# Linear Learning with Sparse Data

Ofer Dekel

oferd@microsoft.com

January 26, 2017

## Abstract

Linear predictors are especially useful when the data is high-dimensional and sparse. One of the standard techniques used to train a linear predictor is the *Averaged Stochastic Gradient Descent* (ASGD) algorithm. We present an efficient implementation of ASGD that avoids dense vector operations. We also describe a translation invariant extension called Centered Averaged Stochastic Gradient Descent (CASGD).

## 1 Introduction

We are given a training set of labeled examples, $\{(x_i, y_i)\}_{i=1}^m$, where each $x_i \in \mathbb{R}^n$ is called a *feature vector* and $y_i \in \mathbb{R}$ is its corresponding label. We are also given a loss function $\ell : \mathbb{R}^2 \mapsto \mathbb{R}$, defined over pairs of labels, where $\ell(p, y)$ is understood to be the penalty associated with predicting the label $p$ when the correct label is known to be $y$. We restrict our discussion to loss functions that are convex in their first argument. Different choices of $\ell$ lead to different learning problems. For example, choosing $\ell$ to be the *absolute loss* or *squared loss* induces a regression problem, whereas choosing the *hinge loss* or *log-loss* induces a binary classification problem (see Table 1 for the definitions of these loss functions).

A linear predictor is a pair $(\mathbf{w}, b)$, where $\mathbf{w} \in \mathbb{R}^n$ is called the *weights* vector and $b \in \mathbb{R}$ is called the *bias*. Given a feature vector $x \in \mathbb{R}^n$, the linear predictor predicts the real-valued label $\mathbf{w} \cdot x + b$. Therefore, the loss incurred by the linear predictor $(\mathbf{w}, b)$ on the training example $(x_i, y_i)$ is $\ell(\mathbf{w} \cdot x_i + b, y_i)$, and the average loss on the entire training set is

$$\frac{1}{m} \sum_{i=1}^m \ell(\mathbf{w} \cdot x_i + b, y_i) \ .$$

To promote statistical generalization, we add a regularization term to the average loss and arrive

| name | definition | (sub)derivative |
|---|---|---|
| absolute loss | $\ell(p, y) = \|p - y\|$ | $\ell'(p, y) = \begin{cases} -1 & \text{if } p \leq y \\ 1 & \text{otherwise} \end{cases}$ |
| squared loss | $\ell(p, y) = \frac{1}{2}(p - y)^2$ | $\ell'(p, y) = p - y$ |
| hinge loss | $\ell(p, y) = \max\{1 - py, 0\}$ | $\ell'(p, y) = \begin{cases} -y & \text{if } py \leq 1 \\ 0 & \text{otherwise} \end{cases}$ |
| log-loss | $\ell(p, y) = \log(1 + \exp(-py))$ | $\ell'(p, y) = \frac{-y}{1 + \exp(py)}$ |

Table 1: Examples of convex loss functions and their (sub)derivatives.

at the objective function

$$F(\mathbf{w}, b) \;=\; \frac{\lambda}{2}(\|\mathbf{w}\|^2 + b^2) \;+\; \frac{1}{m}\sum_{i=1}^{m} \ell(\mathbf{w} \cdot x_i + b, \, y_i) \;\;, \tag{1}$$

where $\lambda$ is a user-defined regularization parameter. The goal of our algorithms is to efficiently find the linear predictor that minimizes $F$. As mentioned above, we solve this optimization problem using the Averaged Stochastic Gradient Descent (ASGD) algorithm.

## 2 Sparse Vector Operations

In many high dimensional machine learning problems, the feature vectors are sparse. Namely, only a small subset of each feature vector's entries are non-zero. Concretely, we assume that, on average, there are $k$ non-zero elements in each feature vector, where $k \ll n$. A good example of a supervised machine learning problem with high dimensional sparse data is text categorization using a bag-of-words feature representation. In this setting, the dimension, $n$, is the number of words in the dictionary, which could be in the millions. On the other hand, the number of non-zeros in each featuer vector, $k$, is the number of unique words in a single document, which could be a few hundreds.

Although the feature vectors are sparse, the linear predictor that optimizes Eq. (1) can have a dense weights vector. To emphasize that some vectors are sparse and others are dense, we denote dense vectors using boldface roman letters, such as $\mathbf{w}$, $\mathbf{v}$, and $\mathbf{u}$.

Sparse vectors can be stored using a space-efficient representation. For example, the non-zero vector elements can be stored as a list of index-value pairs. Moreover, many standard operations involving sparse feature vectors can be done in $\mathcal{O}(k)$ steps, rather than $\mathcal{O}(n)$ steps. We call these operations *sparse vector operations* and distinguish them from the more costly *dense vector operations*. For example, if $\mathbf{v}$ is a dense vector stored in a random-access representation (such as an array), $\alpha$ is a scalar, and $x$ is a sparse vector, then the operation $\mathbf{v} \leftarrow \mathbf{v} + \alpha x$ is a sparse vector operation: iterate over the $k$ non-zero elements of $x$ and update the corresponding entries in $\mathbf{v}$. Similarly, calculating the dot product $\mathbf{v} \cdot x$ requires only $\mathcal{O}(k)$ steps.

Since sparse operations are much faster than dense operations, we want to implement ASGD using only a small constant number of dense operations. Specifically, this implies that we can only perform sparse vector operations inside the gradient descent loop. More precisely, if ASGD performs $T$ gradient descent steps, its total running time should be $\mathcal{O}(n + Tk)$ rather than $\mathcal{O}(Tn)$.

## 3  Stochastic Gradient Descent

As a warm-up to ASGD, we first discuss the simpler Stochastic Gradient Descent (SGD) algorithm [5, 1]. SGD is an interative optimization technique that runs for $T$ steps and produces a sequence of intermediate linear predictors $\big((\mathbf{w}_t, b_t)\big)_{t=0}^{T}$. The first predictor in the sequence, $(\mathbf{w}_0, b_0)$, is initialized to zero. SGD performs $T$ gradient descent steps, each one with respect to an individual training example that is drawn uniformly from the training set. Formally, let $\pi_1, \ldots, \pi_T$ be a sequence of independently drawn random indices, each between 1 and $m$; on iteration $t$ the algorithm processes training example $\pi_t$.

To derive the SGD update, We use the square-bracket notation $[\mathbf{w}, b]$ to denote the concatenation of $\mathbf{w}$ and $b$. Similarly, we use $[x, 1]$ to denote the concatenation of the value 1 to the end of the vector $x$. The subgradient of Eq. (1) is,

$$\nabla F(\mathbf{w}, b) \;=\; \lambda[\mathbf{w}, b] \;+\; \frac{1}{m} \sum_{i=1}^{m} \ell'(\mathbf{w} \cdot x_i + b\,,\, y_i)\,[x_i, 1] \quad. \tag{2}$$

If $\pi$ is a random index, chosen uniformly between 1 and $m$, then

$$\lambda[\mathbf{w}, b] \;+\; \ell'(\mathbf{w} \cdot x_\pi + b\,,\, y_\pi)\,[x_\pi, 1]$$

is an unbiased estimator of Eq. (2), also called a *stochastic gradient* of the objective function in Eq. (1). Each SGD step subtracts a scaled stochastic gradient from the current predictor. The algorithms allows for some flexibility in choosing the size of each step, and we choose the size of step $t$ to be $1/\lambda t$, where $\lambda$ is the regularization parameter in Eq. (1). This step size is motivated by the theoretical convergence analysis of SGD with strongly convex objective functions [2, 7]. Overall, the update on iteration $t$ takes the form

$$[\mathbf{w}_t, b_t] \;=\; [\mathbf{w}_{t-1}, b_{t-1}] \;-\; \frac{1}{\lambda t}\left(\lambda\,[\mathbf{w}_{t-1}, b_{t-1}] \;+\; \ell'(\mathbf{w}_{t-1} \cdot x_{\pi_t} + b_{t-1}\,,\, y_{\pi_t})\,[x_{\pi_t}, 1]\right) \quad.$$

Rearranging terms above gives

$$[\mathbf{w}_t, b_t] \;=\; \left(1 - \frac{1}{t}\right)[\mathbf{w}_{t-1}, b_{t-1}] \;-\; \frac{\ell'(p_t,\, y_{\pi_t})}{\lambda t}\,[x_{\pi_t}, 1] \quad \text{where} \quad p_t = \mathbf{w}_{t-1} \cdot x_{\pi_t} + b_{t-1} \quad. \tag{3}$$

Recall that our goal is to avoid all dense vector operations when performing each SGD step. The vector $[\mathbf{w}_{t-1}, b_{t-1}]$ on the right-hand side above is likely a dense vector, and therefore a naïve implementation of the scaling operation $(1 - t^{-1})[\mathbf{w}_{t-1}, b_{t-1}]$ would require $\mathcal{O}(n)$ steps. To avoid this, we introduce the *gradient sum* variable, defined for each $t$ as

$$[\mathbf{v}_t, a_t] \;=\; \sum_{j=1}^{t} \ell'(p_j,\, y_{\pi_j})\,[x_{\pi_j}, 1] \quad. \tag{4}$$

3

On one hand, the gradient sum can be computed using sparse vector operations. On the other hand, we prove that the linear predictor $[\mathbf{w}_t, b_t]$ can be easily recovered from $[\mathbf{v}_t, a_t]$.

**Lemma 1.** *Let $[\mathbf{w}_t, b_t]$ be as defined in Eq. (3) and let $[\mathbf{v}_t, a_t]$ be as defined in Eq. (4). Then, it holds for all $t \geq 1$ that $\frac{-1}{\lambda t}[\mathbf{v}_t, a_t] = [\mathbf{w}_t, b_t]$.*

*Proof.* It is easier to prove the equivalent opposite direction: we assume that we defined $[\mathbf{v}_t, a_t] = -\lambda t\, [\mathbf{w}_t, b_t]$ and prove that Eq. (4) follows.

For $t = 1$, Eq. (3) implies that $[\mathbf{w}_1, b_1] = \frac{1}{\lambda} - \ell'(0, y_{\pi_t})$. Scaling both sides of this equality by $\lambda$ and using the assumption gives $[\mathbf{v}_1, a_1] = \ell'(0, y_{\pi_t})$, which is consistent with Eq. (4). For $t \geq 2$, we replace $[\mathbf{w}_t, b_t]$ with $\frac{-1}{\lambda t}[\mathbf{v}_t, a_t]$ in Eq. (3) to get

$$\frac{-1}{\lambda t}[\mathbf{v}_t, a_t] \;=\; \left(1 - \frac{1}{t}\right) \frac{-1}{\lambda(t-1)}[\mathbf{v}_{t-1}, a_{t-1}] \;-\; \frac{\ell'(p_t, y_{\pi_t})}{\lambda t}\, [x_{\pi_t}, 1] \;\;.$$

Using the fact that $(1 - \frac{1}{t})\frac{1}{t-1} = \frac{1}{t}$, we multiply both sizes of the equation above by $-\lambda t$ and get

$$[\mathbf{v}_t, a_t] \;=\; [\mathbf{v}_{t-1}, a_{t-1}] \;+\; \ell'(p_t, y_{\pi_t})\, [x_{\pi_t}, 1] \;\;.$$

This implies Eq. (4), which concludes the proof. □

We can now rewrite the prediction $p_t$ in terms of $\mathbf{v}_t$ and $a_t$. For $t = 1$, it simply holds that $p_1 = 0$. For $t \geq 2$, we use Lemma 1 and get

$$p_t \;=\; \frac{-1}{\lambda(t-1)}\big(\mathbf{v}_{t-1} \cdot x_{\pi_t} + a_{t-1}\big) \;\;.$$

We are now ready to design an efficient implementation of SGD. Our algorithm computes the sequence of gradient sums $\big((\mathbf{v}_t, a_t)\big)_{t=0}^{T}$ using only sparse vector operations. Whenever needed, the linear predictor $(\mathbf{w}_t, b_t)$ can be recovered from $(\mathbf{v}_t, a_t)$ by performing a one-time dense rescaling by $\frac{-1}{\lambda t}$. The pseudocode for this algorithm appears in Algorithm 1.

## 4 Averaged Stochastic Gradient Descent

The SGD algorithm in Algorithm 1 implicitly constructs a sequence of intermediate linear predictors and returns the last predictor in the sequence. Ruppert [6] and Polyak [3, 4] independently argued that the last predictor may be suboptimal, and that the *average* of the intermediate predictors is a better choice. Intuitively, the average predictor is more stable than the last predictor, and this stability allows us to prove strong convergence results.

Specifically, we define

$$\overline{\mathbf{w}}_t \;=\; \frac{1}{t}\sum_{j=1}^{t}\mathbf{w}_j \quad \text{and} \quad \overline{b}_t \;=\; \frac{1}{t}\sum_{j=1}^{t} b_j \;\;, \tag{5}$$

and we wish to modify Algorithm 1 to return $(\overline{\mathbf{w}}_T, \overline{b}_T)$. This technique is called Averaged SGD, or ASGD.

**Algorithm 1** SGD for regularized linear learning with sparse data

---

1: **function** SGD$(T, \lambda, \{(x_t, y_t)\}_{i=1}^m)$    // number of steps, regularization parameter, training set

2:    draw random indices $\pi_1, \ldots, \pi_T$

3:    $g \leftarrow \ell'(0, y_{\pi_1})$

4:    $\mathbf{v} \leftarrow g x_{\pi_1}$

5:    $a \leftarrow g$

6:    **for** $t = 2, \ldots, T$ **do**

7:        $d \leftarrow \mathbf{v} \cdot x_{\pi_t}$                    // $\mathcal{O}(k)$ operation

8:        $p \leftarrow \frac{-(d+a)}{\lambda(t-1)}$                // note that $p = \mathbf{w}_{t-1} \cdot x_{\pi_t} + b_{t-1}$

9:        $g \leftarrow \ell'(p, y_{\pi_t})$

10:        $\mathbf{v} \leftarrow \mathbf{v} + g x_{\pi_t}$                    // $\mathcal{O}(k)$ operation

11:        $a \leftarrow a + g$

12:    $\mathbf{w} \leftarrow \frac{-1}{\lambda T} \mathbf{v}$                    // $\mathcal{O}(n)$ operation outside the loop

13:    $b \leftarrow \frac{-a}{\lambda T}$

14:    **return** $[\mathbf{w}, b]$

---

To compute $\bar{b}_t$, we use Lemma 1 and write

$$\bar{b}_t = \frac{1}{t} \sum_{j=1}^t b_j = \frac{-1}{\lambda t} \sum_{j=1}^t \frac{a_j}{j} .$$

Using the above, we can modify Algorithm 1 to incrementally compute the term

$$c_t = \sum_{j=1}^t \frac{a_j}{j} , \tag{6}$$

and when needed, to recover

$$\bar{b}_t = \frac{-c_t}{\lambda t} . \tag{7}$$

Computing $\overline{\mathbf{w}}_t$ requires more care, because the vector addition in Eq. (5) involves dense vectors, and a straightforward computation of $\overline{\mathbf{w}}_t$ would require $\mathcal{O}(tn)$ operations. To avoid these dense vector operations, we apply Lemma 1, and get

$$\mathbf{w}_j = \frac{-1}{\lambda j} \sum_{i=1}^j \ell'(p_i, y_{\pi_i}) x_{\pi_i} .$$

Plugging the above into Eq. (5) gives

$$\overline{\mathbf{w}}_t = \frac{1}{t} \sum_{j=1}^t \left( \frac{-1}{\lambda j} \sum_{i=1}^j \ell'(p_i, y_{\pi_i}) x_{\pi_i} \right) .$$

---

**Algorithm 2** ASGD for regularized linear learning with sparse data

---
1: **function** ASGD$(T, \lambda, \{(x_t, y_t)\}_{i=1}^m)$        // num of steps, regularization param, training set
2:      draw random indices $\pi_1, \ldots, \pi_T$
3:      $g \leftarrow \ell'(0, y_{\pi_1})$ ; $\mathbf{v} \leftarrow gx_{\pi_1}$ ; $a \leftarrow g$                // same as SGD
4:      $\mathbf{u} \leftarrow 0^n$                                        // see Eq. (9)
5:      $c \leftarrow a$                                           // see Eq. (6)
6:      $h \leftarrow 1$                                     // first harmonic number
7:      **for** $t = 2, \ldots, T$ **do**
8:          $d \leftarrow \mathbf{v} \cdot x_{\pi_t}$ ; $p \leftarrow \frac{-(d+a)}{\lambda(t-1)}$ ; $g \leftarrow \ell'(p, y_{\pi_t})$ ; $\mathbf{v} \leftarrow \mathbf{v} + gx_{\pi_t}$ ; $a \leftarrow a + g$    // same as SGD
9:          $\mathbf{u} \leftarrow \mathbf{u} + hgx_{\pi_t}$                      // $\mathcal{O}(k)$ operation, see Eq. (9)
10:        $c \leftarrow c + \frac{a}{t}$                                   // see Eq. (6)
11:        $h \leftarrow h + \frac{1}{t}$                                 // $t$'th harmonic number
12:      $\overline{\mathbf{w}} \leftarrow \frac{-1}{\lambda T}(h\mathbf{v} - \mathbf{u})$               // $\mathcal{O}(n)$ operation outside the loop, see Eq. (10)
13:      $\bar{b} \leftarrow \frac{-c}{\lambda T}$                                      // see Eq. (7)
14:      **return** $[\overline{\mathbf{w}}, \bar{b}]$

---

Rearranging the order of the two sums and using $h_i = \sum_{j=1}^i \frac{1}{j}$ to denote the $i$'th harmonic number, we get

$$
\begin{aligned}
\overline{\mathbf{w}}_t &= \frac{-1}{\lambda t} \sum_{i=1}^t \left( \sum_{j=i}^t \frac{1}{j} \right) \ell'(p_i,\, y_{\pi_i})\, x_{\pi_i} \\
&= \frac{-1}{\lambda t} \sum_{i=1}^t (h_t - h_{i-1})\, \ell'(p_i,\, y_{\pi_i})\, x_{\pi_i} \\
&= \frac{-1}{\lambda t} \left( h_t\, \mathbf{v}_t - \sum_{i=1}^t h_{i-1}\, \ell'(p_i,\, y_{\pi_i})\, x_{\pi_i} \right) \quad .
\end{aligned}
\tag{8}
$$

We modify Algorithm 1 to also incrementally compute the *harmonic gradient sum*,

$$
\mathbf{u}_t = \sum_{i=1}^t h_{i-1}\, \ell'(p_j,\, y_{\pi_j})\, x_{\pi_j} \quad .
\tag{9}
$$

This definition allows us to write Eq. (8) as

$$
\overline{\mathbf{w}}_t = \frac{-1}{\lambda t} \left( h_t\, \mathbf{v}_t - \mathbf{u}_t \right) \quad .
\tag{10}
$$

With the formula above, $\overline{\mathbf{w}}_t$ can be recovered from $h_t$, $\mathbf{v}_t$, and $\mathbf{u}_t$ when needed, via a dense vector operation. The pseudo-code of the resulting ASGD implementation is presented in Algorithm 2.

# 5    Centering and Translation Invariance

A disadvantage of the problem formulation in Eq. (1) is that it is sensitive to translation (a.k.a. offset) of the training data (namely, adding a constant vector to each feature vector in the training set). The root of the problem is the term $b^2$ in Eq. (1), which discourages large values of $b$. There are several different ways to make our algorithms translation invariant. A simple but effective technique is to center the training data. Centering is the process of computing the mean feature vector, $\bar{\mathbf{x}} = \frac{1}{m} \sum_{j=1}^{m} x_j$, and subtracting it from each $x_t$. After we center the data, a predictor with a bias of $b = 0$ is one that passes through the training data's center-of-mass.

If we apply the transformation $x \mapsto x - \bar{\mathbf{x}}$ to the training set and train a predictor $(\mathbf{w}, b)$, we must apply the same centering transformation to new feature vectors before using $(\mathbf{w}, b)$ to predict their labels. There are two equivalent ways of doing this, *explicit centering* and *implicit centering*. Explicit centering involves two consecutive steps: first, create a centered version of the feature vector $x' = x - \bar{\mathbf{x}}$; then, apply the predictor to $x'$ and predict the value $\mathbf{w} \cdot x' + b$. A disadvantage of explicit centering is that it requires us to store $\bar{\mathbf{x}}$ alongside $\mathbf{w}$ and $b$, as part of the predictor definition. On the other hand, implicit centering hides the centering transformation in the bias term. Specifically, define a new bias term

$$b' \;=\; b - \mathbf{w} \cdot \bar{\mathbf{x}} \;, \tag{11}$$

and apply the predictor directly to the original (uncentered) feature vector $x$. In other words, the prediction is computed as $\mathbf{w} \cdot x + b'$. The two centering techniques are equivalent because

$$\mathbf{w} \cdot x' + b \;=\; \mathbf{w} \cdot (x - \bar{\mathbf{x}}) + b \;=\; \mathbf{w} \cdot x + (b - \mathbf{w} \cdot \bar{\mathbf{x}}) \;=\; \mathbf{w} \cdot x + b' \;.$$

The advantage of implicit centering is that it allows us to forget $\bar{\mathbf{x}}$ and to make predictions as if we had not used the centering technique at all. However, note that $b'$ cannot be computed until training concludes and $\mathbf{w}$ is available.

The technical difficulty of centering sparse feature vectors is that $\bar{\mathbf{x}}$ is likely a dense vector, and therefore each centered feature vector, $(x_t - \bar{\mathbf{x}})$, is dense as well. Therefore, explicitly centering the entire training set would require $m$ dense operations and would prevent us from using sparse vector operations during training. In this section, we describe how to apply the centering technique implicitly, without using dense operations.

Imagine repeating the entire derivation from the previous sections, replacing every appearance of $x_{\pi_t}$ with $(x_{\pi_t} - \bar{\mathbf{x}})$. In particular, we would get

$$[\mathbf{w}_t, b_t] \;=\; \frac{-1}{\lambda t} \sum_{j=1}^{t} \ell'(p_j, \, y_{\pi_j}) \, [(x_{\pi_j} - \bar{\mathbf{x}}), 1] \;.$$

Reusing the definition of $[\mathbf{v}_t, a_t]$ from Eq. (4), we rewrite the above as

$$\mathbf{w}_t \;=\; \frac{-1}{\lambda t} (\mathbf{v}_t - a_t \bar{\mathbf{x}}) \quad \text{and} \quad b_t \;=\; \frac{-a_t}{\lambda t} \;. \tag{12}$$

We can now use Eq. (11) to calculate the bias term of the implicit representation,

$$b'_t \;=\; b_t - \mathbf{w}_t \cdot \bar{\mathbf{x}} \;=\; \frac{-a_t}{\lambda t} \;-\; \frac{-1}{\lambda t} (\mathbf{v}_t - a_t \bar{\mathbf{x}}) \cdot \bar{\mathbf{x}} \;=\; \frac{-1}{\lambda t} \Big( a_t (1 + \|\bar{\mathbf{x}}\|^2) - \mathbf{v}_t \cdot \bar{\mathbf{x}} \Big) \;.$$

Consider the amount of work it would take us to obtain each of the terms above. The term $a_t$ can be computed as in Algorithm 2. The mean feature vector $\overline{\mathbf{x}}$ can be precomputed with $\mathcal{O}(mk)$ operations, and $(1 + \|\overline{\mathbf{x}}\|^2)$ can be precomputed using a single dense operation. The only term that poses a potential problem is $\mathbf{v}_t \cdot \overline{\mathbf{x}}$, which is a dot product of two dense vectors. To overcome this problem, we introduce the *projection sum* variable, defined as

$$z_t = \mathbf{v}_t \cdot \overline{\mathbf{x}} = \sum_{j=1}^{t} \ell'(p_j, y_{\pi_j}) \overline{\mathbf{x}} \cdot x_{\pi_j} \ .$$

On one hand, we can modify Algorithm 2 to incrementally compute $z_t$, as

$$z_t = z_{t-1} + \ell'(p_t, y_{\pi_t}) \overline{\mathbf{x}} \cdot x_{\pi_t} \ . \tag{13}$$

On the other hand, with $z_t$ handy, we can easily compute

$$b'_t = \frac{-r_t}{\lambda t} \quad \text{where} \quad r_t = a_t(1 + \|\overline{\mathbf{x}}\|^2) - z_t \ . \tag{14}$$

Plugging Eq. (12) and Eq. (14) into $p_t = \mathbf{w}_{t-1} \cdot x_{\pi_t} + b'_{t-1}$ gives

$$p_t = \frac{-1}{\lambda(t-1)} \left( \mathbf{v}_{t-1} \cdot x_{\pi_t} + r_{t-1} - a_{t-1} \overline{\mathbf{x}} \cdot x_{\pi_t} \right) \ . \tag{15}$$

We have everything we need to implement a centered version of SGD using sparse operations, but we are really after a centered version of ASGD, a.k.a. CASGD. Namely, we need to modify Algorithm 2 to return $(\overline{\mathbf{w}}_T, \overline{b}'_T)$, where

$$\overline{\mathbf{w}}_t = \frac{1}{t} \sum_{j=1}^{t} \mathbf{w}_j \quad \text{and} \quad \overline{b}'_t = \frac{1}{t} \sum_{j=1}^{t} b'_j \ . \tag{16}$$

To help us compute $\overline{b}'_t$, we further modify Algorithm 2 to incrementally compute

$$s_t = \sum_{j=1}^{t} \frac{r_j}{j} \ . \tag{17}$$

The value of $\overline{b}'_t$ can be recovered as

$$\overline{b}'_t = \frac{1}{t} \sum_{j=1}^{t} b'_j = \frac{-1}{\lambda t} \sum_{j=1}^{t} \frac{r_j}{j} = \frac{-s_t}{\lambda t} \ . \tag{18}$$

To compute $\overline{\mathbf{w}}_t$, we plug the definition of $\mathbf{w}_t$ from Eq. (12) into Eq. (16) to get

$$\begin{aligned}
\overline{\mathbf{w}}_t &= \frac{1}{t} \sum_{j=1}^{t} \frac{-1}{\lambda j} (\mathbf{v}_j - a_j \overline{\mathbf{x}}) \\
&= \frac{-1}{\lambda t} \left( \underbrace{\sum_{j=1}^{t} \frac{1}{j} \sum_{i=1}^{j} \ell'(p_i, y_{\pi_i}) \, x_{\pi_i}}_{(i)} - \underbrace{\sum_{j=1}^{t} \frac{a_j}{j} \overline{\mathbf{x}}}_{(ii)} \right) \ .
\end{aligned}$$

---

**Algorithm 3** CASGD for regularized linear learning with sparse data

---

1: **function** $\text{CASGD}(T, \lambda, \{(x_t, y_t)\}_{i=1}^m)$      // num of steps, regularization param, training set

2:      $\overline{\mathbf{x}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i$                   // $\mathcal{O}(mk)$ operation outside the loop

3:      $\theta \leftarrow 1 + \|\overline{\mathbf{x}}\|^2$                        // $\mathcal{O}(n)$ operation outside the loop

4:      draw random indices $\pi_1, \ldots, \pi_T$

5:      $g \leftarrow \ell'(0, y_{\pi_1})$ ; $\mathbf{v} \leftarrow gx_{\pi_1}$ ; $a \leftarrow g$ ; $\mathbf{u} \leftarrow 0^n$ ; $c \leftarrow a$ ; $h \leftarrow 1$      // same as ASGD

6:      $q \leftarrow \overline{\mathbf{x}} \cdot x_{\pi_1}$                  // $\mathcal{O}(k)$ operation

7:      $z \leftarrow gq$                         // see Eq. (13)

8:      $r \leftarrow a\theta - z$                    // see Eq. (14)

9:      $s \leftarrow r$                           // see Eq. (17)

10:      **for** $t = 1, \ldots, T$ **do**

11:          $d \leftarrow \mathbf{v} \cdot x_{\pi_t}$                // same as ASGD

12:          $q \leftarrow \overline{\mathbf{x}} \cdot x_{\pi_t}$               // $\mathcal{O}(k)$ operation

13:          $p \leftarrow \frac{-(d+r-aq)}{\lambda(t-1)}$          // see Eq. (15)

14:          $g \leftarrow \ell'(p, y_{\pi_t})$ ; $\mathbf{v} \leftarrow \mathbf{v} + gx_{\pi_t}$ ; $a \leftarrow a + g$      // same as ASGD

15:          $\mathbf{u} \leftarrow \mathbf{u} + hgx_{\pi_t}$ ; $c \leftarrow c + \frac{a}{t}$ ; $h \leftarrow h + \frac{1}{t}$      // same as ASGD

16:          $z \leftarrow z + gq$              // see Eq. (13)

17:          $r \leftarrow a\theta - z$             // see Eq. (14)

18:          $s \leftarrow s + \frac{r}{t}$             // see Eq. (17)

19:      $\overline{\mathbf{w}} \leftarrow \frac{-1}{\lambda T} \left( h\mathbf{v} - \mathbf{u} - c\overline{\mathbf{x}} \right)$      // $\mathcal{O}(n)$ operation outside the loop, see Eq. (19)

20:      $\overline{b}' \leftarrow \frac{-s}{\lambda T}$                  // see Eq. (18)

21:      **return** $[\overline{\mathbf{w}}, \overline{b}']$

---

Both $(i)$ and $(ii)$ above should look familiar, as we encountered them in the previous section. We rewrite the double sum in $(i)$ as we did in Eq. (10), and we rewrite the sum in $(ii)$ using Eq. (6), to get

$$\overline{\mathbf{w}}_t = \frac{-1}{\lambda t} \left( h_t \mathbf{v}_t - \mathbf{u}_t - c_t \overline{\mathbf{x}} \right) \quad . \tag{19}$$

The pseudo-code the resulting CASGD implementation appears in Algorithm 3

# References

[1] Léon Bottou. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nîmes*, 91(8), 1991.

[2] Elad Hazan, Amit Agarwal, and Satyen Kale. Logarithmic regret algorithms for online convex optimization. *Machine Learning*, 69(2-3):169–192, 2007.

[3] Boris T. Polyak. A new method of stochastic approximation type. *Avtomatika i telemekhanika*, (7):98–107, 1990.

[4] Boris T. Polyak and Anatoli B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

[5] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[6] David Ruppert. Efficient estimations from a slowly convergent robbins-monro process. Technical report, Cornell University Operations Research and Industrial Engineering, 1988.

[7] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.