Converging on the Ultimate Algorithm for Minimizing Convex Sums

Mark Schmidt

University of British Columbia
Minimizing Convex Sums

- We consider the problem of minimizing a finite sum,

\[
f(x) = \frac{1}{n} \sum_{i=1}^{n} f_i(x),
\]

of smooth and convex functions \( f_i \).

- Classic problem frequently arising in machine learning (ML):
  - Basic models like Least squares, logistic regression:

\[
f_i(x) = \frac{1}{2}(a_i^T x - b_i)^2, \quad f_i(x) = \log(1 + \exp(-b_i a_i^T x)),
\]

and more advanced models like conditional random fields:

\[
f_i(x) = -w^T \phi(x_i, y_i) + \log \sum_{y'} \exp(w^T \phi(x_i, y')).
\]

- **Stochastic gradient** methods are traditional approach for large \( n \).
Modern Stochastic Gradient Methods

- Classic stochastic gradient have a *sublinear* convergence rate.
- Since 2012: new stochastic gradient method with *linear rates*.
  - Many papers on this topic (see our tutorial tomorrow).

- Algorithms from papers *often work great in practice*.
  - Sometimes better than existing highly-tuned libraries.
  - Now used in standard ML packages and commercial products

- But they could *potential work much better in practice*.
  - Worst-case analyses don’t account for all structure in the data.
  - There are still a important practical tricks to be discovered.
This Talk: Tricks for Speeding Up SAG and SVRG

- This talk: tricks that could make SAG or SVRG much faster.
  - Same tricks could likely speed up other methods.
  - I’m mostly going to stay away from parallel/distributed issues.

- My goal: build the best “black box” implementation possible.

- What I want from you:
  - If you like to prove, some of these are good challenges.
  - If you like to implement, these could help.
  - If I’m missing tricks, let me know!
The stochastic average gradient (SAG) algorithm has the form

\[ x^{t+1} = x^t - \alpha_t \frac{1}{n} \sum_{i=1}^{n} f_i'(x^t), \]

described as a gradient descent step but with old gradient estimates.

Each iteration evaluates \( f_i'(x^t) \) for a random \( i \).
- We set \( i_t = t \) for this example and \( i_t = i_{t-1} \) for the others.

Number of gradients to reach accuracy \( \epsilon \): \( \tilde{O}(n + \kappa) \).
- Gradient method requires \( \tilde{O}(n\kappa) \).
- Classic stochastic methods require \( O(1/\epsilon) \).
Comparing algorithm from theorem to best implementations:

Sometimes it does better but often it does worse...
Bigger Step Sizes for SAG

- Assumptions in the analysis:
  - Function $f$ is strongly-convex.
  - Gradients $f'_i$ are $L$-Lipschitz continuous.
  - Step-size $\alpha_t$ is set to $1/16L$.

- Algorithm works better in practice with $\alpha_t = 1/L$. 
Even-Bigger Step Sizes for SAG

- In general SAG does not work with $\alpha_t = \frac{10}{L}$ (or even $\frac{1}{L}$).
  - But for some problems it works way better with this choice.
- Why???

- For some problems: local $L$ is much smaller than global $L$. 
Even-Bigger Step Sizes for SAG

- Using global $L$ vs. trying to estimate local $L$:

- See Section 4 of Le Roux et al. [2012] and also Vainsencher et al. [2015].
What about step-sizes depending on $\mu$?

Should we use $\alpha_t = \frac{2}{L + n\mu}$?

Watch out for local $\mu$ vs. global $\mu$.

SDCA uses global $\mu$ so sometimes does really bad:
What about just trying to figure out step-size that works the best?

Mairal [2013] gives a simple line-search method:
  - Search for the best performance on a subset of the data (Bottou trick).

Is there a better method to be discovered?
Disadvantage of SAG is that it has a huge memory requirement.

For many problems, gradient structure allows us to reduce this.
  - Least squares, logistic regression, conditional random fields.

For general problems, we can instead use SVRG.
Stochastic Variance-Reduced Gradient (SVRG)

SVRG algorithm:

- Start with $x_0$
- for $s = 0, 1, 2 \ldots$ (outer loop)
  - $d_s = \frac{1}{N} \sum_{i=1}^{N} f'_i(x_s)$ (full gradient calculation)
  - $x^0 = x_s$
  - for $t = 1, 2, \ldots m$ (inner loop)
    - Randomly pick $i_t \in \{1, 2, \ldots, N\}$
    - $x^t = x^{t-1} - \alpha_t (f'_{i_t}(x^{t-1}) - f'_{i_t}(x_s) + d_s)$ (two gradients per iteration)
    - $x_{s+1} = x^t$ (initialize next outer loop)
- Only need to store $x_s$ and $d_s$.
- Choices that seem to work well are $\alpha_t = 1/L$ and $m = n$.
- Full gradient calculations are wasteful when far from the solution.
Practical SVRG algorithm:

- Start with $x_0$
- for $s = 0, 1, 2 \ldots$
  - $d_s = \frac{1}{|B_s|} \sum_{i \in B_s} f'_i(x_s)$
  - $x^0 = x_s$
  - for $t = 1, 2, \ldots m$
    - Randomly pick $i_t \in \{1, 2, \ldots, N\}$
    - $x^t = x^{t-1} - \alpha_t (f'_{i_t}(x^{t-1}) - f'_{i_t}(x_s) + d_s)$
  - $x_{s+1} = x^t$
- Control variate $d_s$ can be based on a subset of the examples.
- Preserve rate if $B_s$ grows fast enough.
- For example, $B_s = \min\{2^s, n\}$. 

(outer loop)

(batch gradient calculation)

(inner loop)

(two gradients per iteration)

(initialize next outer loop)
Practical SVRG

- SVRG with full-gradient $d^s$ compared to growing batch:

- Is there a better way to grow $d^s$ or choose $B_s$?
- Recent work shows that maybe we should be updating $d^s$ [Ngyuen et al., 2017].
Can we improve performance by non-uniform sampling?

Consider case where each $f_i'$ has Lipschitz constant $L_i$:

- Improve the rate theoretically by sampling biasing towards $L_i$.

Justification: frequently sample gradients that change quickly.

In practice, a huge difference between local $L_i$ and global $L_i$.

[Xiao & Zhang, 2014]
Non-Uniform Sampling

- Uniform vs. non-uniform (global $L_i$) vs. non-uniform (local $L_i$).

- Is this sampling proportional to $L_i$ optimal across iterations?

- Work on stratified sampling and clustering examples.

[Zhao & Zhang, 2014, Hoffman et al., 2015, Allen-Zhu et al., 2016]
Identifying Support Vectors

- A related idea: identifying support vectors.

- Consider a smoothed SVM problem [Rosset & Zhu, 2006]:

\[
\min_{x \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} f(b_i a_i^T x), \quad f(\tau) = \begin{cases} 
0 & \text{if } \tau > 1 + \epsilon, \\
1 - \tau & \text{if } \tau < 1 - \epsilon, \\
\frac{(1 + \epsilon - \tau)^2}{4\epsilon} & \text{if } |1 - \tau| \leq \epsilon.
\end{cases}
\]

- The solution is sparse in the \( f_i \) (has support vectors).
- Keep track of number $z$ of consecutive times $f'_i(x^t)$ was zero.

- If it’s zero at least twice ($z \geq 2$), skip the next $2^{z-2}$ evaluations.
  - May only evaluate non-support examples a logarithmic number of times.

[Babanezhad et al., 2015]
Choosing the Batch

- We can often evaluate several gradients in parallel.

- Logical way to pick the batch size: number of parallel gradients.

- Two possibilities ways to sample the batch:
  - Sample from a fixed set of data “blocks”.
  - Sample the original variables.

- For the original variables, Lipschitz sampling again helps.
- For constructing “blocks”, there may be better strategies.
  - Try to make the blocks have small or varied Lipschitz constants.
Can we accelerate these methods as with gradient methods?

Is $\tilde{O}(n + \kappa)$ the best we can do?

We can’t reduce runtime to $\tilde{O}(n + \sqrt{\kappa})$.

But several authors give algorithms achieving $\tilde{O}(n + \sqrt{n\kappa})$. 
Most common strategy: inexact proximal point methods use

\[ x_{k+1} = \arg\min_x f(x) + \frac{\lambda_k}{2} \|x - x_k\|^2, \]

and solve this up to accuracy \( \epsilon_k \) using stochastic method.

[Shalev-Schwartz & Zhang, 2014]

But needs sequence of parameters and termination criteria.

- Although some nice tricks in Lin et al. [2015].

Recent alternatives don’t need the inner/outer setup.

Newton-Like Methods

- Can we make Newton-like versions of these methods?

- If we use a matrix $H$ and apply the update

\[ x^{t+1} = x^t - \frac{\alpha_t}{n} H \sum_{i=1}^{n} f'_i(x^t_i), \]

then we get the convergence for minimizing $f(H^{1/2}x)$ instead of $f(x)$.

- Can be much faster, but doesn’t give superlinear for any $H$.
  - Superlinear not possible for random, but possible for cyclic [Rodomanov & Kropotov, 2016]

- Not clear how to choose a sequence of $H_t$ matrices.
  - But many recent works on this topic.

- Non-diagonal $H_t$ substantial increase runtime for sparse datasets.
Methods are great in theory, but practical details need to be worked out too.

How do we use/identify bigger step-sizes?

Is sampling based on Lipschitz constants optimal?

- Particularly for accelerated and Newton-like methods.

Can we cleverly choose the batch or batch size?

Can we make accelerated methods adaptive to $\mu$?

Can we design robust/efficient Newton-like method?