Sparse Online Learning via Truncated Gradient

John Langford
Yahoo! Research
jl@yahoo-inc.com

Lihong Li
Department of Computer Science
Rutgers University
lihong@cs.rutgers.edu

Tong Zhang
Department of Statistics
Rutgers University
tongz@rci.rutgers.edu

Abstract

We propose a general method called *truncated gradient* to induce sparsity in the weights of online-learning algorithms with convex loss. This method has several essential properties. First, the degree of sparsity is continuous—a parameter controls the rate of sparsification from no sparsification to total sparsification. Second, 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 small rates of sparsification result in only small additional regret with respect to typical online-learning guarantees. Finally, the approach works well empirically. We apply it to several datasets and find for datasets with large numbers of features, substantial sparsity is discoverable.

1 Introduction

We are concerned with machine learning over large datasets. As an example, the largest dataset 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 dataset into memory or they are not sufficiently efficient. There are roughly two approaches which can work: one is to parallelize a batch learning algorithm over many machines (*e.g.*, [3]), the other is to stream the examples to an online-learning algorithm (*e.g.*, [2, 6]). This paper focuses on the second approach.

Typical online-learning algorithms have at least one weight for every feature, which is too expensive in some applications for a couple reasons. The first is *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. The second is *test-time constraints*: reducing the number of features can significantly reduce the computational time to evaluate a new sample.

This paper addresses the problem of inducing sparsity in learned weights while using an online-learning algorithm. Natural solutions do not work for our problem. For example, either simply adding L_1 regularization to the gradient of an online weight update or simply rounding small weights to zero are problematic. However, these two ideas are closely related to the algorithm we propose and more detailed discussions are found in section 3. A third solution is black-box wrapper approaches which eliminate features and test the impact of the elimination. These approaches typically run an algorithm many times which is particularly undesirable with large datasets.

Similar problems have been considered in various settings before. The Lasso algorithm [12] is commonly used to achieve L_1 regularization for linear regression. 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 soft-regularization with a tunable parameter g:

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

With appropriately chosen g, the two formulations are equivalent. The convex-constraint formulation has a simple online version using the projection idea in [14]. It 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 features are sparse, although important progress was made recently so the complexity is logarithmic in the number of features [5]. In contrast, the soft-regularization formulation (2) is efficient for a batch setting[8] so we pursue it here in an online setting where it has complexity independent of the number of features. In addition to the L_1 regularization formulation (2), the family of online-learning algorithms we consider also includes some non-convex sparsification techniques.

The Forgetron [4] 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 simple linear setting. When applied to a linear kernel, the Forgetron is not computationally or space competitive with approaches operating directly on feature weights.

At a high level, our approach is weight decay to a default value. This simple method enjoys strong performance guarantees (section 3). For instance, the algorithm never performs much worse than a standard online-learning algorithm, and the additional loss due to sparsification is controlled continuously by 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 in section 4 and show how this algorithm can be implemented efficiently in large-scale problems with sparse features. For such problems, truncated gradient enjoys the following properties: (i) It 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; (2) It is *memory efficient*: it maintains a list of active features, and can insert (when the corresponding weight becomes nonzero) and delete (when the corresponding weight becomes zero) features dynamically.

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 5 to show truncated gradient achieves good sparsity in practice. We compare truncated gradient to a few others on small datasets, including the Lasso, online rounding of coefficients to zero, and L_1-regularized subgradient descent. Details of these algorithms are given in section 3.

2 Online Learning with Stochastic Gradient Descent

We are interested in the standard sequential prediction problems where for $i = 1, 2, \ldots$:

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

We want an update rule f 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 subgradient of $L(a, b)$ with respect to the first variable a. The parameter $\eta > 0$ is often referred to as the *learning rate*. In the analysis, we only consider constant learning rate 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 the focus of the present paper is on weight sparsity, rather than choosing the learning rate, 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 (*e.g.*, [2, 6]). 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 [10, 13]. In the scenario outlined in the introduction, online-learning methods are more suitable than some traditional batch learning methods. However, the learning rule (3) itself does not achieve sparsity in the weights, which we address in this paper. Note that variants of SGD exist in the literature, such as exponentiated gradient descent (EG) [6]. Since our focus is sparsity, not SGD vs. EG, we shall only consider modifications of (3) for simplicity.

3 Sparse Online Learning

In this section, we first examine three methods for achieving sparsity in online learning, including a novel algorithm called *truncated gradient*. As we shall see, all these ideas are closely related. Then, we provide theoretical justifications for this algorithm, including a general regret bound and a fundamental connection to the Lasso.

3.1 Simple Coefficient Rounding

In order to achieve sparsity, the most natural method is to round small coefficients (whose magnitudes are below a threshold $\theta > 0$) to zero after every K online steps. That is, if i/K is not an integer, we use the standard SGD rule (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: $\theta \geq 0$ is a threshold, $T_0(v, \theta) = [T_0(v_1, \theta), \ldots, T_0(v_d, \theta)]$ for vector $v = [v_1, \ldots, v_d] \in R^d$, $T_0(v_j, \theta) = v_j I(|v_j| < \theta)$, and $I(\cdot)$ is the set-indicator function. In other words, we first perform a standard stochastic gradient descent, and then round the coefficients to zero. The effect is to remove nonzero and small weights. In general, we should not take $K = 1$, especially when η is small, since in each step w_i is modified 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 must 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 at all. 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. These issues motivate us to consider more principled solutions.

3.2 L_1-Regularized Subgradient

In the experiments, we also combined rounding-in-the-end-of-training with a simple online subgradient method for L_1 regularization with a regularization parameter $g > 0$:

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

where for a vector $v = [v_1, \ldots, v_d]$, $\mathrm{sgn}(v) = [\mathrm{sgn}(v_1), \ldots, \mathrm{sgn}(v_d)]$, and $\mathrm{sgn}(v_j) = 1$ if $v_j > 0$, $\mathrm{sgn}(v_j) = -1$ if $v_j < 0$, and $\mathrm{sgn}(v_j) = 0$ if $v_j = 0$. In the experiments, the online method (5) with rounding in the end is used as a simple baseline. Notice this method does not produce sparse weights online simply because only in very rare cases do two floats add up to 0. Therefore, it is not feasible in 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 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*, where the amount of shrinkage is controlled 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

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

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 $g > 0$. 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 can potentially lead to better sparsity. We also note that when $\eta K g \geq \theta$, truncate gradient coincides with (4). But in practice, as is also verified by the theory, one should adopt a small g; hence, the new learning rule (6) is expected to differ from (4).

In general, the larger the parameters g and θ are, the more sparsity is expected. Due to the extra truncation T_1, this method can lead to sparse solutions, which is confirmed empirically in section 5.

A special case, which we use 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 (in the next subsection) for this method, showing it is theoretically sound. Therefore, it can be regarded as a principled variant of rounding. Another important special case of (6) is setting $\theta = \infty$, in which all weight components shrink in every online step. The method is a modification of the L_1-regularized subgradient descent rule (5). The parameter $g_i \geq 0$ controls the sparsity achieved with the algorithm, and setting $g_i = 0$ gives exactly the standard SGD rule (3). As we show in section 3.5, this special case of truncated gradient can be regarded as an online counterpart of L_1 regularization since it approximately solves an L_1 regularization problem in the limit of $\eta \to 0$. We also show the prediction performance of truncated gradient, measured by total loss, is comparable to standard stochastic gradient descent while introducing sparse weight vectors.

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 $\sup_x \|x\| \leq C$. All of them can be used for binary classification where $y \in \pm 1$, but the last one is more often used in regression where $y \in R$: **Logistic**: $\phi(p, y) = \ln(1 + \exp(-py))$, with $A = 0$ and $B = C^2$; **SVM** (hinge loss): $\phi(p, y) = \max(0, 1 - py)$, with $A = 0$ and $B = C^2$; **Least squares** (square loss): $\phi(p, y) = (p - y)^2$, with $A = 4C^2$ and $B = 0$.

The main result is Theorem 3.1 which is parameterized by A and B. The proof will be provided in a longer paper.

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

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

where $\|v \cdot I(|v'| \leq \theta)\|_1 = \sum_{j=1}^{d} |v_j| I(|v'_j| \leq \theta)$ for vectors $v = [v_1, \ldots, v_d]$ and $v' = [v'_1, \ldots, v'_d]$.

The theorem is stated 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 the focus is on sparsity rather than optimizing 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 regret is of order $O(1/\sqrt{T})$.

In the above theorem, the right-hand side involves a term $g_i\|\bar{w} \cdot I(w_{i+1} \leq \theta)\|_1$ that depends 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 remove the w_{i+1} dependency because the effective regularization condition (as shown on the left-hand side) is the non-convex penalty $g_i\|w \cdot I(|w| \leq \theta)\|_1$. 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 $g_i\|\bar{w} \cdot I(w_{i+1} \leq \theta)\|_1$ on the right-hand side by $g_i\|\bar{w} \cdot I(\bar{w} \leq \theta)\|_1$ because such a formulation implies we could efficiently solve a non-convex problem with a simple online update rule. Still, when $\theta < \infty$, one naturally expects the right-hand side penalty $g_i\|\bar{w} \cdot I(w_{i+1} \leq \theta)\|_1$ 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.

Theorem 3.1 also implies a tradeoff 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 $g\|\bar{w} \cdot I(w_{i+1} \leq \theta)\|_1 \leq g\|\bar{w}\|_1$ on the right-hand side is also small. When g is large, we are able to achieve more sparsity but the regret $g\|\bar{w} \cdot I(w_{i+1} \leq \theta)\|_1$ on the right-hand side also becomes large. Such a tradeoff between sparsity and prediction accuracy is empirically studied in section 5, where we achieve significant sparsity with only a small g (and thus small decrease of performance).

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

$$\frac{1}{T}\sum_{i=1}^{T}[L(w_i, z_i) + g\|w_i\|_1] \leq \inf_{\bar{w} \in R^d}\left[\frac{1}{T}\sum_{i=1}^{T} L(\bar{w}, z_i) + 2g\|\bar{w}\|_1\right] + o(1).$$

follows from Theorem 3.1. 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 $T \to \infty$. This implies truncated gradient 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. In this setting, we can go through training examples one-by-one in an online fashion, and repeat multiple times over the training data. To simplify the analysis, instead of assuming we go through example one by one, we assume each additional example 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 $g_i = g$. The expectation $\mathbf{E}$ is taken over sequences of indices $i_1, \ldots, i_T$.

Theorem 3.2 *(Stochastic Setting) Consider a set of training data $z_i = (x_i, y_i)$ for $1 \leq i \leq n$. 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 $t \geq 1$:

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

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

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

Observe that if we let $\eta \to 0$ and $\eta T \to \infty$, the bound in Theorem 3.2 becomes $\mathbf{E}\left[R(\hat{w}_T, g)\right] \leq \mathbf{E}\left[\frac{1}{T}\sum_{t=1}^{T} R(w_t, g)\right] \leq \inf_{\bar{w}} R(\bar{w}, g) + o(1)$. In other words, on average $\hat{w}_T$ approximately solves the batch L_1-regularization problem $\inf_w \left[\frac{1}{n}\sum_{i=1}^{n} L(w, z_i) + g\|w\|_1\right]$ when T is large. If we choose a random stopping time T, then the above inequalities say that on average w_T also solves this L_1-regularization problem approximately. Thus, we use the last solution w_T instead of the aggregated solution $\hat{w}_T$ in experiments. Since L_1 regularization is often used to achieve sparsity in the batch learning setting, the connection of truncated gradient to L_1 regularization can be regarded as an alternative justification for the sparsity ability of this algorithm.

4 Efficient Implementation of Truncated Gradient for Square Loss

The truncated descent update rule (6) can be applied to least-squares regression using square loss, leading to $f(w_i) = T_1(w_i + 2\eta(y_i - \hat{y}_i)x_i, \eta g_i, \theta)$, where the prediction is given by $\hat{y}_i = \sum_j w_i^j x_i^j$. We altered an efficient SGD implementation, Vowpal Wabbit [7], for least-squares regression according to truncated gradient. The program operates in an entirely online fashion. Features are hashed instead of being stored explicitly, and weights can be easily inserted into or deleted from the table dynamically. So the memory footprint is essentially just the number of nonzero weights, even when the total numbers of data and features are astronomically large.

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. The approach we take is to store a time-stamp τ_j for each feature j. The time-stamp is initialized to the index of the example where feature j becomes nonzero for the first time. During online learning, at each step i, we only go through the nonzero features j of example i, and calculate the un-performed shrinkage of w^j between τ_j and the current time i. These weights are then updated, and their time stamps are reset to i. This lazy-update idea of delaying the shrinkage calculation until needed is the key to efficient implementation of truncated gradient. The implementation satisfies efficiency requirements outlined at the end of the introduction section. A similar time-stamp trick can be applied to the other two algorithms given in section 3.

5 Empirical Results

We applied the algorithm, with the efficiently implemented sparsify option, as described in the previous section, to a selection of datasets, including eleven datasets from the UCI repository [1], the much larger dataset rcv1 [9], and a private large-scale dataset Big_Ads related to ad interest prediction. While UCI datasets are useful for benchmark purposes, rcv1 and Big_Ads are more interesting since they embody real-world datasets with large numbers of features, many of which are less informative for making predictions than others. The UCI datasets we used do not have many features, and it seems a large fraction of these features are useful for making predictions. For comparison purposes and to better demonstrate the behavior of our algorithm, we also added 1000 random binary features to those datasets. Each feature has value 1 with prob. 0.05 and 0 otherwise.

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 dataset, we performed 10-fold cross validation over the training set to identify the best set of parameters, including the learning rate η, the gravity g, number of passes of the training set, and the decay of learning rate across these passes. This set of parameters was then used on the whole training set. Finally, the learned classifier/regressor was evaluated on the test set. We fixed $K = 1$ and $\theta = \infty$ in this set of experiments. The effects of K and θ are included in an extended version of this paper. Figure 1 shows the fraction of reduced features after sparsification is applied to each dataset. For UCI datasets with randomly added features, truncated gradient was able to reduce the number of features by a fraction of more than 90%, except for the ad dataset in which only 71% reduction was observed. This less satisfying result might be improved by a more extensive parameter search in cross validation. However, if

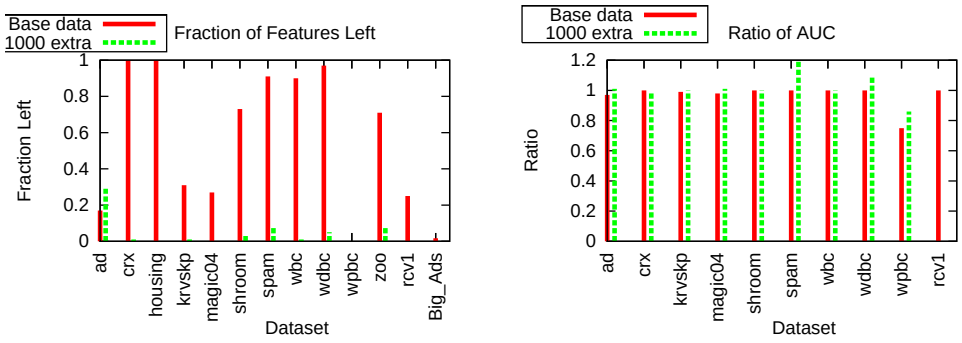

Figure 1: Left: the amount of features left after sparsification for each dataset without 1% performance loss. Right: the ratio of AUC with and without sparsification.

we tolerated 1.3% decrease in accuracy (instead of 1%) during cross validation, truncated gradient was able to achieve 91.4% reduction, indicating a large reduction is still possible at the tiny additional accuracy loss of 0.3%. Even for the original UCI datasets *without* artificially added features, some of the less useful features were removed while the same level of performance was maintained. For classification tasks, we also studied how truncated gradient affects AUC (Area Under the ROC Curve), a standard metric for classification. We use AUC here because it is insensitive to threshold, unlike accuracy. Using the same sets of parameters from 10-fold cross validation described above, we found the criterion was not affected significantly by sparsification and in some cases, it was actually improved, due to removal of some irrelevant features. The ratios of the AUC with and without sparsification for all classification tasks are plotted in Figure 1. Often these ratios are above 98%.

The previous results do not exercise the full power of the approach presented here because they are applied to datasets where the standard Lasso is computationally viable. We have also applied this approach to a large non-public dataset 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 0.9% increase in classification error 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.

The next set of experiments compares truncated gradient to other algorithms regarding their abilities to tradeoff feature sparsification and performance. Again, we focus on the AUC metric in UCI classification tasks. The algorithms for comparison include: (i) the truncated gradient algorithm with $K = 10$ and $\theta = \infty$; (ii) the truncated gradient algorithm with $K = 10$ and $\theta = g$; (iii) the rounding algorithm with $K = 10$; (iv) the L_1-regularized subgradient algorithm with $K = 10$; and (v) the Lasso [12] for batch L_1 regularization (a publicly available implementation [11] was used). We have chosen $K = 10$ since it worked better than $K = 1$, and this choice was especially important for the coefficient rounding algorithm. All unspecified parameters were identified using cross validation. 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 and subgradient descent. Figure 2 gives the results on datasets ad and spambase. Results on other datasets were qualitatively similar. On all datasets, truncated gradient (with $\theta = \infty$) is consistently competitive with the other online algorithms and significantly outperformed them in some problems, implying truncated gradient is generally effective. Moreover, truncated gradient with $\theta = g$ behaves similarly to rounding (and sometimes better). This was expected as truncated gradient with $\theta = g$ can be regarded as a principled variant of rounding with valid theoretical justification. It is also interesting to observe the qualitative behavior of truncated gradient was often similar to LASSO, especially when very sparse weight vectors were allowed (the left sides in the graphs). This is consistent with theorem 3.2 showing the relation between these two algorithms. However, LASSO usually performed worse when the allowed number of nonzero weights was large (the right side of the graphs). In this case, LASSO seemed to overfit while truncated gradient was more robust to overfitting. The robustness of online learning is often attributed to early stopping, which has been extensively studied (e.g., in [13]).

Finally, it is worth emphasizing that these comparison experiments shed some light on the relative strengths of these algorithms in terms of feature sparsification, without considering which one can be efficiently implemented. For large datasets with sparse features, only truncated gradient and the *ad hoc* coefficient rounding algorithm are applicable.

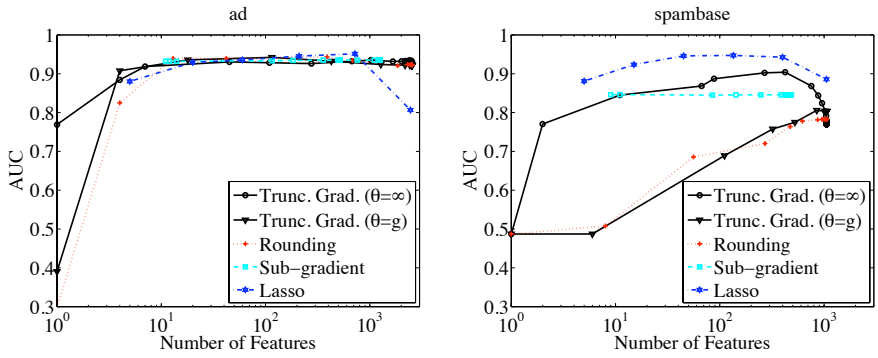

Figure 2: Comparison of the five algorithms in two sample UCI datasets.

6 Conclusion

This paper covers the first efficient sparsification technique for large-scale online learning with strong theoretical guarantees. The algorithm, truncated gradient, is the natural extension of Lasso-style regression 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 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 orders of magnitude reduction in the number of features.

References

[1] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007. UC Irvine.

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

[3] C.-T. Chu, S.K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A.Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In *Advances in Neural Information Processing Systems 20*, pages 281–288, 2008.

[4] O. Dekel, S. Shalev-Schwartz, and Y. Singer. The Forgetron: A kernel-based perceptron on a fixed budget. In *Advances in Neural Information Processing Systems 18*, pages 259–266, 2006.

[5] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the ℓ_1-ball for learning in high dimensions. In *Proceedings of ICML-08*, pages 272–279, 2008.

[6] J. Kivinen and M.K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132(1):1–63, 1997.

[7] J. Langford, L. Li, and A.L. Strehl. Vowpal Wabbit (fast online learning), 2007. http://hunch.net/∼vw/.

[8] Honglak Lee, Alexis Batle, Rajat Raina, and Andrew Y. Ng. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems 19 (NIPS-07)*, 2007.

[9] D.D. Lewis, Y. Yang, T.G. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

[10] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal Estimated sub-GrAdient SOlver for SVM. In *Proceedings of ICML-07*, pages 807–814, 2007.

[11] K. 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.

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

[13] T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of ICML-04*, pages 919–926, 2004.

[14] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of ICML-03*, pages 928–936, 2003.