Convex Optimization for Statistics and Machine Learning

Part II: First-Order Methods

Ryan Tibshirani

Depts. of Statistics & Machine Learning
Carnegie Mellon University

http://www.stat.cmu.edu/~ryantibs/talks/cuso-part2-2019.pdf

Outline for Part II

- Part A. Gradient descent
- Part B. Subgradients
- Part C. Proximal methods
- Part D. Stochastic methods

Part II: First-order methods

A. Gradient descent

Gradient descent

Consider unconstrained, smooth convex optimization

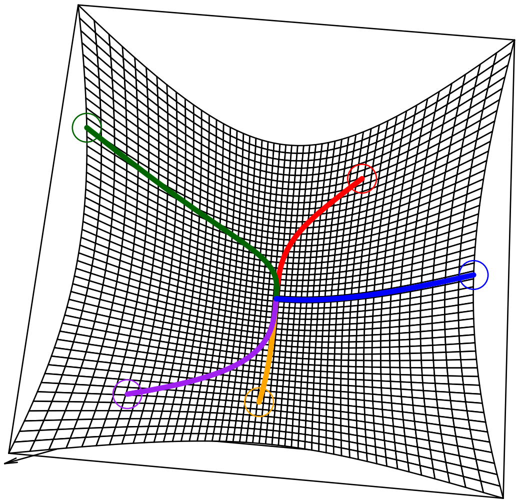
$$\min_x \ f(x)$$

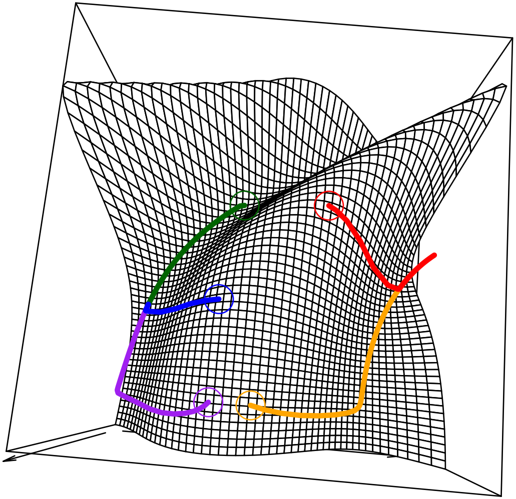
That is, f is convex and differentiable with $\mathrm{dom}(f) = \mathbb{R}^n$. Denote optimal criterion value by $f^\star = \min_x \ f(x)$, and a solution by $x^\star$

Gradient descent: choose initial point $x^{(0)} \in \mathbb{R}^n$, repeat:

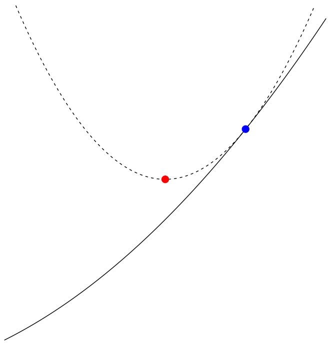
$$x^{(k)} = x^{(k-1)} - t_k \cdot \nabla f(x^{(k-1)}), \quad k = 1, 2, 3, \dots$$

Stop at some point





Gradient descent interpretation



At each iteration, consider quadratic approximation:

$$f(y) \approx f(x) + \nabla f(x)^T (y-x) + \frac{1}{2t}\|y-x\|_2^2$$

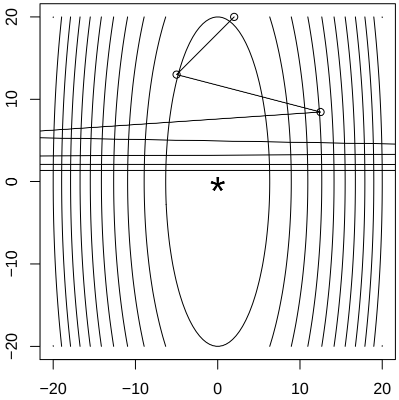
Note the Hessian $\nabla^2 f(x)$ is replaced by $\frac{1}{t}I$

Minimizing the quadratic approximation over y gives

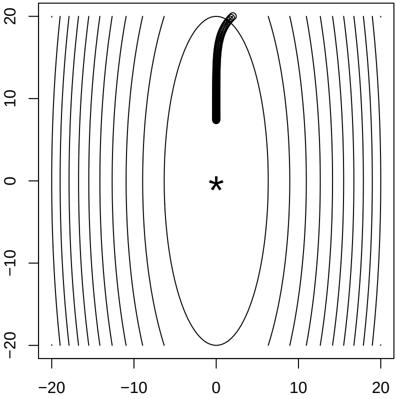
$$x^+ = x - t\nabla f(x)$$

Fixed step size

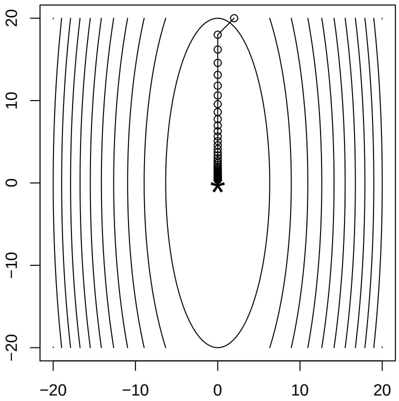
Simply take $t_k = t$ for all $k = 1, 2, 3, \ldots$, can diverge if t is too big. Consider $f(x) = (10x_1^2 + x_2^2)/2$, gradient descent after 8 steps:



Can be slow if t is too small. Same example, gradient descent after 100 steps:



Converges nicely when t is "just right". Same example, 40 steps:



Convergence analysis later will give us a precise idea of "just right"

Backtracking line search

One way to adaptively choose the step size is to use backtracking line search:

- First fix parameters $0 < \beta < 1$ and $0 < \alpha \leq 1/2$
- At each iteration, start with $t = t_{\mathsf{init}}$, and while

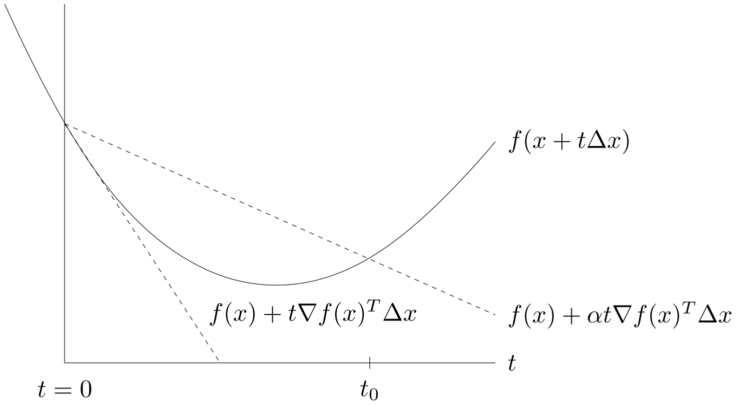
  $$f(x - t\nabla f(x)) > f(x) - \alpha t \|\nabla f(x)\|_2^2$$

  shrink $t = \beta t$. Else perform gradient descent update

  $$x^+ = x - t\nabla f(x)$$

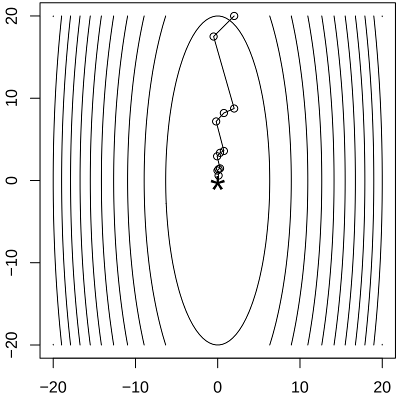
Simple and tends to work well in practice (further simplification: just take $\alpha = 1/2$)

Backtracking interpretation


$f(x + t\Delta x)$
$f(x) + t\nabla f(x)^T\Delta x$
$f(x) + \alpha t\nabla f(x)^T\Delta x$
t
$t = 0$
t_0

For us $\Delta x = -\nabla f(x)$

Setting $\alpha = \beta = 0.5$, backtracking picks up roughly the right step size (12 outer steps, 40 steps total),



Convergence analysis

Assume that f convex and differentiable, with $\mathrm{dom}(f) = \mathbb{R}^n$, and additionally that ∇f is Lipschitz continuous with constant $L > 0$,

$$\|\nabla f(x) - \nabla f(y)\|_2 \le L\|x - y\|_2 \quad \text{for any } x, y$$

(Or when twice differentiable: $\nabla^2 f(x) \preceq LI$)

Theorem: Gradient descent with fixed step size $t \le 1/L$ satisfies

$$f(x^{(k)}) - f^\star \le \frac{\|x^{(0)} - x^\star\|_2^2}{2tk}$$

and same result holds for backtracking, with t replaced by β/L

We say gradient descent has convergence rate $O(1/k)$. That is, it finds ϵ-suboptimal point in $O(1/\epsilon)$ iterations

Analysis for strong convexity

Reminder: strong convexity of f means $f(x) - \frac{m}{2}\|x\|_2^2$ is convex for some $m > 0$ (when twice differentiable: $\nabla^2 f(x) \succeq mI$)

Assuming Lipschitz gradient as before, and also strong convexity:

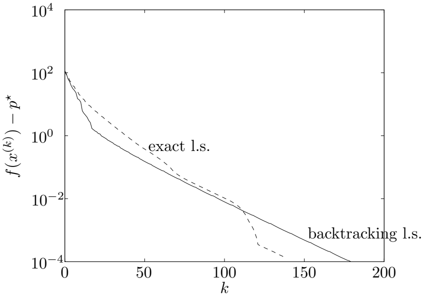
Theorem: Gradient descent with fixed step size $t \le 2/(m+L)$ or with backtracking line search search satisfies

$$f(x^{(k)}) - f^\star \le \gamma^k \frac{L}{2}\|x^{(0)} - x^\star\|_2^2$$

where $0 < \gamma < 1$

Rate under strong convexity is $O(\gamma^k)$, exponentially fast! That is, it finds ϵ-suboptimal point in $O(\log(1/\epsilon))$ iterations

Called linear convergence, because looks linear on a semi-log plot



(From B & V page 487)

Important note: contraction factor c in rate depends adversely on condition number L/m: higher condition number $\Rightarrow$ slower rate

Affects not only our upper bound ... very apparent in practice too

Can we do better?

Gradient descent has $O(1/\epsilon)$ convergence rate over problem class of convex, differentiable functions with Lipschitz gradients

First-order method: iterative method, which updates $x^{(k)}$ in

$$x^{(0)} + \text{span}\{\nabla f(x^{(0)}), \nabla f(x^{(1)}), \ldots \nabla f(x^{(k-1)})\}$$

Theorem (Nesterov): For any $k \leq (n-1)/2$ and any starting point $x^{(0)}$, there is a function f in the problem class such that any first-order method satisfies

$$f(x^{(k)}) - f^\star \geq \frac{3L\|x^{(0)} - x^\star\|_2^2}{32(k+1)^2}$$

Can attain rate $O(1/k^2)$, or $O(1/\sqrt{\epsilon})$? Answer: yes (we'll see)!

Analysis for nonconvex case

Assume f is differentiable with Lipschitz gradient as before, but now nonconvex. Asking for optimality is too much. So we'll settle for x such that $\|\nabla f(x)\|_2 \leq \epsilon$, called ϵ-stationarity

Theorem: Gradient descent with fixed step size $t \leq 1/L$ satisfies

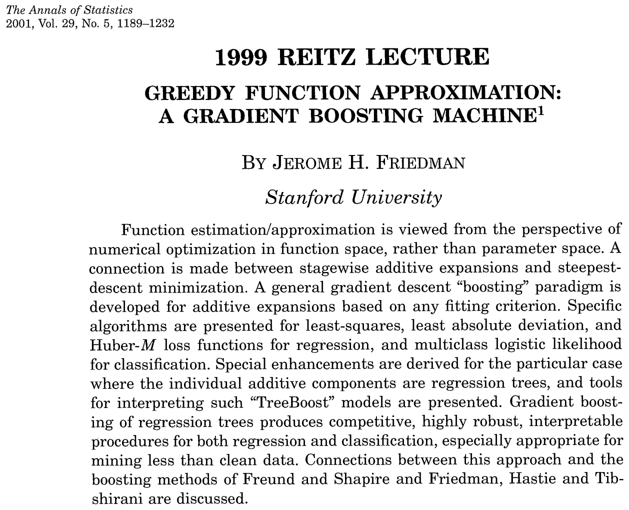
$$\min_{i=0,\ldots,k} \|\nabla f(x^{(i)})\|_2 \leq \sqrt{\frac{2(f(x^{(0)}) - f^\star)}{t(k+1)}}$$

Thus gradient descent has rate $O(1/\sqrt{k})$, or $O(1/\epsilon^2)$, even in the nonconvex case for finding stationary points

This rate cannot be improved (over class of differentiable functions with Lipschitz gradients) by any deterministic algorithm[1]

[1]Carmon et al. (2017), "Lower bounds for finding stationary points I"

Gradient boosting



The Annals of Statistics
2001, Vol. 29, No. 5, 1189–1232

1999 REITZ LECTURE

GREEDY FUNCTION APPROXIMATION: A GRADIENT BOOSTING MACHINE[1]

BY JEROME H. FRIEDMAN

Stanford University

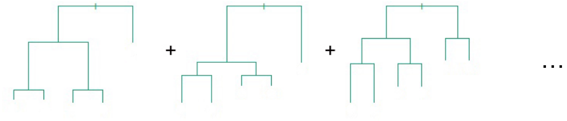
Function estimation/approximation is viewed from the perspective of numerical optimization in function space, rather than parameter space. A connection is made between stagewise additive expansions and steepest-descent minimization. A general gradient descent "boosting" paradigm is developed for additive expansions based on any fitting criterion. Specific algorithms are presented for least-squares, least absolute deviation, and Huber-M loss functions for regression, and multiclass logistic likelihood for classification. Special enhancements are derived for the particular case where the individual additive components are regression trees, and tools for interpreting such "TreeBoost" models are presented. Gradient boosting of regression trees produces competitive, highly robust, interpretable procedures for both regression and classification, especially appropriate for mining less than clean data. Connections between this approach and the boosting methods of Freund and Shapire and Friedman, Hastie and Tibshirani are discussed.

Given responses $y_i \in \mathbb{R}$ and features $x_i \in \mathbb{R}^p$, $i = 1, \ldots, n$

Want to construct a flexible (nonlinear) model for response based on features. Weighted sum of trees:

$$u_i = \sum_{j=1}^{m} \beta_j \cdot T_j(x_i), \quad i = 1, \ldots, n$$

Each tree T_j inputs x_i, outputs predicted response. Typically trees are pretty short



Pick a loss function L to reflect setting. For continuous responses, e.g., could take $L(y_i, u_i) = (y_i - u_i)^2$

Want to solve

$$\min_{\beta} \sum_{i=1}^{n} L\Big(y_i, \sum_{j=1}^{M} \beta_j \cdot T_j(x_i)\Big)$$

Indexes all trees of a fixed size (e.g., depth $= 5$), so M is huge. Space is simply too big to optimize

Gradient boosting: basically a version of gradient descent that is forced to work with trees

First think of optimization as $\min_u \; f(u)$, over predicted values u, subject to u coming from trees

Start with initial model, a single tree $u^{(0)} = T_0$. Repeat:

- Compute negative gradient d at latest prediction $u^{(k-1)}$,

$$d_i = -\left[\frac{\partial L(y_i, u_i)}{\partial u_i}\right]\bigg|_{u_i = u_i^{(k-1)}}, \quad i = 1, \ldots, n$$

- Find a tree T_k that is close to a, i.e., according to

$$\min_{\text{trees } T} \sum_{i=1}^{n} (d_i - T(x_i))^2$$

  Not hard to (approximately) solve for a single tree

- Compute step size α_k, and update our prediction:

$$u^{(k)} = u^{(k-1)} + \alpha_k \cdot T_k$$

  Note: predictions are weighted sums of trees, as desired

Part II: First-order methods

B. Subgradients

Subgradients

Recall that for convex and differentiable f,

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) \quad \text{for all } x, y$$

That is, linear approximation always underestimates f

A subgradient of a convex function f at x is any $g \in \mathbb{R}^n$ such that

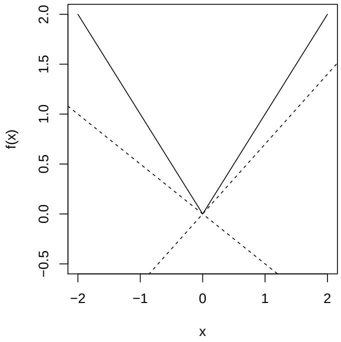
$$f(y) \geq f(x) + g^T (y - x) \quad \text{for all } y$$

- Always exists[2]
- If f differentiable at x, then $g = \nabla f(x)$ uniquely
- Same definition works for nonconvex f (however, subgradients need not exist)

[2]On the relative interior of $\mathrm{dom}(f)$

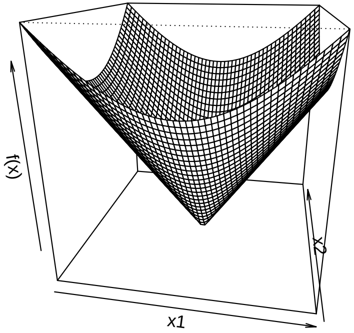
Examples of subgradients

Consider $f : \mathbb{R} \to \mathbb{R}$, $f(x) = |x|$



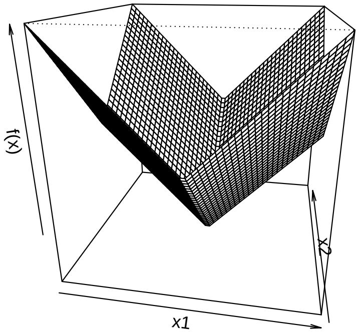
- For $x \neq 0$, unique subgradient $g = \operatorname{sign}(x)$
- For $x = 0$, subgradient g is any element of $[-1, 1]$

Consider $f : \mathbb{R}^n \to \mathbb{R}$, $f(x) = \|x\|_2$



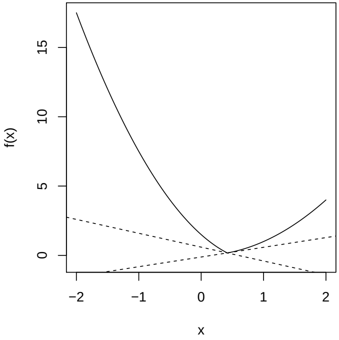
- For $x \neq 0$, unique subgradient $g = x/\|x\|_2$
- For $x = 0$, subgradient g is any element of $\{z : \|z\|_2 \leq 1\}$

Consider $f : \mathbb{R}^n \to \mathbb{R}$, $f(x) = \|x\|_1$



- For $x_i \neq 0$, unique ith component $g_i = \text{sign}(x_i)$
- For $x_i = 0$, ith component g_i is any element of $[-1, 1]$

Consider $f(x) = \max\{f_1(x), f_2(x)\}$, for $f_1, f_2 : \mathbb{R}^n \to \mathbb{R}$ convex, differentiable



- For $f_1(x) > f_2(x)$, unique subgradient $g = \nabla f_1(x)$
- For $f_2(x) > f_1(x)$, unique subgradient $g = \nabla f_2(x)$
- For $f_1(x) = f_2(x)$, subgradient g is any point on line segment between $\nabla f_1(x)$ and $\nabla f_2(x)$

Subdifferential

Set of all subgradients of convex f is called the subdifferential:

$$\partial f(x) = \{g \in \mathbb{R}^n : g \text{ is a subgradient of } f \text{ at } x\}$$

- Nonempty (only for convex f)
- $\partial f(x)$ is closed and convex (even for nonconvex f)
- If f is differentiable at x, then $\partial f(x) = \{\nabla f(x)\}$
- If $\partial f(x) = \{g\}$, then f is differentiable at x and $\nabla f(x) = g$

Connection to convex geometry

Convex set $C \subseteq \mathbb{R}^n$, consider indicator function $I_C : \mathbb{R}^n \to \mathbb{R}$,

$$I_C(x) = I\{x \in C\} = \begin{cases} 0 & \text{if } x \in C \\ \infty & \text{if } x \notin C \end{cases}$$

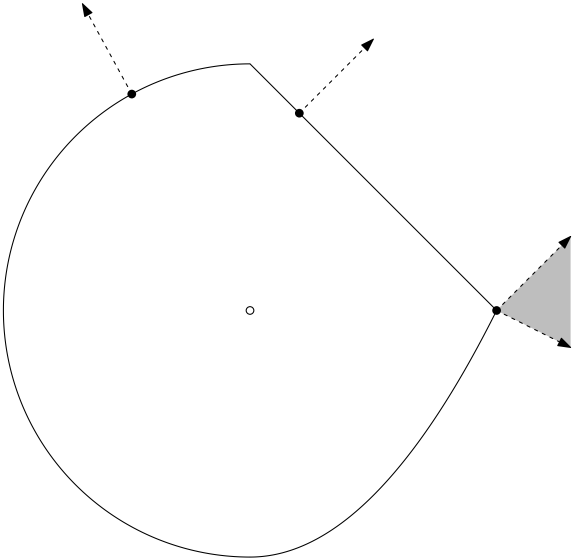
For $x \in C$, $\partial I_C(x) = \mathcal{N}_C(x)$, the normal cone of C at x is, recall

$$\mathcal{N}_C(x) = \{g \in \mathbb{R}^n : g^T x \geq g^T y \text{ for any } y \in C\}$$

Why? By definition of subgradient g,

$$I_C(y) \geq I_C(x) + g^T(y - x) \quad \text{for all } y$$

- For $y \notin C$, $I_C(y) = \infty$
- For $y \in C$, this means $0 \geq g^T(y - x)$



Subgradient calculus

Basic rules for convex functions:

- Scaling: $\partial(af) = a \cdot \partial f$ provided $a > 0$
- Addition: $\partial(f_1 + f_2) = \partial f_1 + \partial f_2$
- Affine composition: if $g(x) = f(Ax + b)$, then

$$\partial g(x) = A^T \partial f(Ax + b)$$

- Finite pointwise maximum: if $f(x) = \max_{i=1,\ldots,m} f_i(x)$, then

$$\partial f(x) = \mathrm{conv}\Bigg(\bigcup_{i: f_i(x) = f(x)} \partial f_i(x) \Bigg)$$

  convex hull of union of subdifferentials of active functions at x

- General pointwise maximum: if $f(x) = \max_{s \in S} f_s(x)$, then

$$\partial f(x) \supseteq \mathrm{cl}\left\{ \mathrm{conv}\left(\bigcup_{s: f_s(x) = f(x)} \partial f_s(x) \right) \right\}$$

  Under some regularity conditions (on S, f_s), we get equality

- Norms: important special case, $f(x) = \|x\|_p$. Let q be such that $1/p + 1/q = 1$, then

$$\|x\|_p = \max_{\|z\|_q \leq 1} \; z^T x$$

  And

$$\partial f(x) = \underset{\|z\|_q \leq 1}{\mathrm{argmax}} \; z^T x$$

Optimality condition

For any f (convex or not),

$$f(x^\star) = \min_x \; f(x) \quad \Longleftrightarrow \quad 0 \in \partial f(x^\star)$$

That is, $x^\star$ is a minimizer if and only if 0 is a subgradient of f at $x^\star$. This is called the subgradient optimality condition

Why? Easy: $g = 0$ being a subgradient means that for all y

$$f(y) \geq f(x^\star) + 0^T(y - x^\star) = f(x^\star)$$

Note the implication for a convex and differentiable function f, with $\partial f(x) = \{\nabla f(x)\}$

Derivation of first-order optimality

Example of the power of subgradients: we can use what we have learned so far to derive the first-order optimality condition. Recall

$$\min_x \ f(x) \ \text{ subject to } \ x \in C$$

is solved at x, for f convex and differentiable, if and only if

$$\nabla f(x)^T(y-x) \geq 0 \ \text{ for all } \ y \in C$$

Intuitively: says that gradient increases as we move away from x. How to prove it? First recast problem as

$$\min_x \ f(x) + I_C(x)$$

Now apply subgradient optimality: $0 \in \partial(f(x) + I_C(x))$

Observe

$$\begin{aligned} 0 \in \partial\big(f(x) + I_C(x)\big) \\ &\iff 0 \in \{\nabla f(x)\} + \mathcal{N}_C(x) \\ &\iff -\nabla f(x) \in \mathcal{N}_C(x) \\ &\iff -\nabla f(x)^T x \geq -\nabla f(x)^T y \text{ for all } y \in C \\ &\iff \nabla f(x)^T (y-x) \geq 0 \text{ for all } y \in C \end{aligned}$$

as desired

Note: the condition $0 \in \partial f(x) + \mathcal{N}_C(x)$ is a fully general condition for optimality in convex problems. But it's not always easy to work with (KKT conditions, later, are easier)

Example: lasso optimality conditions

Given $y \in \mathbb{R}^n$, $X \in \mathbb{R}^{n \times p}$, lasso problem can be parametrized as

$$\min_\beta \ \frac{1}{2}\|y - X\beta\|_2^2 + \lambda\|\beta\|_1$$

where $\lambda \geq 0$. Subgradient optimality:

$$\begin{aligned} 0 \in \partial\Big(\frac{1}{2}\|y - X\beta\|_2^2 + \lambda\|\beta\|_1\Big) & \\ \iff & \ 0 \in -X^T(y - X\beta) + \lambda\partial\|\beta\|_1 \\ \iff & \ X^T(y - X\beta) = \lambda v \end{aligned}$$

for some $v \in \partial\|\beta\|_1$, i.e.,

$$v_i \in \begin{cases} \{1\} & \text{if } \beta_i > 0 \\ \{-1\} & \text{if } \beta_i < 0 \\ [-1, 1] & \text{if } \beta_i = 0 \end{cases}, \quad i = 1, \ldots, p$$

Write $X_1, \ldots, X_p$ for columns of X. Then our condition reads:

$$\begin{cases} X_i^T(y - X\beta) = \lambda \cdot \operatorname{sign}(\beta_i) & \text{if } \beta_i \neq 0 \\ |X_i^T(y - X\beta)| \leq \lambda & \text{if } \beta_i = 0 \end{cases}$$

Note: subgradient optimality conditions don't lead to closed-form expression for a lasso solution ... however they do provide a way to check lasso optimality

They are also helpful in understanding the lasso estimator; e.g., if $|X_i^T(y - X\beta)| < \lambda$, then $\beta_i = 0$ (used by screening rules, later?)

Example: soft-thresholding

Simplfied lasso problem with $X = I$:

$$\min_{\beta} \ \frac{1}{2}\|y - \beta\|_2^2 + \lambda\|\beta\|_1$$

This we can solve directly using subgradient optimality. Solution is $\beta = S_\lambda(y)$, where S_λ is the soft-thresholding operator:

$$[S_\lambda(y)]_i = \begin{cases} y_i - \lambda & \text{if } y_i > \lambda \\ 0 & \text{if } -\lambda \le y_i \le \lambda \\ y_i + \lambda & \text{if } y_i < -\lambda \end{cases}, \quad i = 1, \ldots, n$$

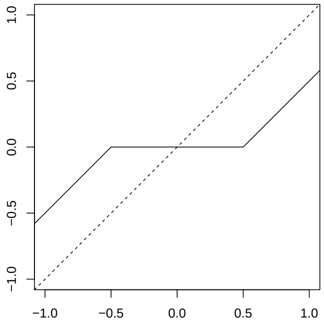
Check: from last slide, subgradient optimality conditions are

$$\begin{cases} y_i - \beta_i = \lambda \cdot \mathrm{sign}(\beta_i) & \text{if } \beta_i \ne 0 \\ |y_i - \beta_i| \le \lambda & \text{if } \beta_i = 0 \end{cases}$$

Now plug in $\beta = S_\lambda(y)$ and check these are satisfied:

- When $y_i > \lambda$, $\beta_i = y_i - \lambda > 0$, so $y_i - \beta_i = \lambda = \lambda \cdot 1$
- When $y_i < -\lambda$, argument is similar
- When $|y_i| \le \lambda$, $\beta_i = 0$, and $|y_i - \beta_i| = |y_i| \le \lambda$

Soft-thresholding in one variable:



Subgradient method

Now consider f convex, having $\mathrm{dom}(f) = \mathbb{R}^n$, but not necessarily differentiable

Subgradient method: like gradient descent, but replacing gradients with subgradients. Initialize $x^{(0)}$, repeat:

$$x^{(k)} = x^{(k-1)} - t_k \cdot g^{(k-1)}, \quad k = 1, 2, 3, \ldots$$

where $g^{(k-1)} \in \partial f(x^{(k-1)})$, any subgradient of f at $x^{(k-1)}$

Subgradient method is not necessarily a descent method, so we keep track of best iterate $x^{(k)}_{\text{best}}$ among $x^{(0)}, \ldots, x^{(k)}$ so far, i.e.,

$$f(x^{(k)}_{\text{best}}) = \min_{i=0,\ldots,k} f(x^{(i)})$$

Step size choices

- Fixed step sizes: $t_k = t$ all $k = 1, 2, 3, \ldots$
- Diminishing step sizes: choose to meet conditions

$$\sum_{k=1}^{\infty} t_k^2 < \infty, \quad \sum_{k=1}^{\infty} t_k = \infty,$$

  i.e., square summable but not summable. Important here that step sizes go to zero, but not too fast

There are several other options too, but key difference to gradient descent: step sizes are pre-specified, not adaptively computed

Convergence analysis

Assume that f convex, $\mathrm{dom}(f) = \mathbb{R}^n$, and also that f is Lipschitz continuous with constant $G > 0$, i.e.,

$$|f(x) - f(y)| \le G\|x - y\|_2 \quad \text{for all } x, y$$

Theorem: For a fixed step size t, subgradient method satisfies

$$\lim_{k\to\infty} f(x_{\text{best}}^{(k)}) \le f^\star + G^2 t/2.$$

For diminishing step sizes, subgradient method satisfies

$$\lim_{k\to\infty} f(x_{\text{best}}^{(k)}) = f^\star$$

(Lipschitz condition can be removed with diminishing step sizes)

Convergence rate

The proof of these results tells us that after k steps, we have

$$f(x^{(k)}_{\text{best}}) - f(x^\star) \le \frac{R^2 + G^2 \sum_{i=1}^k t_i^2}{2\sum_{i=1}^k t_i}$$

With fixed step size t, this gives

$$f(x^{(k)}_{\text{best}}) - f^\star \le \frac{R^2}{2kt} + \frac{G^2 t}{2}$$

For this to be $\le \epsilon$, let's make each term $\le \epsilon/2$. So we can choose $t = \epsilon/G^2$, and $k = R^2/t \cdot 1/\epsilon = R^2G^2/\epsilon^2$

That is, subgradient method has convergence rate $O(1/\epsilon^2)$ ... this is slower than $O(1/\epsilon)$ rate of gradient descent

Example: regularized logistic regression

Given $(x_i, y_i) \in \mathbb{R}^p \times \{0, 1\}$ for $i = 1, \ldots, n$, the logistic regression loss is

$$f(\beta) = \sum_{i=1}^{n} \Big(-y_i x_i^T \beta + \log(1 + \exp(x_i^T \beta)) \Big)$$

This is a smooth and convex, with

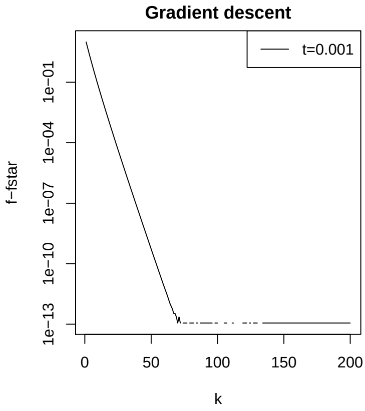
$$\nabla f(\beta) = \sum_{i=1}^{n} \big(y_i - p_i(\beta) \big) x_i$$

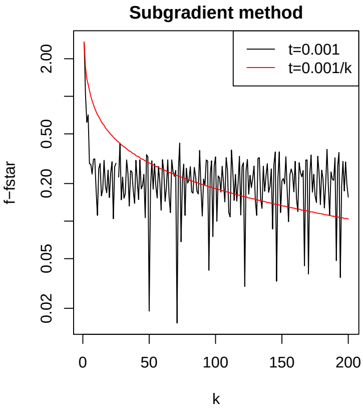
where $p_i(\beta) = \exp(x_i^T \beta)/(1 + \exp(x_i^T \beta))$, $i = 1, \ldots, n$. Consider the regularized problem:

$$\min_{\beta} \; f(\beta) + \lambda \cdot P(\beta)$$

where $P(\beta) = \|\beta\|_2^2$, ridge penalty; or $P(\beta) = \|\beta\|_1$, lasso penalty

Ridge: use gradients; lasso: use subgradients. Example here has $n = 1000$, $p = 20$:





Step sizes hand-tuned to be favorable for each method (of course comparison is imperfect, but it reveals the convergence behaviors)

Can we do better?

Nonsmooth first-order methods: iterative methods updating $x^{(k)}$ in

$$x^{(0)} + \mathrm{span}\{g^{(0)}, g^{(1)}, \ldots, g^{(k-1)}\}$$

where subgradients $g^{(0)}, g^{(1)}, \ldots, g^{(k-1)}$ come from weak oracle

Theorem (Nesterov): For any $k \leq n-1$ and starting point $x^{(0)}$, there is a function in the problem class such that any nonsmooth first-order method satisfies

$$f(x^{(k)}) - f^\star \geq \frac{RG}{2(1+\sqrt{k+1})}$$

In words, we cannot do better than the $O(1/\epsilon^2)$ rate of subgradient method (unless we go beyond nonsmooth first-order methods) ...
so we will focus on $f = g + h$, where g is smooth and h is “simple”

Part II: First-order methods

C. Proximal methods

Composite functions

Suppose

$$f(x) = g(x) + h(x)$$

- g is convex, differentiable, $\mathrm{dom}(g) = \mathbb{R}^n$
- h is convex, not necessarily differentiable

If f were differentiable, then gradient descent update would be:

$$x^+ = x - t \cdot \nabla f(x)$$

Recall motivation: minimize quadratic approximation to f around x, replace $\nabla^2 f(x)$ by $\frac{1}{t} I$,

$$x^+ = \underset{z}{\mathrm{argmin}} \; \underbrace{f(x) + \nabla f(x)^T (z - x) + \frac{1}{2t}\|z - x\|_2^2}_{\widetilde{f}_t(z)}$$

In our case f is not differentiable, but $f = g + h$, g differentiable. Why don't we make quadratic approximation to g, leave h alone?

That is, update

$$\begin{aligned} x^+ &= \underset{z}{\operatorname{argmin}}\ \widetilde{g}_t(z) + h(z) \\ &= \underset{z}{\operatorname{argmin}}\ g(x) + \nabla g(x)^T(z - x) + \frac{1}{2t}\|z - x\|_2^2 + h(z) \\ &= \underbrace{\underset{z}{\operatorname{argmin}}\ \frac{1}{2t}\big\|z - \big(x - t\nabla g(x)\big)\big\|_2^2 + h(z)}_{\operatorname{prox}_t(x)} \end{aligned}$$

We call $\operatorname{prox}_t(\cdot)$ the proximal mapping

Proximal gradient descent

Proximal gradient descent: choose initialize $x^{(0)}$, repeat:

$$x^{(k)} = \mathrm{prox}_{t_k}\big(x^{(k-1)} - t_k \nabla g(x^{(k-1)})\big), \quad k = 1, 2, 3, \ldots$$

To make this update step look familiar, can rewrite it as

$$x^{(k)} = x^{(k-1)} - t_k \cdot G_{t_k}(x^{(k-1)})$$

where $G_t(x) = \frac{x - \mathrm{prox}_t(x - t\nabla g(x))}{t}$ is called the generalized gradient

Notes:

- Mapping $\mathrm{prox}_t(\cdot)$ doesn't depend on g at all, only on h
- Smooth part g can be complicated, we only need to compute its gradients

Example: ISTA

Given $y \in \mathbb{R}^n$, $X \in \mathbb{R}^{n\times p}$, recall lasso criterion:

$$f(\beta) = \underbrace{\frac{1}{2}\|y - X\beta\|_2^2}_{g(\beta)} + \underbrace{\lambda\|\beta\|_1}_{h(\beta)}$$

Prox mapping is now

$$\begin{aligned} \mathrm{prox}_t(\beta) &= \underset{z}{\mathrm{argmin}}\ \frac{1}{2t}\|\beta - z\|_2^2 + \lambda\|z\|_1 \\ &= S_{\lambda t}(\beta) \end{aligned}$$

where $S_\lambda(\beta)$ is the soft-thresholding operator,

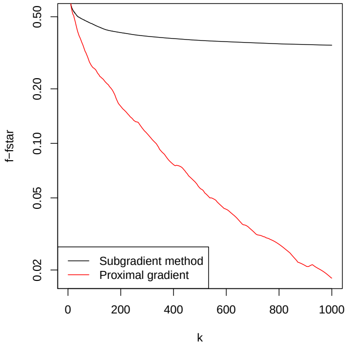
$$[S_\lambda(\beta)]_i = \begin{cases} \beta_i - \lambda & \text{if } \beta_i > \lambda \\ 0 & \text{if } -\lambda \le \beta_i \le \lambda\,, \\ \beta_i + \lambda & \text{if } \beta_i < -\lambda \end{cases} \quad i = 1, \dots, n$$

Recall $\nabla g(\beta) = -X^T(y - X\beta)$, hence proximal gradient update is:

$$\beta^+ = S_{\lambda t}\big(\beta + tX^T(y - X\beta)\big)$$

Often called the iterative soft-thresholding algorithm (ISTA).[3] Very simple algorithm

Example of proximal gradient (ISTA) vs. subgradient method convergence rates



[3]Beck and Teboulle (2008), "A fast iterative shrinkage-thresholding algorithm for linear inverse problems"

Backtracking line search

Backtracking for prox gradient descent works similar as before (in gradient descent), but operates on g and not f

Choose parameter $0 < \beta < 1$. At each iteration, start at $t = t_{\text{init}}$, and while

$$g\big(x - tG_t(x)\big) > g(x) - t\nabla g(x)^T G_t(x) + \frac{t}{2}\|G_t(x)\|_2^2$$

shrink $t = \beta t$, for some $0 < \beta < 1$. Else perform proximal gradient update

(Alternative formulations exist that require less computation, i.e., fewer calls to prox)

Convergence analysis

For criterion $f(x) = g(x) + h(x)$, we assume:

- g is convex, differentiable, $\mathrm{dom}(g) = \mathbb{R}^n$, and ∇g is Lipschitz continuous with constant $L > 0$
- h is convex, $\mathrm{prox}_t(x) = \mathrm{argmin}_z\{\|x - z\|_2^2/(2t) + h(z)\}$ can be evaluated

Theorem: Proximal gradient descent with fixed step size $t \leq 1/L$ satisfies

$$f(x^{(k)}) - f^\star \leq \frac{\|x^{(0)} - x^\star\|_2^2}{2tk}$$

and same result holds for backtracking, with t replaced by β/L

Proximal gradient descent has convergence rate $O(1/k)$ or $O(1/\epsilon)$. Same as gradient descent! (But remember, prox cost matters ...)

Example: matrix completion

Given a matrix $Y \in \mathbb{R}^{m \times n}$, and only observe entries Y_{ij}, $(i,j) \in \Omega$. Suppose we want to fill in missing entries (e.g., for a recommender system), so we solve a matrix completion problem:

$$\min_B \ \frac{1}{2} \sum_{(i,j)\in\Omega} (Y_{ij} - B_{ij})^2 + \lambda \|B\|_{\mathrm{tr}}$$

Here $\|B\|_{\mathrm{tr}}$ is the trace (or nuclear) norm of B,

$$\|B\|_{\mathrm{tr}} = \sum_{i=1}^{r} \sigma_i(B)$$

where $r = \mathrm{rank}(B)$ and $\sigma_1(X) \geq \ldots \geq \sigma_r(X) \geq 0$ are the singular values

Define P_Ω, projection operator onto observed set:

$$[P_\Omega(B)]_{ij} = \begin{cases} B_{ij} & (i,j) \in \Omega \\ 0 & (i,j) \notin \Omega \end{cases}$$

Then the criterion is

$$f(B) = \underbrace{\frac{1}{2}\|P_\Omega(Y) - P_\Omega(B)\|_F^2}_{g(B)} + \underbrace{\lambda\|B\|_{\mathrm{tr}}}_{h(B)}$$

Two ingredients needed for proximal gradient descent:

- Gradient calculation: $\nabla g(B) = -(P_\Omega(Y) - P_\Omega(B))$
- Prox function:

$$\mathrm{prox}_t(B) = \underset{Z}{\mathrm{argmin}} \; \frac{1}{2t}\|B - Z\|_F^2 + \lambda\|Z\|_{\mathrm{tr}}$$

Fact: $\mathrm{prox}_t(B) = S_{\lambda t}(B)$, matrix soft-thresholding at the level λ. Here $S_\lambda(B)$ is defined by

$$S_\lambda(B) = U\Sigma_\lambda V^T$$

where $B = U\Sigma V^T$ is an SVD, and $(\Sigma_\lambda)_{ii} = \max\{\Sigma_{ii} - \lambda, 0\}$

Hence proximal gradient update step is:

$$B^+ = S_{\lambda t}\Big(B + t\big(P_\Omega(Y) - P_\Omega(B)\big)\Big)$$

Note that $\nabla g(B)$ is Lipschitz with $L = 1$, so we can take $t = 1$:

$$B^+ = S_\lambda\big(P_\Omega(Y) + P_\Omega^\perp(B)\big)$$

where $P_\Omega(B) + P_\Omega^\perp(B) = B$. This is the soft-impute algorithm[4]

[4]Mazumder et al. (2011), "Spectral regularization algorithms for learning large incomplete matrices"

Special cases

Proximal gradient descent also called composite gradient descent, or generalized gradient descent

Why "generalized"? This refers to the several special cases, when minimizing $f = g + h$:

- $h = 0$: gradient descent
- $h = I_C$: projected gradient descent
- $g = 0$: proximal minimization algorithm

Therefore these algorithms all have $O(1/\epsilon)$ convergence rate

Projected gradient descent

Given closed, convex set $C \in \mathbb{R}^n$,

$$\min_{x \in C} \; g(x) \iff \min_{x} \; g(x) + I_C(x)$$

where $I_C(x) = \begin{cases} 0 & x \in C \\ \infty & x \notin C \end{cases}$ is the indicator function of C

Hence

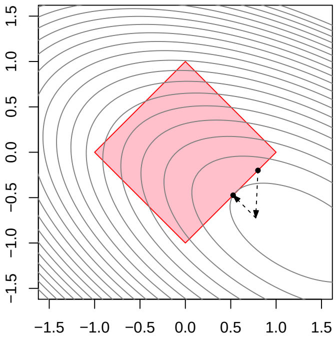
$$\begin{aligned} \mathrm{prox}_t(x) &= \underset{z}{\mathrm{argmin}} \; \frac{1}{2t}\|x - z\|_2^2 + I_C(z) \\ &= \underset{z \in C}{\mathrm{argmin}} \; \|x - z\|_2^2 \end{aligned}$$

That is, $\mathrm{prox}_t(x) = P_C(x)$, projection operator onto C

Therefore proximal gradient update step is:

$$x^+ = P_C\big(x - t\nabla g(x)\big)$$

That is, perform usual gradient update and then project back onto C. Called projected gradient descent



Acceleration

Turns out we can accelerate proximal gradient descent in order to achieve the optimal $O(1/\sqrt{\epsilon})$ convergence rate. Four ideas (three acceleration methods) by Nesterov:

- 1983: original acceleration idea for smooth functions
- 1988: another acceleration idea for smooth functions
- 2005: smoothing techniques for nonsmooth functions, coupled with original acceleration idea
- 2007: acceleration idea for composite functions[5]

We will follow Beck and Teboulle (2008), an extension of Nesterov (1983) to composite functions[6]

[5]Each step uses entire history of previous steps and makes two prox calls

[6]Each step uses information from two last steps and makes one prox call

Accelerated proximal gradient method

As before, consider:

$$\min_x \; g(x) + h(x)$$

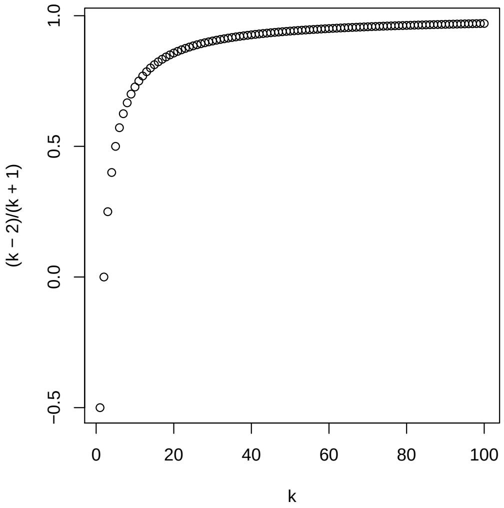
where g convex, differentiable, and h convex. Accelerated proximal gradient method: choose initial point $x^{(0)} = x^{(-1)} \in \mathbb{R}^n$, repeat:

$$v = x^{(k-1)} + \frac{k-2}{k+1}(x^{(k-1)} - x^{(k-2)})$$
$$x^{(k)} = \mathrm{prox}_{t_k}\big(v - t_k \nabla g(v)\big)$$

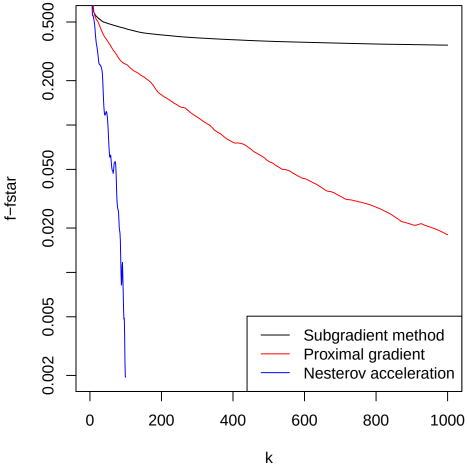
for $k = 1, 2, 3, \ldots$

- First step $k = 1$ is just usual proximal gradient update
- After that, $v = x^{(k-1)} + \frac{k-2}{k+1}(x^{(k-1)} - x^{(k-2)})$ carries some "momentum" from previous iterations
- $h = 0$ gives accelerated gradient method

Momentum weights:



Back to lasso example: acceleration can really help!



Note: accelerated proximal gradient is not a descent method

Convergence analysis

For criterion $f(x) = g(x) + h(x)$, we assume as before:

- g is convex, differentiable, $\mathrm{dom}(g) = \mathbb{R}^n$, and ∇g is Lipschitz continuous with constant $L > 0$
- h is convex, $\mathrm{prox}_t(x) = \mathrm{argmin}_z\{\|x - z\|_2^2/(2t) + h(z)\}$ can be evaluated

Theorem: Accelerated proximal gradient method with fixed step size $t \le 1/L$ satisfies

$$f(x^{(k)}) - f^\star \le \frac{2\|x^{(0)} - x^\star\|_2^2}{t(k+1)^2}$$

and same result holds for backtracking, with t replaced by β/L

Achieves optimal rate $O(1/k^2)$ or $O(1/\sqrt{\epsilon})$ for first-order methods

FISTA

Back to lasso problem:

$$\min_{\beta} \ \frac{1}{2}\|y - X\beta\|_2^2 + \lambda\|\beta\|_1$$

Recall ISTA (Iterative Soft-thresholding Algorithm):

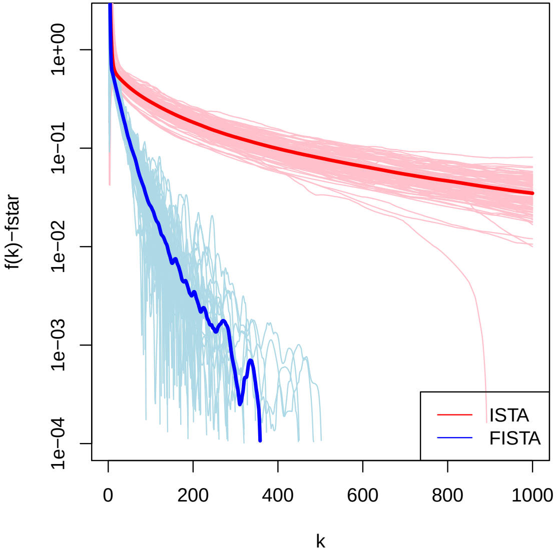
$$\beta^{(k)} = S_{\lambda t_k}(\beta^{(k-1)} + t_k X^T(y - X\beta^{(k-1)})), \quad k = 1, 2, 3, \ldots$$

$S_\lambda(\cdot)$ being vector soft-thresholding. Applying acceleration gives us FISTA (F is for Fast):[7] for $k = 1, 2, 3, \ldots$,

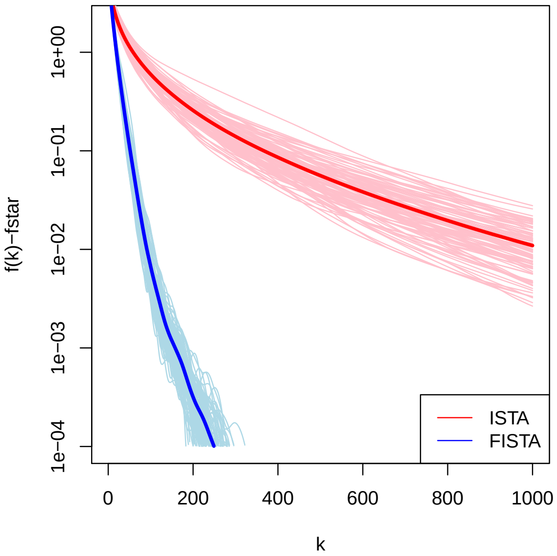
$$\begin{aligned} v &= \beta^{(k-1)} + \frac{k-2}{k+1}(\beta^{(k-1)} - \beta^{(k-2)}) \\ \beta^{(k)} &= S_{\lambda t_k}\big(v + t_k X^T(y - Xv)\big), \end{aligned}$$

[7]Beck and Teboulle (2008) actually call their general acceleration technique (for general g, h) FISTA, which may be somewhat confusing

Lasso regression: 100 instances (with $n = 100$, $p = 500$):



Lasso logistic regression: 100 instances ($n = 100$, $p = 500$):



Part II: First-order methods

D. Stochastic methods

Stochastic gradient descent

Consider minimizing an average of functions

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

As $\nabla \sum_{i=1}^m f_i(x) = \sum_{i=1}^m \nabla f_i(x)$, gradient descent would repeat:

$$x^{(k)} = x^{(k-1)} - t_k \cdot \frac{1}{m}\sum_{i=1}^m \nabla f_i(x^{(k-1)}), \quad k = 1, 2, 3, \ldots$$

In comparison, stochastic gradient descent or SGD (or incremental gradient descent) repeats:

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

where $i_k \in \{1, \ldots, m\}$ is some chosen index at iteration k

Two rules for choosing index i_k at iteration k:

- Randomized rule: choose $i_k \in \{1, \ldots, m\}$ uniformly at random
- Cyclic rule: choose $i_k = 1, 2, \ldots, m, 1, 2, \ldots, m, \ldots$

Randomized rule is more common in practice. For randomized rule, note that

$$\mathbb{E}[\nabla f_{i_k}(x)] = \nabla f(x)$$

so we can view SGD as using an unbiased estimate of the gradient at each step

Main appeal of SGD:

- Iteration cost is independent of m (number of functions)
- Can also be a big savings in terms of memory useage

Example: stochastic logistic regression

Given $(x_i, y_i) \in \mathbb{R}^p \times \{0,1\}$, $i = 1,\ldots,n$, recall logistic regression:

$$\min_\beta \; f(\beta) = \frac{1}{n}\sum_{i=1}^n \underbrace{\Big(- y_i x_i^T\beta + \log(1+\exp(x_i^T\beta))\Big)}_{f_i(\beta)}$$

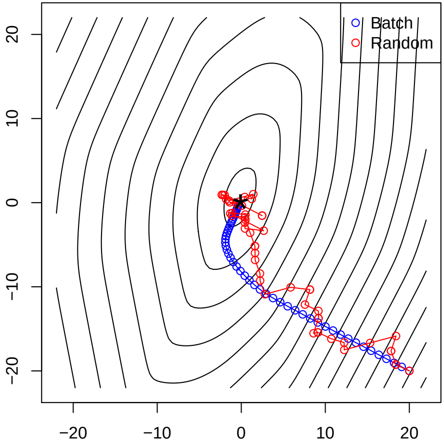
Gradient computation $\nabla f(\beta) = \frac{1}{n}\sum_{i=1}^n \big(y_i - p_i(\beta)\big)x_i$ is doable when n is moderate, but not when n is huge

Full gradient (also called batch) versus stochastic gradient:

- One batch update costs $O(np)$
- One stochastic update costs $O(p)$

Clearly, e.g., 10K stochastic steps are much more affordable

Small example with $n = 10$, $p = 2$ to show the “classic picture” for batch versus stochastic methods:



Blue: batch steps, $O(np)$
Red: stochastic steps, $O(p)$

Rule of thumb for stochastic methods:

- generally thrive far from optimum
- generally struggle close to optimum

Step sizes

Standard in SGD is to use diminishing step sizes, e.g., $t_k = 1/k$

Why not fixed step sizes? Here's some intuition. Suppose we take cyclic rule for simplicity. Set $t_k = t$ for m updates in a row, we get:

$$x^{(k+m)} = x^{(k)} - t \sum_{i=1}^{m} \nabla f_i(x^{(k+i-1)})$$

Meanwhile, full gradient with step size mt would give:

$$x^{(k+1)} = x^{(k)} - t \sum_{i=1}^{m} \nabla f_i(x^{(k)})$$

The difference here: $t \sum_{i=1}^{m} [\nabla f_i(x^{(k+i-1)}) - \nabla f_i(x^{(k)})]$, and if we hold t constant, this difference will not generally be going to zero

Convergence rates

Recall: for convex f, gradient descent with diminishing step sizes satisfies

$$f(x^{(k)}) - f^\star = O(1/\sqrt{k})$$

When f is differentiable with Lipschitz gradient, we get for suitable fixed step sizes

$$f(x^{(k)}) - f^\star = O(1/k)$$

What about SGD? For convex f, SGD with diminishing step sizes satisfies[8]

$$\mathbb{E}[f(x^{(k)})] - f^\star = O(1/\sqrt{k})$$

Unfortunately this does not improve when we further assume f has Lipschitz gradient

[8]For example, Nemirosvki et al. (2009), "Robust stochastic optimization approach to stochastic programming"

Even worse is the following discrepancy!

When f is strongly convex and has a Lipschitz gradient, gradient descent satisfies

$$f(x^{(k)}) - f^\star = O(\gamma^k)$$

where $0 < \gamma < 1$. But under same conditions, SGD gives us[9]

$$\mathbb{E}[f(x^{(k)})] - f^\star = O(1/k)$$

So stochastic methods do not enjoy the linear convergence rate of gradient descent under strong convexity

What can we do to improve SGD?

[9]For example, Nemirosvki et al. (2009), "Robust stochastic optimization approach to stochastic programming"

Mini-batches

Also common is mini-batch stochastic gradient descent, where we choose a random subset $I_k \subseteq \{1, \ldots, m\}$, $|I_k| = b \ll m$, repeat:

$$x^{(k)} = x^{(k-1)} - t_k \cdot \frac{1}{b} \sum_{i \in I_k} \nabla f_i(x^{(k-1)}), \quad k = 1, 2, 3, \ldots$$

Again, we are approximating full gradient by an unbiased estimate:

$$\mathbb{E}\left[\frac{1}{b} \sum_{i \in I_k} \nabla f_i(x)\right] = \nabla f(x)$$

Using mini-batches reduces variance by a factor $1/b$, but is also b times more expensive. Theory justifies this, but only to an extent: under Lipschitz gradient, rate goes from $O(1/\sqrt{k})$ to $O(1/\sqrt{bk})$[10]

[10] For example, Dekel et al. (2012), "Optimal distributed online prediction using mini-batches"

Back to logistic regression, let's now consider a regularized version:

$$\min_{\beta} \; \frac{1}{n} \sum_{i=1}^{n} \Big(-y_i x_i^T \beta + \log(1 + e^{x_i^T \beta}) \Big) + \frac{\lambda}{2} \|\beta\|_2^2$$

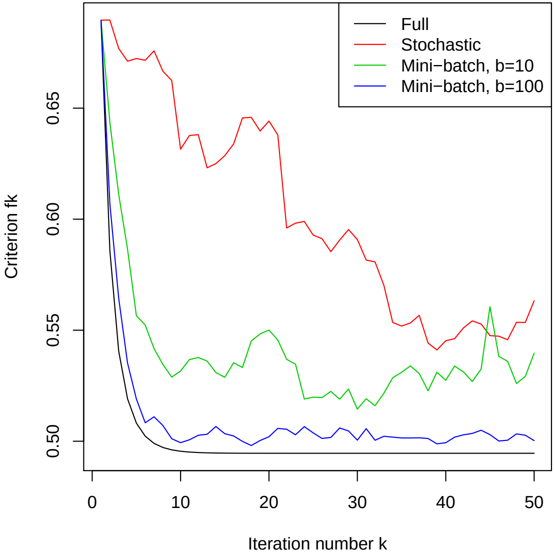
Write the criterion as

$$f(\beta) = \frac{1}{n} \sum_{i=1}^{n} f_i(\beta), \quad f_i(\beta) = -y_i x_i^T \beta + \log(1 + e^{x_i^T \beta}) + \frac{\lambda}{2} \|\beta\|_2^2$$

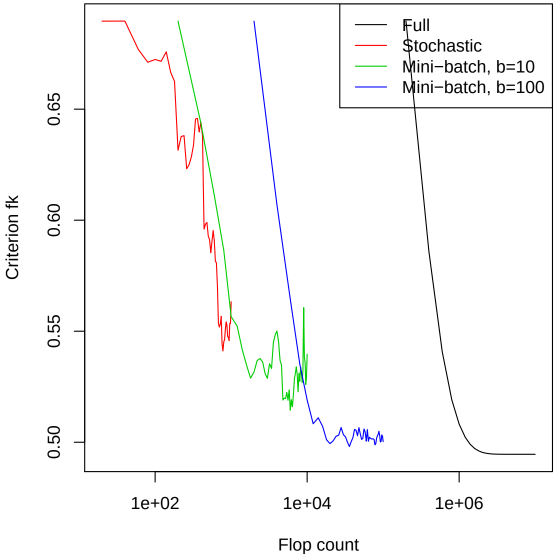
Full gradient computation is $\nabla f(\beta) = \frac{1}{n} \sum_{i=1}^{n} \big(y_i - p_i(\beta)\big) x_i + \lambda \beta$. Comparison between methods:

- One batch update costs $O(np)$
- One mini-batch update costs $O(bp)$
- One stochastic update costs $O(p)$

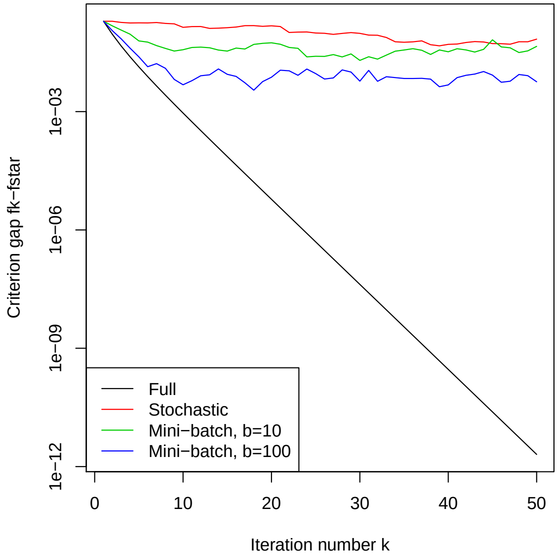
Example with $n = 10,000$, $p = 20$, all methods use fixed step sizes:



What's happening? Now let's parametrize by flops:



Finally, looking at suboptimality gap (on log scale):



End of the story?

Short story:

- SGD can be super effective in terms of iteration cost, memory
- But SGD is slow to converge to high accuracy solutions, can't adapt to strong convexity
- And mini-batches seem to be a wash in terms of flops (though they can still be useful in practice)

Is this the end of the story for SGD?

For a while, the answer was believed to be yes. Slow convergence for strongly convex functions was believed inevitable, as Nemirovski and others established matching lower bounds ... but this was for a more general stochastic problem, where $f(x) = \int F(x, \xi)\, dP(\xi)$

Variance reduction

New wave of "variance reduction" work shows we can modify SGD to converge much faster for finite sums, $f(x) = \frac{1}{m}\sum_{i=1}^m f_i(x)$: see SAG, SDCA, SVRG, S2GD, SAGA, etc. Here we describe SAGA:[11]

- Maintain table, containing gradient g_i of f_i, $i = 1, \ldots, n$
- At steps $k = 1, 2, 3, \ldots$, pick random $i_k \in \{1, \ldots n\}$, then let

$$g_{i_k}^{(k)} = \nabla f_{i_k}(x^{(k-1)}) \quad \text{(most recent gradient of } f_{i_k}\text{)}$$

  Set all other $g_i^{(k)} = g_i^{(k-1)}$, $i \neq i_k$, i.e., these stay the same
- Update

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

[11] Defasio et al. (2014), "SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives". This paper also gives a nice literature review on variance reduction

Notes on SAGA

- SAGA uses gradient estimate: $g_{i_k}^{(k)} - g_{i_k}^{(k-1)} + \frac{1}{n}\sum_{i=1}^n g_i^{(k-1)}$
- Its predecessor, SAG, uses: $\frac{1}{n}g_{i_k}^{(k)} - \frac{1}{n}g_{i_k}^{(k-1)} + \frac{1}{n}\sum_{i=1}^n g_i^{(k-1)}$
- Common footing for both: consider family of estimators

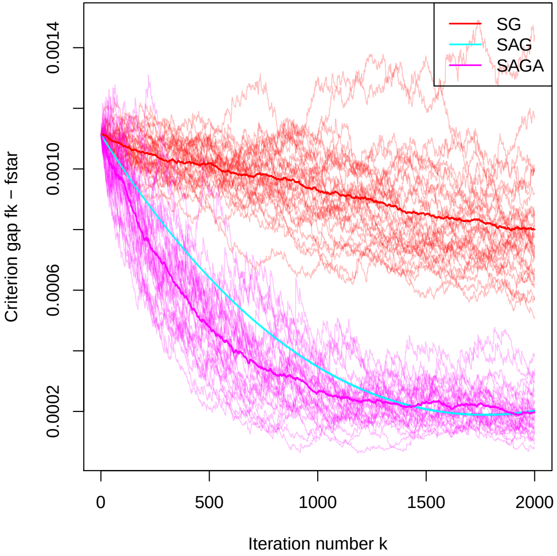
$$\theta_\alpha = \alpha(X - Y) + \mathbb{E}(Y)$$

  for $\mathbb{E}(X)$, where $\alpha \in [0,1]$, and X, Y are correlated. We have

$$\mathbb{E}(\theta_\alpha) = \alpha\mathbb{E}(X) + (1-\alpha)\mathbb{E}(Y)$$
$$\mathrm{Var}(\theta_\alpha) = \alpha^2\big(\mathrm{Var}(X) + \mathrm{Var}(Y) - 2\mathrm{Cov}(X,Y)\big)$$

  SAGA uses $\alpha = 1$: unbiased, SAG uses $\alpha = 1/n$: biased
- Remarkably, both SAG and SAGA restore convergence rates of full gradient! For Lipschitz gradient: $O(1/k)$, and additionally strong convexity: $O(\gamma^k)$

Our logistic regression example with 30 runs of SGD, SAG, SAGA:



SGD in large-scale ML

SGD has really taken off in large-scale machine learning

- In many ML problems we don't care about optimizing to high accuracy, it doesn't pay off in terms of statistical performance
- Thus (in contrast to what classic theory says) fixed step sizes are commonly used in ML applications
- One trick is to experiment with step sizes using small fraction of training before running SGD on full data set ... many other heuristics are common[12]
- SGD in the continuous, nonconvex world is extremely popular but poorly undestood
- Many variants currently being explored: momentum, adaptive step sizes, averaging, early stopping, etc.

[12]For example, Bottou (2012), "Stochastic gradient descent tricks"

References and further reading

Part A:

- S. Boyd and L. Vandenberghe (2004), "Convex optimization", Chapter 9
- Y. Nesterov (1998), "Introductory lectures on convex optimization: a basic course", Chapter 2

Part B:

- Y. Nesterov (1998), "Introductory lectures on convex optimization: a basic course", Chapter 3
- B. Polyak (1987), "Introduction to optimization", Chapter 5
- R. T. Rockafellar (1970), "Convex analysis", Chapters 23–25

Part C:

- A. Beck and M. Teboulle (2008), "A fast iterative shrinkage-thresholding algorithm for linear inverse problems"
- Y. Nesterov (1998), "Introductory lectures on convex optimization: a basic course", Chapter 2

Part D:

- D. Bertsekas (2010), "Incremental gradient, subgradient, and proximal methods for convex optimization: a survey"
- A. Defasio and F. Bach and S. Lacoste-Julien (2014), "SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives"
- A. Nemirovski and A. Juditsky and G. Lan and A. Shapiro (2009), "Robust stochastic optimization approach to stochastic programming"