Numerical Optimization for Machine Learning

Variance Reduction and 1.5-Order Methods

Mark Schmidt

University of British Columbia

Summer 2022

Last Time: Constant Steps, Mini-Batches, and Over-Parameterization

- With constant step size α, under PL SGD satisfies

$$f(w^k) - f^* \leq \rho(\alpha)^k (f(w^0) - f^*) + O(\alpha\sigma^2/m),$$

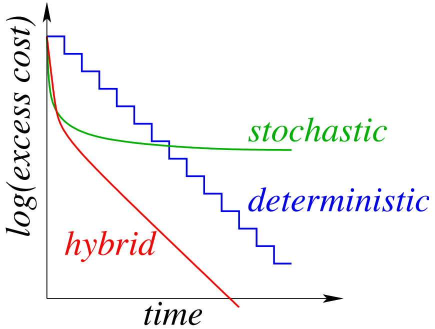
  where m is the mini-batch size.
  - Linear convergence up to some solution accuracy.
  - Solution accuracy proportional to step size and inversely proportional to batch size.

- We discussed growing batch strategies and over-parameterization:
  - Gives fast convergence of SGD with constant step size.
  - Allows using deterministic tricks like line search.
    - But over-parameterization is a strong assumtpion and growing batches increases cost.

- Today: avoiding high iteration costs or over-parameterization assumptions?

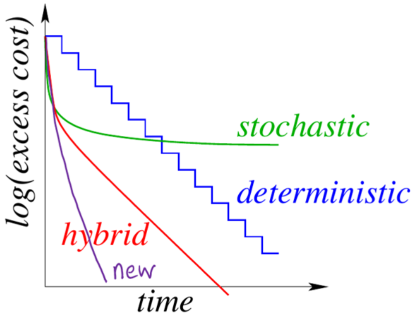
Outline

Deterministic vs. Stochastic vs. Hybrid



- 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.

Deterministic vs. Stochastic vs. Hybrid



- 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:

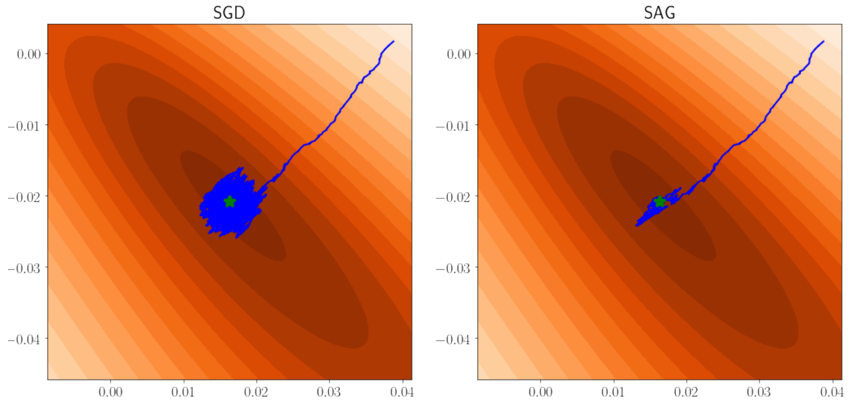
$$\begin{bmatrix} \text{———} & v_1 & \text{———} \\ \text{———} & v_2 & \text{———} \\ & \vdots & \\ \text{———} & v_n & \text{———} \end{bmatrix},$$

  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.

SGD vs. SAG

- SAG update leads to convergence with a constant step size:



- Without needing to assume over-parameterization or growing batches.

Stochastic Average Gradient

- Basic SAG algorithm (maintains $g = \sum_{i=1}^n v_i$):
  - Set $g = 0$ and gradient approximation $v_i = 0$ for $i = 1, 2, \dots, n$.
  - while(1)
    - Sample i from $\{1, 2, \dots, n\}$.
    - 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: $v_i^k \rightarrow \nabla f_i(w^*)$ at the same rate that $w^k \rightarrow w^*$:
  - So the variance $\|e_k\|^2$ ("bad term") converges linearly to 0.

Convergence Rate of SAG

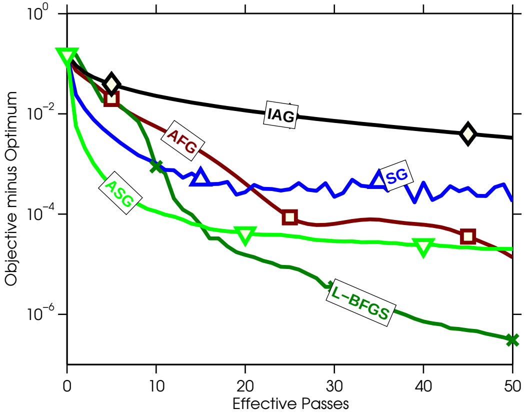
If each ∇f_i is $L-$continuous and f is strongly-convex, with $\alpha_k = 1/16L$ SAG has

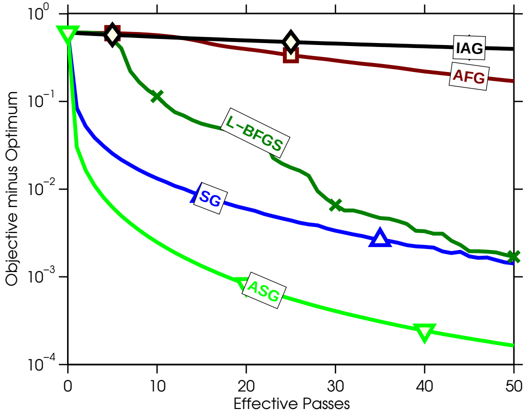
$$\mathbb{E}[f(w^k) - f(w^*)] \leqslant O\left(\left(1 - \min\left\{\frac{\mu}{16L}, \frac{1}{8n}\right\}\right)^k\right)$$

- 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 different between algorithms.

Comparing Deterministic and Stochastic Methods

- Two benchmark L2-regularized logistic regression datasets:

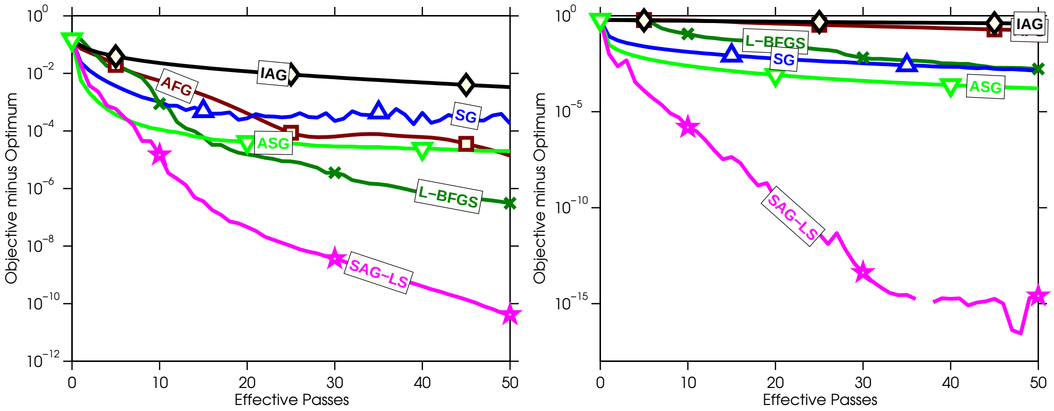




- Averaging makes SG work better, deterministic methods eventually catch up.

SAG Compared to Deterministic/Stochastic Methods

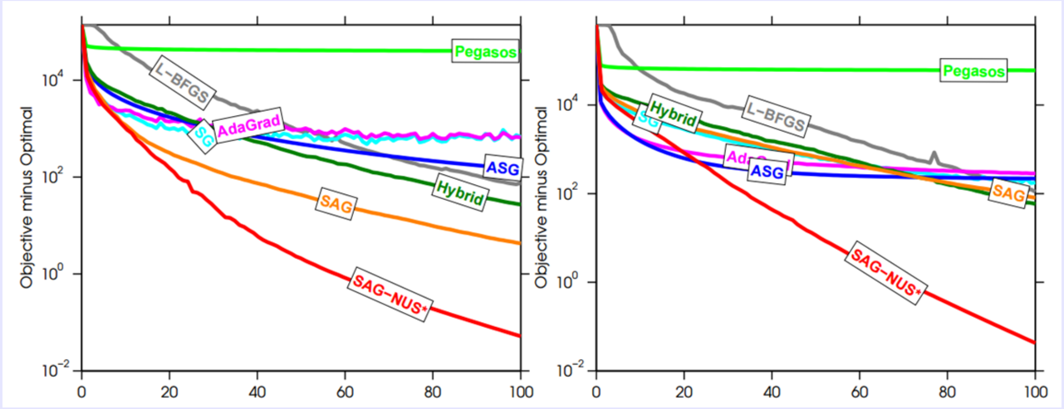
- Two benchmark L2-regularized logistic regression datasets:



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

SAG Compared to Deterministic/Stochastic Methods

- Comparison of methods to train a conditional random field:



- SAG-NUS* is a variation on Lipschitz sampling using local approximations $\hat{L}_i$.
  - Bonus slide discusses various practical implementation issues.

Outline

Variance-Reduced Stochastic Gradient Methods

- Now exists a variety of fast stochastic finite for finite-sum problems:
    - SDCA, MISO, mixedGrad, SVRG, Finito, SAGA, SARAH, SPIDER, and so on.

- Strategies to develop faster methods:
    - Non-uniform sampling using Lipschitz constants of examples:
        - Improves complexity from $\tilde{O}(n + L_{\max}/\mu))$ to $\tilde{O}(n + \bar{L}/\mu)$.
    - Accelerated methods:
        - Improves complexity from $\tilde{O}(n + L_{\max}/\mu)$ to $\tilde{O}(n + \sqrt{nL_{\max}/\mu})$.
    - Newton-like methods and methods designed for non-convex problems.
        - Still active area of research, achieve faster rates in some settings.

- There are also methods that reduce the memory to $O(d)$.
    - Most common approach is stochastic variance-reduced gradient (SVRG).
    - We will first cover a simpler but non-implementable method called SGD*.

SVRG Warm-Up: SGD*

- Suppose we knew w^*, and use the following SGD* iteration:

$$w^{k+1} = w^k - \alpha_k(\underbrace{\nabla f_{i_k}(w^k) - \nabla f_{i_k}(w^*)}_{g_k}).$$

- Similar to SGD, using g_k gives an unbiased gradient approximation:

$$\begin{aligned}\mathbb{E}[g_k] &= \mathbb{E}[\nabla f_{i_k}(w^k)] - \mathbb{E}[\nabla f_{i_k}(w^*)] \\ &= \nabla f(w^k) - \underbrace{\nabla f(w^*)}_{0} = \nabla f(w^k).\end{aligned}$$

- But (for convex f_i) you can show gradient approximation goes to 0 as $w^k \to w^*$,

$$\mathbb{E}[\|g_k\|^2] \leq 2L_{\max}(f(w^k) - f^*).$$

- This makes SGD* behave like over-parameterized SGD.
  - And for over-parameterized problems, SGD* is just SGD since $\nabla f_i(w^*) = 0$ for all i.

SGD* Convergence Rate (using Descent Lemma)

- Recall our progress bound for any unbiased SGD method:

$$\mathbb{E}[f(w^{k+1})] \leq f(w^k) - \alpha_k \underbrace{\|g^k\|^2}_{\text{good}} + \underbrace{\alpha_k^2 \frac{L}{2}\mathbb{E}[\|g_k\|^2]}_{\text{bad}}.$$

- Using PL ($\|\nabla f(w^k)\|^2 \geq 2\mu(f(w^k) - f^*)$) and $\mathbb{E}[\|g_k\|^2]$ bound (previous slide),

$$\mathbb{E}[f(w^{k+1})] \leq f(w^k) - 2\alpha_k\mu(f(w^k) - f^*) + \alpha_k^2 L L_{\max}(f(w^k) - f^*).$$

- If you subtact f^* and recurse, then with $\alpha_k = \mu/LL_{\max}$ SGD* satisfies

$$\mathbb{E}[f(w^k) - f^*] \leq \left(1 - \frac{\mu^2}{LL_{\max}}\right)^k [f(w^0) - f^*].$$

- Get a $(1 - \mu/L_{\max})$ rate by analyzing $\|w^k - w^*\|$ instead (using $\alpha_k = 1/L_{\max}$)).
  - We will consider this proof technique later when we disucss non-smooth optimization.

From SGD* to SVRG

- Since $\nabla f(w^*) = 0$, we can re-write SGD* as
$$w^{k+1} = w^k - \alpha_k(\nabla f_{i_k}(w^k) - \nabla f_{i_k}(w^*) + \underbrace{\nabla f(w^*)}_{0}),$$
which achieves fast rate without a memory by evaluating 2 gradients per iteration.
    - We evalute ∇f_{i_k} at w^k and w^*.
- This is a special case of using a control variate estimate of the graident.
    - "Add random variable and subtract its mean" .
    - Gives an unbiased Monte Carlo estimate, that can have reduced variance.
- Stochastic variance-reduced gradient (SVRG) uses a similar control variate:
$$w^{k+1} = w^k - \alpha_k(\nabla f_{i_k}(w^k) - \nabla f_{i_k}(v^k) + \nabla f(v^k)),$$
where v^k is some previous iterate rather than the global minimum w^*.

Stochastic Variance Reduced Gradient Method

- The SVRG iteration

$$w^{k+1} = w^k - \alpha_k \underbrace{(\nabla f_{i_k}(w^k) - \nabla f_{i_k}(v^k) + \nabla f(v^k))}_{g_k}.$$

- Unlike SAG, but similar to SGD*, this gives an unbiased gradient approximation:

$$\mathbb{E}[g_k] = \nabla f(w^k) \underbrace{-\mathbb{E}[\nabla f_{i_k}(v^k)] + \nabla f(v^k)}_{0}.$$

- And can show that gradient approximation goes to 0 as w^k and v^k approach w^*,

$$\mathbb{E}[\|g_k\|^2] \leq 4L(f(w^k) - f^*) + 4L(f(v^k) - f^*).$$

Stochastic Variance Reduced Gradient Method

- To implement the SVRG iterations,

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

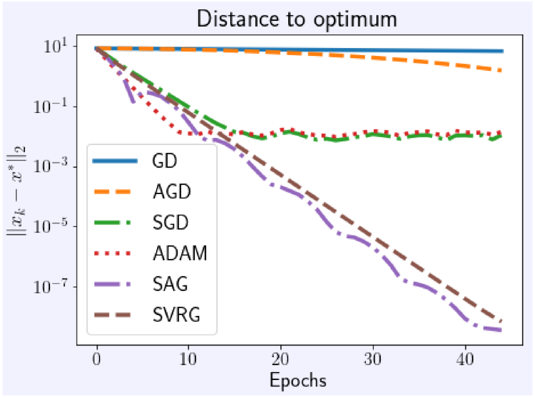
  we have two types of iterations:
  1. On most iterations we set $v^k = v^{k-1}$ ("cheap iterations").
     - These iterations only require 2 gradient evaluations if we have stored $\nabla f(v^{k-1})$.
  2. On some iterations we set $v^k = w^k$ ("expensive iterations").
     - These iterations cost n gradients evaluations to update $\nabla f(v^k)$.

- Obtain fast rates under appropriate α_k and expensive iteration frequency.
  - If $\alpha_k = 1/6L$ and we update v^k with probability $1/n$:
    - SVRG achieves the SAG complexity of $\tilde{O}((n + L_{\text{max}}/\mu))$.
    - With the standard $O(d)$ memory and an average cost of 3 gradients per iteration.

- In practice, using $\alpha_k = 1/L$ and updating v^k every n iterations often works well.
  - And you can/should using a growing-batch estimate of $\nabla f(v^k)$ ("practical SVRG").

Stochastic Variance Reduced Gradient Method

- Comparison of various methods for fitting a logistic regerssion model:



- The above is with “no tricks”. With tricks SAG tends to outperform SVRG.

SVRG for Deep Learning?

- Variance-reduced methods are not typically used for deep learning.
    - SVRG does not converge noticeably faster for neural networks.
- This might be because we often use over-parameterized neural networks.
    - For over-parameterzied models SVRG still needs $\tilde{O}(n + L_{\text{max}}/\mu)$ iterations.
    - But plain SGD only needs $\tilde{O}(L_{\text{max}}/\mu)$ iterations.
- Or it could be that networks are close to over-parameterized.
    - Or that we do not need to run the methods long enough to see a difference.
- Recent work argues that variance reduction may be useful for GANs.

Outline

Motivation: Cost of Newton Iterations

- Newton's method is expensive if dimension d is large:
  - Requires solving $\nabla^2 f(w^k)d^k = \nabla f(w^k)$.
  - For logistic regression, this costs $O(nd^2)$ to form Hessian and $O(d^3)$ to solve.

- Many methods proposed to approximate Newton's method at reduced cost.
  1. Cheaper Hessian approximations.
  2. Hessian-free Newton methods.
  3. Quasi-Newton methods.

- We will overview some representative ones.

Cheaper Hessian Approximation #1: Diagonal Hessian

- A simple strategy is to use a diagonal approximation of the Hessian,
$$[\nabla^2 f(w^k)]^{-1} \approx D^k,$$
where D^k is a diagonal matrix.
- This gives the (damped) Newton step the form
$$w^{k+1} = w^k - \alpha_k D^k \nabla f(w^k),$$
which only costs $O(d)$ instead of $O(d^3)$ to compute.
- A common choice is using inverse of Hessian diagonals $D^k_{ii} = (\nabla^2_{ii} f(w^k))^{-1}$.
  - Corresponding to a coordinate-wise Newton step along each dimension.
- Diagonal approximations lose superlinear convergence.
  - For some problems Hessian diagonals outperforms gradient descent.
  - For many problems using Hessian diagonals is worse than gradient descent.

Cheap Hessian Approximation #2: Preconditioning

- Some methods use a Newton-style update with a positive-definite fixed matrix,

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

- Matrix M could be chosen to include some second-order information.
  - And may be chosen so that multiplication by M costs less than $O(d^2)$.

- We call this approach preconditioning (details in bonus slides).
  - It can be viewed as performing gradient descent under change of variables.
    - Choosing a matrix R such that $RR^T = M$ (like Cholesky factorization).
    - Preconditioned update corresponds to gradient descent on v, where $w = Rv$.
  - Convergence rate (for C^2) functions depends on $R^T\nabla^2 f(Rw)R$ instead of $\nabla^2 f(w)$.
    - For strongly-convex quadratics, ideal preconditioner would be $M = [\nabla^2 f(w)]^{-1}$.

Preconditioner Variation: Matrix Upper Bound

- Our usual Lipschitz continuity assumption on the gradient is that

$$\|\nabla f(w) - \nabla f(v)\| \leq L\|w - v\|.$$

- We could instead assume 1-Lipschitz continuity with respect to a matrix M,

$$\|\nabla f(w) - \nabla f(v)\|_{M^{-1}} \leq \|w - v\|_M,$$

  where $\|d\|_M = \sqrt{d^T M d}$ and we assume M is positive definite.

- For quadratic functions, we can use $M = \nabla^2 f(w)$ and we get Newton.

- For binary logistic regression, we can use $M = \frac{1}{4}X^T X$.
  - We have $\nabla^2 f(w) = X^T D(w) X$, where diagonal $D(w)$ has diagonal entries $\leq \frac{1}{4}$.

Preconditioner Variation: Matrix Upper Bound

- The matrix-norm Lipschitz continuity leads to a descent lemma of the form

$$f(w^{k+1}) \leq f(w^k) + \nabla f(w^k)^T(w^{k+1} - w^k) + \frac{1}{2}\|w^{k+1} - w^k\|_M^2,$$

  and minimizing the righ side yields the Newton-like step

$$w^{k+1} = w^k - M^{-1}\nabla f(w^k).$$

- This step does not require a step size and guarantees descent.
  - With appropriate M guarantees more progress per iteration than gradient descent.
    - Which is more than we can say about Newton when not close to the solution.
  - Though in practice you may get better performance using a line-search.

- But loses superlinear convergence and cost is still $O(d^2)$ per iteration.
  - Or $O(d)$ if M is diagonal.
- And not obvious how to find upper-bound matrices M (backtracking?).

Cheaper Hessian Approximation #3: Mini-Batch Hessian

- For ML problems some have explored using a mini-batch Hessian approximation,

$$\nabla^2 f(w^k) = \frac{1}{n}\sum_{i=1}^{n} \nabla^2 f_i(w^k) \approx \frac{1}{|\mathcal{B}|}\sum_{i \in \mathcal{B}} \nabla^2 f_i(w^k),$$

  which removes dependence on n in the Hessian calculation.
  - Newton update can be solved quickly under this approximation for some problems.
  - For L2-regularized logistic regression, costs $O(|\mathcal{B}|^2 d + |\mathcal{B}|^3)$.
    - By using kernelized version.

- Leads to superlinear convergence if batch size grows fast enough.
- But for general problems still require $O(d^3)$ to solve Newton system.

Hessian-Free Newton Methods ("Truncated Newton")

- Cheap Hessian methods approximate $\nabla^2 f(w^k)$, and lose superlinear convergence.
- Hessian-free Newton use exact $\nabla^2 f(w^k)$ but approximates the Newton direction.

- As an example, for strongly-convex f Newton's method minimizes a quadratic,

$$\underset{g}{\text{argmin}}\, f(w^k) + \nabla f(w^k)^T g + \frac{1}{2} g^T \nabla^2 f(w^k) g,$$

- We have good first-order methods for minimizing strongly-convex quadratics.
  - You could use conjugate gradient (heavy-ball with optimal α_k and β_k on each step).
- To use a first-order method we need to compute gradient with respect to g,

$$0 + \nabla f(w^k) + \nabla^2 f(w^k) g,$$

  which requires Hessian-vector products.
  - So why is it called "Hessian-free"?

Hessian-Vector Products are Cheap

- Cost of a Hessian-vector product is at most the cost of computing gradient.

- Example: for binary logistic regression we have

$$\nabla f(w) = X^T r(w), \quad \nabla^2 f(w) = X^T D(w) X,$$

where $r(w)$ and $D(w)$ each cost $O(n)$ to compute for n training examples.

  - Cost of computing gradient is $O(nd)$ due to the matrix-vector product.
  - Cost of computing Hessian is $O(nd^2)$ due to the matrix-matrix product.

  - But cost of computing Hessian-vector product is only $O(nd)$,

$$\begin{aligned}\nabla^2 f(w)d &= X^T D(w) X d \\ &= X^T (D(w)(Xd)).\end{aligned}$$

due to the matrix-vector products.

Hessian-Vector Products and Automatic Differentiation

- More generally, Hessian-vector product is a directional derivative of gradient,

$$\nabla^2 f(w)g = \lim_{\delta \to 0} \frac{1}{\delta}(\nabla f(w + \delta g) - \nabla f(w)).$$

- You could thus use a finite-difference approximation of Hessian-vector product.
- Or you could compute exactly with forward-mode automatic differentiation.
    - This is different than the usual "reverse mode" we use to get gradients.
- Gives Hessian-vector product for cost of computing gradient.
    - Does not have the high memory requirements of reverse mode.
    - No need to worry about things like "checkpointing".
- Bonus slide shows complex-step derivative if you do not have AD code.
    - Allows computing Hessian-vector products to arbitrary accuracy using complex numbers.

Hessian-Free Newton - Local Convergence Rates

- Key ideas behind Hessian-free Newton methods:
  - Approximately compute Newton direction using conjugate gradient.
  - Each iteration of conjugate gradient only needs a (cheap) Hessian-vector product.
- Key to reducing iteration cost compared to exact Newton method:
  - We do not run conjugate gradient to convergence.
    - Hessian-free Newton is also called "truncated Newton" or "inexact Newton".
- Local convergence rates of Hessian-free Newton depend on the accuracy:
  - Let $r^k = \nabla f(w^k) + \nabla^2 f(w^k) g^k$ be the gradient for the final g^k.
    - We get linear convergence if $\|r^k\| \leq \eta_k \|\nabla f(w^k)\|$ for $\eta_k \leq \eta < 1$.
    - We get superlinear convergence if the above holds with $\eta_k \to 0$.
    - We get quadratic convergence if $\eta_k = O(\|\nabla f(w^k)\|)$.
  - For superlinear convergence, a typical forcing sequence is

$$\eta_k = \min\{0.5, \sqrt{\|\nabla f(w^k)\|}\},$$

  which forces CG to solve Newton system more accurately as the gradient decreases.

Hessian-Free Netwon - Globalization and Negative Curvature

- To ensure convergence, you still need to use a globalization strategy.
    - Use the approximate Newton direction generated by CG within a line-search.
    - Or run CG until you are outside the trust region raidus.

- Conjugate gradient only applies to convex quadratic functions.
    - For non-convex problems, the Hessian may have negative eigenvalues.

- During the CG iterations, we can test whether $d^T \nabla^2 f(w^k) d < 0$.
    - If so, we have detected a direction of negative curvature.
        - We usually stop running CG if this is detected.
    - Descent directions of negative curvature can make excellent search directions.
        - "The function starts decreasing faster if we move in this direction".
        - Some codes switch to using a precise line-search if such a direction is found.

Sketched Newton - Random Hessian-Vector Products

- Several recent works consider sketched Newton methods.
  - Performs Hessian-vector products with random directions.
  - Uses the resulting vectors to build an approximation to the Hessian.
- Usually converges to Newton direction slower than running conjugate gradient.
  - But random Hessian-vector products can be computed in parallel.
  - Alternately, for some problems allows faster Hessian-vector products.
    - By using the structure of the Hessian and sparse random vectors.

2.5-Order Methods

- Key to Hessian-free Newton methods is cost of Hessian-vector products.
  - Hessian-vector products have same cost as computing gradient.
  - Allows us to implement an approximate second-order method.
- But consider a scenario where we can afford to compute the Hessian.
  - Can compute tensor-vector products with 3rd-order tensor for same cost.
  - Allows us to implement an approximate third-order method.
- A "tensor-free" method might use tensor-vector products to try to minimize

$$f(w^k) + \nabla f(w^k)^T g + \frac{1}{2} g^T \nabla^2 f(w^k) g + \frac{1}{6} \nabla^3 f(w^k)[g]^3 + \frac{T}{24}\|g\|^4,$$

  where T is the Lipschitz constant of the third-order tensor.
  - Third-order methods give faster rates.
  - In practice I have found that they usually only save 1 iteration compared to Newton.
    - This is sensible because unless g is close to 0 the approximation is not good.

Outline

Quasi-Newton Methods: Overview

- We have discussed methods that use limited information about current Hessian.
  - Diagonals of Hessian, Hessian-vector products, and so on.

- Quasi-Newton build a sequence of Hessian approximations B_0, B_1, B_2,..., and use
$$w^{k+1} = w^k - \alpha_k B_k^{-1} \nabla f(w^k),$$
with the goal that approximations eventually act like the Hessian.
  - Typically used with a line-search that initially tries $\alpha_k = 1$.

- Classic quasi-Newton methods choose B_k to satisfy the secant equations,
$$B_{k+1}(w^k - w^{k-1}) = \nabla f(w^k) - \nabla f(w^{k-1}),$$
which only uses iterate and gradient differences (no Hessian information).
  - Roughly, "multiplying by B_{k+1} acts like a Hessian vector product".
  - Secant equations give superlinear local convergence for one-dimensional problems.

Barzilai-Borwein Method: Quasi-Newton with Scaled Identity

- A simple quasi-Newton method is the Barzilai-Borwein method.
- Uses an approximation of the form $B_k = (1/\alpha_k)I$ for a scalar α_k.
  - So it is equivalent to gradient descent with a particular step size.
- This B_k cannot always solve the secant equations, so we minimize squared error,
$$\alpha_{k+1} \in \underset{\alpha}{\text{argmin}} \|B_{k+1}(w^k - w^{k-1}) - (\nabla f(w^k) - \nabla f(w^{k-1}))\|^2,$$
which gives
$$\alpha_{k+1} = \frac{\|w^k - w^{k-1}\|^2}{(w^k - w^{k-1})^T(\nabla f(w^k) - \nabla f(w^{k-1}))}$$
- Barzilai and Borwein showed this gives superlinear convergence for 2d quadratics.
  - Now extended to 3d quadratics, but not faster than gradient descent in general.
- Often used with safeguards and "non-monotonic Armijo" line-search.
  - Empirical convergence rate is often very fast, but almost no theory on why.

Alternate Secant Equations and BB Step Size

- Usual secant equations are

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

  but we could alternately require inverse to satisfy secant equations,

$$(w^k - w^{k-1}) = [B_{k+1}]^{-1} \nabla f(w^k) - \nabla f(w^{k-1}).$$

- This gives an alternate Barzilai-Borwein step size of

$$\alpha_{k+1} = \frac{(w^k - w^{k-1})^T(\nabla f(w^k) - \nabla f(w^{k-1}))}{\|\nabla f(w^k) - \nabla f(w^{k-1})\|^2},$$

  which is the one used in *findMin* and my previous demos.

BFGS Quasi-Newton Method

- Most quasi-Newton methods use dense matrices B_k.
  - In this case there may be an infinite number of solutions to secant equations.
- Many methods exist, and typical methods also require:
  - B_{k+1} to be symmetric.
  - B_{k+1} to be close to B_k under some norm.
- Most popular is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) update:

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k},$$

where $s_k = w^k - w^{k-1}$ and $y_k = \nabla f(w^k) - \nabla f(w^{k-1})$.

- Derived as rank-2 update which stays close to previous matrix in some norm.

BFGS Convergence

- If $y_k^T s_k > 0$ and B_k is positive-definite, then B_{k+1} is positive-definite.
  - Some implementations "skip" updates when this does not hold (other "dampen").
  - Guaranteed to have $y_k^T s_k > 0$ if f is strongly-convex or we use Wolfe line-search.
- BFGS with standard line-searches converges if $\gamma_1 I \preceq B_k \preceq \gamma_2 I$.
  - For some $\gamma_1 > 0$ and $\gamma_2 < \infty$, for all k.
  - Unfortunately, this may not hold for the updates.
- If BFGS does converge to minimizer, then local rate is suplinear.
  - Under typical assumptions like strong convexity and Lipshictz-continuity of Hessian.

Limited-Memory BFGS (L-BFGS)

- Cost of inverting a dense B_k is $O(d^3)$.
  - For BFGS this can reduced to $O(d^2)$ using a matrix inversion formula.

- Limited-memory BFGS (L-BFGS) reduces the cost/memory to $O(md)$.
  - Instead of storing B_k, only stores m vectors s_k and y_k.
  - Uses an update based on a matrix H_k and this "limited" memory.
  - Applies the BFGS update m times starting from H_k.
    - Recursive algorithm costs $O(md)$, plus the cost of inverting H_k.
  - Typically we choose $H_k = \alpha_k I$ for some α_k.

- L-BFGS is widely-used and is often the "default" deterministic optimizer.
  - Hard to beat on many problems, and linear cost allows scaling to large problems.
  - With limited memory, L-BFGS loses the superlinear convergence of BFGS.
    - And inherits the potential for non-convergence.
  - Explaining when L-BFGS works and does not work is an open problem.

Initializing BFGS and L-BFGS

- Performance of BFGS depends heavily on B_0.
  - A poor choice can lead to many poor iterations at the start.
- A choice that often works well is $B_0 = \alpha_{\text{BB}}^{-1} I$.
  - Where α_{BB} is the alternate Barzilai-Borwein step size after the first iteration.
    - So we do a gradient descent step on iteration 1, then choose the "initial" matrix.
- For L-BFGS, we can use this scaling on each iteration.
  - We do this by setting H_k to the the Barzilai-Borwein approximation.
  - This "trick" often drastically improves performance of L-BFGS, even over BFGS.

Other Quasi-Newton Methods

- An alternative to BFGS is the symmetric rank-1 (SR1) update.
  - A rank-1 update that gives a better Hessian approximation than BFGS.
  - Does not maintain positive-definitness.
    - This is annoying for line-search methods but may be better for non-convex problems.

- There exist methods that combine Hessian-free and quasi-Newton methods.
  - For example, use L-BFGS matrix as a preconditioner for Hessian-free Newton.
  - For some problems this drastically reduces number of CG iterations.

- In the last few years, explicit superlinear rates have been derived.
  - First papers considered greedy/random quasi-Newton methods.
  - More recently, explicit rates have been derived for BFGS and SR1.

Numerical Comparison with minFunc

In my experience L-BFGS performs best for many problems.

- But for some problems Hessian-free Newton or non-linear CG are better.
- Barzilai-Borwein is a great choice if you have to implement from scratch.

Result after 25 evaluations of limited-memory solvers on 2D rosenbrock:

---

x1 = 0.0000, x2 = 0.0000 (starting point)
x1 = 1.0000, x2 = 1.0000 (optimal solution)

---

x1 = 0.3654, x2 = 0.1230 (minFunc with gradient descent)
x1 = 0.8756, x2 = 0.7661 (minFunc with Barzilai-Borwein)
x1 = 0.5840, x2 = 0.3169 (minFunc with Hessian-free Newton)
x1 = 0.7478, x2 = 0.5559 (minFunc with preconditioned Hessian-free Newton)
x1 = 1.0010, x2 = 1.0020 (minFunc with non-linear conjugate gradient)
x1 = 1.0000, x2 = 1.0000 (minFunc with limited-memory BFGS - default)

Summary

- Stochastic average gradient: $O(\log(1/\epsilon))$ iterations with 1 gradient per iteration.
- SVRG removes the memory requirement of SAG.
- Cheap Hessian approximations are used to reduce cost of Newton.
  - Diagonal approximations are the most common.
- Hessian-free Newton uses first-order method to solve Newton system.
  - Relies on cheap Hessian-vector products, and usually conjugate gradient.
- Quasi-Newton build a sequence of approximations to the Hessian.
  - Most popular quasi-Newton methods are variants of BFGS.
    - Superlinear local convergence but convergence not guaranteed.
  - Limited-memory BFGS (L-BFGS) is a variant with linear iteration cost.
    - Only linear convergence but often works well in practice.

- Next time: will probably be in 2 weeks (but might be longer, check website).

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.

Complex-Step Derivative

- The usual finite-difference approximation of derivative:

$$f'(w) \approx \frac{f(w+\delta) - f(w)}{\delta}.$$

  - Has $O(\delta^2)$ error from Taylor expansion.

$$f(w+\delta) = f(w) + \delta f'(w) + O(\delta^2).$$

  - But h cannot be too small: floating-point cancellation in $f(w+\delta) - f(w)$.
- For analytic functions, the complex-step derivative uses

$$f(w+i\delta) = f(w) + i\delta f'(w) + O(\delta^2),$$

  that also gives function and derivative to accuracy $O(\delta^2)$:

$$\text{real}(f(w+i\delta)) = f(w) + O(\delta^2), \quad \frac{\text{imag}(f(w+i\delta))}{\delta} = f'(w) + O(h^2),$$

  which we can use to get Hessian-vector products of arbitrary accuracy.
- First appearance is apparently Squire & Trapp [1998].

Preconditioning and Re-Parameterization

- Consider the preconditioned gradient descent iteration

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

  for some positive-definite matrix M.
- We can interpret this as gradient descent under a change of variables $w^k = Rv^k$ where $M = RR^T$,

$$v^{k+1} = v^k - \alpha_k \nabla g(v^k),$$

  where $g(v) = f(Rw)$.
- Using that $\nabla g(v) = R^T \nabla f(Rv)$ and multiplying update by R gives

$$\underbrace{Rv^{k+1}}_{w^{k+1}} = \underbrace{Rv^k}_{w^k} - \alpha_k \underbrace{RR^T}_{M} \nabla f(\underbrace{Rv^k}_{w^k}),$$

  which in w^k space gives the preconditioned gradient descent iteration above.

Preconditioning and Re-Parameterization

- Previous slide: preconditioning by M can be viewed as gradient descent on $f(Rw)$.

- Changes convergence rate since (if C^2) Hessian of re-parameterized problem is
$$R^T\nabla^2 f(Rw)R.$$

- So instead of usual $\mu I \preceq \nabla^2 f(w) \preceq LI$, we care about eigenvalues of above matrices.
- If we have a quadratic with fixed Hessian H, choose $RR^T = H^{-1}$ and get
$$R^T\nabla^2 f(Rw)R = R^T H^{-1} R = R^T(RR^T)^{-1}R = R^T R^{-T} R^{-1} R = I,$$
so $L = \mu$ and we converge in 1 step.

Preconditioning and Re-Parameterization

- Should we scale the momentum term too?
- If we apply the heavy-ball method in the v space we get

$$v^{k+1} = v^k - \alpha_k \nabla g(v^k) + \beta^k (v^k - v^{k-1}),$$

which in the w space corresponds to (by multiplying by R),

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

so under this logic you would not scale the momentum term.