CPSC 540: Machine Learning

Group L1-Regularization, Proximal-Gradient

Mark Schmidt

University of British Columbia

Winter 2017

Admin

- Assignment 1:
  - 1 late day to hand it in tonight.
  - 2 late days to hand it in next Monday.
- Assignment 2:
  - Out soon.
  - Due February 6.

Last Time: Convex Optimization Zoo

- We discussed the convex optimization zoo:
  - Iteration complexity of algorithms under different assumptions.

Assumption	Algorithm	Convex	Strongly-Convex
Subgradient bounded	Subgradient	$O(1/\epsilon^2)$	$O(1/\epsilon)$
Gradient is Lipschitz	Gradient	$O(1/\epsilon)$	$O\left(\frac{L}{\mu}\log(1/\epsilon)\right)$
Gradient is Lipschitz	Nesterov	$O(1/\sqrt{\epsilon})$	$O\left(\sqrt{\frac{L}{\mu}}\log(1/\epsilon)\right)$

- Smoothing gets faster rate only if you use Nesterov-style algorithms.

- Asymptotically-Newton methods get superlinear convergence.
  - Assuming strong-convexity, gradient is Lipschitz, and Hessian is Lipschitz.
  - Not achieved by $O(d)$ time/space practical methods.

Last Time: Weaker Conditions for Linear Convergence

- We argued gradient descent converges linearly under weaker assumptions.
  - No need to know L, it holds for various step-size stragies.
- No need for "strong-smoothness.
  - Just need Lipschitz-continuous gradient,

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|.$$

  - Or just for all t and some L that

$$L[f(x^{t+1}) - f(x^t)] \leq -\frac{1}{2}\|\nabla f(x^t)\|^2.$$

- No need for "strong-convexity", we just need the PL inequality,

$$\mu[f(x) - f^*] \leq \frac{1}{2}\|\nabla f(x)\|^2,$$

or if f is convex we can make it strongly-convex by addng L2-regularization.

Last Time: L1-Regularization

- We considered regularization by the L1-norm,

$$\operatorname*{argmin}_{x \in \mathbb{R}^d} g(x) + \lambda\|x\|_1.$$

  - Encourages solution x^* to be sparse.

- Convex approach to regularization and pruning irrelevant features.
  - Not perfect, but very fast.
  - Could be used as filter, or to initialize NP-hard solver.

- Non-smooth, but non-smooth part is separable,

$$\lambda\|x\|_1 = \sum_{j=1}^{d} \lambda|x_j| = \sum_{j=1}^{d} h_j(x_j).$$

which allows coordinate optimization.

Last Time: Coordinate Optimization

- In coordinate optimization each iteration t only updates one variable.

- More efficient than gradient descent if the iterations are d-times cheaper.
- This holds for the problem class

$$f(x) = g(Ax) + \sum_{j=1}^{d} h_j(x_j) + \sum_{i=1}^{d}\sum_{j=1}^{d} g_{ij}(x_i, x_j),$$

for smooth g and g_{ij} (and where g costs $O(n)$).

- We usually analyze it assuming partial derivatives are Lipschitz,

$$|\nabla_j f(x) - \nabla_j f(y)| \leq L|x_j - y_j|,$$

for some L whenever x and y only differ in coordinate j.
  - This is often easier to compute than L for the full gradient.

Convergence Rate of Randomized Coordinate Optimization

- Last time we analyzed coordinate optimization assuming that:
  - Partial derivative are Lipschitz and f satisfies PL inequality.
  - We choose coordinate to update j_t uniformly at random.
  - Given j_t, we take a gradient step on x_{j_t} with step-size $\alpha_t = 1/L$.
- We showed that this leads to the bound

$$\mathbb{E}[f(x^{t+1})] - f(x^*) \leq \left(1 - \frac{\mu}{dL}\right)[f(x^t) - f(x^*)].$$

- By recursing we get linear convergence rate,

$$\mathbb{E}[\mathbb{E}[f(x^{t+1})]] - f(x^*) \leq \mathbb{E}[\left(1 - \frac{\mu}{dL}\right)[f(x^t) - f(x^*)]] \quad \text{(expectation wrt } j_{t-1}\text{)}$$

$$\mathbb{E}[f(x^{t+1})] - f(x^*) \leq \left(1 - \frac{\mu}{dL}\right)\mathbb{E}[f(x^t) - f(x^*)] \quad \text{(iterated expectation)}$$

$$\leq \left(1 - \frac{\mu}{dL}\right)^2[f(x^{t-1}) - f(x^*)]$$

Randomized Coordinate Optimization vs. Gradient Descent

- So our rate for coordinate optimization is

$$\mathbb{E}[f(x^t] - f(x^*) \leq \left(1 - \frac{\mu}{dL}\right)^t [f(x^0) - f(x^*)],$$

  which means we need $O\left(d\frac{L}{\mu}\log(1/\epsilon)\right)$ iterations.

- Remember that gradient descent needs $O\left(\frac{L}{\mu}\log(1/\epsilon)\right)$ iterations.

- So coordinate optimzation is slower?
  - Yes, but remember we'll assume coordinate optimization steps are d-times cheaper.
  - So we should divide the coordinate optimization complexity by d.

Randomized Coordinate Optimization vs. Gradient Descent

- So for problems where coordinate steps are d-times cheaper we have

$$O\left(\frac{L}{\mu}\log(1/\epsilon)\right),$$

  for both algorithms in terms of gradient descent iteration costs.

- So why prefer coordinate optimization?

- The Lipschitz constants are different.
  - Gradient descent uses L_f and coordinate optimization uses L_c.
- $L_c \leq L_f$, so coordinate optimization is faster when steps are d-times cheaper.

Lipschitz Sampling

- Can we do better than choosing j_t uniformly at random?
- You can go faster if you have an L_j for each coordinate:

$$|\nabla_j f(x+\gamma e_j) - \nabla_j f(x)| \leq L_j |\gamma|.$$

- Using L_{j_t} as the step-size and sampling j_t proportional to L_j gives

$$\mathbb{E}[f(x^t)] - f(x^*) \leq \left(1 - \frac{\mu}{d\bar{L}}\right)^t [f(x^0) - f(x^*)],$$

  where $\bar{L}$ as the average Lipschitz constant (previously we used the maximum L_j).

- There are also greedy selection rules...

Gauss-Southwell Selection Rule

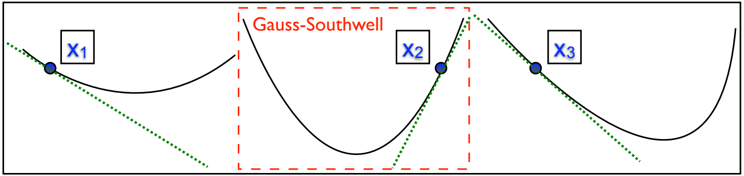
- Our bound on the progress if we choose coordinate j_t is

$$f(x^{t+1}) \leq f(x^t) - \frac{1}{2L}|\nabla_{j_t} f(x^t)|^2.$$

- The "best" j_t according to the bound is

$$j_t \in \underset{j}{\text{argmax}}\{|\nabla_j f(x^t)|\},$$

which is called greedy selection or the Gauss-Southwell rule.



Gauss-Southwell Selection Rule

- Our bound on the progress if we choose coordinate j_t is

$$f(x^{t+1}) \leq f(x^t) - \frac{1}{2L}|\nabla_{j_t} f(x^t)|^2.$$

- The "best" j_t according to the bound is

$$j_t \in \underset{j}{\text{argmax}}\{|\nabla_j f(x^t)|\},$$

  which is called greedy selection or the Gauss-Southwell rule.
- This gives a faster rate than uniformly at random.
  - You can prove this using that $|\nabla_{j_t} f(x^t)| = \|\nabla f(x^t)\|_\infty$.
  - And measuring PL in the ∞-norm,

$$\mu[f(x) - f(x^*)] \leq \frac{1}{2}\|f(x)\|_\infty^2.$$

  - But typically this can't be implemented d times faster than gradient descent.
    - You need an extra sparsity condition.

Gauss-Southwell-Lipschitz

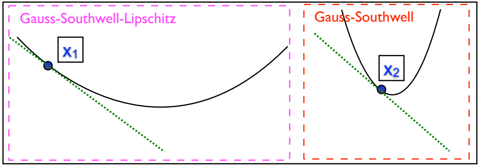
- Our bound on the progress with an L_j for each coordinate is

$$f(x^{t+1}) \leq f(x^t) - \frac{1}{2L_{j_t}}|\nabla_{j_t} f(x^t)|^2.$$

- The best coordinate to update according to this bound is

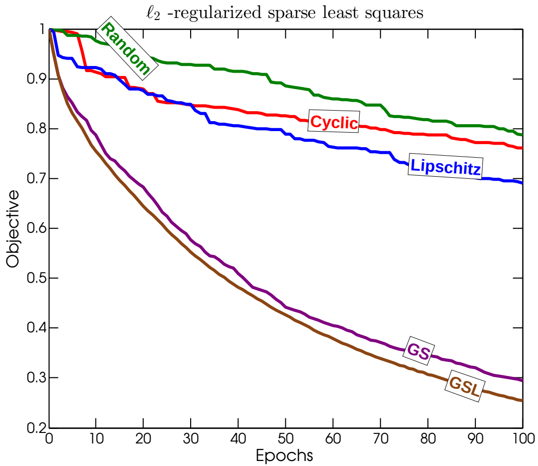
$$j_t \in \underset{j}{\text{argmax}} \frac{|\nabla_j f(x^t)|^2}{L_j}$$

which is called the Gauss-Southwell-Lipschitz rule.



- This is the optimal update for quadratic functions.

Numerical Comparison of Coordinate Selection Rules



Comparison on problem where Gauss-Southwell has similar cost to random:

Coordinate Optimization for Non-Smooth Objectives

- Last time we considered problems of the form

$$\underset{x\in\mathbb{R}^d}{\text{argmin}} \underbrace{g(x)}_{\text{smooth}} + \underbrace{\sum_{j=1}^{d} h_j(x_j)}_{\text{separable}},$$

  which includes L1-regularized least squares.
- Let's assume that
  - g is coordinate-wise Lipschitz continuous and μ-strongly convex.
  - h_j are general convex functions (could be non-smooth).
  - You do exact coordinate optimization.
- Then we can show that

$$\mathbb{E}[f(x^t)] - f(x^*) \leq \left(1 - \frac{\mu}{dL}\right)^t [f(x^0) - f(x^*)],$$

  the same convergence linear rate as if the non-smooth h_j were not there.

(and faster than the sublinear $O(1/\epsilon)$ for solving non-smooth strongly-convex problems)

Outline

Motivation for Group Sparsity

- Recall that multi-class logistic regression uses

$$\hat{y}^i = \underset{c}{\text{argmax}}\{w_c^T x^i\},$$

where we have a parameter vector w_c for each class c.

- We typically use softmax loss and write our parameters as a matrix,

$$W = \begin{bmatrix} | & | & | & & | \\ w_1 & w_2 & w_3 & \cdots & w_k \\ | & | & | & & | \end{bmatrix}$$

- Suppose we want to use L1-regularization for feature selection,

$$\underset{W \in \mathbb{R}^{d \times k}}{\text{argmin}} \underbrace{f(W)}_{\text{softmax loss}} + \underbrace{\lambda \sum_{c=1}^{k} \|w_c\|_1}_{\text{L1-regularization}}.$$

- Unfortunately, setting elements of W to zero may not select features.

Motivation for Group Sparsity

- Suppose L1-regularizationgives a sparse W with a non-zero in each row:

$$W = \begin{bmatrix} -0.83 & 0 & 0 & 0 \\ 0 & 0 & 0.62 & 0 \\ 0 & 0 & 0 & -0.06 \\ 0 & 0.72 & 0 & 0 \end{bmatrix}.$$

- Even though it's very sparse, it uses all features.
    - Feature 1 is used in w_1.
    - Feature 2 is used in w_3.
    - Feature 3 is used in w_4.
    - Feature 4 is used in w_2.
- The classifier multiplies feature j by each value in row j.
- In order to remove a feature, we need its entire row to be zero.

Motivation for Group Sparsity

- What we want is group sparsity:

$$W = \begin{bmatrix} -0.77 & 0.04 & -0.03 & -0.09 \\ 0 & 0 & 0 & 0 \\ 0.04 & -0.08 & 0.01 & -0.06 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

- Each row is a group, and we want groups (rows) of variables that have all zeroes.
  - If row j is zero, then x_j is not used by the model.
- Pattern arises in other settings where each row gives parameters for one feature:
  - Multiple regression, multi-label classification, and multi-task classification.

Motivation for Group Sparsiy

- Categorical features are another setting where group sparsity is needed.
- Consider categorical features encoded as binary indicator features:

City	Age
Vancouver	22
Burnaby	35
Vancouver	28

Vancouver	Burnaby	Surrey	Age ≤ 20	20 < Age ≤ 30	Age > 30
1	0	0	0	1	0
0	1	0	0	0	1
1	0	0	0	1	0

- A linear model would use

$$\hat{y}^i = w_1 x_{\text{van}} + w_2 x_{\text{bur}} + w_3 x_{\text{sur}} + w_4 x_{\leq 20} + w_5 x_{21-30} + w_6 x_{>30}.$$

- If we want feature selection of original categorical variables, we have 2 groups:
  - $\{w_1, w_2, w_3\}$ correspond to "City" and $\{w_4, w_5, w_6\}$ correspond to "Age".

Group L1-Regularization

- Consider a problem with a set of disjoint groups $\mathcal{G}$.
  - For example, $\mathcal{G} = \{\{1,2\},\{3,4\}\}$.

- Minimizing a function f with group L1-regularization:

$$\operatorname*{argmin}_{w \in \mathbb{R}^d} f(w) + \lambda \sum_{g \in \mathcal{G}} \|w_g\|_p,$$

where g refers to individual group indices and $\|\cdot\|_p$ is some norm.

- For certain norms, it encourages sparsity in terms of groups g.
  - Variables x_1 and x_2 will either be both zero or both non-zero.
  - Variables x_3 and x_4 will either be both zero or both non-zero.

Group L1-Regularization

- Why is it called group L1-regularization?

- Consider $G = \{\{1,2\},\{3,4\}\}$ and using L2-norm,

$$\sum_{g \in G} \|x_g\|_2 = \sqrt{x_1^2 + x_2^2} + \sqrt{x_3^2 + x_4^2}.$$

- If vector v contains the group norms, it's the L1-norm of v:

$$\text{If } v \triangleq \begin{bmatrix} \|x_{12}\|_2 \\ \|x_{34}\|_2 \end{bmatrix} \text{ then } \sum_{g \in G} \|x_g\|_2 = \|x_{12}\|_2 + \|x_{34}\|_2 = v_1 + v_2 = |v_1| + |v_2| = \|v\|_1.$$

- So L1-regularization encourages sparsity in the group norms.
  - When the norm of the group is 0, all group elements are 0.

Group L1-Regularization: Choice of Norm

- The group L1-regularizer is sometimes written as a "mixed" norm,

$$\|w\|_{1,p} \triangleq \sum_{g \in \mathcal{G}} \|w_g\|_p.$$

- The most common choice for the norm is the L2-norm:
    - If $\mathcal{G} = \{\{1,2\},\{3,4\}\}$ we obtain

$$\|w\|_{1,2} = \sqrt{w_1^2 + w_2^2} + \sqrt{w_3^2 + w_4^2}.$$

- Another common choice is the L∞-norm,

$$\|w\|_{1,\infty} = \max\{|w_1|,|w_2|\} + \max\{|w_3|,|w_4|\}.$$

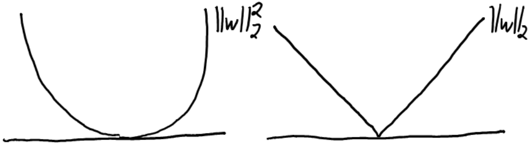
- But note that the L1-norm does not give group sparsity,

$$\|w\|_{1,1} = |w_1| + |w_2| + |w_3| + |w_4| = \|w\|_1,$$

as it's equivalent to non-group L1-reuglarization.

Sparsity from the L2-Norm?

- Didn't we say sparsity comes from the L1-norm and not the L2-norm?
  - Yes, but we were using the squared L2-norm.
- Squared vs. non-squared L2-norm in 1D:



- Non-squared L2-norm is absolute value.
  - It will set $w = 0$ for some finite λ.
- Squaring the L2-norm gives a smooth function and destorys sparsity.

Sparsity from the L2-Norm?

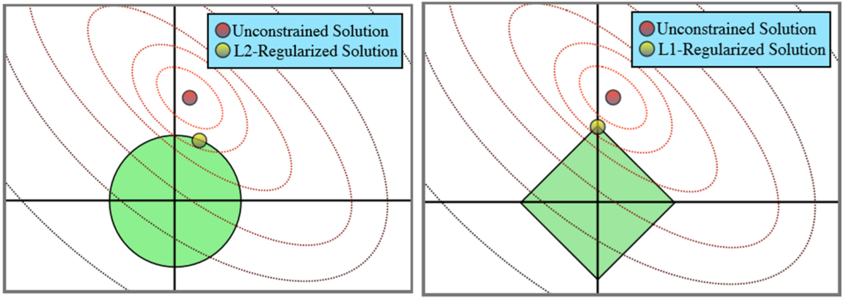
- Squared vs. non-squared L2-norm in 2D:

- The squared L2-norm is smooth and has no sparsity.
- For some finite λ, non-squared L2-norm simultaneously sets all variables to zero.

L1-Regularization vs. L2-Regularization

- Last time we looked at sparsity using our constraint trick,

$$\underset{w\in\mathbb{R}^d}{\text{argmin}}\, f(w) + \lambda\|w\|_p \quad \Leftrightarrow \quad \underset{w\in\mathbb{R}^d,\tau\in\mathbb{R}}{\text{argmin}}\ f(w) + \lambda\tau \text{ with } \tau \geq \|w\|_p.$$



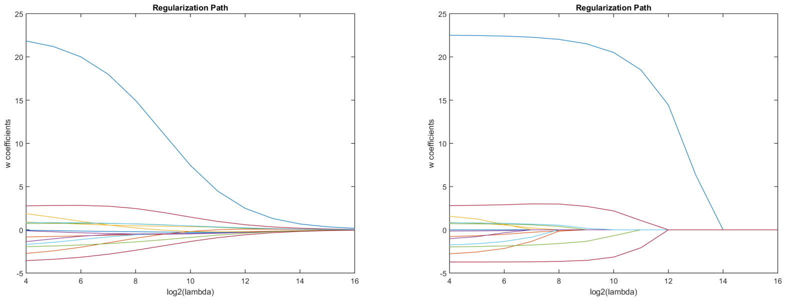
- Note that we're also minimizing the radius τ.
  - If τ shrinks to zero, all w are set to zero.
  - But if τ is squared there is virtually no penalty for having τ non-zero.

L2 and L1 Regularization Paths

- The regularization path is the set of w values as λ varies,

$$w^{\lambda} = \underset{w \in \mathbb{R}^d}{\text{argmin}}\ f(w) + \lambda r(w),$$

- Squared L2-regularization path vs. L1-regularization path:



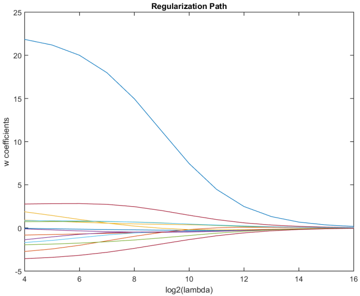
- With $r(w) = \|w\|^2$, each w_j gets close to 0 but is never exactly 0.
- With $r(w) = \|w\|_1$, each w_j gets set to exactly zero for a finite λ.

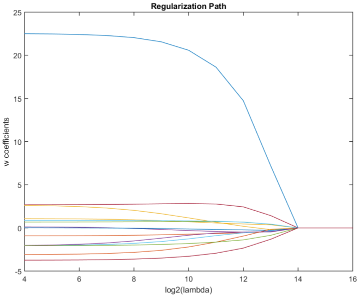
$L2^2$ and L2 Regularization Paths

- The regularization path is the set of w values as λ varies,

$$w^\lambda = \mathop{\mathrm{argmin}}_{w \in \mathbb{R}^d} f(w) + \lambda r(w),$$

- Squared L2-regularization path vs. non-squared path:



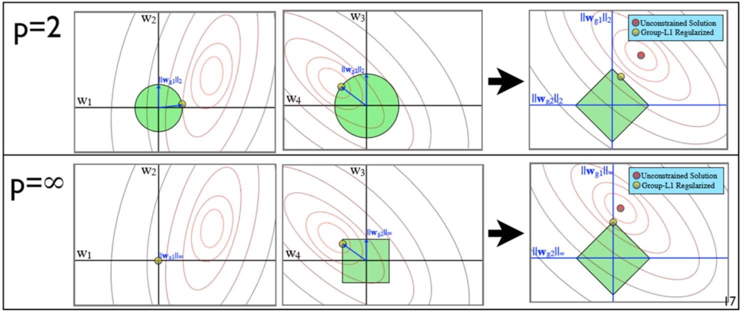


- With $r(w) = \|w\|^2$, each w_j gets close to 0 but is never exactly 0.
- With $r(w) = \|w\|_2$, all w_j get set to exactly zero for same finite λ.

Group L1-Regularization

- Minimizing a function f with group L1-regularization,

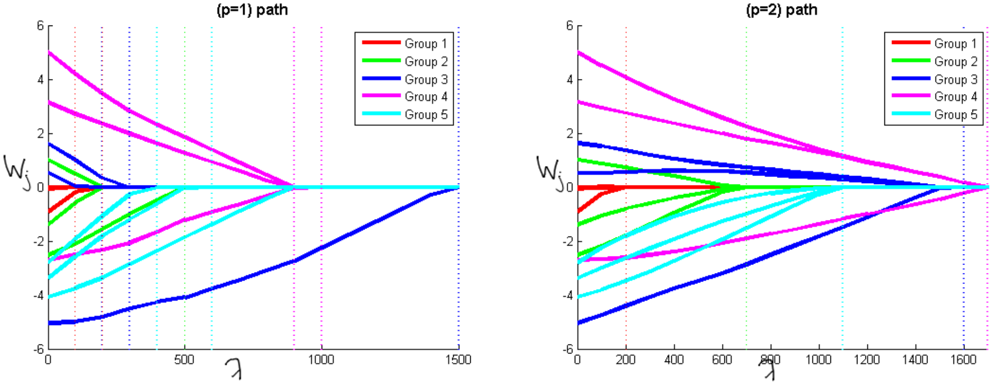
$$\operatorname*{argmin}_{w \in \mathbb{R}^d} f(w) + \lambda \|w\|_{1,p} \quad \Leftrightarrow \quad \operatorname*{argmin}_{w \in \mathbb{R}^d, \tau \in \mathbb{R}^{|\mathcal{G}|}} f(w) + \lambda \sum_{g=1}^{|\mathcal{G}|} \tau_g \text{ with } \tau_g \geq \|w\|_p.$$



- We're minimizing $f(w)$ plus the radiuses τ_g for each group g.
  - If τ_g shrinks to zero, all w_g are set to zero.

Group L1-Regularization Paths

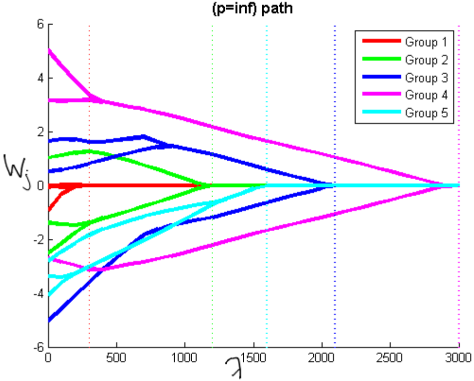
- The regularization path for group L1-regularizaiton for different p values:

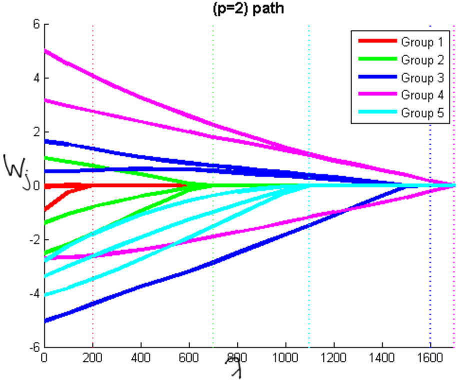


- With $p = 1$ there is no grouping effect.
- With $p = 2$ the groups become zero at the same time.

Group L1-Regularization Paths

- The regularization path for group L1-regularizaiton for different p values:





- With $p = 1$ there is no grouping effect.
- With $p = 2$ the groups become zero at the same time.
- With $p = \infty$ the groups converge to same magnitude which then goes to 0.

Outline

Solving Group L1-Regularization Problems

- The group L1-regularizer is non-differentiable for any norm.
- It's also non-separable, so we can't apply coordinate optimization.
  - You can do block coordinate optimization, but that won't work for other problems.

- A different problem structure we can use is

$$\operatorname*{argmin}_{x \in \mathbb{R}^d} \underbrace{g(x)}_{\text{smooth}} + \underbrace{r(x)}_{\text{"simple"}},$$

that it's the sum of a smooth function and a "simple" function.
  - We'll define "simple" later, but simple functions can be non-smooth.

- We can efficiently solve such problems with proximal-gradient methods.
  - A generalization of projected gradient methods.

Projected-Gradient for Non-Negative Constraints

- We used projected gradient in 340 for NMF to find non-negative solutions,

$$\underset{x \geq 0}{\operatorname{argmin}}\, f(x).$$

- In this case the algorithm has a simple form,

$$x^{t+1} = \max\{0, x^t - \alpha_t \nabla f(x^t)\},$$

  where the $\max$ is taken element-wise.
  - "Do a gradient descent step, set negative values to 0."

- An obvious algorithm to try, and works as well as unconstrained gradient descent.

Broken "Projected-Gradient" Algorithms

- Based on our intuition, maybe we can go faster using a Newton-like step,

$$x^{t+1} = \max\{0, x^t - \alpha_t[\nabla^2 f(x^t)]^{-1}\nabla f(x^t)\},$$

- We might also think that if we want x to be a probability

$$\operatorname*{argmin}_{x \geq 0,\, 1^T x = 1} f(x),$$

we could take a gradient step, set negative values to zero, and divide by the sum.

- Both of the above algorithms will NOT work.

Optimization with Simple Constraints

- Recall that we can view gradient descent as a minimizing quadratic approximation

$$x^{t+1} \in \underset{y}{\operatorname{argmin}} \left\{ f(x^t) + \nabla f(x^t)(y - x^t) + \frac{1}{2\alpha_t}\|y - x^t\|^2 \right\},$$

where we have a general step-size α_t instead of $1/L$.

- Now we want to optimize x over some convex set $\mathcal{C}$,

$$\underset{x \in \mathcal{C}}{\operatorname{argmin}}\, f(x).$$

- We could minimize quadratic approximation to f subject to the constraints,

$$x^{t+1} \in \underset{y \in \mathcal{C}}{\operatorname{argmin}} \left\{ f(x^t) + \nabla f(x^t)^T(y - x^t) + \frac{1}{2\alpha_t}\|y - x^t\|^2 \right\},$$

Projected Gradient

- We can re-write this iteration as

$$x^{t+1} \in \underset{y \in \mathcal{C}}{\text{argmin}} \left\{ f(x^t) + \nabla f(x^t)^T (y - x^t) + \frac{1}{2\alpha_t} \|y - x^t\|^2 \right\}$$

$$\equiv \underset{y \in \mathcal{C}}{\text{argmin}} \left\{ \alpha_t f(x^t) + \alpha_t \nabla f(x^t)^T (y - x^t) + \frac{1}{2} \|y - x^t\|^2 \right\} \quad \text{(multiply by } \alpha_t\text{)}$$

$$\equiv \underset{y \in \mathcal{C}}{\text{argmin}} \left\{ \frac{\alpha_t^2}{2} \|\nabla f(x^t)\|^2 + \alpha_t \nabla f(x^t)^T (y - x^t) + \frac{1}{2} \|y - x^t\|^2 \right\} \quad \text{(add constant)}$$

$$\equiv \underset{y \in \mathcal{C}}{\text{argmin}} \left\{ \|(y - x^t) + \alpha_t \nabla f(x^t)\|^2 \right\} \quad \text{(complete the square)}$$

$$\equiv \underset{y \in \mathcal{C}}{\text{argmin}} \left\{ \|y - \underbrace{(x^t - \alpha_t \nabla f(x^t))}_{\text{gradient descent}}\| \right\},$$

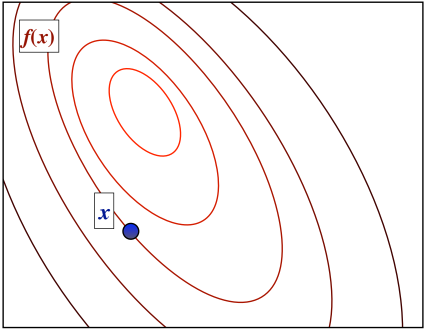
and this is called the projected-gradient algorithm.

Projected-Gradient

- We can view the projected-gradient algorithm as having two steps:
  1. Perform an unconstrained gradient descent step,
  $$x^{t+\frac{1}{2}} = x^t - \alpha_t \nabla f(x^t).$$
  2. Computed the projection onto the set $\mathcal{C}$,
  $$x^{t+1} \in \underset{y \in \mathcal{C}}{\text{argmin}} \|y - x^{t+\frac{1}{2}}\|.$$
- Projection is the closest point that satisfies the constraints.
  - Generalizes "projection" from linear algebra.
  - We'll also write projection of x onto $\mathcal{C}$ as
  $$\text{proj}_{\mathcal{C}}[x] = \underset{y \in \mathcal{C}}{\text{argmin}} \|y - x\|,$$
  and for convex $\mathcal{C}$ it's unique.

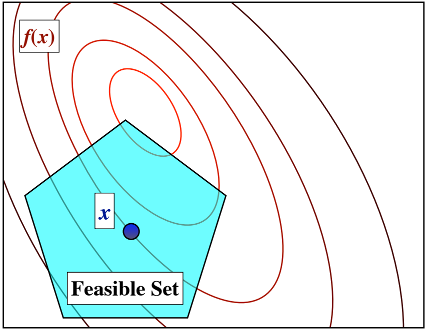
Projected-Gradient

$$x^{t+1} \in \underbrace{\operatorname*{argmin}_{y \in \mathcal{C}} \|y - x^{t+\frac{1}{2}}\|}_{\text{projection}}, \quad x^{t+\frac{1}{2}} = \underbrace{x^t - \alpha_t \nabla f(x^t)}_{\text{gradient}}.$$



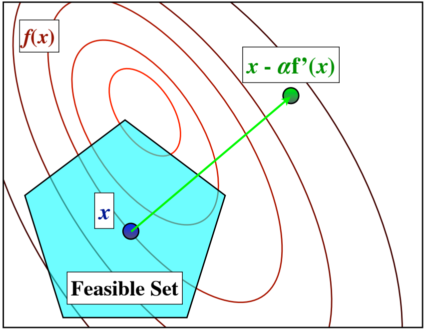
Projected-Gradient

$$x^{t+1} \in \underbrace{\mathop{\mathrm{argmin}}_{y \in \mathcal{C}} \|y - x^{t+\frac{1}{2}}\|}_{\text{projection}}, \quad x^{t+\frac{1}{2}} = \underbrace{x^t - \alpha_t \nabla f(x^t)}_{\text{gradient}}.$$



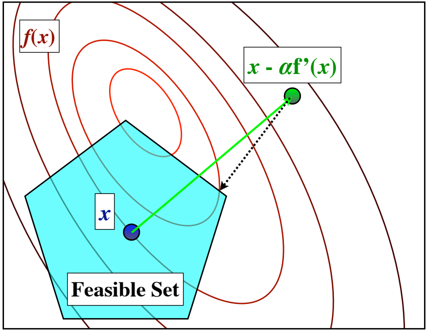
Projected-Gradient

$$x^{t+1} \in \underbrace{\operatorname*{argmin}_{y \in \mathcal{C}} \|y - x^{t+\frac{1}{2}}\|}_{\text{projection}}, \quad x^{t+\frac{1}{2}} = \underbrace{x^t - \alpha_t \nabla f(x^t)}_{\text{gradient}}.$$



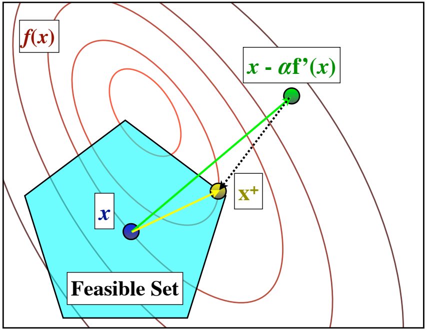
Projected-Gradient

$$x^{t+1} \in \underbrace{\operatorname*{argmin}_{y \in \mathcal{C}} \|y - x^{t+\frac{1}{2}}\|}_{\text{projection}}, \quad x^{t+\frac{1}{2}} = \underbrace{x^t - \alpha_t \nabla f(x^t)}_{\text{gradient}}.$$



Projected-Gradient

$$x^{t+1} \in \underbrace{\underset{y \in \mathcal{C}}{\text{argmin}} \|y - x^{t+\frac{1}{2}}\|}_{\text{projection}}, \quad x^{t+\frac{1}{2}} = \underbrace{x^t - \alpha_t \nabla f(x^t)}_{\text{gradient}}.$$



Convergence Rate of Projected Gradient

- Iteration complexity of projection-gradient:

Assumption	Algorithm	Convex	Strongly-Convex
Subgradient bounded	Subgradient	$O(1/\epsilon^2)$	$O(1/\epsilon)$
Gradient is Lipschitz	Gradient	$O(1/\epsilon)$	$O\left(\frac{L}{\mu}\log(1/\epsilon)\right)$
Gradient is Lipschitz	Nesterov	$O(1/\sqrt{\epsilon})$	$O\left(\sqrt{\frac{L}{\mu}}\log(1/\epsilon)\right)$

- These are the same rates we had for unconstrained optimization.

- Other nice properties:
  - With $\alpha_t < 2/L$, guaranteed to decrease objective.
  - For convex f the only "fixed points" are optimal solutions,

$$x^* = \text{proj}_{\mathcal{C}}[x^* - \alpha \nabla f(x^*)],$$

  for any step-size $\alpha > 0$:

Simple Convex Sets

- Projected-gradient is only efficient if the projection is cheap.
- We say that $\mathcal{C}$ is simple if the projection is cheap.
  - For example, if it costs $O(d)$ then it adds no cost to the algorithm.
- For example, if want $x \geq 0$ then projection sets negative values to 0.
  - Non-negative constraints are "simple".
- Another example if $x \geq 0$ and $x^T 1 = 1$, the probability simplex.
  - There are $O(d)$ algorithm to compute this projection.

Simple Convex Sets

- Other examples of simple convex sets:
  - Having upper and lower bounds on the variables, $LB \leq x \leq UB$.
  - Having a linear equality constraint, $a^T x = b$, or a small number of them.
  - Having a half-space constraint, $a^T x \leq b$, or a small number of them.
  - Having a norm-ball constraint, $\|x\|_p \leq \tau$, for $p = 1, 2, \infty$ (fixed τ).
  - Having a norm-cone constraint, $\|x\|_p \leq \tau$, for $p = 1, 2, \infty$ (variable τ).

Group L1-Regularization

- We can convert the non-smooth group L1-regularization problem,

$$\underset{x \in \mathbb{R}^d}{\operatorname{argmin}}\, g(x) + \lambda \sum_{g \in G} \|x_g\|_2,$$

into a smooth problem with simple constraints:

$$\underset{x \in \mathbb{R}^d}{\operatorname{argmin}}\, \underbrace{g(x) + \lambda \sum_{g \in G} r_g}_{f}, \text{ subject to } r_g \geq \|x_g\|_2 \text{ for all } g.$$

- Here the constraitnts are separable:
  - We can project onto each norm-cone separately.

- Since norm-cones are simple we can solve this with projected-gradient,

Faster Projected-Gradient Methods

- Accelerated projected-gradient method has the form

$$x^{t+1} = \text{proj}_{\mathcal{C}}[y^t - \alpha_t \nabla f(x^t)]$$
$$y^{t+1} = x^t + \beta_t(x^{t+1} - x^t).$$

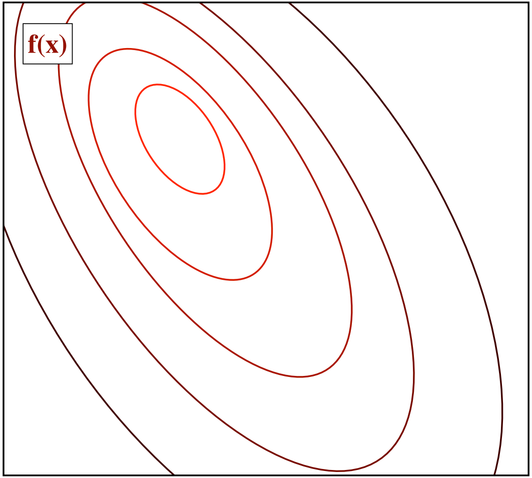
- We could alternately use the Barzilai-Borwein step-size.
  - Known as spectral projected-gradient.

- The naive Newton-like methods with Hessian approximation H_t,

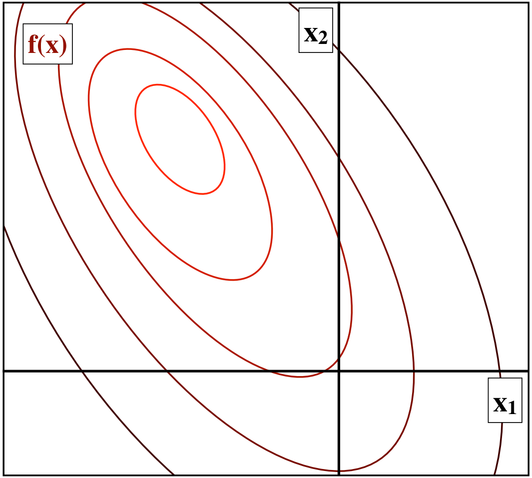
$$x^{t+1} = \text{proj}_{\mathcal{C}}[x^t - \alpha_t [H_t]^{-1} \nabla f(x^t)],$$

does NOT work.

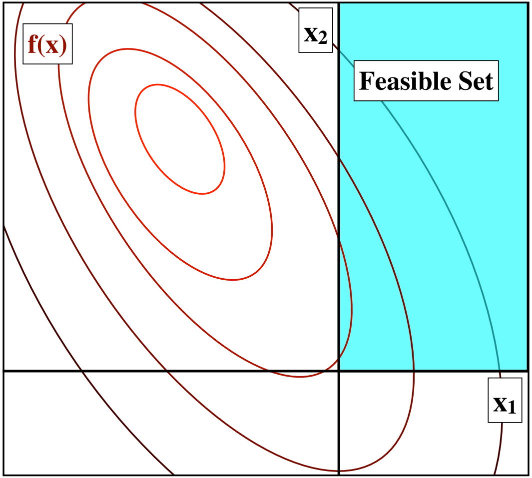
Naive Projected-Newton



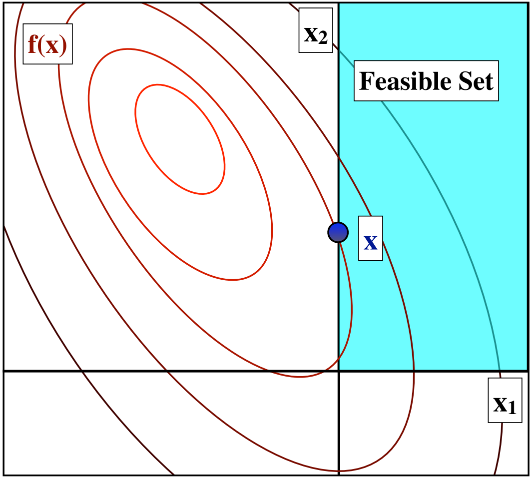
Naive Projected-Newton



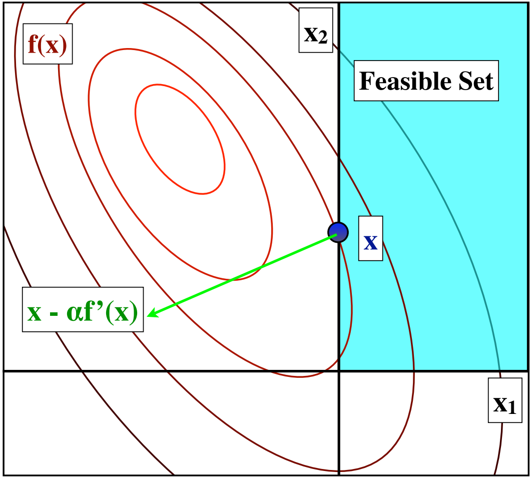
Naive Projected-Newton



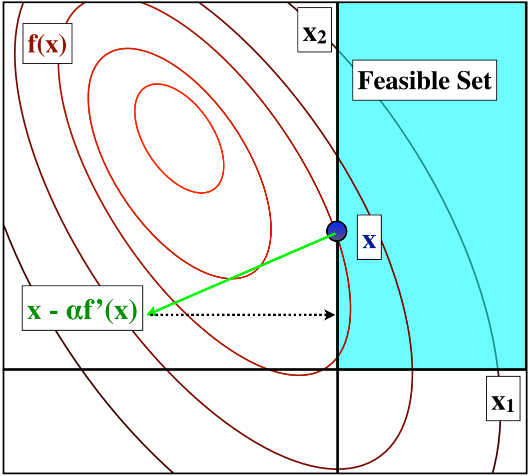
Naive Projected-Newton



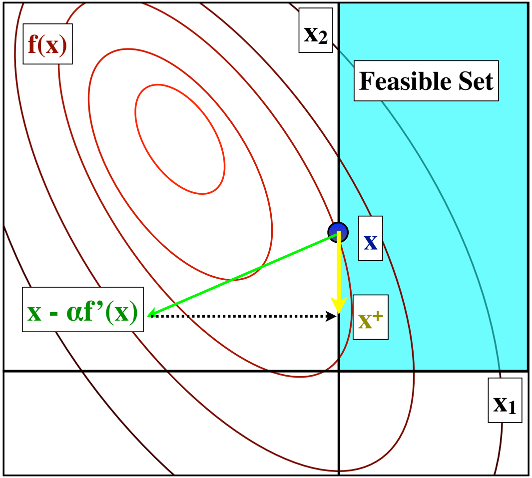
Naive Projected-Newton



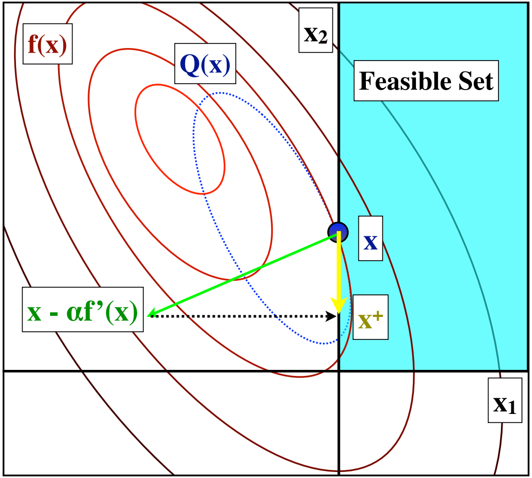
Naive Projected-Newton



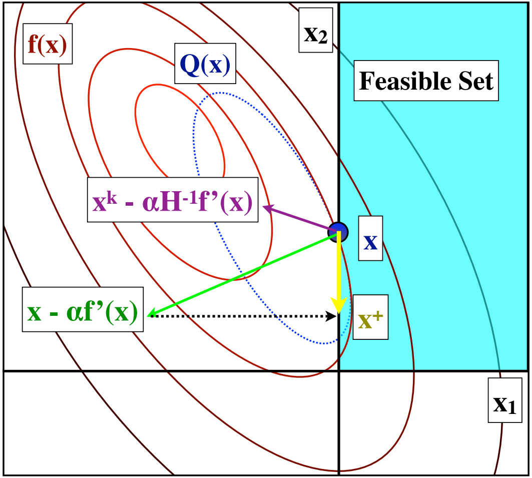
Naive Projected-Newton



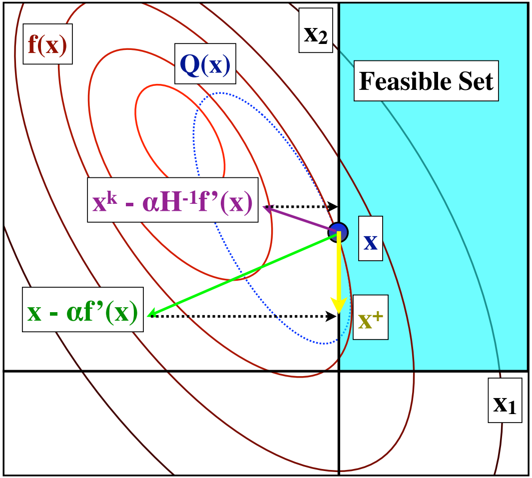
Naive Projected-Newton



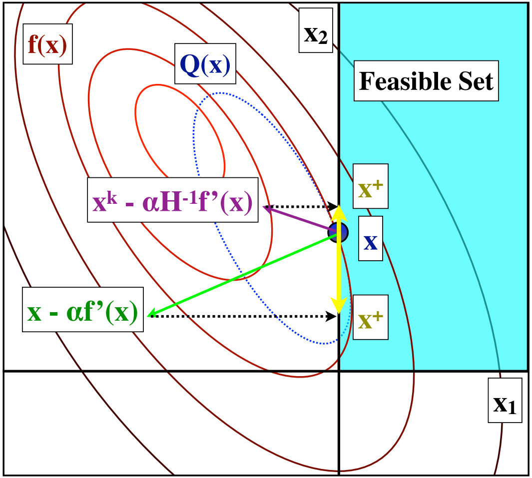
Naive Projected-Newton



Naive Projected-Newton



Naive Projected-Newton



Projected-Newton Method

- Projected-gradient minimizes quadratic approximation,

$$x^{t+1} = \operatorname*{argmin}_{y \in C} \left\{ f(x^t) + \nabla f(x^t)(y - x^t) + \frac{1}{2\alpha_t}\|y - x^t\|^2 \right\}.$$

- Newton's method can be viewed as quadratic approximation (wth $H_t \approx \nabla^2 f(x^t)$):

$$x^{t+1} = \operatorname*{argmin}_{y \in \mathbb{R}^d} \left\{ f(x^t) + \nabla f(x^t)(y - x^t) + \frac{1}{2\alpha_t}(y - x^t)H_t(y - x^t) \right\}.$$

- Projected Newton minimizes constrained quadratic approximation:

$$x^{t+1} = \operatorname*{argmin}_{y \in C} \left\{ f(x^t) + \nabla f(x^t)(y - x^t) + \frac{1}{2\alpha_t}(y - x^t)H_t(y - x^t) \right\}.$$

- Equivalently, we project Newton step under different Hessian-defined norm,

$$x^{t+1} = \operatorname*{argmin}_{y \in C} \|y - (x^t - \alpha_t H_t^{-1}\nabla f(x^t))\|_{H_t},$$

where general "quadratic norm" is $\|z\|_A = \sqrt{z^T A z}$ for $A \succ 0$.

Discussion of Projected-Newton

- Projected-Newton iteration is given by

$$x^{t+1} = \operatorname*{argmin}_{y \in C} \left\{ f(x^t) + \nabla f(x^t)(y - x^t) + \frac{1}{2\alpha_t}(y - x^t)H_t(y - x^t) \right\}.$$

- But this is expensive even when $\mathcal{C}$ is simple.

- There are a variety of practical alternatives:
  - If H_t is diagonal then this is typically simple to solve.
  - Two-metric projection methods are special algorithms for upper/lower bounds.
    - Fix problem of naive method in this case by making H_t partially diagonal.
  - Inexact projected-Newton: solve the above approximately.
    - Useful when f is very expensive but H_t and $\mathcal{C}$ are simple.
    - "Costly functions with simple constraints".

Outline

Should we use projected-gradient for non-smooth problems?

- We converted non-smooth problem into smooth with simple constraints.

- But transforming might make problem harder:
  - For L1-regularization least squares,

$$\operatorname*{argmin}_{w \in \mathbb{R}^d} \frac{1}{2}\|Xw - y\|^2 + \lambda \|w\|_1,$$

  we can re-write as a smooth problem with bound constraints,

$$\operatorname*{argmin}_{w_+ \geq 0,\, w_- \geq 0} \|X(w_+ - w_-) - y\|^2 + \lambda \sum_{j=1}^{d} (w_+ + w_-).$$

  - Transformed problem is not strongly convex even if the original was.
- Proximal-gradient methods apply to analogous non-smooth problems,

$$\operatorname*{argmin}_{w \in \mathbb{R}^d} \underbrace{g(w)}_{\text{smooth}} + \underbrace{r(w)}_{\text{simple}}.$$

Gradient Method

- We want to solve a smooth optimization problem:

$$\operatorname*{argmin}_{x \in \mathbb{R}^d} f(x).$$

- Iteration x^t works with a quadratic approximation to f:

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

$$x^{t+1} = \operatorname*{argmin}_{y \in \mathbb{R}^d} \left\{ f(x^t) + \nabla f(x^t)^T (y - x^t) + \frac{1}{2\alpha_t}\|y - x^t\|^2 \right\}.$$

We can equivalently write this as the quadratic optimization:

$$x^{t+1} = \operatorname*{argmin}_{y \in \mathbb{R}^d} \left\{ \frac{1}{2}\|y - (x^t - \alpha_t \nabla f(x^t))\|^2 \right\},$$

and the solution is the gradient algorithm:

$$x^{t+1} = x^t - \alpha_t \nabla f(x^t).$$

Proximal-Gradient Method

- We want to solve a smooth plus non-smooth optimization problem:

$$\operatorname*{argmin}_{x \in \mathbb{R}^d} f(x) + r(x).$$

- Iteration x^t works with a quadratic approximation to f:

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

$$x^{t+1} = \operatorname*{argmin}_{y \in \mathbb{R}^d} \left\{ f(x^t) + \nabla f(x^t)^T (y - x^t) + \frac{1}{2\alpha_t}\|y - x^t\|^2 + r(y) \right\}.$$

We can equivalently write this as the proximal optimization:

$$x^{t+1} = \operatorname*{argmin}_{y \in \mathbb{R}^d} \left\{ \frac{1}{2}\|y - (x^t - \alpha_t \nabla f(x^t))\|^2 + \alpha_t r(y) \right\},$$

and the solution is the proximal-gradient algorithm:

$$x^{t+1} = \text{prox}_{\alpha r}[x^t - \alpha_t \nabla f(x^t)].$$

Proximal-Gradient Method

- So proximal-gradient step takes the form:

$$x^{t+\frac{1}{2}} = x^t - \alpha_t \nabla f(x^t)$$
$$x^{t+1} = \operatorname*{argmin}_{y \in \mathbb{R}^d} \left\{ \frac{1}{2}\|y - x^{t+\frac{1}{2}}\|^2 + \alpha_t r(y) \right\}.$$

- Second part is called the proximal operator with respect to $\alpha_t r$.

- Convergence rates are still the same as for minimizing f alone:
  - E.g, if ∇f is L-Lipschitz, f is μ-strongly convex and r is convex, then

$$F(x^t) - F(x^*) \leq \left(1 - \frac{\mu}{L}\right)^t \left[F(x^0) - F(x^*)\right],$$

  where $F(x) = f(x) + r(x)$.

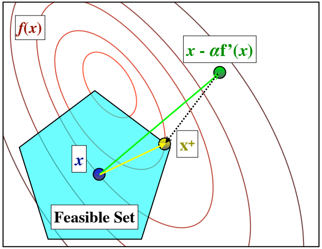
Special case of Projected-Gradient Methods

- Projected-gradient methods are a special case:

$$r(y) = \begin{cases} 0 & \text{if } x \in \mathcal{C} \\ \infty & \text{if } x \notin \mathcal{C} \end{cases}, \quad (\text{indicator function for convex set } \mathcal{C})$$

gives

$$x^{t+1} = \operatorname*{argmin}_{y \in \mathbb{R}^d} \frac{1}{2}\|y - x\|^2 + r(y) = \operatorname*{argmin}_{y \in \mathcal{C}} \frac{1}{2}\|y - x\|^2 = \operatorname*{argmin}_{y \in \mathcal{C}} \|y - x\|.$$



Proximal Operator, Iterative Soft Thresholding

- The proximal operator is the solution to

$$\text{prox}_r[x] = \underset{y \in \mathbb{R}^d}{\text{argmin}} \ \frac{1}{2}\|y - x\|^2 + r(y).$$

- If $r(y) = \lambda\|y\|_1$, proximal operator is soft-threshold:
  - Apply $x_j = \text{sign}(x_j)\max\{0, |x_j| - \lambda\}$ element-wise.
  - An example with $\lambda = 1$:

$$\begin{array}{ccc} \text{Input} & \text{Threshold} & \text{Soft-Threshold} \\ \begin{bmatrix} 0.6715 \\ -1.2075 \\ 0.7172 \\ 1.6302 \\ 0.4889 \end{bmatrix} & \begin{bmatrix} 0 \\ -1.2075 \\ 0 \\ 1.6302 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ -0.2075 \\ 0 \\ 0.6302 \\ 0 \end{bmatrix} \end{array}$$

- Has the nice property that iterations x^t are sparse.

Proximal-Gradient for L1-Regularization

- The proximal operator for L1-regularization when using step-size α_t,

$$\operatorname*{argmin}_{y \in \mathbb{R}^d} \left\{ \frac{1}{2}\|y - x\|^2 + \alpha_t \lambda \|y\|_1 \right\},$$

applies soft-threshold element-wise,

$$x_j = \frac{x_j}{|x_j|}\max\{0, |x_j| - \alpha_t \lambda\}.$$

- w_j with absolute values below $\alpha_t\lambda$ get set to 0.
- w_j with absolute values above $\alpha_t\lambda$ get shrunk by $\alpha_t\lambda$.

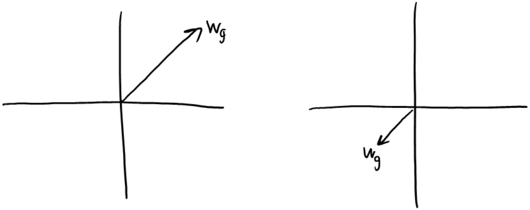
Proximal-Gradient for Group L1-Regularization

- The proximal operator for group L1-regularization,

$$\underset{y \in \mathbb{R}^d}{\operatorname{argmin}} \left\{ \frac{1}{2}\|y - x\|^2 + \alpha_t \lambda \sum_{g \in G} \|y\|_2 \right\},$$

applies a soft-threshold group-wise,

$$x_g = \frac{x_g}{\|x_g\|_2} \max\{0, \|x_g\|_2 - \alpha_t \lambda\}.$$



- So we can solve group L1-regularization problems as fast as smooth problems.

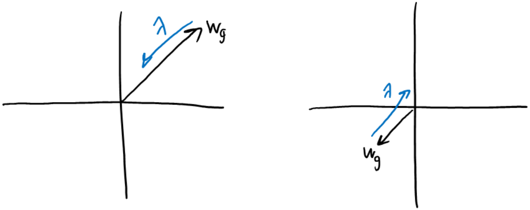
Proximal-Gradient for Group L1-Regularization

- The proximal operator for group L1-regularization,

$$\underset{y \in \mathbb{R}^d}{\text{argmin}} \left\{ \frac{1}{2}\|y - x\|^2 + \alpha_t \lambda \sum_{g \in G} \|y\|_2 \right\},$$

  applies a soft-threshold group-wise,

$$x_g = \frac{x_g}{\|x_g\|_2} \max\{0, \|x_g\|_2 - \alpha_t \lambda\}.$$



- So we can solve group L1-regularization problems as fast as smooth problems.

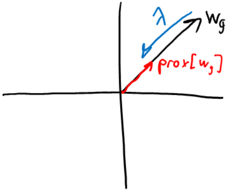
Proximal-Gradient for Group L1-Regularization

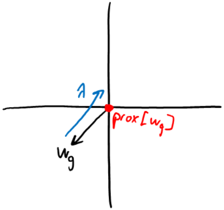
- The proximal operator for group L1-regularization,

$$\operatorname*{argmin}_{y \in \mathbb{R}^d} \left\{ \frac{1}{2}\|y - x\|^2 + \alpha_t \lambda \sum_{g \in G} \|y\|_2 \right\},$$

  applies a soft-threshold group-wise,

$$x_g = \frac{x_g}{\|x_g\|_2} \max\{0, \|x_g\|_2 - \alpha_t \lambda\}.$$





- So we can solve group L1-regularization problems as fast as smooth problems.

Summary

- Group L1-regularization encourages sparsity in variable groups.
- Projected-gradient allows optimization with simple constraints.
- Projected-Newton: even faster rates in special cases.
- Proximal-gradient: linear rates for sum of smooth and simple non-smooth.

- Next time: what if the number of training examples n is huge?