10-725/36-725: Convex Optimization **Fall 2018**

Lecture 9: September 26

Lecturer: Ryan Tibshirani *Scribes: Emre Yolcu*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

9.1 Finite sum problems

Consider minimizing an average of functions

$$\min_x \frac{1}{m} \sum_{i=1}^m f_i(x).$$

This setting is common in machine learning, where have a dataset of m points y_i and each f_i is an error function evaluated at the ith datapoint.

As $\nabla \sum_{i=1}^m f_i(x) = \sum_{i=1}^m \nabla f_i(x)$, gradient descent updates are as follows.

$$x^{(k)} = x^{(k-1)} - t_k \frac{1}{m} \sum_{i=1}^m \nabla f_i(x^{(k-1)}), \quad k = 1, 2, 3, \ldots$$

9.2 Stochastic gradient descent

When m is large in the above update rule, computing the gradient for all i becomes costly, so **stochastic gradient descent** or SGD (or incremental gradient descent) updates are made the following way.

$$x^{(k)} = x^{(k-1)} - t_k \nabla f_{i_k}(x^{(k-1)}), \quad k = 1, 2, 3, \ldots$$

where $i_k \in \{1, \ldots, m\}$ is some index chosen at iteration k.

There are two commonly used rules for selecting index i_k at iteration k:

- **Randomized rule**: choose $i_k \in \{1, \ldots, m\}$ uniformly at random (this is easier to analyze theoretically).
- **Cyclic rule**: choose $i_k = 1, 2, \ldots, m, 1, 2, \ldots, m, \ldots$ (this is more difficult to analyze in general).

In practice, randomized rule is more commonly used. Note that for this rule, we can write

$$\mathbb{E}[\nabla f_{i_k}(x)] = \frac{1}{m} \sum_{i=1}^m \nabla f_i(x)$$

which implies that we can view SGD as using an **unbiased estimate** of the gradient at each step.

Main appeals of SGD are:

- Iteration cost is independent of m (the number of functions, or in some cases, the number of points in the data set), and depends only on the dimensions of the parameters.

- It saves memory, since it could be costly to store the data in memory, and SGD can load the data in smaller chunks.

9.3 Example: stochastic logistic regression

Given data $(x_i, y_i) \in \mathbb{R}^p \times \{0, 1\}$, $i = 1, \ldots, n$, the logistic regression objective is

$$\min_{\beta} f(\beta) = \sum_{i=1}^{n} \big(- y_i x_i^T \beta + \log(1 + e^{x_i^T \beta}) \big)$$

In this setting, when n is moderate, the gradient computation is doable, but not when n is huge. For full gradient descent, one batch update costs $O(np)$ while for SGD, one stochastic update costs $O(p)$, and clearly, for instance, 10000 stochastic steps are much more affordable.

Below picture shows the behavior of full (batch) gradient descent compared to stochastic gradient descent for an example with $n = 10$, $p = 2$ (although this is not the setting that SGD is usually used in).

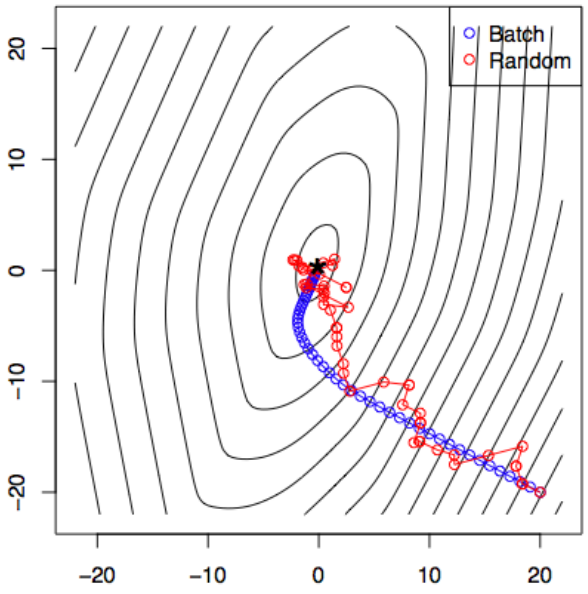

Figure 9.1: Blue: batch steps, $O(np)$
Red: stochastic steps, $O(p)$

Far from the optimum, SGD moves faster. When close to the optimum, gradient descent converges quickly while SGD struggles. This behavior is in agreement with the general characteristics stochastic methods.

9.4 Step sizes

In SGD, it is standard to use (at least in the optimization community) to use diminishing step sizes, such as $t_k = 1/k$ for $k = 1, 2, 3, \ldots$. For an argument against fixed step sizes, consider the cyclic rule for simplicity. Let $t_k = t$ for m updates in a row. Then, by expanding the iterate at each iteration in terms of the previous iterate, we can write

$$\begin{aligned} x^{(k+m)} &= x^{(k+m-1)} - t\nabla f_i(x^{(k+m-1)}) \\ &= x^{(k+m-2)} - t\nabla f_i(x^{(k+m-2)}) - t\nabla f_i(x^{(k+m-1)}) \\ &\vdots \\ &= x^{(k)} - t\sum_{i=1}^{m} \nabla f_i(x^{(k+i-1)}). \end{aligned}$$

Meanwhile, for the full gradient descent with step size t we would have

$$x^{(k+1)} = x^{(k)} - t\sum_{i=1}^{m} \nabla f_i(x^{(k)}).$$

If we hold t constant, the difference

$$t\sum_{i=1}^{m}[\nabla f_i(x^{(k+i-1)}) - \nabla f_i(x^{(k)})]$$

does not generally go to zero, meaning the algorithm does not converge.

9.5 Convergence rates

9.5.1 Gradient descent

Let us recall that for a convex function f, gradient descent with diminishing step sizes has the convergence rate

$$f(x^{(k)}) - f^\star = O(1/\sqrt{k}).$$

If f is differentiable and ∇f is Lipschitz continuous, with suitable sizes we get

$$f(x^{(k)}) - f^\star = O(1/k).$$

9.5.2 Stochastic dradient descent

For convex f, SGD with diminishing step sizes satisfies

$$\mathbb{E}[f(x^{(k)})] - f^\star = O(1/\sqrt{k}),$$

which unfortunately does not improve with the assumption that ∇f is Lipschitz continuous.

In addition, when f is strongly convex and ∇f is Lipschitz, full gradient descent satisfies

$$f(x^{(k)}) - f^\star = O(c^k)$$

where $c < 1$ is the condition number. Under same conditions, SGD can only give us

$$\mathbb{E}[f(x^{(k)})] - f^\star = O(1/k).$$

Thus, SGD does not achieve the linear convergence rate of full gradient descent even with the assumption of strong convexity. Things brings up the question: what can we do to improve SGD? One answer to this, especially from the machine learning community, is that these shortcomings of SGD are not a very critical point, because for better out-of-sample performance in our estimators, we do not necessarily want to fully optimize the objective. Another answer is the mini-batch SGD, which we cover in the next section.

9.6 Mini-batch SGD

In this setup, we choose a random subset $I_k \subseteq \{1, \dots, m\}$ of size $|I_k| = b \ll m$, and use the update rule

$$x^{(k)} = x^{(k-1)} - t_k \frac{1}{b} \sum_{i \in I_k} \nabla f_i(x^{(k-1)}), \quad k = 1, 2, 3, \dots$$

As in the random case, the gradient estimate is again unbiased

$$\mathbb{E}\Big[\frac{1}{b} \sum_{i \in I_k} \nabla f_i(x)\Big] = \nabla f(x)$$

But this time, the variance of the estimate is reduced by a factor of $1/b$, but the cost of an iteration is also b times more expensive, so mini-batch SGD is a compromise between random SGD and full GD.

Going back to the logistic regression example, consider a regularized version:

$$\min_{\beta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} \big(- y_i x_i^T \beta + \log(1 + e^{x_i^T \beta}) \big) + \frac{\lambda}{2} \|\beta\|_2^2$$

The gradient computation is $\nabla f(\beta) = \sum_{i=1}^{n} (y_i - p_i(\beta)) x_i + \lambda \beta$. To compare:

- one batch update costs $O(np)$
- one mini-batch update costs $O(bp)$
- one stochastic update costs $O(p)$

  Below is a comparison of the behavior of different configurations with $n = 10000, p = 20$ and all methods using fixed step sizes:

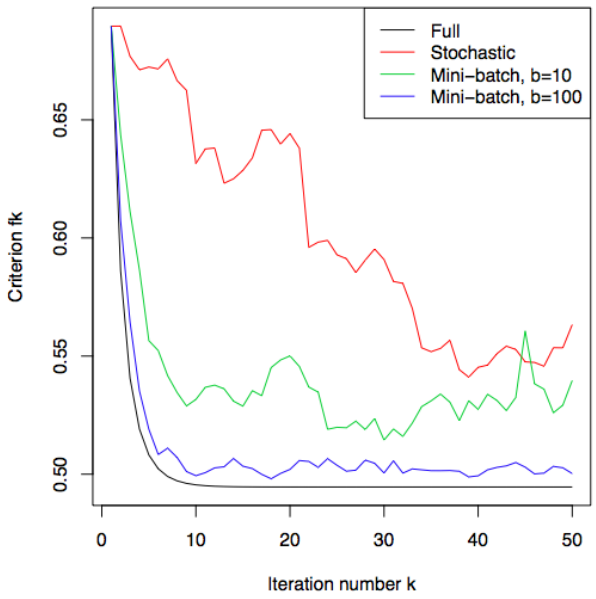

Figure 9.2: In terms of per iteration performance, using mini-batches does help.

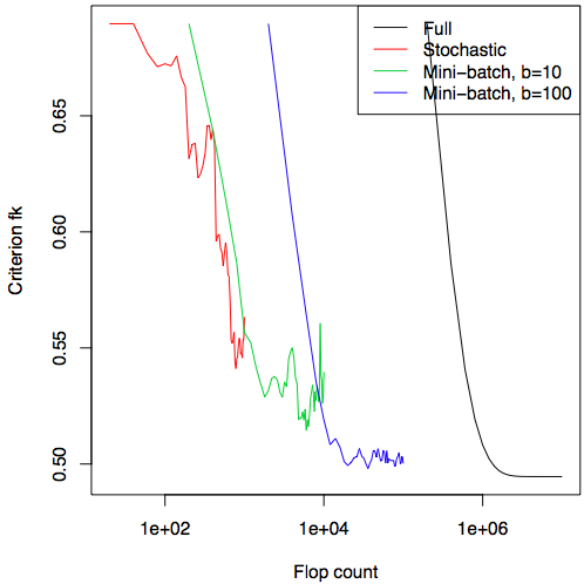

Figure 9.3: Parameterized by flops

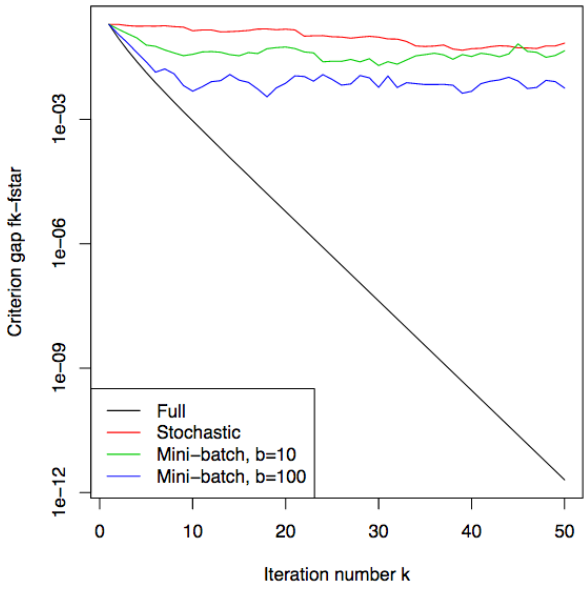

Figure 9.4: Suboptimality gap: this demonstrates that mini-batch SGD does not really help in terms of optimality

9.7 Recap

Let us review the properties of SGD we have seen so far.

- SGD can be very effective in terms of resources (iteration cost, memory), and for this reason it is still the standard for solving large problems.
- But it is also slow to converge, and does not benefit from strong convexity unlike full gradient descent.
- Mini-batches offer a partial solution: they are not that beneficial in terms of flops, but they can still be useful in practice in settings where the cost of computation over a batch can be brought closer to the cost for a point. Still, while mini-batches help reduce the variance, they do not necessarily lead to faster convergence.

Due to these properties, it was thought for a while that SGD would not be commonly useful. [NJLS09] and others had established lower bounds indicating that slow convergence for strongly convex functions was inevitable.

It was recognized very recently that these lower bounds (which were established for a more general stochastic problem) do not apply to finite sums, and with the new wave of variance reduction methods, it was shown that we can modify SGD to converge much faster for finite sums.

9.8 SGD in large-scale ML

In large-scale ML, SGD sees a large amount of use.

- As mentioned before, for machine learning the goal is not always to optimize to high accuracy, because it does not pay off in terms of statistical performance. For difficult prediction problems, suboptimal optimization can be "good enough". As a result of this, contrary to theory, fixed step sizes are commonly used.
- In order to find the appropriate step size, one heuristic is to use a small part of the data to pick a good learning rate before running SGD on the full data set.
- Recently, many variants have been proposed.
  - Adaptive methods: AdaGrad, Adam, AdaMax, ...
  - Variance reduction methods: SVRG, SAG, SAGA, ...

9.9 Early stopping

Going back to the logistic regression example, suppose p is large and we want to fit a model to data $(x_i, y_i) \in \mathbb{R}^p \times \{0, 1\}$, $i = 1, \ldots, n$. An option is to solve the (explicitly) L2 regularized logistic regression problem:

$$\min_{\beta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} \big(- y_i x_i^T \beta + \log(1 + e^{x_i^T \beta}) \big) \quad \text{subject to} \quad \|\beta\|_2 \leq t$$

An alternative is to run gradient descent on the unregularized objective

$$\min_{\beta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} \big(- y_i x_i^T \beta + \log(1 + e^{x_i^T \beta}) \big)$$

and terminate the algorithm well-short of the minimum. This is called **early stopping** (an instance of implicit regularization) for gradient descent. A reason to use this is that it is both more convenient and potentially more efficient than explicit regularization.

Early stopping is not very well-understood theoretically, and it has intriguing properties:

- In general, gradient descent path does not coincide with the explicitly regularized path, yet in practice it is competitive in terms of statistical performance.
- The idea can be extended to mimick a generic regularizer.

References

[NJLS09] A. Nemirovski and A. Juditsky and G. Lan and A. Shapiro (2009), "Robust stochastic optimization approach to stochastic programming"