \cdot I(|w_{i+1}| \leq \theta)\|_1] \\
\leq & \sum_{i=1}^{T} \left[\frac{\|\bar{w} - w_i\|^2 - \|\bar{w} - w_{i+1}\|^2}{2\eta} + L(\bar{w}, z_i) + g_i \|\bar{w} \cdot I(|w_{i+1}| \leq \theta)\|_1 + \frac{\eta}{2} B \right] \\
= & \frac{\|\bar{w} - w_1\|^2 - \|\bar{w} - w_T\|^2}{2\eta} + \frac{\eta}{2} TB + \sum_{i=1}^{T} [L(\bar{w}, z_i) + g_i \|\bar{w} \cdot I(|w_{i+1}| \leq \theta)\|_1] \\
\leq & \frac{\|\bar{w}\|^2}{2\eta} + \frac{\eta}{2} TB + \sum_{i=1}^{T} [L(\bar{w}, z_i) + g_i \|\bar{w} \cdot I(|w_{i+1}| \leq \theta)\|_1].
\end{aligned}$$

The first equality follows from the telescoping sum and the second inequality follows from the initial condition (all weights are zero) and dropping negative quantities. The theorem follows by dividing with respect to T and rearranging terms. ■

References

Arthur Asuncion and David J. Newman. UCI machine learning repository, 2007. University of California, Irvine, School of Information and Computer Sciences, http://www.ics.uci.edu/∼mlearn/MLRepository.html.

Suhrid Balakrishnan and David Madigan. Algorithms for sparse linear classifiers in the massive data setting. *Journal of Machine Learning Research*, 9:313–337, 2008.

Bob Carpenter. Lazy sparse stochastic gradient descent for regularized multinomial logistic regression. Technical report, April 2008.

Nicolò Cesa-Bianchi, Philip M. Long, and Manfred Warmuth. Worst-case quadratic loss bounds for prediction using linear functions and gradient descent. *IEEE Transactions on Neural Networks*, 7(3):604–619, 1996.

Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Y. Ng, and Kunle Olukotun. Map-reduce for machine learning on multicore. In *Advances in Neural Information Processing Systems 20 (NIPS-07)*, 2008.

Ofer Dekel, Shai Shalev-Schwartz, and Yoram Singer. The Forgetron: A kernel-based perceptron on a fixed budget. In *Advances in Neural Information Processing Systems 18 (NIPS-05)*, pages 259–266, 2006.

John Duchi and Yoram Singer. Online and batch learning using forward looking subgradients. Unpublished manuscript, September 2008.

John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. Efficient projections onto the ℓ_1-ball for learning in high dimensions. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML-08)*, pages 272–279, 2008.

Jyrki Kivinen and Manfred K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132(1):1–63, 1997.

John Langford, Lihong Li, and Alexander L. Strehl. Vowpal Wabbit (fast online learning), 2007. http://hunch.net/∼vw/.

Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y. Ng. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems 19 (NIPS-06)*, pages 801–808, 2007.

David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithms. *Machine Learning*, 2(4):285–318, 1988.

Nick Littlestone, Philip M. Long, and Manfred K. Warmuth. On-line learning of linear functions. *Computational Complexity*, 5(2):1–23, 1995.

Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning (ICML-07)*, 2007.

Karl Sjöstrand. Matlab implementation of LASSO, LARS, the elastic net and SPCA, June 2005. Version 2.0, http://www2.imm.dtu.dk/pubdb/p.php?3897.

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

Tong Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the Twenty-First International Conference on Machine Learning (ICML-04)*, pages 919–926, 2004.

Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-03)*, pages 928–936, 2003.