CPSC 540: Machine Learning
Stochastic Average Gradient

Mark Schmidt
University of British Columbia

Winter 2019
Stochastic Average Gradient Structured Prediction

Last Time: Better Methods for Smooth Objectives and Finite Datasets

Stochastic vs. deterministic methods
• Goal = best of both worlds: linear rate with $O(1)$ iteration cost

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 $|B^k|$ 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

\[ w^{k+1} = w^k - \frac{\alpha_k}{n} \sum_{i=1}^{n} v_{i_k}^k, \]

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 out of date.

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

- Key proof idea: \( v_{i_k}^k \to \nabla f_i(w^*) \) 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 $\nabla f_i$ is $L$-continuous and $f$ is strongly-convex, with $\alpha_k = 1/16L$ SAG has

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

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

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:

- 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.

- Start with $w_0$
- for $s = 0, 1, 2 \ldots$
  - $\nabla f(w_s) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w_s)$
  - $w^0 = w_s$
  - for $k = 0, 2, \ldots m - 1$
    - Randomly pick $i_k \in \{1, 2, \ldots, n\}$
    - $w^{k+1} = w^k - \alpha_k (\nabla f_{i_k}(w^k) - \nabla f_{i_k}(w_s) + \nabla f(w_s))$.
  - $w_{s+1} = w^m$.

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 $L, \mu$, 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.
Stochastic Gradient for Stochastic Objectives

- Our analysis of stochastic gradient only used two assumptions on $\nabla f_i(w^k)$:
  1. Unbiased approximation of subgradient: $\mathbb{E}[\nabla f_i(w^k)] = \nabla f(w^k)$.
  2. Variance is bounded: $\mathbb{E}[\|\nabla f_i(w^k)\|^2] \leq \sigma^2$.

- Unlike SAG/SVRG, “classic” SGD does not need to assume dataset is finite.

- We can apply stochastic gradient to minimize objectives written as expectations,

$$\arg\min_{w \in \mathbb{R}^d} \mathbb{E}[f_i(w)],$$

as long as we can sample unbiased estimates of the gradient.

- For example, drop out adds randomization to each example.
- Most important example is the test loss....
Consider a scenario where we have infinite number of IID samples:

- We can optimize the test loss,

\[ \arg\min_{w \in \mathbb{R}^d} \mathbb{E}[f_i(w)], \]

by applying stochastic gradient on a new IID sample on each iteration.

- In this setting, we are directly optimizing test loss and cannot overfit.
- We require \(O(1/\epsilon)\) iterations/samples to reach test loss accuracy of \(\epsilon\) (under PL).

Though keep in mind that the test loss may not be the test error.

- Linear classifiers approximate 0-1 loss (test error) with logistic/hinge loss (test loss).
Infinite-Data Optimization (“Trade-Offs of Large-Scale Learning”)

- Consider number of training examples so large we can’t go through all examples.
  - Stochastic gradient gets within $\epsilon$ of optimal test loss after $t = O(1/\epsilon)$ iterations.

- How does this compare to sampling $t$ examples and optimizing on these?
  - What we usually do: “minimize regularized training loss”.

- How many samples $t$ before training objective is within $\epsilon$ of test objective?
  - Minimum possible assumptions: $t = O(1/\epsilon^2)$.
  - Realistic assumptions: $t = O(1/\epsilon)$.
  - Strong assumptions: $t = O(\log(1/\epsilon))$.

- “Realistic”: $n$ iterations of stochastic gradient on $n$ examples is optimal!?!?
  - Almost always worse empirically than methods which do multiple passes.
  - Constants matter for test data (better optimization improves constants).
Strong Growth Condition

- Consider the following assumption ("strong growth condition"),
  \[ \mathbb{E}[\|\nabla f_i(x)\|^2] \leq \sigma \|\nabla f(x)\|^2. \]

- With this assumption, stochastic gradient converges faster (constant step-size):
  - \(O(1/t)\) rate for non-convex functions, instead of \(O(1/\sqrt{t})\).
  - \(O(\rho^t)\) rate for PL functions, instead of \(O(1/t)\).
    - Unlike usual stochastic setting, Nesterov acceleration works.

- **Ridiculous assumption:** \(\nabla f(w) = 0\) implies \(\nabla f_i(w) = 0\) (\(w^*\) minimizes all \(f_i\)).
  - You fit every data-point exactly (data is "interpolated").
  - Makes variance go to 0 as you approach \(w^*\) (no need for SAG/SVRG).

- **Not-ridiculous assumption for over-parameterized models?**
  - Universal kernels or deep neural networks where you can fit every data point.
  - Why constant step-size SGD + momentum is tough to beat for deep learning?
Typical ML problems are written as optimization problem

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

Convex optimization packages:
- For the special case when $F$ is convex and $d$ is small.
- Many objectives can be re-written as linear or quadratic programs.

Gradient descent:
- Applies when $F$ is differentiable, yields iteration cost that is linear in $d$.
- Needs $O(1/\epsilon)$ iterations in general, only $O(\log(1/\epsilon))$ for PL functions.
- Faster versions like Nesterov’s and Newton-like methods exist.

Proximal gradient:
- Applies when $f_i$ is differentiable and $r$ is “simple” (like L1-regularization).
- Similar convergence properties to gradient descent, even for non-smooth $r$.
- Special case is projected gradient, which allows “simple” constraints.
End of Part 1: Key Ideas

- Typical ML problems are written as optimization problem

\[
\arg \min_{w \in \mathbb{R}^d} F(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w^\top x^i) + \lambda r(w).
\]

- 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, and allows \(n = \infty\).
  - For non-smooth problems, convergence rate is same as subgradient method.
  - For smooth problems, number of iterations is much higher than gradient descent.

- SAG and SVRG:
  - Special case when \(F\) is smooth.
  - Same low cost as stochastic gradient methods.
  - But similar convergence rate to gradient descent.
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:
  - Asycronous 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).
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
Outline

1 Stochastic Average Gradient

2 Structured Prediction
Motivation: Structured Prediction

Classic **supervised learning** focuses on predicting single discrete/continuous label:

Input: $P$

Output: "P"

Structured prediction allows **general objects** as labels:

Input: Paris

Output: "Paris"
Two ways to formulate as “classic” machine learning:

1. Treat each word as a different class label.
   - Problem: there are too many possible words.
   - You will never recognize new words.

2. Predict each letter individually:
   - Works if you are really good at predicting individual letters.
   - But some tasks don’t have a natural decomposition.
   - Ignores dependencies between letters.
Motivation: Structured Prediction

- What letter is this?
  - v

- What are these letters?
  - Vancouver

- Predict each letter using “classic” ML and features from neighbouring images?
  - Can be good or bad depending on goal:
    - Good if you want to predict individual letters.
    - Bad if goal is to predict entire word.
Stochastic Average Gradient

Examples of Structured Prediction

Translate

I moved to Canada in 2013, as indicated on my 2013 declaration of revenue. I received no income from French sources in 2014. How can I owe 12 thousand Euros?


The teacher praised the student.
Examples of Structured Prediction

Coding Regions

Non-coding Regions
(Containing large TE content)
Examples of Structured Prediction
Does the brain do structured prediction?

Gestalt effect: “whole is other than the sum of the parts”.

What do you see?
By shifting perspective you might see an old woman or a young woman.
Summary

- **Stochastic average gradient**: $O(\log(1/\epsilon))$ iterations with 1 gradient per iteration.
- **SVRG** removes the memory requirement of SAG.
- **Infinite datasets** can be handled with stochastic subgradient methods.
  - This is theoretically “optimal” in some settings, not optimal in practice.
- **Strong growth condition** might be the right way to view neural network objectives.
- **Structured prediction**: supervised learning with complicated “labels”.

- Next time: everyone’s favourite distributions...
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 satisfy
      \[
      f_{i_k} \left( w^k - \frac{1}{\hat{L}} \nabla f_{i_k} (w^k) \right) \leq f_{i_k} (w^k) - \frac{1}{2\hat{L}} \| \nabla f_{i_k} (w^k) \|^2,
      \]
      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.

- 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.