First-Order Optimization Algorithms for Machine Learning

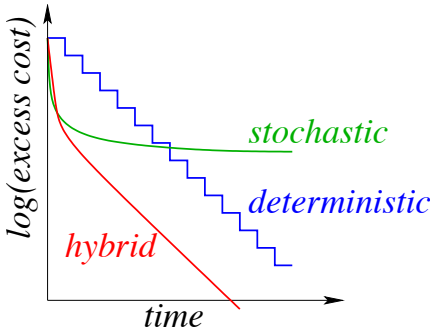
Variance-Reduced Stochastic Gradient

Mark Schmidt

University of British Columbia

Summer 2020

Better Methods for Smooth Objectives and Finite Datasets?



- Stochastic methods:
  - $O(1/\epsilon)$ iterations but requires 1 gradient per iterations.
  - Rates are unimprovable for general stochastic objectives.
- Deterministic methods:
  - $O(\log(1/\epsilon))$ iterations but requires n gradients per iteration.
  - The faster rate is possible because n is finite.
- For finite n, can we design a better method?

Hybrid Deterministic-Stochastic

- Approach 1: control the sample size.
- Deterministic method uses all n gradients,

$$\nabla f(w^k) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(w^k).$$

- Stochastic method approximates it with 1 sample,

$$\nabla f_{i_k}(w^k) \approx \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(w^k).$$

- A common variant is to use larger sample $\mathcal{B}^k$ ("mini-batch"),

$$\frac{1}{|\mathcal{B}^k|}\sum_{i\in\mathcal{B}^k} \nabla f_i(w^k) \approx \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(w^k),$$

particularly useful for vectorization/parallelization.

- For example, with 16 cores set $|\mathcal{B}^k| = 16$ and compute 16 gradients at once.

Mini-Batching as Gradient Descent with Error

- The SG method with a sample $\mathcal{B}^k$ ("mini-batch") uses iterations

$$w^{k+1} = w^k - \frac{\alpha_k}{|\mathcal{B}^k|} \sum_{i \in \mathcal{B}^k} \nabla f_i(w^k).$$

- Let's view this as a "gradient method with error",

$$w^{k+1} = w^k - \alpha_k(\nabla f(w^k) + e^k),$$

where e^k is the difference between approximate and true gradient.

($e^k = g^k - \nabla f(w^k)$ for approximation g^k)

- If you use $\alpha_k = 1/L$, then using descent lemma this algorithm has

$$f(w^{k+1}) \leq f(w^k) - \underbrace{\frac{1}{2L}\|\nabla f(w^k)\|^2}_{\text{good}} + \underbrace{\frac{1}{2L}\|e^k\|^2}_{\text{bad}},$$

for any error e^k (not necessarily unbiased or even stochastic).

Effect of Error on Convergence Rate

- Our progress bound with $\alpha_k = 1/L$ and error in the gradient of e^k is

$$f(w^{k+1}) \leq f(w^k) - \underbrace{\frac{1}{2L}\|\nabla f(w^k)\|^2}_{\text{good}} + \underbrace{\frac{1}{2L}\|e^k\|^2}_{\text{bad}},$$

  and notice that you are guaranteed to decrease f is $\|e^k\| < \|\nabla f(w^k)\|$.

- Connection between "error-free" rate and "with error" rate:
  - If "error-free" rate is $O(1/k)$, you maintain this rate if $\|e^k\|^2 = O(1/k)$.
  - If "error-free" rate is $O(\rho^k)$, you maintain this rate if $\|e^k\|^2 = O(\rho^k)$.
  - If error goes to zero more slowly, then rate that it goes to zero becomes bottleneck.

- So to understanding effect of batch-size, need to know how $|\mathcal{B}^k|$ affects $\|e^k\|^2$.

Effect of Batch Size on Error

- Effect of batch size $|\mathcal{B}^k|$ control error size e^k.
  - If we sample with replacement we get

$$\mathbb{E}[\|e^k\|^2] = \frac{1}{|\mathcal{B}^k|}\sigma^2,$$

    where σ^2 is the variance of the gradient norms.
    - "Doubling the batch size cuts the error in half".
  - If we sample without replacement from a training set of size n we get

$$\mathbb{E}[\|e^k\|^2] = \frac{n - |\mathcal{B}^k|}{n}\frac{1}{|\mathcal{B}^k|}\sigma^2,$$

    which drives error to zero as batch size approaches n.
- For $O(\rho^k)$ linear convergence, need a schedule like $|\mathcal{B}^{k+1}| = |\mathcal{B}^k|/\rho$.
- For $O(1/k)$ sublinear convergence, need a schedule like $|\mathcal{B}^{k+1}| = |\mathcal{B}^k| + \text{const}$.

Batching: Growing-Batch-Size Methods

- The SG method with a sample $\mathcal{B}^k$ uses iterations

$$w^{k+1} = w^k - \frac{\alpha_k}{|\mathcal{B}^k|} \sum_{i \in \mathcal{B}^k} \nabla f_i(w^k).$$

- For a fixed sample size $|\mathcal{B}^k|$, the rate is sublinear.
  - With fixed step-size, doubling batch size halves radius of "ball" around solution.
  - Still need step-size to go to zero to get convergence.

- But we can grow $|\mathcal{B}^k|$ to achieve a faster rate:
  - Early iterations are cheap like SG iterations.
  - Later iterations can use a sophisticated gradient method.
    - No need to set a magical step-size: use a line-search.
    - Can incorporate linear-time approximations to Newton.

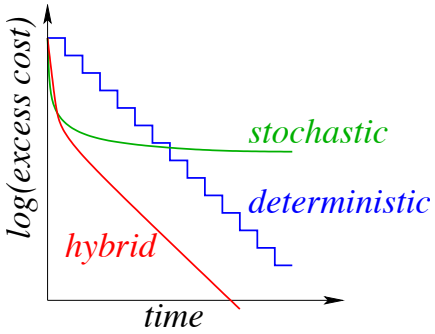
- Another approach: at some point switch from stochastic to deterministic:
  - Often after a small number of passes (but hard to know when to switch).

Variance-Reduction

- Increasing the batch size is a form of variance-reduction.
  - A way to decrease the size of the variance in SGD ("bad" term).
- Many other forms of variance reduction exist.
  - Control variates, importance sampling, re-parameterization trick, and so on.
- These improve constants in SGD convergence rate.
  - But don't improve rate unless objecctive is smooth and variance goes to zero.

Outline

Previously: Better Methods for Smooth Objectives and Finite Datasets



- Stochastic methods:
  - $O(1/\epsilon)$ iterations but requires 1 gradient per iterations.
- Deterministic methods:
  - $O(\log(1/\epsilon))$ iterations but requires n gradients per iteration.
- Growing-batch ("batching") or "switching" methods:
  - $O(\log(1/\epsilon))$ iterations, requires fewer than n gradients in early iterations.

Stochastic Average Gradient

- Growing $|\mathcal{B}^k|$ eventually requires O(n) iteration cost.
- **Can we have 1 gradient per iteration and only $O(\log(1/\epsilon))$ iterations?**
  - YES! First method was the stochastic average gradient (SAG) algorithm in 2012.
- To motivate SAG, let's view gradient descent as performing the iteration

$$w^{k+1} = w^k - \frac{\alpha_k}{n} \sum_{i=1}^{n} v_i^k,$$

  where on each step we set $v_i^k = \nabla f_i(w^k)$ for all i.
- SAG method: only set $v_{i_k}^k = \nabla f_{i_k}(w^k)$ for a randomly-chosen i_k.
  - All other v_i^k are kept at their previous value.

Stochastic Average Gradient

- We can think of SAG as having a memory:

$$\begin{bmatrix} — & v_1 & — \\ — & v_2 & — \\ & \vdots & \\ — & v_n & — \end{bmatrix},$$

where v_i^k is the gradient $\nabla f_i(w^k)$ from the last k where i was selected.

- On each iteration we:
  - Randomly choose one of the v_i and update it to the current gradient.
  - We take a step in the direction of the average of these v_i.

Stochastic Average Gradient

- Basic SAG algorithm (maintains $g = \sum_{i=1}^{n} v_i$):
  - Set $g = 0$ and gradient approximation $v_i = 0$ for $i = 1, 2, \ldots, n$.
  - while(1)
    - Sample i from $\{1, 2, \ldots, n\}$.
    - Compute $\nabla f_i(w)$.
    - $g = g - v_i + \nabla f_i(w)$.
    - $v_i = \nabla f_i(w)$.
    - $w = w - \frac{\alpha}{n} g$.

- Iteration cost is $O(d)$, and "lazy updates" allow $O(z)$ with sparse gradients.
- For linear models where $f_i(w) = h(w^\top x^i)$, it only requires $O(n)$ memory:

$$\nabla f_i(w) = \underbrace{h'(w^\top x^i)}_{\text{scalar}} \underbrace{x^i}_{\text{data}}.$$

  - Least squares is $h(z) = \frac{1}{2}(z - y^i)^2$, logistic is $h(z) = \log(1 + \exp(-y^i z))$, etc.
- For neural networks, would need to store all activations (typically impractical).

Stochastic Average Gradient

- The SAG iteration is

$$w^{k+1} = w^k - \frac{\alpha_k}{n} \sum_{i=1}^{n} v_i^k,$$

where on each iteration we set $v_{i_k}^k = \nabla f_{i_k}(w^k)$ for a randomly-chosen i_k.

- Unlike batching, we use a gradient for every example.
  - But the gradients might be out of date.

- Stochastic variant of earlier increment aggregated gradient (IAG).
  - Selects i_k cyclically, which destroys performance.

- Key proof idea: $v_i^k \to \nabla f_i(w^*)$ at the same rate that $w^k \to w^*$:
  - So the variance $\|e_k\|^2$ ("bad term") converges linearly to 0.

Convergence Rate of SAG

If each ∇f_i is $L-$continuous and f is strongly-convex, with $\alpha_k = 1/16L$ SAG has

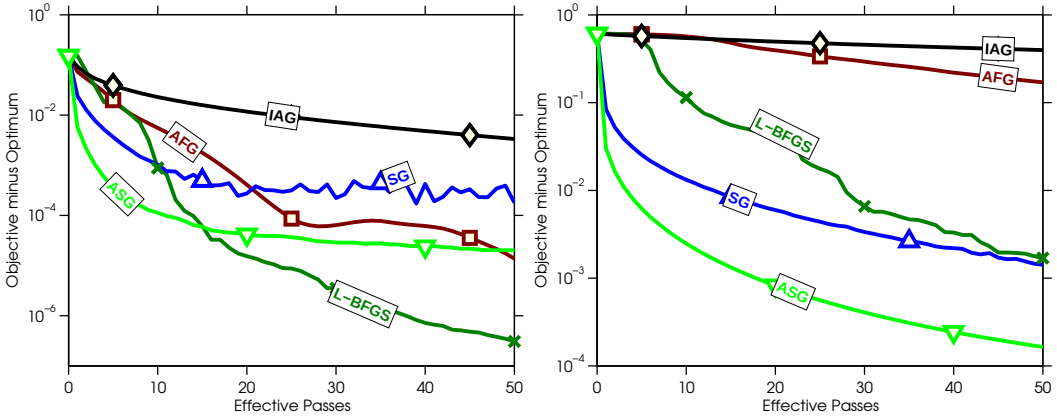
$$\mathbb{E}[f(w^k) - f(w^*)] \leqslant O\left(\left(1 - \min\left\{\frac{\mu}{16L}, \frac{1}{8n}\right\}\right)^k\right)$$

- Number of ∇f_i evaluations to reach accuracy ϵ:
  - Stochastic: $O(\frac{L}{\mu}(1/\epsilon))$. (Best when n is enormous)
  - Gradient: $O(n\frac{L}{\mu}\log(1/\epsilon))$.
  - Nesterov: $O(n\sqrt{\frac{L}{\mu}}\log(1/\epsilon))$. (Best when n is small and L/μ is big)
  - SAG: $O(\max\{n, \frac{L}{\mu}\}\log(1/\epsilon))$.

- But note that the L values are again different between algorithms.

Comparing Deterministic and Stochastic Methods

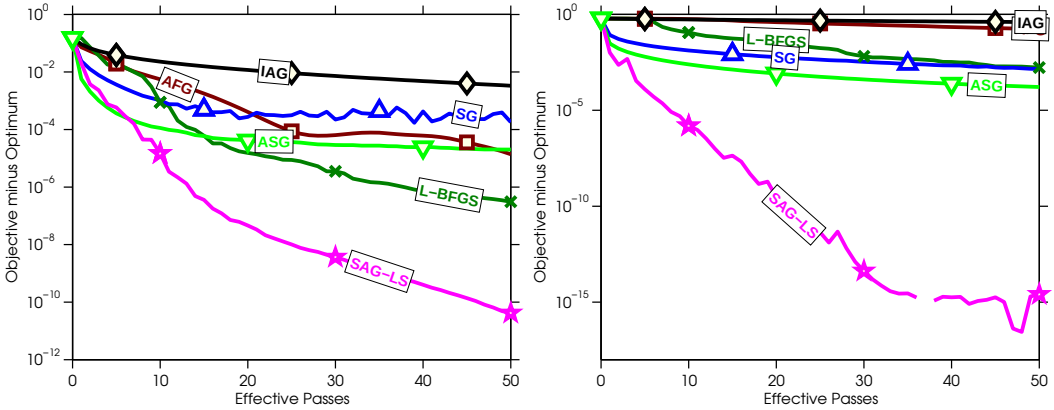
- Two benchmark L2-regularized logistic regression datasets:



- Averaging makes SG work better, deterministic methods eventually catch up.

SAG Compared to Deterministic/Stochastic Methods

- Two benchmark L2-regularized logistic regression datasets:



- Starts like stochastic but linear rate, SAG step-size set to $\hat{L}$ approximation.

Discussion of SAG and Beyond

- Bonus slides discuss practical issues related to SAG:
  - Setting step-size with an approximation to L.
  - Deciding when to stop.
  - Lipschitz sampling of training examples.
    - Improves rate for SAG, only changes constants for SG.

- There are now a bunch of stochastic algorithm with fast rates:
  - SDCA, MISO, mixedGrad, SVRG, S2GD, Finito, SAGA, etc.
  - Accelerated/Newton-like/coordinate-wise/proximal/ADMM versions.
  - Analysis in non-convex settings, including new algorithms for PCA.
  - You can apparently get medals for research:

    `https://ismp2018.sciencesconf.org/data/pages/_SJP8196.jpg`

- Most notable variation is SVRG which gets rid of the memory...

Stochastic Variance-Reduced Gradient (SVRG)

SVRG algorithm: gets rid of memory by occasionally computing exact gradient.

$$w^{k+1} = w^k - \alpha_k(\nabla f_{i_k}(w^k) - \underbrace{\nabla f_{i_k}(w_s) + \nabla f(w_s)}_{\text{mean zero}}),$$

where w_s is updated every m iterations.

Convergence properties similar to SAG (for suitable m).

- Unbiased: $\mathbb{E}[\nabla f_{i_k}(w_s)] = \nabla f(w_s)$ (special case of "control variate").
- Theoretically m depends on L, μ, and n (some analyses randomize it).
- In practice $m = n$ seems to work well.
  - $O(d)$ storage at average cost of 3 gradients per iteration.

End of Part 2: Key Ideas

- Typical ML problems are written as optimization problem

$$\operatorname*{argmin}_{w \in \mathbb{R}^d} F(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w^\top x^i) + \lambda r(w).$$

- Coordinate optimization:
  - Faster than gradient descent if iterations are d-times cheaper.
  - Allows non-smooth r if it's separable.
- Stochastic subgradient:
  - Iteration cost is n-times cheaper than [sub]gradient descent.
  - For non-smooth problems, convergence rate is same as subgradient method.
  - For smooth problems, number of iterations is much higher than gradient descent.
  - Effect of constant step size and batch size.
- SAG and SVRG:
  - Special case when F is smooth.
  - Same low cost as stochastic gradient methods.
  - But similar convergence rate to gradient descent (many extensions exist).

Even Bigger Problems?

- What about datasets that don't fit on one machine?
  - We need to consider parallel and distributed optimization.
- New issues:
  - Synchronization: we may not want to wait for the slowest machine.
  - Communication: it's expensive to transfer data and parameters across machines.
  - Failures: in huge-scale settings, machine failure probability is non-trivial.
  - Batch size: for SGD is it better to get more parallelism or more iterations?

- "Embarassingly" parallel solution:
  - Split data across machines, each machine computes gradient of their subset.
  - Papers present more fancy methods, but always try this first ("linear speedup").

- Fancier methods:
  - Asyncronous stochastic subgradient (works fine if you make the step-size smaller).
  - Parallel coordinate optimization (works fine if you make the step-size smaller).
  - Decentralized gradient (needs a smaller step-size and an "EXTRA" trick).

Skipped Topics: Kernel Methods and Dual Methods

- In previous years, I've covered the following topics:
  1. Kernel methods:
     - Allows using some exponential- or infinite-sized feature sets.
     - Allows defining a "similarity" between training examples rather than features.
     - Mercer's theorem and how to determine if a kernel is valid.
     - Representer theorem and models allowing kernel trick.
     - Multiple kernel learning and connection to structured sparsity.
     - Large-scale kernel approximations that avoid the high cost.
  2. Dual methods:
     - Lagrangian function, dual function, and convex conjugate.
     - Fenchel dual for deriving duals of "loss plus regularizer" problems.
     - Connection between stochastic subgradient method and dual coordinate ascent.
     - Turning non-smooth problems into equivalent smooth problems.
     - Line-search for stochastic subgradient methods.
- If you're interested, I put the slides on these topics here:

https://www.cs.ubc.ca/~schmidtm/Courses/540-W19/L12.5.pdf

Summary

- Mini-batches and effect of batch size.:
  - Doubling batch size halves the variance.
  - Growing batch size leads to faster rate in terms of iterations.
    - And makes it easier to set the step-size and use Newton-like methods.

- Stochastic average gradient: $O(\log(1/\epsilon))$ iterations with 1 gradient per iteration.
- SVRG removes the memory requirement of SAG.

- Next time: optimization with $n = \infty$ (possibly non-IID).

SAG Practical Implementation Issues

- Implementation tricks:
  - Improve performance at start using $\frac{1}{m}g$ instead of $\frac{1}{n}g$.
    - m is the number of examples visited.
  - Common to use $\alpha_k = 1/L$ and use adaptive L.
    - Start with $\hat{L} = 1$ and double it whenever we don't satisfiy

$$f_{i_k}\left(w^k - \frac{1}{\hat{L}}\nabla f_{i_k}(w^k)\right) \leq f_{i_k}(w^k) - \frac{1}{2\hat{L}}\|\nabla f_{i_k}(w^k)\|^2,$$

      and $\|\nabla f_{i_k}(w^k)\|$ is non-trivial. Costs $O(1)$ for linear models in terms of n and d.
  - Can use $\|w^{k+1} - w^k\|/\alpha = \frac{1}{n}\|g\| \approx \|\nabla f(w^k)\|$ to decide when to stop.
  - Lipschitz sampling of examples improves convergence rate:
    - As with coordinate descent, sample the ones that can change quickly more often.
    - For classic SG methods, this only changes constants.