CSE599s, Spring 2012, Online Learning Lecture 7 - 04/17/2012

Adaptive per-coordinate Learning

Lecturer: Brendan McMahan *Scribe: Paris Koutris*

1 Online Gradient Descent Revisited

As we have seen in previous lectures, the standard version of the OGD algorithm has a regret bound

$$\text{Regret}_T \leq \frac{2R^2}{\eta_T} + \frac{1}{2}\sum_{t=1}^{T} \eta_t g_t^2, \tag{1}$$

where η_t is the *global learning rate* at round t. In general, the only restriction we need on the learning rate is that it is a non-increasing function of time (number of rounds).

Different choices for the learning rates lead to different regret bounds. For example, by choosing a learning rate $\eta_t = \frac{R\sqrt{2}}{G\sqrt{t}}$, and assuming that the subgradients g_t are upper bounded, i.e. $\|g_t\| \leq G$, we obtain the following regret bound,

$$\text{Regret}_T \leq 2\sqrt{2}RG\sqrt{T}. \tag{2}$$

The problem with this choice boils down to the fact that the learning rate is oblivious of the subgradients g_t we have already seen (recall the bad example from the previous lecture where the sequence of one-dimensional g_t included a lot of zeros). In order to overcome this problem, the choice of the learning rate as $\eta_t = \frac{\sqrt{2}R}{\sqrt{\sum_{s=1}^{t} g_s^2}}$ gives an even better regret bound,

$$\text{Regret}_T \leq 2\sqrt{2}R\sqrt{\sum_{t=1}^{T} g_t^2}. \tag{3}$$

This choice of learning rate does not lead to a better worst-case algorithm, but provides us with a more efficient learning algorithm when the problem has a non-adverserial structure. The question is now: can we make this approach even more powerful? The crucial idea here is to adapt the learning rate not only to the subgradients, but also to specific coordinates. We call this *per-coordinate learning*. First, let us provide some motivation of why per-coordinate learning may perform better in some settings.

Example 1 Consider a typical learning setting where labeled examples of the form (x_t, y_t) arrive, where $x_t \in \mathbb{R}^n$ is the feature vector of the example and $y_t \in \{0, 1\}$ its label. The learning problem will admit a loss function of the form $f_t(w) = \ell(w \cdot x_t; y_t)$, where different choices of ℓ lead to different learning algorithms. The subgradient of this function will typically be $g_t = \ell'(w \cdot x_t; y_t)x_t$.

In many real-world cases (for example, the bag-of-words model), the feature vectors x_t (where $x_{t,i}$ may correspond to the indicator variable of the fact: does word i belong to document t?) is very sparse. For example, n may be in the order of 10^7, while a feature vector may typically have 100 non-zero entries. In this case, we would like to keep a distinct learning rate for each coordinate (i.e. each word) instead of a global learning rate.

The algorithm we describe next (PC-OGD) tackles the above issue. In order to keep the description simple, we will assume that the feasible set for the vectors w_t will be $\mathcal{W} = \{w | w_i \in [-B, B]\}$ for some constant B. This simplifies the treatment of the problem, since we can handle each coordinate *independently* of the other coordinates. Compare this with a feasible set of the form $\{w | w_1 + w_2 \leq 1\}$, where the choice for w_1 depends on the choice of w_2 (for example, they can not be both 1). We should note here that the algorithm we will present admits the same regret bound for any feasible convex set, but projection needs to be handled carefully and the analysis is more technical.

Algorithm PC-OGD. The update at step $t+1$ is

$$w_{t+1,i} = \Pi(w_{t,i} - \eta_{t,i} \cdot g_{t,i}) \quad \text{where} \quad \eta_{t,i} = \frac{B\sqrt{2}}{\sqrt{\sum_{s=1}^{t} g_{s,i}^2}}. \tag{4}$$

Notice that the algorithm keeps a different learning rate for each coordinate and adapts it according to the subgradient at this specific coordinate. The following theorem gives us the regret bound for the above algorithm.

Theorem 1. *For PC-OGD,* $\text{Regret}_T \leq \sum_{i=1}^{n} 2\sqrt{2}B\sqrt{\sum_{t=1}^{T} g_{t,i}^2}$.

Proof. The basic idea is that the linearity of the regret allows to reorganize the sum and then bound the regret for each coordinate separately. Indeed, we have:

$$\begin{aligned}
\text{Regret}_T &= \sum_{t=1}^{T} \{g_t \cdot w_t - g_t \cdot w^*\} \\
&= \sum_{t=1}^{T} \{\sum_{i=1}^{n} g_{t,i} w_{t,i} - \sum_{i=1}^{n} g_{t,i} w_i^*\} \\
&= \sum_{i=1}^{n} \{\sum_{t=1}^{T} g_{t,i} w_{t,i} - \sum_{t=1}^{T} g_{t,i} w_i^*\} \\
&= \sum_{i=1}^{n} \text{Regret}_T^i
\end{aligned}$$

where Regret_T^i is defined as the regret in the case where the loss function is $f_{t,i}(w) = g_{t,i} w_i$. Using the regret bounds we have previously proved, we can bound $\text{Regret}_T^i \leq 2\sqrt{2}B\sqrt{\sum_{t=1}^{T} g_{t,i}^2}$. This completes the proof of the regret bound. □

We next show that the regret bound we obtained is at least as good as the bound for OGD with a global rate.

Lemma 2. *The PC-OGD regret bound is at least as good as the OGD bound.*

Proof. Let us define $\mathbf{b} = (B, B, \ldots, B)$. Notice that $R = \|\mathbf{b}\|$. Define $g_i = \sqrt{\sum_{s=1}^{t} g_{t,i}^2}$ and $\mathbf{g} = (g_1, \ldots, g_n)$. We have that

$$\|\mathbf{g}\| = \sqrt{\sum_i g_i^2} = \sqrt{\sum_i \sum_t g_{t,i}^2} = \sqrt{\sum_t \sum_i g_{t,i}^2} = \sqrt{\sum_t g_t^2}$$

Now, we can show that:

$$\sum_{i=1}^{n} 2\sqrt{2}B\sqrt{\sum_{t=1}^{T} g_{t,i}^2} = 2\sqrt{2}\mathbf{b} \cdot \mathbf{g} \leq 2\sqrt{2}\,\|\mathbf{b}\|\,\|\mathbf{g}\| = 2\sqrt{2}R\sqrt{\sum_t g_t^2} \leq 2\sqrt{2}RG\sqrt{T}$$

where the inequality is an application of Cauchy-Schwartz. The right-hand side is the same as Eq. (2). □

The per-coordinate learning rate performs better with more independence among the coordinates. This approach can be further generalized by modeling correlations between the coordinates of the feature vector. More precisely, the updates will now be of the form $w_{t+1} = w_t - D_t g_t$, where D_t is a $n \times n$ matrix. The PC-OGD algorithm corresponds to the case where D_t is a diagonal matrix. The disadvantage of introducing correlations is that the space requirements for keeping the learning rate matrix become too large.

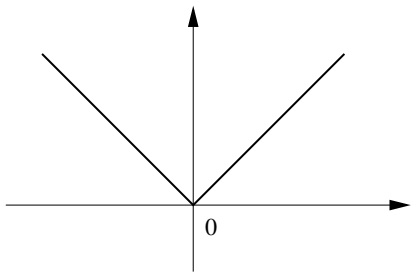
2 Sparse Models

In this section, we discuss how we can modify the learning algorithms we have seen so far in order to produce sparse models. A sparse model corresponds to a vector w with few non-zero entries. Sparse models are appealing for the following reasons:

- *Model storage*: The storage requirement for a non-sparse vector will be $O(n)$, where n is the dimension of the vector. For high dimensional models, this may be infeasible. A sparse vector w can be stored using other representations with much less space.
- *Feature selection*: In the case where $w_i = 0$, we know that feature i is irrelevant for our problem. This allows for potentially better feature selection in the future.

A first naive approach to this problem would be to restrict the feasible set of vectors w to be sparse. More precisely, define $\mathcal{W}$ to contain only vectors where the number of non-zero entries is bounded: for example, $\mathcal{W} = \{w \in \mathbb{R}^{10^7} | \#\text{non\_zero\_entries} \leq 10^4\}$. Unfortunately, such a feasible set is not convex. Additionally, this problem is NP-hard even in the offline case, i.e. when all the data is available to us.

The approach we follow is to use L_1 regularization. Intuitively, we penalize a vector whenever it introduces non-zero entries. The following figure shows the penalty introduced by the L_1 regularizer in the 1-dimensional case. We should note here that using instead L_2 regularization does not lead to sparser vectors.



For example, we can modify the loss function f_t to penalize for non-zeros as follows:

$$\hat{f}_t(w) = \ell(w \cdot x_t; y_t) + \lambda \|w\|_1$$

Now we use the subgradient $\hat{g}_t$ of $\hat{f}_t$ to perform the updates according to the OGD algorithm: $w_{t+1} = w_t - \eta \hat{g}_t$. However, this approach does not work well, as the next example shows.

Example 2 Consider the 1-dimensional case, where the sequence of subgradients g_t is: $(0.5, -1, +1, \dots)$. Using $\hat{g}_t$ in the update step of OGD, the vector w_t will oscillate between negative and positive values, without ever converging to 0, which is the best solution and is also sparse.

Instead, we can use the idea of L_1 regularization by viewing OGD from a different angle. One can show that OGD is equivalent to the following update:

$$w_{t+1} = \arg\min_w \{g_t \cdot w + \frac{1}{2\eta} \|w - w_t\|^2\} \tag{5}$$

Indeed, in order to find the w that minimizes the above expression, we want a w such that $\nabla\{g_t \cdot w + \frac{1}{2\eta} \|w - w_t\|^2\} = 0$, or equivalently $g_t + \frac{1}{\eta}(w - w_t) = 0$.

Having the updates into this form, we can now add directly the L_1 regularization to come up with an update step that produces sparse solutions:

$$w_{t+1} = \arg\min_w \{g_t \cdot w + \lambda \|w\|_1 + \frac{1}{2\eta} \|w - w_t\|^2\} \tag{6}$$

We next switch to the FTRL algorithm. Let us remind that the updates for round $t+1$ when the regularization $r(w)$ is fixed are of the form:

$$w_{t+1} = \arg\min_{w}\{g_{1:t} \cdot w + r(w)\}$$

For $r(w) = \frac{1}{2\eta}\|w\|^2$, it can be shown that FTRL is equivalent to OGD with constant learning rate η (if we don't project). We can next generalize this approach by adding a regularizer r_t for each step t. In this case, the updates will be as follows:

$$w_{t+1} = \arg\min_{w}\{g_{1:t} \cdot w + r_{1:t}(w)\} \tag{7}$$

By choosing $r_t(w) = \frac{\sigma_t}{2}\|w - w_t\|^2$, we obtain an algorithm that is equivalent to OGD with adaptive learning rate $\eta_t \approx \frac{1}{\sigma_{1:t}}$. In this setting, adding L_1 regularization as we did with OGD will lead to the following update step:

$$w_{t+1} = \arg\min_{w} g_{1:t} \cdot w + t\lambda\|w\|_1 + r_{1:t}(w) \tag{8}$$

We should notice here the factor t that appears in the regularization term. We will show in later lectures that this factor will actually make L_1 regularization much more effective for FTRL in comparison to OGD.