Chapter 12 Supplementary Material

Last update on 2025-04-27 10:25:48+08:00

Table of contents

Subgradients

Coordinate Descent

Stochastic Gradient Descent

Alternating Direction Method of Multipliers (ADMM)

Subgradients

Coordinate Descent

Stochastic Gradient Descent

Alternating Direction Method of Multipliers (ADMM)

definition

Recall that for convex and differentiable f,

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

This is first-order condition. Linear approximation always underestimates f

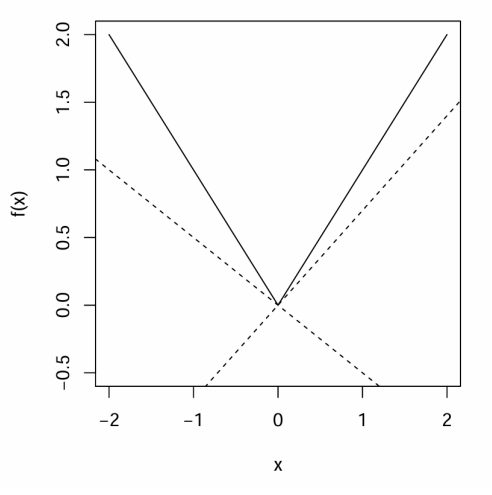
A subgradient of a convex function f at x is any $g \in \mathbb{R}^n$ such that

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

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

Examples of subgradients

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



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

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 $\nabla f(x) = g$

Optimality condition

For any f (convex or not),

$$f(x^*) = \min_x f(x) \iff 0 \in \partial f(x^*)$$

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

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

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

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

Subgradient method

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

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

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

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, thus we keep track of best iterate $x^{(k)}_{\text{best}}$ among $x^{(0)}, ..., x^{(k)}$ so far, i.e.,

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

Subgradients

Coordinate Descent

Stochastic Gradient Descent

Alternating Direction Method of Multipliers (ADMM)

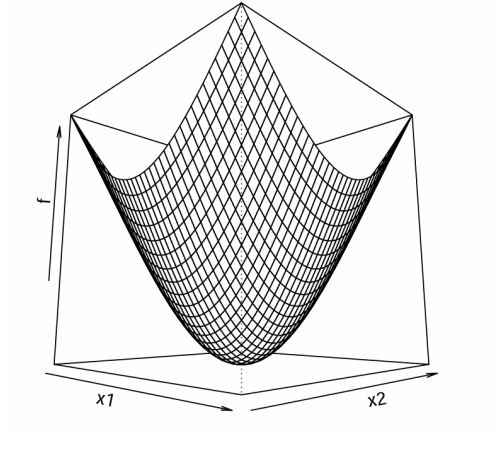
Coordinatewise optimality

We now focus on a very simple technique that can be surprisingly efficient, scalable: coordinate descent, or more appropriately called coordinatewise minimization

Q: Given convex, differentiable $f : \mathbb{R}^n \to \mathbb{R}$, if we are at a point x such that $f(x)$ is minimized along each coordinate axis, then have we found a global minimizer?

That is, does $f(x + \delta e_i) \geq f(x)$ for all δ, $i \implies f(x) = \min_z f(z)$?

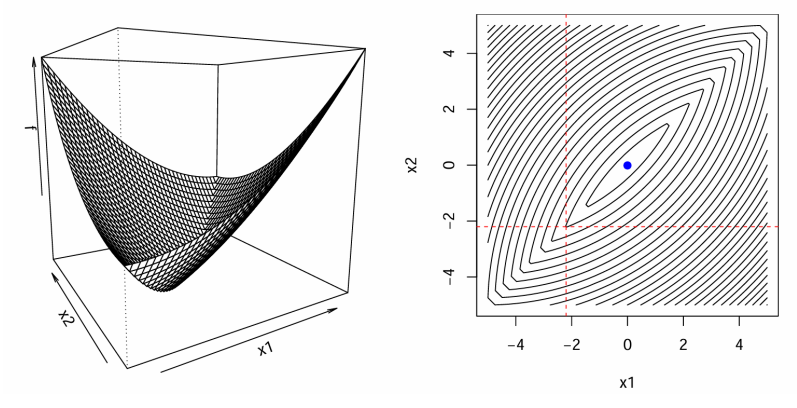
(Here $e_i = (0, ..., 1, ..., 0) \in \mathbb{R}^n$ is the ith standard basis vector)



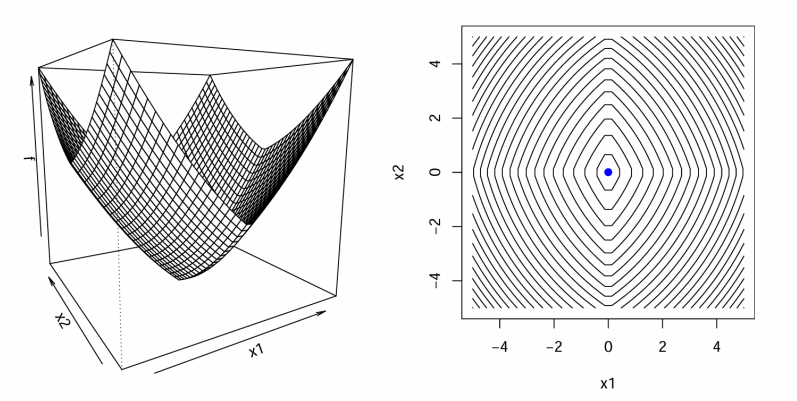
- A: Yes! Proof:

$$0 = \nabla f(x) = \Big(\frac{\partial f}{\partial x_1}(x), ..., \frac{\partial f}{\partial x_n}(x) \Big)$$

- Q: Same question, but now for f convex, and not differentiable?



- A: No! Look at the above counterexample
- Q: Same, now $f(x) = g(x) + \sum_{i=1}^n h_i(x_i)$, with g convex, smooth, and each h_i convex? (Here the nonsmooth part is called separable)



- A: Yes! Proof: using convexity of g and subgradient optimality

$$
\begin{aligned}
f(y) - f(x) &= g(y) - g(x) + \sum_{i=1}^{n} [h_i(y_i) - h_i(x_i)] \\
&\geq \sum_{i=1}^{n} \underbrace{\left[\partial_{x_i} g(x)(y_i - x_i) + h_i(y_i) - h_i(x_i)\right]}_{\geq 0} \geq 0
\end{aligned}
$$

Coordinate descent

This suggests that for the problem

$$\min_x f(x)$$

where $f(x) = g(x) + \sum_{i=1}^n h_i(x_i)$, with g convex and differentiable and each h_i convex, we can use coordinate descent: let $x^{(0)} \in \mathbb{R}^n$, and repeat

$$x_i^{(k)} = \underset{x_i}{\text{argmin}}\, f\big(x_1^{(k)}, ..., x_{i-1}^{(k)}, x_i, x_{i+1}^{(k-1)}, ..., x_n^{(k-1)}\big), \qquad i = 1, ..., n$$

for $k = 1, 2, 3, ...$

Important note: we always use most recent information possible

Tseng (2001) showed that for such f (provided f is continuous on compact set $\{x : f(x) \leq f(x^{(0)})\}$ and f attains its minimum), any limit point of $x^{(k)},\ k = 1, 2, 3, ...$ is a minimizer of f

Notes:

- Order of cycle through coordinates is arbitrary, can use any permutation of $\{1, 2, ..., n\}$
- Can everywhere replace individual coordinates with blocks of coordinates
- "One-at-a-time" update scheme is critical, and "all-at-once" scheme does not necessarily converge
- The analogy for solving linear systems: Gauss-Seidel versus Jacobi method

Example: linear regression

Given $y \in \mathbb{R}^n$, and $X \in \mathbb{R}^{n \times p}$ with columns $X_1, ..., X_p$, consider the linear regression problem:

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

Minimizing over β_i, with all $\beta_j, j \neq i$ fixed:

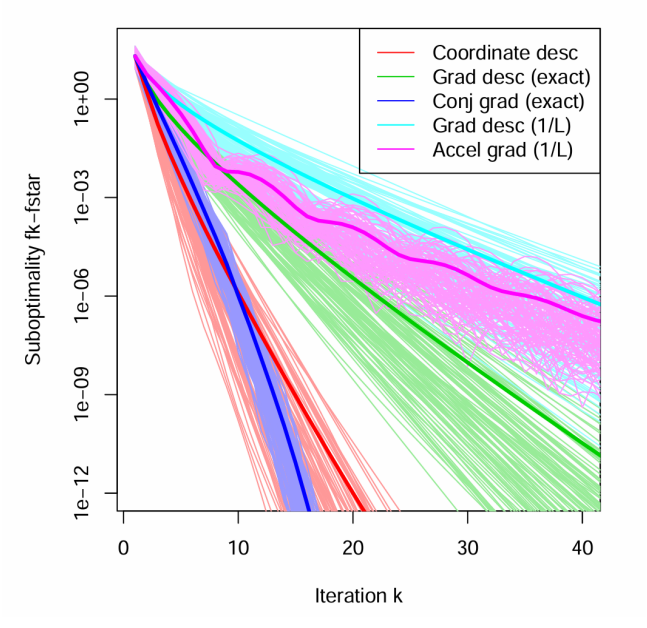
$$0 = \partial_{\beta_i} f(\beta) = X_i^T(X\beta - y) = X_i^T(X_i\beta_i + X_{-i}\beta_{-i} - y)$$

i.e., we take

$$\beta_i = \frac{X_i^T(y - X_{-i}\beta_{-i})}{X_i^T X_i}$$

Coordinate descent repeats this update for $i = 1, 2, ..., p, 1, 2, ...$ Note that this is exactly Gauss-Seidl for the system $X^T X\beta = X^T y$

Coordinate descent vs gradient descent for linear regression: 100 random instances with $n = 100$, $p = 20$



Is it fair to compare 1 cycle of coordinate descent to 1 iteration of gradient descent? Yes, if were clever

- Gradient descent: $\beta \leftarrow \beta + tX^T(y - X\beta)$, costs $O(np)$ flops
- Coordinate descent, one coordinate update:

$$\beta_i \leftarrow \frac{X_i^T(y - X_{-i}\beta_{-i})}{X_i^T X_i} = \frac{X_i^T r}{\|X_i\|_2^2} + \beta_i$$

  where $r = y - X\beta$
- Each coordinate costs $O(n)$ flops: $O(n)$ to update r, $O(n)$ to compute $X_i^T r$
- One cycle of coordinate descent costs $O(np)$ operations, same as gradient descent

Example: lasso regression

Consider the lasso problem:

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

Note that nonsmooth part here is separable: $\|\beta\|_1 = \sum_{i=1}^p |\beta_i|$. Minimizing over β_i, with β_j, $j \neq i$ fixed:

$$0 = X_i^T X_i \beta_i + X_i^T (X_{-i}\beta_{-i} - y) + \lambda s_i$$

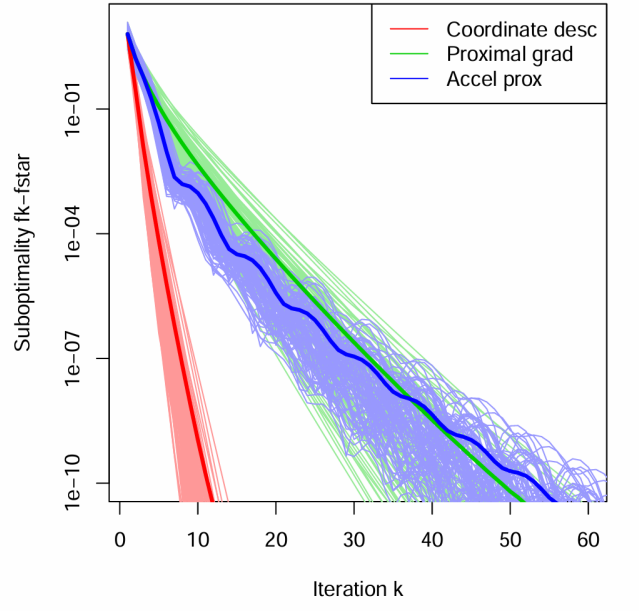
where $s_i \in \partial|\beta_i|$. Solution is simply given by soft-thresholding

$$\beta_i = S_{\lambda/\|X_i\|_2^2}\left(\frac{X_i^T(y - X_{-i}\beta_{-i})}{X_i^T X_i}\right)$$

Repeat this for $i = 1, 2, ..., p, 1, 2, ...$ Here, the soft-thresholding operator S_t is defined as

$$S_t(x_j) = \text{sign}(x_j)\,\mathbf{max}\{|x_j| - t, 0\} = \begin{cases} x_j - t, & x_j > t \\ 0, & -t \leq x_j \leq t, \\ x_j + t & x_j < -t \end{cases} \qquad j = 1, ..., p$$

Coordinate descent vs proximal gradient for lasso regression: 100 random instances with $n = 200,\ p = 50$ (all methods cost $O(np)$ per iter)



Coordinate descent in statistics and ML

History in statistics/ML:

- Idea appeared in Fu (1998), and then again in Daubechies et al. (2004), but was inexplicably ignored
- Later, three papers in 2007, especially Friedman et al. (2007), really sparked interest in statistics and ML communities

Why is it used?

- Very simple and easy to implement
- Careful implementations can achieve state-of-the-art
- Scalable, e.g., don't need to keep full data in memory

Examples: lasso regression, lasso GLMs (under proximal Newton), SVMs, group lasso, graphical lasso (applied to the dual), additive modeling, matrix completion, regression with nonconvex penalties

Coordinate gradient descent

For a smooth function f, the iterations

$$x_i^{(k)} = x_i^{(k-1)} - t_{ki} \cdot \partial_{x_i} f\big(x_1^{(k)}, ..., x_{i-1}^{(k)}, x_i, x_{i+1}^{(k-1)}, ..., x_n^{(k-1)}\big), \qquad i = 1, ..., n$$

for $k = 1, 2, 3, ...$ are called coordinate gradient descent, and when $f = g + h$, with g smooth and $h = \sum_{i=1}^n h_i$, the iterations

$$x_i^{(k)} = \mathrm{prox}_{h_i, t_{ki}}\Big(x_i^{(k-1)} - t_{ki} \cdot \partial_{x_i} g\big(x_1^{(k)}, ..., x_{i-1}^{(k)}, x_i, x_{i+1}^{(k-1)}, ..., x_n^{(k-1)}\big)\Big), \quad i = 1, ..., n$$

for $k = 1, 2, 3, ...$ are called coordinate proximal gradient descent

When g is quadratic, (proximal) coordinate gradient descent is the same as coordinate descent under proper step sizes

Subgradients

Coordinate Descent

Stochastic Gradient Descent

Alternating Direction Method of Multipliers (ADMM)

Stochastic gradient descent

Consider minimizing an average of functions

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

Gradient descent would repeat:

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

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

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

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 $i_k = 1, 2, ..., m, 1, 2, ..., m, ...$

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

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

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

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

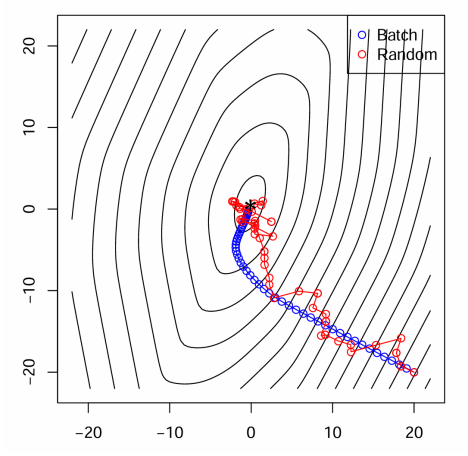
Gradient computation $\nabla f(\beta) = \frac{1}{n} \sum_{i=1}^{n} \big(y_i - p_i(\beta)\big) x_i$ is doable when n is moderate, but not when n is huge

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

Mini-batches

See more about step sizes and convergence rates

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

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

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

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

Using mini-batches reduces variance by a factor $1/b$, but is also b times more expensive. Theory is not convincing: under Lipschitz gradient, rate goes from $O(1/\sqrt{k})$ to $O(1/\sqrt{bk} + 1/k)$[3]

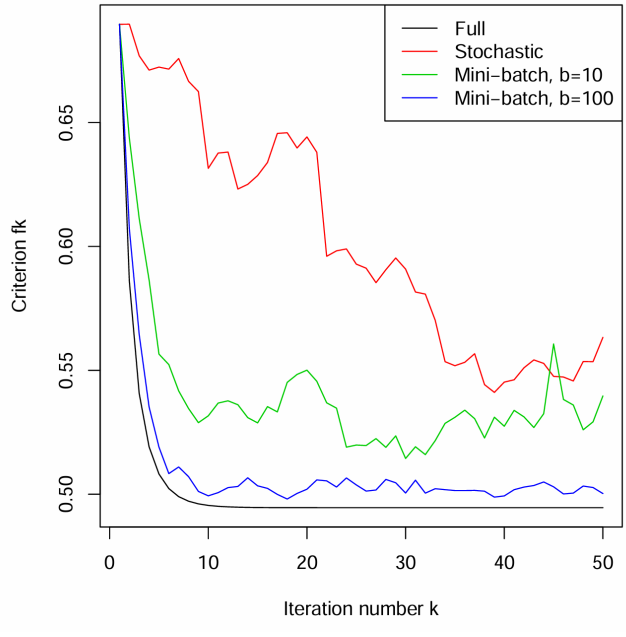
Back to logistic regression, lets now consider a regularized version:

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

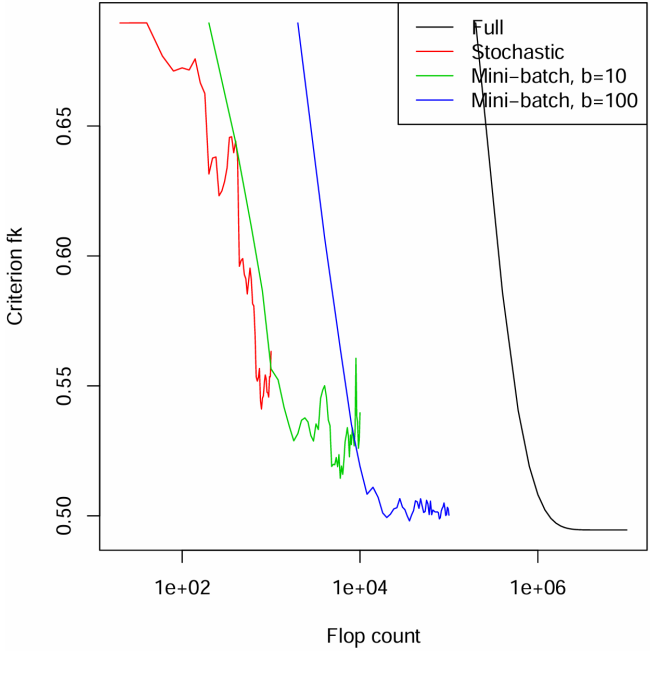
Comparison between methods:

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

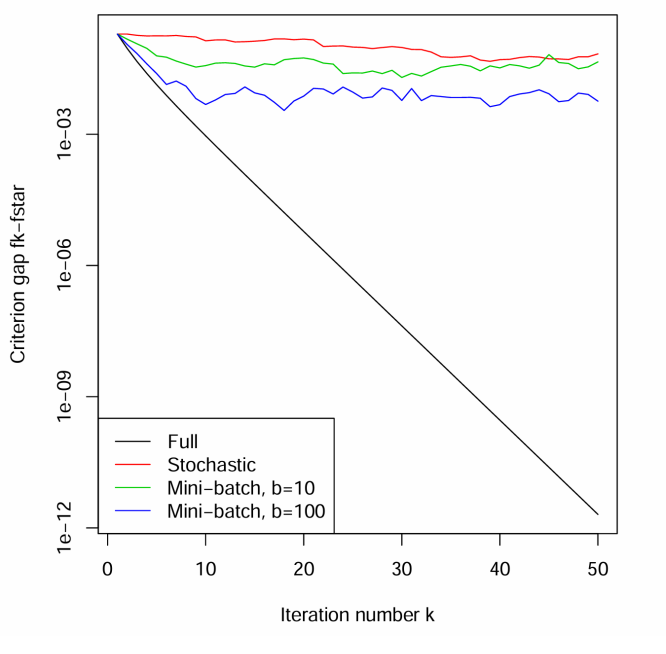
Example with $n = 10000, p = 20$, all methods use fixed step sizes:



Whats happening? Now lets 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, 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)

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
- Momentum/acceleration, averaging, adaptive step sizes are all popular variants in practice
- SGD is especially popular in large-scale, continuous, nonconvex optimization, but it is still not particular well-understood there (a big open issue is that of implicit regularization)

Subgradients

Coordinate Descent

Stochastic Gradient Descent

Alternating Direction Method of Multipliers (ADMM)

Framework of ADMM

- solve the following problem using ADMM

$$\begin{aligned} \underset{x,z}{\text{minimize}} \quad & f(x) + g(z) \\ \text{subject to} \quad & Ax + Bz = c \end{aligned}$$

where $x \in \mathbb{R}^p, z \in \mathbb{R}^q, A \in \mathbb{R}^{m \times p}, B \in \mathbb{R}^{m \times q}, c \in \mathbb{R}^k$ and

$$f : \mathbb{R}^p \to \mathbb{R}, \qquad g : \mathbb{R}^q \to \mathbb{R}$$

- the objective function is separable and constraint only contains equalities
- augmented Lagrangian method modifies the augmented objective function

$$\begin{aligned} \underset{x,z}{\text{minimize}} \quad & Q_\rho(x, z) = f(x) + g(z) + \frac{\rho}{2}\|Ax + Bz - c\|_2^2 \\ \text{subject to} \quad & Ax + Bz = c \end{aligned}$$

- augmented Lagrangian

$$\begin{aligned} L_\rho(x,z,\nu) &= Q_\rho(x,z) + \nu^T(Ax+Bz-c) \\ &= f(x)+g(z)+\frac{\rho}{2}\|Ax+Bz-c\|_2^2+\nu^T(Ax+Bz-c) \end{aligned}$$

- algorithm for the kth iteration:
  1. update x: $x^{(k)} = \mathsf{argmin}_x L_\rho(x, z^{(k-1)}, \nu^{(k-1)})$
  2. update z: $z^{(k)} = \mathsf{argmin}_z L_\rho(x^{(k)}, z, \nu^{(k-1)})$
  3. update ν: $\nu^{(k)} = \nu^{(k-1)} + \rho(Ax^{(k)} + Bz^{(k)} - c)$

conjugate function

- the basic idea is to use the connection between primal problem and dual problem. We solve the dual problem using gradient descent method so that we obtain the optimal of primal problem at the same time
- recall that the conjugate function is

$$f^*(y) = \max_x \ x^T y - f(x) = \max_x \ L(x, y) = L(x^*, y)$$

- even $f(x)$ is not convex, its conjugate $f^*(y)$ is always convex
- according to envelop theorem,

$$\frac{\partial f^*(y)}{\partial y} = \frac{\partial L(x^*, y)}{\partial y} = x^* = \operatorname*{argmin}_x \ x^T y - f(x)$$

Dual gradient ascent

- the primal problem

$$\begin{array}{ll} \underset{x}{\text{minimize}} & f(x) \\ \text{subject to} & Ax = c \end{array}$$

- the Lagrangian is

$$L(x, \nu) = f(x) + \nu^T (Ax - c)$$

- dual function is

$$g(\nu) = \min_x \; L(x, \nu) = -f^*(-A^T \nu) - c^T \nu$$

Dual gradient ascent

- to solve the unconstrained dual problem, we need to compute the gradient of the objective

$$\nabla g(\nu) = -\frac{\partial f^*(-A^T\nu)}{\partial \nu} - c = A\frac{\partial f^*(-A^T\nu)}{\partial(-A^T\nu)} - c = Ax^* - c$$

- the iteration with the learning rate α_k is

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

- the algorithm of dual gradient ascent is:
  1. update x: $x^{(k)} = \text{argmin}_x \; L(x, \nu^{(k-1)})$
  2. update ν: $\nu^{(k)} = \nu^{(k-1)} + \alpha_k[Ax^{(k)} - c]$
- dual decomposition can be used as a trick in dual gradient ascent method when the objective $f(x)$ is separable

Augmented Lagrangian method

- consider the augmented form of the primal problem

$$\begin{array}{ll} \underset{x}{\text{minimize}} & f(x) + (\rho/2)\|Ax - c\|_2^2 \\ \text{subject to} & Ax = c \end{array}$$

- ρ is a penalty parameter and the whole penalty term is to "increase" the convexity of the problem
- the augmented Lagrangian is

$$L_\rho(x, \nu) = f(x) + \frac{\rho}{2}\|Ax - c\|_2^2 + \nu^T(Ax - c)$$

- the algorithm of the augmented problem using dual gradient ascent is:
  1. update x: $x^{(k)} = \text{argmin}_x \; L_\rho(x, \nu^{(k-1)})$
  2. update ν: $\nu^{(k)} = \nu^{(k-1)} + \rho[Ax^{(k)} - c]$
- the learning rate is taken to be ρ which speeds up the convergence

Scaled form

- augmented Lagrangian

$$L_\rho(x, z, \nu) \quad = \quad f(x) + g(z) + \frac{\rho}{2}\|Ax + Bz - c\|_2^2 + \nu^T(Ax + Bz - c)$$

- Scaled form: let $u = \nu/\rho$, then augmented Lagrangian becomes

$$L_\rho(x, z, u) = f(x) + g(z) + \frac{\rho}{2}\|Ax + Bz - c + u\|_2^2 + C,$$

  where C is independent of x, z
- algorithm for the kth iteration of ADMM updates:
  1. update x: $x^{(k)} = \mathsf{argmin}_x L_\rho(x, z^{(k-1)}, u^{(k-1)})$
  2. update z: $z^{(k)} = \mathsf{argmin}_z L_\rho(x^{(k)}, z, u^{(k-1)})$
  3. update ν: $u^{(k)} = u^{(k-1)} + (Ax^{(k)} + Bz^{(k)} - c)$

Example: lasso regression

- Lasso = loss + ℓ^1 penalty
- gradient descent or Newton method are not applicable
- proximal gradient descent, coordinate descent, ADMM can be considered
- reference
  1. https://zhuanlan.zhihu.com/p/448289351
  2. Boyd ADMM paper
  3. CMU convex optimization course

Example: lasso regression

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

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

We can rewrite this as:

$$\begin{aligned} \underset{\beta,\alpha}{\text{minimize}} \quad & \frac{1}{2}\|y - X\beta\|_2^2 + \lambda\|\alpha\|_1 \\ \text{subject to} \quad & \beta = \alpha \end{aligned}$$

augmented Lagrangian

$$L_\rho(\beta, \alpha, w) = \frac{1}{2}\|y - X\beta\|_2^2 + \lambda\|\alpha\|_1 + \frac{\rho}{2}\|\beta - \alpha + w\|_2^2$$

Scaled form ADMM steps:

1. update β:

$$\beta^{(k)} = \mathsf{argmin}_\beta \frac{1}{2}\|y - X\beta\|_2^2 + \frac{\rho}{2}\|\beta - \alpha^{(k-1)} + w^{(k-1)}\|_2^2$$

2. update α:

$$\alpha^{(k)} = \mathsf{argmin}_\alpha \lambda\|\alpha\|_1 + \frac{\rho}{2}\|\beta^{(k)} - \alpha + w^{(k-1)}\|_2^2 = S_{\lambda/\rho}(\beta^{(k)} + w^{(k-1)})$$

3. update w:

$$w^{(k)} = w^{(k-1)} + (\beta^{(k)} - \alpha^{(k)})$$

Scaled form ADMM steps:

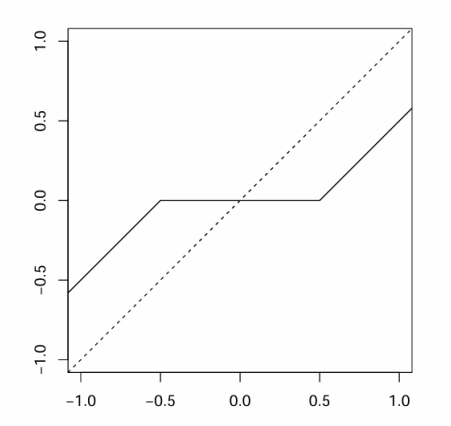
$$\begin{aligned}
\beta^{(k)} &= (X^T X + \rho I)^{-1}\big(X^T y + \rho(\alpha^{(k-1)} - w^{(k-1)})\big) \\
\alpha^{(k)} &= S_{\lambda/\rho}(\beta^{(k)} + w^{(k-1)}) \\
w^{(k)} &= w^{(k-1)} + \beta^{(k)} - \alpha^{(k)}
\end{aligned}$$

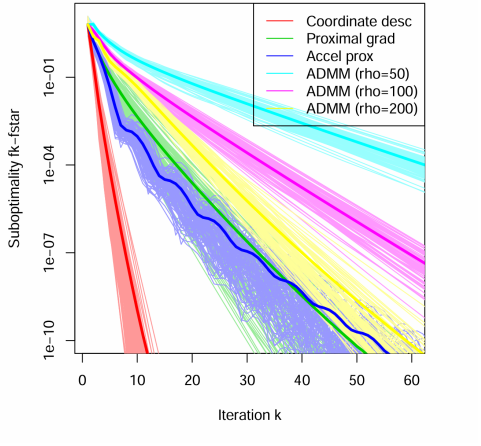
Notes:

- The matrix $X^T X + \rho I$ is always invertible, regardless of X
- If we compute a factorization (say Cholesky) in $O(p^3)$ flops, then each β update takes $O(p^2)$ flops
- The α update applies the soft-thresholding operator S_t, which recall is defined as

$$[S_t(x)]_j = \text{sign}(x_j) \max\{|x_j| - t, 0\} = \begin{cases} x_j - t, & x_j > t \\ 0, & -t \leq x_j \leq t \\ x_j + t & x_j < -t \end{cases} \qquad j = 1, ..., p$$

- ADMM steps are almost like repeated soft-thresholding of ridge regression coefficients





- Soft-thresholding in one variable
- Comparison of various algorithms for lasso regression: 100 random instances with $n = 200, p = 50$