Last time

- subgradient method
- used for convex opts with non smooth criterion functions
- use pre set step sizes
- Upside: very generic
- Downside: can be slow: convergence rate is $O(1/\epsilon^2)$, vs gradient descent, which is $O(1/\epsilon)$
  - For any first order method that is a subgradient method, this is as good as it gets per Nesterov

Can we improve on subgradient method somehow?

- Yes! **proximal gradient descent**

A. Context: Decomposable functions

Suppose $f(x) = g(x) + h(x)$
where $g(x)$is convex, differentiable, $dom(g) = R^n$
h(x) is convex, not diff
so f(x) is not diff

If $f(x)$ were differentiable, the gradient descent update would be
$x^+ = x - t.\nabla f(x)$

Recall the motivation for this: minimize the quadratic approximation to f around x, replace hessian with $\frac{1}{t}I$

$$x^+ = argmin_z \; f(x) + \nabla f(x)^T(z-x) + \frac{1}{2t}||z-x||_2^2 = argmin \; \tilde{f}_t(z)$$

In our case, f is not diff but g is.

B. Motivation: Can we take quadratic approx to g and leave h alone?

so we update

$$x^+ = argmin_z \; \tilde{g_t}(z) + h(z)$$

$$= argmin_z \; g(x) + \nabla g(x)^T(z-x) + \frac{1}{2t}||z-x||_2^2 + h(z)$$

$first\ term\ has\ no\ reliance\ on\ z$

$$= argmin_z \; \frac{1}{2t}\left(z^Tz - z^Tx - x^Tz + x^Tx + 2t\nabla g(x)^Tz\right)$$

$$= argmin_z \frac{1}{2t}\left(z^T z - z^T x - x^T z + x^T x + 2t\nabla g(x)^T z + t||\nabla g(x)||_2^2 - t\nabla g(x)^T x - x^T t\nabla g(x)\right)$$

adding terms that have no reliance on z

$$= argmin_z \frac{1}{2t}||z-(x-t\nabla g(x))||_2^2 + h(z)$$

Yielding a nice interpretation: squared norm of (z − gradient update) + h(z)

$$argmin_z \frac{1}{2t}||z-(x-t\nabla g(x))||_2^2 + h(z)$$

The first term says 'stay close to gradient update for g'

The second terms says 'also make h small'

- interpretation: find the vector that minimizes the diff from the gradient update for g, and keeps h minimal

This is the definition of the **proximal mapping.**

$$prox_t(x;\ h) = argmin_z \left(\frac{1}{2t}||x-z||_2^2 + h(z)\right)$$

- interpretation: the proximal mapping is the vector 'closest' to x that minimizes h()
- Note that this is a minimization problem of its own! Albeit for the most part well defined, at least for many known h() - see below

Question: is this a well defined mapping? That is, does $prox()$ for a given x and h have a unique solution?

- Yes, because the first term is strictly convex
- So the sum will be strictly convex
- This guarantees a single solution

C. Proximal gradient descent algorithm

1. initialize $x^{(0)}$

2. repeat $x^{(k)} = prox_{t_k}\left(x^{(k-1)} - t_k \nabla g\left(x^{(k-1)}\right)\right),\ k = 1,\ 2,\ 3,\ ...$

To make this update step look familiar, rewrite it as

$$x^{(k)} = x^{(k-1)} - t_k.\ G_{t_k}\left(x^{(k-1)}\right)$$

where $G_t(x) = \frac{x - prox_t(x - t\nabla g(x))}{t}$, known as the generalized gradient of f

So,

$$x^{(k)} = x^{(k-1)} - t_k.\ G_{t_k}\left(x^{(k-1)}\right) = x^{(k-1)} - t_k\left(\frac{x^{(k-1)} - prox_t\left(x^{(k-1)} - t_k\nabla g\left(x^{(k-1)}\right)\right)}{t_k}\right)$$

- everything cancels out nicely. By design!

D. Why did this help?
Didn't we just swap one minimization problem for another? We still need to figure out how to minimize h?

Yes. But $prox()$ can be computed analytically for a lot of important functions h.
Note that
1. mapping $prox()$ doesn't depend on g at all, only on h
2. smooth part g can be complicated, but it doesn't matter! We only need to calculate $\nabla g(x)$

So when applied to a known problem, using the prox is useful

Convergence analysis will be in terms of number of iterations of the algorithm. However do remember that each iteration will evaluate $prox()$ once, and this can be cheap or expensive depending on h.

Example: ISTA
Another algorithm for the lasso.
Last time we saw subgradient method - didn't do very well.
We can try using proximal gradient method here, for which we need to calculate

$$G_t(x) = \frac{x - prox_t(x - t\nabla g(x))}{t},$$

That is, we need to know
1. what prox() is
2. what $\nabla g(x)$ is

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

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

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

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

8

1. You get $prox(B)$ by setting the subgradient of the function $\frac{1}{2t}||B - z||_2^2 + \lambda||z||_1$ to zero to get the argmin
But we already solved this exact problem! See Lecture 6, soft thresholding

So $prox_t(B) = argmin_z \frac{1}{2t}||B - z||_2^2 + \lambda||z||_1 = S_{\lambda t}(B)$

2. We know already that $\nabla g(B) = -X^T(y - XB)$ from all the least squares work we have seen so far...

So, putting this together, the update rule is

$B^k = prox_{t_k}\left(B^{(k-1)} - t_k \nabla g\left(B^{(k-1)}\right)\right) = S_{\lambda t}\left(B^{(k-1)} + t_k X^T(y - XB)\right)$

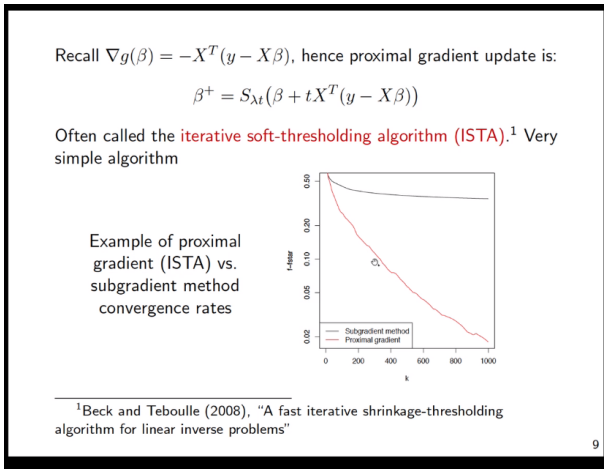
This is called the iterative soft thresholing algorithm = ista

It soft threshold the usual gradient update fothe unconstrained least squares problem!

- a very simple and intuitive algorithm

This shows directly from the algorithm that we get a sparse solution for the lasso

- sparsity inducing - it sets some of the B values to zero by soft thresholding!



E. Backtracking line search to get t

For proximal gradient this works imiliart to gradient descent, but operates on g and not f

Choose parameter $0 < B < 1$.

At each iteration:

- start at $t = t_{init}$
- 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 $t = Bt$
  - if $G_t(x)$ were the same as $\nabla g(x)$, this would yield the same formulation as the backtracking check in Lecture 5 on gradient descent, assuming $\alpha = 1/2$
- else perform $x^{(k)} = x^{(k-1)} - t_k . G_{t_k}\left(x^{(k-1)}\right)$

Note that every evaluation of G() also involves a call to prox() - so this might be expensive if prox is expensive to evaluate! There are alternative formulations that require less computation i.e. fewer calls to prox()

F. Convergence analysis

Same as gradient descent!

Except in every step, it does something more expensive than for gradient descent (i.e. the prox call)

- this is a good result if the prox cost is low

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 ...)

In the ISTA example, the prox cost is low. Soft thresholding is O(p) operation since it just needs one comparison per B value.

Here is an exmaple with high prox cost.

Example: matrix completion

Netflix prize problem - recommender systems

- assume some low rank latent structure: if Y were fully observed, it would be low rank

Given matrix $Y \in R^{m\,x\,n}$, only observe entries $Y_{ij},\ i,\ j \in \Omega$

We want to fill in the missing entries, i.e. we want to solve a matrix completion problem

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

- the trace norm is the sum of a matrix's singular values: $||B||_{tr} = \sum_{i=1}^{r} \sigma_i(B)$ where r = rank(B) and the sigmas are the singular values
- it is a convex approximation to the matrix's rank

- If B was a diagonal matrix:
  - rank = L0 norm
  - trace = L1 norm
- So by penalizing the trace norm in this problem, we are encouraging a low rank in the solution
  - approximate Y, but least complicated

We will try solving this using proximal gradient descent.
As a reminder

- we find the generalized gradient G for the criterion function f(x), using the gradient of the differentiable part g(x)
  - $G_t(x) = \frac{x - prox(x - t\nabla g(x))}{t}$
- Then do $x^+ = x - tG_t(x)$

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

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

Two ingredients needed for proximal gradient descent:

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

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

13

Q: how can you calculate the prox? how to solve this minimization problem?

Claim: $\text{prox}_t(B) = S_{\lambda t}(B)$, 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 Σ_λ is diagonal with

$$(\Sigma_\lambda)_{ii} = \max\{\Sigma_{ii} - \lambda, 0\}$$

Proof: note that $\text{prox}_t(B) = Z$, where Z satisfies

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

Helpful fact: if $Z = U\Sigma V^T$, then

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

Now plug in $Z = S_{\lambda t}(B)$ and check that we can get 0

14

The matrix soft thresholding adjusts (downwards) the singular values of the matrix

This is a challenging prox to derive! We need to minimize a function with a trace norm.

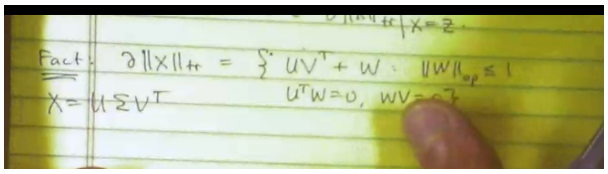
Right now, the only way we know how to minimize a non smooth function is subgradient optimality.

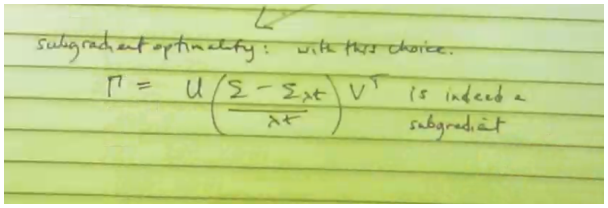
$prox_z(B) = argmin_z \frac{1}{2t}||B-Z||_2^2 + \lambda||Z||_{tr} = argmin_z \frac{1}{2}||B-Z||_2^2 + \lambda t||Z||_{tr}$

So at optimal z^*, $0 \in \partial prox_z(B) = Z^* - B + t\lambda\partial\left(||Z^*||_{tr}\right) \implies Z^* = B - t\lambda\Gamma$

Let $\Gamma \in \partial(||Z||_{tr})$. We need to find some Γ in this subdifferential, to help us calculate the value Z^*.

As a reminder, we know from the structure of the prox function that there is just one unique solution, as the first term is strictly convex.





The claim is that $prox(B) = Z^* = B - t\lambda\Gamma = S_{\lambda t}(B)$

Conclusion:

The proximal gradient step is

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

- Note that this involves computes the SVD of $P_\Omega(Y) - P_\Omega(B)$ and shrinking the singular values
- but Y is sparse (hence the problem) and B is designed to be low rank. This hopefully keeps the cost of performing SVD low - therefore making this an efficient algorithm
  - there are good numerical routines to do this

Similar to lasso - soft threshold the gradient step!

Hence proximal gradient update step is:

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

Note that $\nabla g(B)$ is Lipschitz continuous with $L = 1$, so we can choose fixed step size $t = 1$. Update step is now:

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

where $P_\Omega^\perp$ projects onto unobserved set, $P_\Omega(B) + P_\Omega^\perp(B) = B$

This is the soft-impute algorithm[2], simple and effective method for matrix completion

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

15

Update rule interpretation:
At every step, take observed entries of Y, fill in all the unobserved entries with what you thuoght they should have been in the last iteration. Then take this matrix and reduce its rank.

G. Special cases

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

- it is a superset of the other gradient descent methods we have seen - just adjust the notation/reshape the problem appropriately

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

1

H. Projected gradient descent

The prox if h = indicator just projects onto the set C
So the proximal gradient update step = project the gradient update step onto C!

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

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

I.e., $\text{prox}_t(x) = P_C(x)$, projection operator onto C

17

I. What if you cannot evaluate prox?

What happens if we can't evaluate prox?

Theory for proximal gradient, with $f = g + h$, assumes that prox function can be evaluated, i.e., assumes the minimization

$$\text{prox}_t(x) = \underset{z}{\text{argmin}} \ \frac{1}{2t}\|x - z\|_2^2 + h(z)$$

can be done exactly. In general, not clear what happens if we just minimize this approximately

But, if you can precisely control the errors in approximating the prox operator, then you can recover the original convergence rates[3]

In practice, if prox evaluation is done approximately, then it should be done to decently high accuracy

[3] Schmidt et al. (2011), "Convergence rates of inexact proximal-gradient methods for convex optimization"

20

- as the algorithm progresses, if (estimated prox - real prox) decreases, we get pretty close to the true solution

J. Acceleration

You can accelerate proximal gradient descent in order to achieve the optimal $O\left(1/\sqrt{\epsilon}\right)$ rate.

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[4]

We will follow Beck and Teboulle (2008), an extension of Nesterov (1983) to composite functions[5]

[4]Each step uses entire history of previous steps and makes two prox calls
[5]Each step uses information from two last steps and makes one prox call

21

Imagine the same setting as before

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

Accelerated proximal gradient method

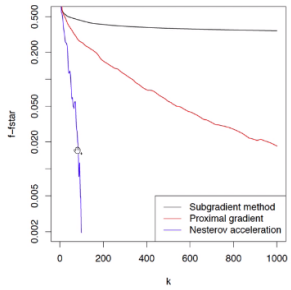
1. choose initial point $x^{(0)} = x^{(-1)} \in R^n$ (just for notation consistency)
2. repeat

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

$$x^{(k)} = prox_{t_k}(v - t_k \nabla g(v))$$

- usually, we just perform $x^{(k)} = prox_{t_k}(x^{(k-1)} - t_k \nabla g\left(x^{(k-1)}\right)$
- here, we find a new point v, which carries some past information from previous iterations
- if $h = 0$ this is the accelerated gradient method
- 'if you were travelling in a direction, this pushes you to continue traveling in that direction'
- the momentum weight increases as the number of iterations increases, eventually approaching 1
  - the intuition is that as you get closer to the optimum, the gradient gets smaller, so your gradient update is smaller too
  - but you don't want to slow down! want to keep continuing on

Back to lasso example: acceleration can really help!



Note: accelerated proximal gradient is not a descent method