10-725/36-725: Convex Optimization **Fall 2019**

Lecture 8: September 23

Lecturer: Ryan Tibshirani *Scribes: Youngseog Chung, Audrey Huang, Jeffrey Li*

This lecture's notes illustrate some uses of various LaTeX macros. Take a look at this and imitate.

8.1 More About Subgradient Methods

8.1.1 Review of last week

Last time we talked about the subgradient method, which is essentially gradient descent using the subgradient instead of the gradient. It is the most general method we have seen thus far because it can be applied to arbitrary complex criterion whose subgradient we can compute. However, it converges more slowly than gradient descent; even with appropriate step sizes, it converges at an order of magnitude slower than gradient descent at $O(\frac{1}{\epsilon^2})$.

8.1.2 Projected Subgradient Method

Suppose we wanted to solve the following convex minimization problem

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

This problem can be solved by pushing C into the objective function using the indicator function, and then using subgradient descent.

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

If we try to apply the subgradient method to this objective function, we have to take the subgradient of the indicator function. The subgradient of $\partial I_C(x) = N_C(x)$ the normal cone of C at x, which is generally pretty difficult to compute.

Instead, we can use projected subgradient descent. It has the same update as the subgradient method, except we project onto C instead of calculating the subgradient at each step

$$x^{(k)} = P_C(x^{(k-1)} - t_k g^{(k-1)})$$

If this projection can be done, the projected subgradient method achieves the same convergence guarantees as subgradient descent.

Not all sets are easy to project onto. It is possible to have a simple expression for C (e.g. the polyhedron $C = \{x : Ax \leq b\}$) but have projection be difficult or impossible to write down in closed form. However, some projections are easy (e,g, for projecting x onto the L2 norm, just divide x by its L2 norm). Some other easy projections are:

- affine images $\{Ax + b : x \in \mathbb{R}^n\}$
- solution set of linear system $\{x : Ax + b\}$
- nonnegative orthant $\mathbb{R}^n_+ = \{x : x \geq 0\}$
- some norm balls $\{||x||_p \leq 1\}$ for $p = 1, 2, \infty$
- some simple polyhedra and simple cones

8.1.3 Convergence of the Subgradient Method

Though the broad applicability of the subgradient method is desirable, for convex and Lipschitz methods it has a much slower convergence rate of $O(\frac{1}{\epsilon^2})$ than gradient descent, $O(\frac{1}{\epsilon})$. We cannot do better than this rate for the following reason. For nonsmooth first-order methods such as the subgradient method where $x^{(k)}$ is updated as

$$x^{(0)} + \text{span}(g^{(0)} ... g^{(k-1)})$$

where the subgradients come from a weak oracle, the following theorem tell us that the convergence rate is $O(\frac{1}{\epsilon^2})$.

Theorem 8.1 *(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*

$$f(x^{(0)}) - f^* \geq \frac{RG}{2(1 + \sqrt{k+1})}$$

As a result we cannot do better than $O(\frac{1}{\epsilon^2})$. In order to attain faster convergence, we would need to explore second order methods.

8.2 Proximal Gradient Descent

Instead of trying to improve rate of convergence for all types of problems, we instead focus on minimizing composite functions of the form

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

where g is convex and differentiable, and h is simple and convex but nonsmooth. This relaxes the class of functions we can minimize compared to gradient descent, but for many problems we can still achieve the $O(\frac{1}{\epsilon})$ rate of convergence from gradient descent.

Since h is not differentiable, we cannot directly take the gradient of f and apply the gradient descent update:

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

Instead, we can try another method motivated by the same principles as gradient descent. Recall that the gradient descent method is motivated by minimizing a quadratic approximation to f around x, replaying the Hessian with $\frac{1}{t}I$. Instead of trying to minimize the quadratic around all of f, which we can't do because

h is not differentiable, we can minimize just the quadratic approximation to g and leave h alone. Make the update:

$$\begin{aligned} x^+ &= \text{argmin}_z \bar{g}_t(z) + h(z) \\ &= \text{argmin}_z g(x) + \nabla g(x)^T(z-x) + \frac{1}{2t}||z-x||_2^2 + h(z) \\ &= \text{argmin}_z \frac{1}{2t}||z-(x-t\nabla g(x))||_2^2 + h(z) \end{aligned}$$

The first term $||z-(x-t\nabla g(x))||_2^2$ forces us to stay close to the gradient update for g and the second terms forces us to make $h(z)$ small. This is the principle behind proximal mapping.

8.2.1 Proximal Mapping

Proximal mapping is given by

$$\text{prox}_{h,t}(x) = \arg\min_z ||x-z||_2^2 + h(z)$$

In *proximal gradient descent* we choose an initial $x^{(0)}$ and iteratively update

$$x^{(k)} = \text{prox}_{h,t_k}(x^{(k-1)} - t_k \nabla g(x^{(k-1)}))$$

We can rewrite in the same form as a gradient step by defining

$$G_t(x) = \frac{x - \text{prox}_{h,t}(x - t\nabla g(x))}{t}$$

then rewriting the update as:

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

The advantage of the proximal mapping is that $\text{prox}_{h,t}(x)$ has a closed-form solution for many important functions h, which may not be differentiable but are simple. Making the proximal mapping only depends on g, and because g is smooth we can compute its gradients even though it may be very complicated. The computational cost of making the mapping, however, depends on the function and can be either expensive or cheap.

8.2.2 Example: ISTA

Given $y \in \mathbb{R}^n$, $X \in \mathbb{R}^{n\times p}$, the lasso criterion is given by

$$\begin{aligned} f(\beta) &= \frac{1}{2}||y - X\beta||_2^2 + \lambda||\beta||_1 \\ &= g(\beta) + h(\beta) \end{aligned}$$

Then the proximal mapping for $h(\beta) = \lambda||\beta||_1$ is

$$\begin{aligned} \text{prox}_{h,t}(\beta) &= \arg\min_z \frac{1}{2t}||x-z||_2^2 + \lambda||z||_1 \\ &= S_{\lambda t}(\beta) \end{aligned}$$

where $S_t(\beta)$ is the soft thresholding operator.

$$[S_t(\beta)]_i = \begin{cases} \beta_i - \lambda & \beta_i > \lambda \\ 0 & -\lambda \le \beta_i \le \lambda \\ \beta_i + \lambda & \beta_i < -\lambda \end{cases}$$

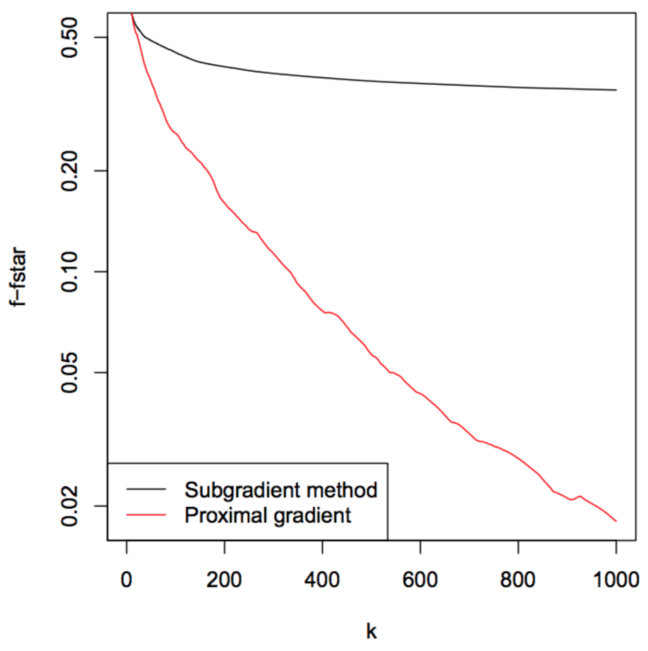
The proximal update is then given by

$$\beta^+ = \text{prox}_{h,t}(\beta - t\nabla g(\beta))$$

Using the fact that $\nabla g(\beta) = -X^T(y - X\beta)$ and the definition of $S_{\lambda t}$,

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

This is called the *iterative soft thresholding algorithm* and demonstrates the faster convergence speed of the proximal gradient method over the subgradient method in the following graph:



8.2.3 Backtracking Line Search

Backtracking works the same as it does for gradient descent, but because we require the derivative we use backtracking only on g, not f. Choose a parameter $0 < \beta < 1$. At each iteration start at $t = t_{init}$ and while

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

shrink the step size $t = \beta t$. Otherwise perform the proximal gradient update.

8.2.4 Convergence

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

- g is convex and differentiable with $\text{dom}(g) \in \mathbb{R}^n$, ∇g is L-Lipshitz

- h is convex, $\text{prox}_{h,t}(x)$ can be evaluated

Then we have the following theorem:

Theorem 8.2 *Proximal gradient descent with step size $t \leq \frac{1}{L}$ satisfies*

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

and the same result holds for backtracking, with t replaced by $\frac{\beta}{L}$

Thus proximal gradient descent has a convergence rate of $O(\frac{1}{\epsilon})$, same as gradient descent. However, we must include the cost of calculating the prox, which could be expensive.

8.3 Example: Matrix Completion

We now turn to another application of Proximal GD called Matrix Completion. This is an example of a problem where the prox operator is much more expensive to compute, despite it having a closed form. That being said, it is still a useful algorithm given the difficulty of the problem at hand, which involves trace norms (a very common application for Proximal GD next to the L_1 norm).

Matrix completion assumes there is a matrix $Y \in \mathbb{R}^{m \times n}$ which is only partially (often very sparsely) filled with observed entries $Y_{ij}, (i,j) \in \Omega$ where Ω is some observed set of entries. The goal is to find a matrix $B \in \mathbb{R}^{m \times n}$ which is both (1) close to the entries we did observe and (2) a low rank approximation of Y. More formally the objective can be stated as

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

The first term ensures B matches Y on known entries. For $\text{rank}(B) = r$ and singular values $\sigma_1(B) \geq \cdots \geq \sigma_r(B)$, the trace norm in the second term can be defined as $||B||_{\text{tr}} = \sum_{i=1}^{r} \sigma_i(B)$. This can be thought of as a convex but non-smooth relaxation of $\text{rank}(B)$ (analogous to the L_1 norm being a relaxation of L_0).

As a motivating example, one can think of a recommender system, whose Y is a rating matrix from m users and n movies and whose goal is to find a B to "fill-in" or predict ratings for users on movies they haven't seen. The low-rank provision comes from thinking about applications where there is an assumed low dimensional latent factor model that explains Y (ie. that there are canonical groups of people, and canonical groups of movies).

To apply Proximal GD to this problem we need to know how to calculate two pieces: $\nabla g(B)$ and $\text{prox}_t(B)$. The first is relatively easy but the second is more involved.

$\underline{\nabla g(B)}$:

To be more compact, we consider P_Ω, a projection onto the observed set (and zeros out all other observations)

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

which allows us to re-write $g(B) = \frac{1}{2}||P_\Omega(Y) - P_\Omega(B)||_F^2$ and gives us via chain-rule (and P_Ω being a linear operator)

$$\nabla g(B) = -(P_\Omega(Y) - P_\Omega(B))$$

$\underline{\text{prox}_t(B)}$:

Many of the details for this derivation are in Homework 2 but we outline the general approach and state the final result here. We have by definition that

$$\text{prox}_t(B) = \arg\min_Z \frac{1}{2t}||B - F||_F^2 + \lambda||Z||_{\text{tr}}$$

Subgradient optimality tells us that $\text{prox}_t(B) = Z$, where Z satisfies

$$0 \in Z - B + \lambda t \cdot \partial||Z||_{\text{tr}}$$

As shown in Homework 2 (Problem 2g), if $Z = U\Sigma V^T$ is a SVD for Z, then the sub-differential is

$$\partial||Z||_{\text{tr}} = \{UV^T + W : ||W||_{\text{op}} \leq 1, U^TW = 0, WV = 0\}$$

We can use this to verify that matrix soft-thresholding (defined below) is indeed the prox for the trace norm by showing that plugging in this specification for the prox for Z results in 0 being in the sub-differential at Z (Problem 3e):

$$\text{prox}_t(B) = S_{\lambda t}(B) = U\Sigma_{\lambda t}V^T$$

Here we slightly abuse notation as we now refer to U, Σ, V as coming from the SVD of B ($B = U\Sigma V^T$). We define $\Sigma_{\lambda t}$ to be the diagonal matrix with entries $(\Sigma_{\lambda t})_{ii} = \max\{\Sigma_{ii} - \lambda t, 0\}$. That is, the $\text{prox}_t(B)$ simply takes the SVD of B and soft-thresholds each of the diagonal entries in the Σ matrix (keeping U, V the same).

In summary, we can combine everything together to arrive at our Proximal GD update for this problem.

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

or equivalently (using the fact that $\nabla g(B)$ is 1-Lipschitz and choosing $t = \frac{1}{L} = 1$)

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

This algorithm is called the *soft-impute* algorithm. Because the prox requires calculating an SVD, this is much costlier of an evaluation than the soft-thresholding for vectors. In practice, the SVD can be done partially (ie. if singular values are found in order, once a singular value falls below the threshold, the algorithm terminates). Similar to LASSO, we can see exactly from the algorithm how we tend towards getting a low-rank solution.

8.4 Special Cases of Proximal Gradient Descent

Proximal gradient descent is also referred to as composite gradient descent (criterion is a composition of 2 functions) or generalized gradient descent because it is a generalization of gradient descent, and encompasses many variants of gradient descent, 3 of which are listed as follows.

Given a criterion $f = g + h$,

- Case 1: $h = 0$

  - $f = g$
  - $\text{prox}_{h,t}(x) = \arg\min_z \frac{1}{2t}||x - z||_2^2 + 0 = x$
  - $x^{(k)} = \text{prox}_{h,t_k}(x^{(k-1)} - t_k \nabla g(x^{(k-1)}) = x^{(k-1)} - t_k \nabla g(x^{(k-1)}$
  - This update is identical to **gradient descent**

- Case 2: $h = I_C$ (where C is the closed and convex set that g is optimized over)
  - $f = g + I_C(x)$ where $I_C = 0$ if $x \in C$, ∞ if $x \notin C$
  - $\text{prox}_{h,t}(x) = \arg\min_z \frac{1}{2t}||x - z||_2^2 + I_C(z) = \arg\min_{z\in C} ||x - z||_2^2 = P_C(x)$
  - Here, $P_C(x)$ is the projection operator onto C, i.e. the projection of x onto C.
  - $x^{(k)} = \text{prox}_{h,t}(x^{(k-1)} - t_k \nabla g(x^{(k-1)})) = P_C(x^{(k-1)} - t_k \nabla g(x^{(k-1)}))$
  - This is equivalent to taking a gradient descent step, then projecting the next vector back to C, which is equivalent to **projected gradient descent** over set C.

- **C**ase 3: $g = 0$
  - $f = h$
    This is equivalent to minimizing a non-differentiable convex function, which we already know a method for (subgradient method)
  - $x^{(k)} = \text{prox}_{h,t}(x^{(k-1)} - t_k \nabla g(x^{(k-1)})) = \text{prox}_{h,t}(x^{(k-1)}) = \arg\min_z \frac{1}{2t}||x^{(k-1)} - z||_2^2 + h(z)$
  - This is called **proximal minimization**.
  - This method predates proximal gradient descent and under certain assumptions, guarantees $O(1/\epsilon)$ convergence rate, which is faster than the $O(1/\epsilon^2)$ convergence of subgradient methods. This hints that proximal minimization is a method that uses much more knowledge than just subgradients.

8.5 Acceleration

Consider the generalized gradient descent setting,

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

where g is convex and differentiable, and h is convex, but not necessarily differentiable.

In choosing the next point to query, $x^{(k)}$, instead of considering $x^{(k-1)}$ and the ∇g at $x^{(k-1)}$, we consider

$$v = x^{(k-1)} + \frac{k-2}{k+1}(x^{(k-1)} - x^{(k-2)})$$

This is looking into the history of how we arrived at $x^{(k-1)}$, which is from the direction $x^{(k-1)} - x^{(k-2)}$. Then we go further in this most recent descent direction by $\frac{k-2}{k+1}$. This point will be denoted v.
Then we proceed in the same proximal gradient step at this point v by taking the proximal operator of the gradient step taken at v.

The main idea behind this method is that as we approach the solution, the gradient is vanishing, making the gradient step sizes smaller. Given that you are approaching the solution in a smooth path, the history (and directions taken) tell you something about which direction is "good", and using this information makes a notable difference when the gradient is vanishing.

- Accelerated Proximal Gradient Descent
  $x^{(k)} = \text{prox}_{h,t_k}(v - t_k \nabla g(v))$
- Accelerated Gradient Descent (when $h = 0$)
  $x^{(k)} = v - t_k \nabla g(v)$

Note: Convergence rates for accelerated proximal gradient descent will be discussed in the next lecture, and the sequence $\frac{k-2}{k+1}$ (referred to as "momentum weights") is crucial for this convergence rate to hold. The sequence $\frac{k-2}{k+1}$ dictates that when k is small (earlier steps of optimization), there is very little momentum because the direction is uninformative, then the momentum weights become bigger as you approach the solution and the directions are more informative, which counteracts vanishing gradients near the solution.