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

Lecture 24: November 26

Lecturer: Ryan Tibshirani *Scribes: Maximilian Sieb, Byeongjoo Ahn, Anqi Yang*

24.1 Stochastic Gradient Descent

Consider minimizing an average of functions

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

This setting is common in machine learning, where this average of functions is equivalent to a loss function and each $f_i(x)$ is associated to the loss term of an individual sample point x_i. The full gradient descent step is given by

$$x^{(k)} = x^{(k-1)} - t_k \cdot \frac{1}{n}\sum_{i=1}^n \nabla f_i(x^{(k-1)}), \qquad k = 1, 2, 3, ...$$

The idea is now to just use a subset of all samples, i.e. all possible $f_i(x)$'s to approximate the full gradient. This is called **stochastic gradient descent**, or short, **SGD**. More formally, stochastic gradient repeats

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

where $i_k \in \{1, ..., n\}$ is a randomly chosen idnex at iteration k. Because we have $\mathbb{E}[\nabla f_{i_k}(x)] = \nabla f(x)$, the estimate is unbiased. The indicies i_k are usually chosen without replacement until we complete one full cycle through the entire data set.

24.1.1 Mini-batching

A common technique employed with SGD is **mini-batching**, where we choose a random subset $I_k \subseteq \{1, ..., n\}$ with size $|I_k| = b << n$. We then repeat

$$x^{(k)} = x^{(k-1)} - t_k \cdot \frac{1}{b}\sum_{i \in I_k} \nabla f_{i_k}(x^{(k-1)}), \qquad k = 1, 2, 3, ...$$

Because we have $\mathbb{E}[\frac{1}{b}\sum_{i_k \in I_k} \nabla f_{i_k}(x)] = \nabla f(x)$, we still have an unbiased estimate of the full gradient. Furthermore, we now reduced the *variance* of our gradient estimate by $\frac{1}{b}$, but we have also incurred b times more

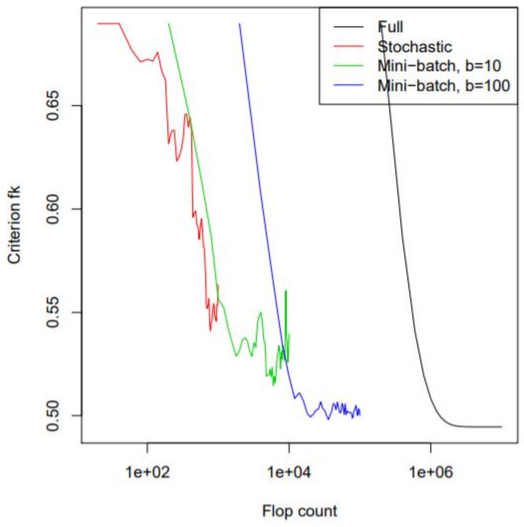


Figure 24.1: Criterion value vs. number of flops

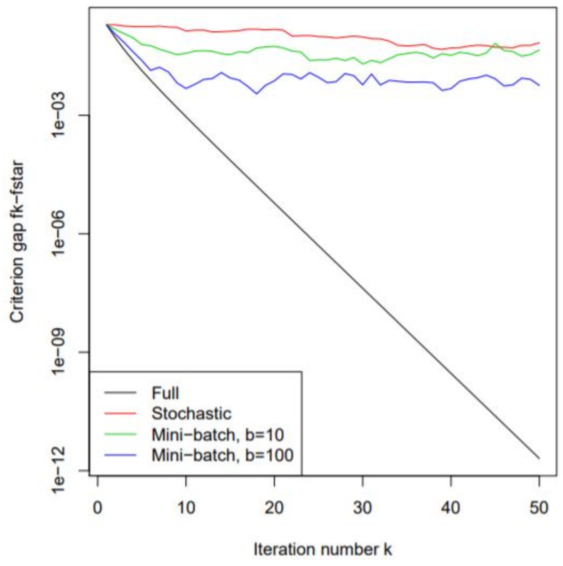


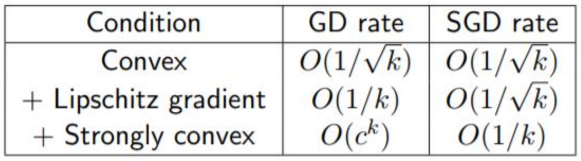
Figure 24.2: Optimality gap vs. number of iterations

computational cost. We see that smaller batch sizes would converge more quickly to a less optimal value, so in theory we do not really gain much from using larger batches if we use diminishing step sizes. Note, however, that in many settings the batch-update can be parallelized and we then indeed gain computational savings.

For a problem with n data points, mini-batch size b and feature dimension p, we obtain the following costs of standard SGD and batch-SGD:

1. full gradient: $O(np)$
2. mini-batch: $O(bp)$
3. standard SGD: $O(p)$

In terms of convergance rates, SGD has worse performance guarantees than standard gradient descent. Also, for SGD to converge we have to decrease our step-sizes for all settings, while for standard Gradient descent we can keep constant step-sizes if the Lipschitz gradient exists or if the problem is strongly convex.



Condition	GD rate	SGD rate
Convex	$O(1/\sqrt{k})$	$O(1/\sqrt{k})$
+ Lipschitz gradient	$O(1/k)$	$O(1/\sqrt{k})$
+ Strongly convex	$O(c^k)$	$O(1/k)$

Figure 24.3: Convergence rates of Standard Gradient Descent vs. Stochastic Gradient Descent

24.2 Variance Reduction

24.2.1 Stochastic average gradient

Stochastic average gradient (SAG) is a breakthrough method in stochastic optimization that greatly reduced variance though it is biased. The main idea of SAG is maintaining the table of the most recent gradient $g_i = \nabla f_i$. After initializing $x^{(0)}$ and $g_i^{(0)} = \nabla f_i(x^{(0)})$, the method is as follows:

- At steps $k = 1, 2, 3, ...$, pick random $i_k \in \{1, ..., n\}$, then let
$$g_{i_k}^{(k)} = \nabla f_{i_k}(x^{(k-1)}),$$
which is the most recent gradient of f_{i_k}. Then, set all other $g_i^{(k)} = g_i^{(k-1)}, i \neq i_k$, i.e., these stay the same.
- Update
$$x^{(k)} = x^{(k-1)} - t_k \cdot \frac{1}{n} \sum_{i=1}^{n} g_i^{(k)}$$

Note that

- Key of SAG is to allow each $f_i, i = 1, ..., n$ to communicate a part of the gradient estimate at each step
- This basic idea can be traced back to incremental aggregated gradient
- SAG gradient estimates are no longer unbiased, but they have greatly reduced variance

- Although SAG maintain the table of gradients and require to average all these gradients, it is not expensive. Basically it is just as efficient as SGD if we compute the average as follows:

$$x^{(k)} = x^{(k-1)} - t_k \cdot \underbrace{\left(\frac{g_{i_k}^{(k)}}{n} - \frac{g_{i_k}^{(k-1)}}{n} + \underbrace{\frac{1}{n}\sum_{i=1}^{n} g_i^{(k-1)}}_{\text{old table average}} \right)}_{\text{new table average}}$$

24.2.1.1 SAG convergence analysis

Assume that $f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$, where each f_i is differentiable, and ∇f_i is Lipschitz with constant L, and denote $\bar{x}^{(k)} = \frac{1}{k}\sum_{l=0}^{k-1} x^l$, the average iterate after $k-1$ steps.

Theorem (Schmidt, Le Roux, Bach): SAG, with a fixed step size $t = 1/(16L)$, and the initialization

$$g_i^{(0)} = \nabla f_i(x^{(0)}) - \nabla f(x^{(0)}), i = 1, ..., n$$

satisfies

$$\mathbb{E}[f(\bar{x}^{(k)})] - f^* \leq \frac{48n}{k}(f(x^{(0)}) - f^*) + \frac{128L}{k}\|x^{(0)} - x^*\|_2^2$$

where the expectation is taken over random choices of indices.

Note that

- Result stated in terms of the average iterate $\bar{x}^{(k)}$, but also can be shown to hold for best iterate $x_{\text{best}}^{(k)}$ seen so far.
- This is $O(1/k)$ convergence rate for SAG. Compare to $O(1/k)$ rate for GD, and $O(1/\sqrt{k})$ rate for SGD.
- But, the constants are different! Bound after k steps:

$$\text{GD}: \frac{L}{2k}\|x^{(0)} - x^*\|_2^2$$
$$\text{SAG}: \frac{48n}{k}(f(x^{(0)}) - f^*) + \frac{128L}{k}\|x^{(0)} - x^*\|_2^2$$

- So first term in SAG bound suffers from factor of n; authors suggest smarter initialization to make $f(x^{(0)}) - f^*$ small (e.g., they suggest using result of n SGD steps)
- Zero initialization gives us worse results.

24.2.1.2 Convergence under strong convexity

Assume further that each f_i is strongly convex with parameter m

Theorem (Schmidt, Le Roux, Bach): SAG, with a step size $t = 1/(16L)$ and the same initialization as before, satisfies

$$\mathbb{E}[f(x^{(k)})] - f^* \leq \left(1 - \min\left\{\frac{m}{16L}, \frac{1}{8n}\right\}\right)^k \times \left(\frac{3}{2}(f(x^{(0)}) - f^*) + \frac{4L}{n}\|x^{(0)} - x^*\|_2^2\right)$$

Note that

- This is linear convergence rate $O(c^k)$ for SAG. Compare this to $O(c^k)$ for GD, and only $O(1/k)$ for SGD
- Like GD, SAG is adaptive to strong convexity (achieves better rate with same settings)
- Proofs of these results are not easy because it is biased.

24.2.1.3 Example: logistic regression

We compare SGD versus SAG in logistic regression, over 30 reruns of these randomized algorithms. SAG provides strikingly lower variance and looks deterministic as shown in Figure 24.4.

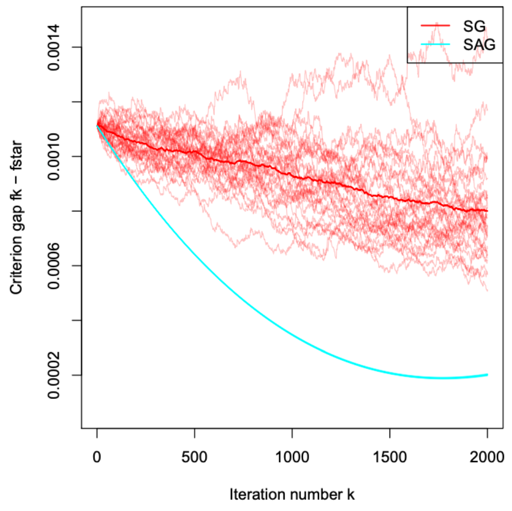


Figure 24.4: Logistic regression using SG and SAG.

- SAG does well, but did not work out of the box; required a specific setup
- Took one full cycle of SGD (one pass over the data) to get $\beta^{(0)}$, and then started SGD and SAG both from $\beta^{(0)}$. This warm start helped a lot
- SAG initialized at $g_i^{(0)} = \nabla f_i(\beta^{(0)}, i = 1, ..., n$, computed during initial SGD cycle. Centering these gradients was much worse (and so was initializing them at 0)
- Tuning the fixed step sizes for SAG was very finicky; here now hand-tuned to be about as large as possible before it diverges
- Authors of SAG conveyed that this algorithm will work the best, relative to SGD, for ill-conditioned problems (the current problem not being ill-conditioned at all)

24.2.2 SAGA

SAGA is a follow-up work on the SAG work. The algorithm is as the following:

- Maintain table, containing gradient g_i of $f_i, i = 1, ..., n$
- Initialize $x^{(0)}$, and $g^{(0)} = \nabla f_i(x^{(0)}), i = 1, ..., n$
- At steps $k = 1, 2, 3, ...,$ pick random $i_k \in \{1, ..., n\}$, then let

  $$g_{i_k}^{(k)} = \nabla f_{i_k}(x^{(k-1)})$$

  (update $g_{i_k}^{(k)}$ to the most recent gradient of f_{i_k}) Set all the other $g_i^{(k)} = g_i^{(k-1)}, i \neq i_k$, i.e, these stay the same.
- Update

  $$x^{(k)} = x^{(k-1)} - t_k \cdot (g_{i_k}^{(k)} - g_{i_k}^{(k-1} + \frac{1}{n}\sum_{i=1}^{n} g_i^{(k-1)})$$

Note that

- SAGA and SAG are almost the same, except for the last updating step. SAGA gradient estimate $g_{i_k}^{(k)} - g_{i_k}^{(k-1} + \frac{1}{n}\sum_{i=1}^{n} g_i^{(k-1)}$, while SAG estimate $\frac{1}{n}g_{i_k}^{(k)} - \frac{1}{n}g_{i_k}^{(k-1} + \frac{1}{n}\sum_{i=1}^{n} g_i^{(k-1)}$.
- This change makes SAGA estimate have a much higher variance, but remarkably, its estimate is *unbiased*. Here is a simple explanation: Consider family of estimators for $\mathbb{E}(X)$,

  $$\theta_\alpha = \alpha(X - Y) + \mathbb{Y}$$

  where $\alpha \in [0, 1]$, and X, Y are presumed correlated. We compute the first two moments of θ_α

  $$\begin{aligned} \mathbb{E}(\theta_\alpha) &= \alpha\mathbb{E}(X) - \alpha\mathbb{E}(Y) + \mathbb{E}(Y) \\ &= \alpha\mathbb{E}(X) + (1-\alpha)\mathbb{E}(Y) \end{aligned}$$

  $$\begin{aligned} \mathrm{Var}(\theta_\alpha) &= \alpha^2 \mathrm{Var}(X - Y) \\ &= \alpha^2(\mathrm{Var}(X) + \mathrm{Var}(Y) - 2\mathrm{Cov}(X, Y)) \end{aligned}$$

  Let's consider $g_{i_k}^{(k)}$ as X, and $g_{i_k}^{(k-1)}$ as Y. SAGA uses $\alpha = 1$ and thus the estimation is unbiased. While SAGA uses $\alpha = 1/n$, which leads to dramatically low variance.
- SAGA matches convergence rates of SAG, but has much simpler proofs.

Here is an example for logistic regression example, now adding SAGA to mix.
SAGA works well, but it again required somewhat specific setup. Note that the warm start helped a lot, by taking a full cycle of SGD (one pass over the data) to get $\beta^{(0)}$, and then started SGD, SAG, SAGA all from $\beta^{(0)}$. Also, SAGA initialized at $g_i^{(0)} = \nabla f_i(\beta^{(0)}), i = 1, ..., n$ has better performance than initializing them at 0 or centering these gradients. Interestingly, the SAGA criterion curves look like SGD curves except that they are jagged and highly variable. While SAG looks very different, and this really emphasizes the fact that its update have much lower variance.

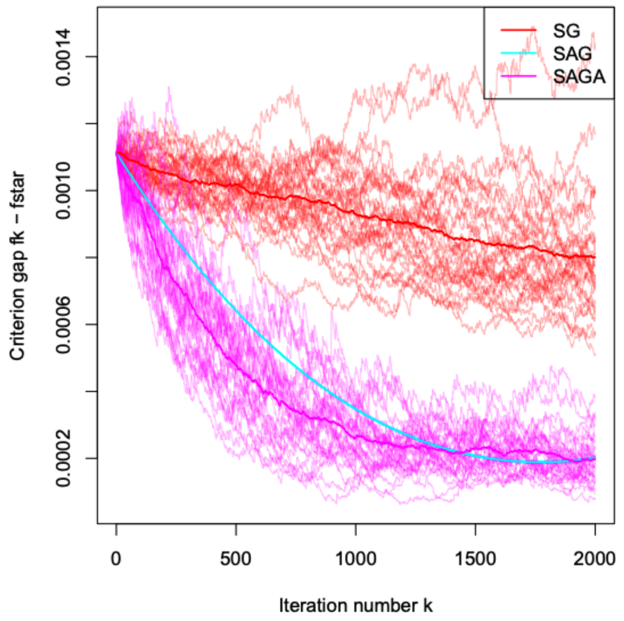


Figure 24.5: Logistic regression using SG, SAG, and SAGA.

24.2.3 Other Stochastic Methods

SDCA (Shalev-Schwartz, Zhang, 2013): applies coordinate ascent to the dual of ridge regularized problems, and uses randomly selected coordinates. Effective primal updates are similar to SAG/SAGA.
SVRG(Johnson, Zhang, 2013): like SAG/SAGA, but does not store a full table of gradients, just an average, and updates this occasionally.
Others S2GD (Konecny, Richtarik, 2014), MISO (Mairal, 2013), Finito (Defazio, Caetano, Domke, 2014), etc.

24.3 Acceleration and Momentum

In the setting of strong convex, SAG and SAGA (others) have the iteration complexity of

$$O((n+\frac{L}{m})\log(1/\epsilon))$$

and the lower bound of

$$O((n+\sqrt{\frac{nL}{m}})\log(1/\epsilon))$$

Can we do better? We can use acceleration (Lan and Zhou 2015, Lin et al., 2015). Variance reduction + acceleration completely solve the finite sum case. However, in convex problems, if we move the criterion from finite sum to general stochastic settings $f(x) = \mathbb{E}_{\xi}[F(x,\xi)]$, the performance of SGD cannot be improved. In nonconvex problems, variance reduction is not necessarily useful since variance can help with getting over local minimums. But momentum is popular in nonconvex acceleration. One of the most classic method is Polyak's heavy ball method, which works really well in practice,

$$x^{(k)} = x^{(k-1)} + \alpha(x^{(k-1)} - x^{(k-2)}) - t_k \nabla f_{i_k}(x^{(k-1)})$$

It push to the direction by adding a term $\alpha(x^{(k-1)} - x^{(k-2)})$, where α is a fixed constant. But this method can somewhat fragile. Here is an example of Polyak's heavy ball versus Nesterov acceleration, in optimizing a convex quadratic (from Shi et al., 2018):

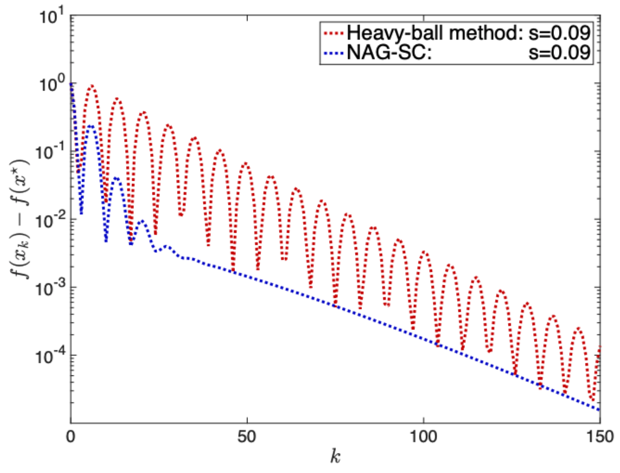


Figure 24.6: Comparison of Polyak's heavy ball and Nesterov methods.

24.4 Adaptive Step Sizes

As mentioned before, we have to diminish our step size over time for SGD to ensure convergence. A problem with this is that features which do not provide a rich gradient signal will soon not receive any more update due to diminishing step sizes.
A prossible solution is to adaptively change the step size based on how large the gradients were for the respective feature dimension for prior time-steps.

24.5 AdaGrad

A very popular adaptive method is called *AdaGrad*. Let $g^{(k)} = \nabla f_{i_k}(x^{(k-1)})$. Then for $j = 1, ..., p$ do:

$$x_j^{(k)} = x_j^{(k-1)} - \alpha \cdot \frac{g_j^{(k)}}{\sqrt{\sum_{l=1}^{k} (g_j^{(l)})^2}}$$

What did we gain? First, we do not need to tune our learning rate anymore becaause α is just a fixed hyperparamter. In sparse problems, AdaGrad performs much better than standard SGD.
There have been numerous extension on this algorithm such as Adam, RMSProp, etc.

References

[1] J. Duchi and E. Hazan and Y. Singer (2010), Adaptive subgradient methods for online learning and stochastic optimization

[2] A. Defasio and F. Bach and S. Lacoste-Julien (2014), SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives

[3] G. Lan and Y. Zhou (2015), An optimal randomized incremental gradient method

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

[5] M. Schmidt and N. Le Roux and F. Bach (2013), Minimizing finite sums with the stochastic average gradient