10-725/36-725: Convex Optimization **Fall 2019**

Lecture 9: September 25

Lecturer: Lecturer: Ryan Tibshirani *Scribes: Scribes: Ziheng Cai, Jinjin Tian, Xuejian Wang*

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

9.1 Motivation

To begin with, consider the following problem of minimizing an average of functions:

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

The gradient of the above objective function will be

$$\nabla \sum_{i=1}^m f_i(x) = \sum_{i=1}^m \nabla f_i(x)$$

If we apply gradient descent algorithm, we would be repeating the following steps:

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

However, gradient descent will be very costly if we have m in the order of, say, 1 millions. Instead, we can apply the following algorithm:

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

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

This algorithm is called the **stochastic gradient descent** or **SGD** (or incremental gradient descent). The main appeal and motivations of using SGD are:

- The iteration cost of SGD is independent of m
- SGD can be a big savings in terms of memory usage.

Note: SGD is not necessarily a descent method!

9.2 Choosing Index i_k

In general, there are two ways of choosing the index i_k at iteration k of the SGD algorithm:

- **Randomized Rule:** Every iteration we choose $i_k \in \{1, \cdots, m\}$ uniformly at random
- **Cyclic Rule:** We choose i_k in the pattern of $1, 2, \cdots, m, 1, 2, \cdots, m, \cdots$

In practice, randomized rule is more common. For randomized rule, we have

$$\mathbb{E}[\nabla f_{i_k}(x)] = \nabla f(x)$$

so we can view SGD as an **unbiased estimate** of the gradient at each step.

9.3 Example: Stochastic Logistic Regression

To further illustrate the SGD algorithm, recall the formulation of **logistic regression**:

Given $(x_i, y_i) \in \mathbb{R}^p \times \{0, 1\}$ with $i = 1, \cdots, n$, we want to minimize $f(\beta)$ with the following expression:

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

The gradient of $f(\beta)$ is $\nabla f(\beta) = -\frac{1}{n} \sum_{i=1}^{n} (y_i - p_i(\beta)) x_i$, where $p_i(\beta) = \exp(x_i^T \beta)/(1 + \exp(x_i^T \beta))$. This computation is doable when n is moderate, but **not when n is huge**. In comparison, in each iteration step of SGD we only need to compute $-(y_i - p_i(\beta))x_i$ for some $i \in \{1, \cdots, n\}$. In terms of time complexity, we have

- **Full Gradient:** one batch update costs $O(np)$
- **Stochastic Gradient:** one stochastic update costs $O(p)$

Clearly, the stochastic steps are much more affordable.

Figure 9.1 shows the "classic picture" for batch versus stochastic methods. The experiment is stochastic logistic regression with $n = 10$ and $p = 2$. We can see that SGD trades stability for faster speed, since it generally thrives when far from optimum but struggles when close to optimum. This is considered as the rule of thumb for stochastic method.

9.4 Step Sizes

It is standard to use **diminishing step sizes** in SGD (e.g., $t_k = 1/k$). The short (and intuitive) explanation for why we do not use fixed step sizes is the following. Suppose we take cyclic rule for simplicity. We have $t_k = t$ for m updates in a row, so for SGD we get

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

Meanwhile, for full gradient method with step size mt, we have

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

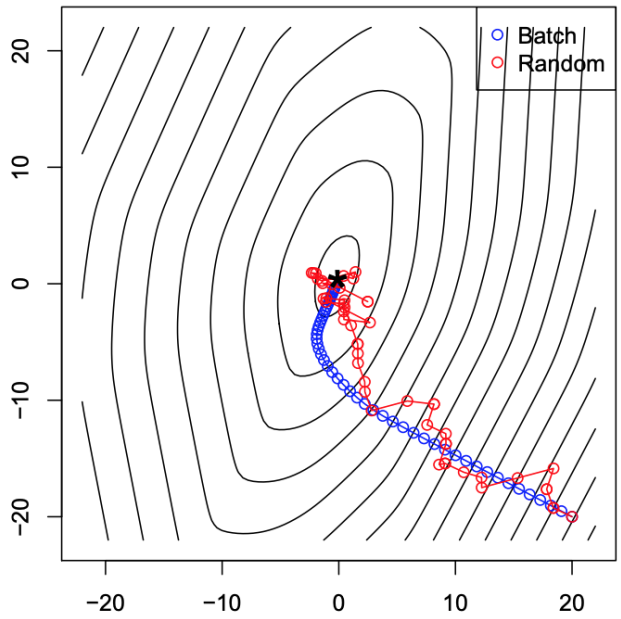

Figure 9.1: Comparison between batch methods and stochastic methods. The blue dots represent the batch steps, which takes $O(np)$ running time. The red dots represent the stochastic steps, which takes $O(p)$ running time

Consider the error between $x^{(k+m)}$ and x^{k+1}:

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

If we keep t constant, this difference will in general not go to zero. Hence, usually using a diminishing step sizes will make stochastic update a more accurate estimate of the full gradient estimate.

9.5 Convergence rate of SGD

We have learnt that for a convex function f, gradient descent with diminishing step sizes satisfies

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

and in addition, if f is differentiable and ∇f is Liptchitz continuous, with suitable sizes we get

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

Now, as for SGD, as shown by Nemirovski et al. [1], though we have that for convex function f, with diminishing step sizes, it satisfies

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

and the bound does not improve when we further assuming ∇f is Liptchitz continuous. In fact, the discrepency can be worse for f being strongly convex and ∇f is Liptchitz continuous. Specifically, full gradient decent satisfies

$$f(x^{(k)}) - f^* = O(\gamma^k),$$

where $0 < \gamma < 1$ is the condition number; while SGD gives us

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

Therefore, stochastic methods do not enjoy the linear convergence rate of gradient descent under strong convexity.

9.6 Mini-batch SGD

Previous section says that SGD do not enjoy the linear convergence rate of full gradient descent even with the assumption of strong convexity. However, this is usually not a real concern in machine learning community, since people often want better performance on the samples outside the training sets, which makes fully optimization of the objectives unnecessary, therefore we can always stop early on an "okay" solution. To people actually want the optimization, we show a common way to improve the convergence rate of SGD, which the Mini-batch SGD.

Mini-batch SGD For the optimization problem described early, during the k-th update, we choose a random subset $I_k \subset \{1, \ldots, m\}$, where $|I_k| = b << m$, and use the following update rule:

$$x^{(k)} = x^{(k-1)} - \frac{t_k}{b} \sum_{i \in I_k} \nabla f_i(x^{(k-1)}).$$

As before, the gradient estimate is unbiased as

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

and its variance is reduced by a factor $1/b$, though at the cost of b times more expensive running time at each iteration. Therefore, one may regard Mini-batch SGD as a compromise between SGD and full gradient descent.

Dekel et al. [2] had shown that, under the Lipschitz gradient, the convergence rate for Mini-batch becomes $O(\sqrt{b/k} + b/k)$. Though it is not convincing to use Mini-batch SGD from the theory aspect, Mini-batch SGD turns out performing not too bad in practice, as shown in the following example.

Example Consider the regularized version of logistic regression in $\mathbb{R}^p$:

$$\min_{\beta \in \mathbb{R}^p} f(\beta), \quad \text{where} \quad f = \frac{1}{n} \sum_{i=1}^{n} f_i, \quad \text{and} \quad f_i(\beta) = -y_i x_i^T \beta + \log\left(1 + \exp\left(x_i^T \beta\right)\right) + \frac{\lambda}{2} ||\beta||_2^2.$$

The full gradient computation is $\nabla f(\beta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - p_i(\beta)) x_i + \lambda \beta$, where $p_i(\beta) = \exp\left(x_i^T \beta\right)/(1 + \exp\left(x_i^T \beta\right))$. Then, as for the time cost during each iteration, denote p as the time to compute the the full gradient update costs $O(np)$, and the mini-batch update with batch size b costs $O(bp)$, and the SGD update costs $O(p)$. 9.2 shows the comparison between full gradient descent, mini-batch, and SGD, in terms of convergence rate between, where $n = 10^4$, $p = 20$, and the step size is fixed.

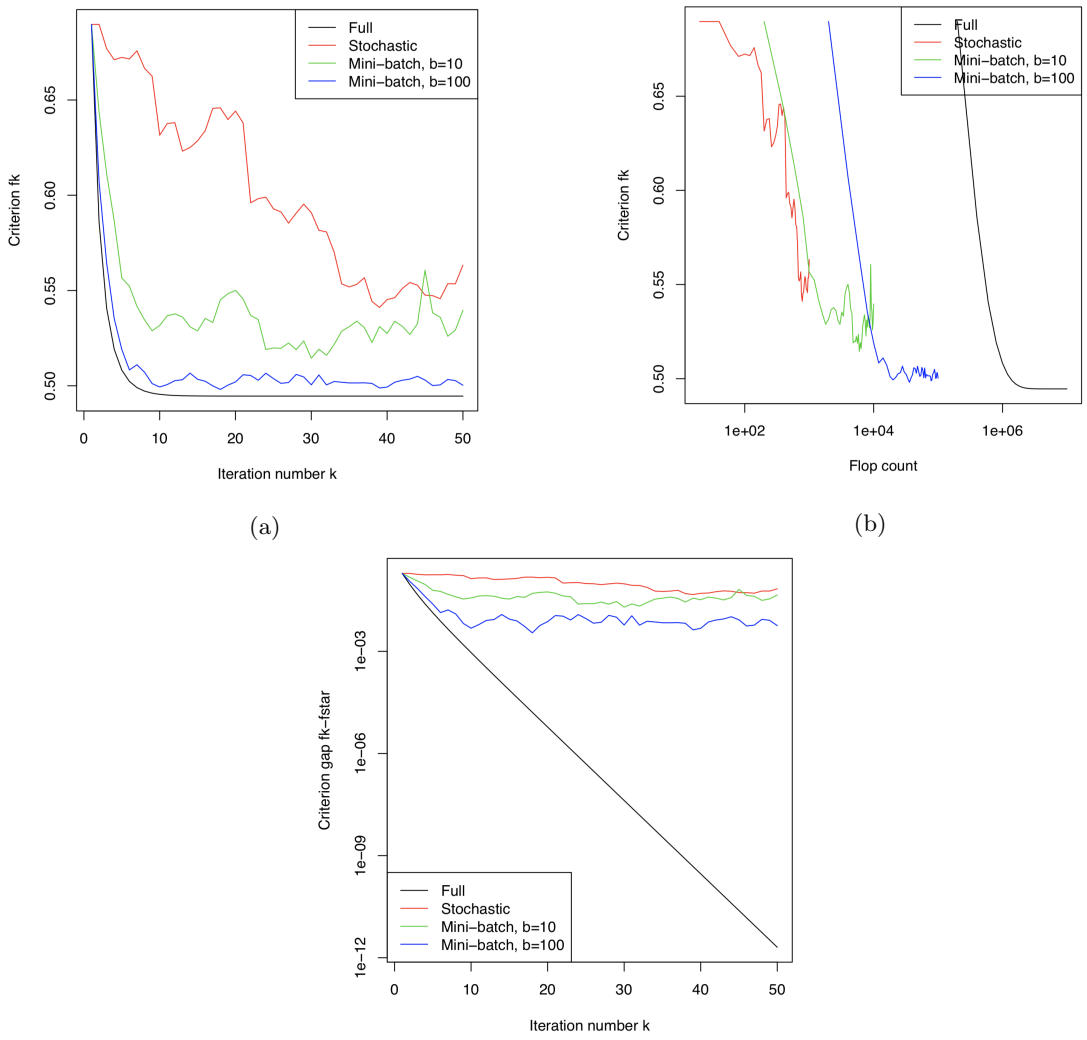

Figure 9.2: Comparison of convergence rate between full gradient descent, mini-batch, and SGD. Figure (a) shows that mini-batch does help in terms of per iteration performance; Figure (b) is shows that mini-batch usually do not give much improvement in terms of total cost (where flop count equals iteration index times time cost each iteration); Figure (c) shows that Mini-batch SGD does not really help in terms of optimality (where the y axis is the suboptimiality gap on a log scale).

9.7 Remark

As we have discussed, SGD can be very effective in terms of resources (per iteration cost, memory), and for this reason it is still the standard for solving large problems. However SGD is slow to converge, and does not benefit from strong convexity unlike full gradient descent. Mini-batch SGD can be an approach to trade off effectiveness and convergence rate: it is not that beneficial in terms of flops, but it 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 reducing the variance, they do not necessarily lead to faster convergence in the

theoretical domain.

Due to these properties, it was thought for a while that SGD would not be commonly useful. Nemirovski et al. [1] 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 ML

In many machine learning problems, we care more about the overall error instead of optimization error. When the optimization error is lower than a certain threshold, it is not the dominant term of the overall risk any more and we do not need to optimize to really high accuracy.

Thus in SGD, *fixed step sizes* are commonly used instead of diminishing. Someone may wonder how to choose step size t. A trick is to do tuning with different step sizes over a small fraction of training data, which is shown in Bottou [3].

Variants: Adagrad, Adam, AdaMax, SVRG, SAG, SAGA, ...

9.9 Early stopping

Suppose our data is $(x_i, y_i) \in \mathbb{R}^p \times \{0, 1\}, i = 1, ..., n$, where p is large.

It turns out that solve a ℓ_2 regularized logistic regression:

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

is similar to running gradient descent on the unregularized problem:

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

and stop early. By "early", it means to stop really early instead of stopping when we are bouncing around the minima.

More formally, early stopping is:

- Start at $\beta^{(0)} = 0$, could be seen as solution to regularized problem at $t = 0$
- Perform GD:

$$\beta^{(k)} = \beta^{(k-1)} - \epsilon \cdot \frac{1}{n} \sum_{i=1}^{n} (y_i - p_i(\beta^{(k-1)})) x_i, \quad k = 1, 2, 3, ... \quad \epsilon \text{ is a very small constant}$$

- Treat $\beta^{(k)}$ as an approximate solution to regulartized problem with $t = \|\beta^{(k)}\|_2$

With early stopping, it is both more convenient and much more efficient than using explicit regularization. Because in this way, it does not require running with different parameter t, just running gradient descent. And it turns out when we plot gradient descent iterates, it resembles the solution path of the ℓ_2 regularized problem for varying t!

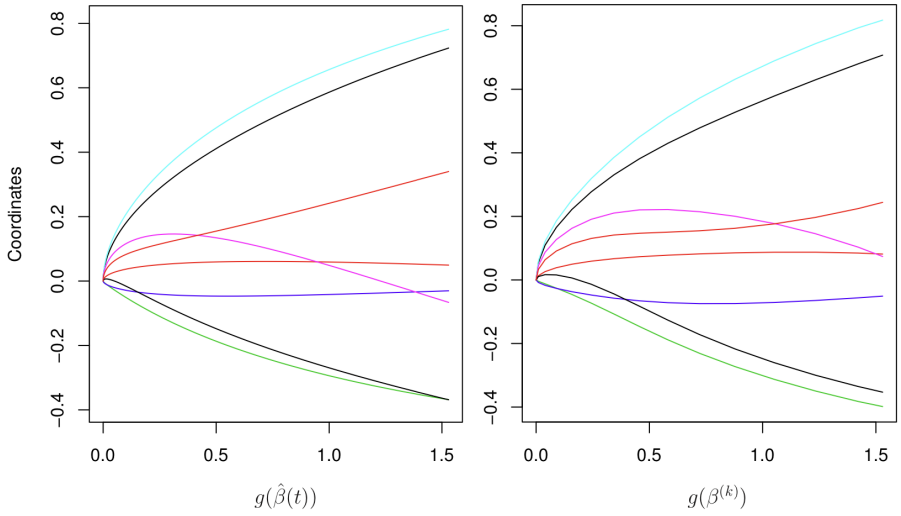

Figure 9.3: Logistic example with p = 8, solution path and grad descent path

What's the connection? The reason for such coincidence is remained to be explored more. However, intuitively, we can view it from the *steepest descent* view of gradient descent.

Let $\|\cdot\|$ and $\|\cdot\|_*$ be dual norms (e.g. ℓ_p and ℓ_q norms with $\frac{1}{p} + \frac{1}{q} = 1$)

Steepest descent updates are $x^+ = x + t \cdot \Delta x$, where

$$\Delta x = \|\nabla f(x)\|_* \cdot u$$
$$u = \underset{\|v\| \leq 1}{\operatorname{argmin}} \nabla f(x)^T v$$

If $p = 2$, then Δx is exactly $-\nabla f(x)$ and this is just gradient descent. Thus at each iteration, gradient descent moves in a direction that balances **decreasing f** and **increasing the ℓ_2 norm**, which is the same as in the regularized problem.

References

[1] Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on optimization*, 19(4):1574–1609, 2009.

[2] Ofer Dekel, Ran Gilad-Bachrach, Ohad Shamir, and Lin Xiao. Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13(Jan):165–202, 2012.

[3] Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.