# CPSC 540: Machine Learning

## Subgradients and Projected Gradient

Mark Schmidt

University of British Columbia

Winter 2018

# Admin

- Auditting/registration forms:
  - Submit them at end of class, pick them up end of next class.
  - I need your prereq form before I'll sign registration forms.
  - I wrote comments on the back of some forms.
- Assignment 1:
  - 1 late day to hand in tonight, 2 late days for Wednesday.

# Last Time: Iteration Complexity

- We discussed the iteration complexity of an algorithm for a problem class:
  - "How many iterations $t$ before we guarantee an accuracy $\epsilon$"?

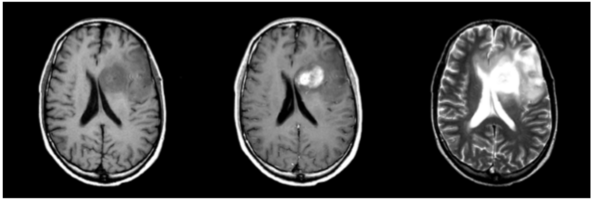
- Iteration complexity of gradient descent when $\nabla f$ is Lipschitz continuous:

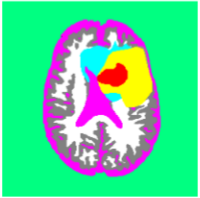
| Assumption | Iteration Complexity | Quantity |
|---|---|---|
| Non-Convex | $t = O(1/\epsilon)$ | $\min_{k=1,2,\dots,t} \|\nabla f(w^k)\|^2 \leq \epsilon$ |
| Convex | $t = O(1/\epsilon)$ | $f(w^t) - f^* \leq \epsilon$ |
| Strongly-Convex | $t = O(\log(1/\epsilon))$ | $f(w^t) - f^* \leq \epsilon$ |

- Adding L2-regularization to a convex function gives a strongly-convex function.
  - So L2-regularization can make gradient descent converge much faster.

# Motivation: Automatic Brain Tumour Segmentation

- Task: identifying tumours in multi-modal MRI data.

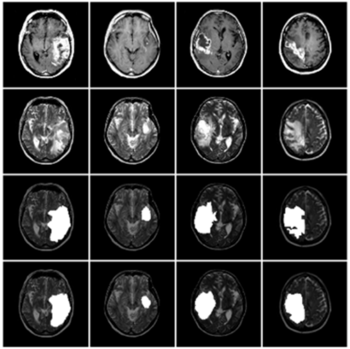


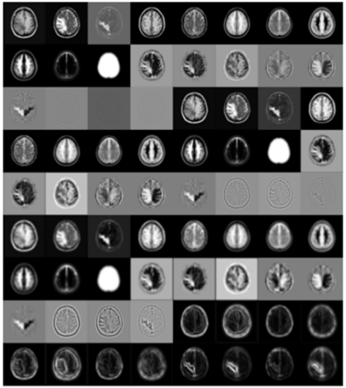


- Applications:
  - Image-guided surgery.
  - Radiation target planning.
  - Quantifying treatment response.
  - Discovering growth patterns.

# Motivation: Automatic Brain Tumour Segmentation

- Formulate as supervised learning:
  - Pixel-level classifier that predicts "tumour" or "non-tumour".
  - Features: convolutions, expected values (in aligned template), and symmetry.
    - All at multiple scales.





# Motivation: Automatic Brain Tumour Segmentation

- Logistic regression was among the most effective, with the right features.
- But if you used all features, it overfit.
  - We needed feature selection.

- Classical approach:
  - Define some score: AIC, BIC, cross-validation error, etc.
  - Search for features that optimize score:
    - Usually NP-hard, so we use greedy: forward selection, backward selection,. . .
  - In brain tumour application, even greedy methods were too slow.
    - Just one image gives 8 million training examples.

# Feature Selection

- General feature selection problem:
  - Given our usual $X$ and $y$, we'll use $x_j$ to represent column $j$:

$$X = \begin{bmatrix} | & | & & | \\ x_1 & x_2 & \dots & x_d \\ | & | & & | \end{bmatrix}, \quad y = \begin{bmatrix} | \\ y \\ | \end{bmatrix}.$$

  - We think some features/columns $x_j$ are irrelevant for predicting $y$.
- We want to fit a model that uses the "best" set of features.
- One of most important problems in ML/statistics, but very very messy.
  - In 340 we saw how difficult it is to define what "relevant" means.

# L1-Regularization

- A popular appraoch to feature selection we saw in 340 is L1-regularization:

$$F(w) = f(w) + \lambda \|w\|_1.$$

- Advantages:
  - Fast: can apply to large datasets, just minimizing one function.
    - Convex if $f$ is convex.
  - Reduces overfitting because it simultaneously regularizes.
- Disadvantages:
  - Prone to false positives, particularly if you pick $\lambda$ by cross-validation.
  - Not unique: there may be infinite solutions.

- There exist many extensions:
  - "Elastic net" adds L2-regularization to make solution unique.
  - "Bolasso" applies this on bootstrap samples to reduce false positives.
  - Non-convex regularizers reduce false positives but are NP-hard.

# L1-Regularization

- Key property of L1-regularization: if $\lambda$ is large, solution $w^*$ is sparse:
  - $w^*$ has many values that are exactly zero.

- How setting variables to exactly 0 performs feature selection in linear models:

$$\hat{y}^i = w_1 x_1^i + w_2 x_2^i + w_3 x_3^i + w_4 x_4^i + w_5 x_5^i.$$

- If $w = \begin{bmatrix} 0 & 0 & 3 & 0 & -2 \end{bmatrix}^T$ then:

$$\begin{aligned} \hat{y}^i &= 0x_1^i + 0x_2^i + 3x_3^i + 0x_4^i + (-2)x_5^i \\ &= 3x_3^i - 2x_5^i. \end{aligned}$$

- Features $\{1, 2, 4\}$ are not used in making predictions: we “selected” $\{2, 5\}$.
  - To understand why variables are set to exactly 0, we need the notion of subgradient.

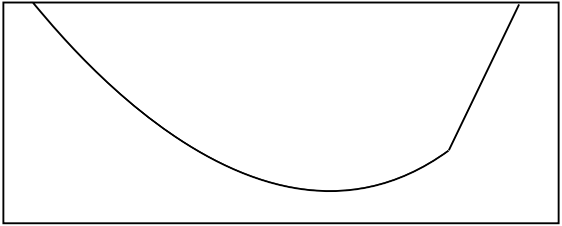
# Sub-Gradients and Sub-Differentials

*Differentiable* convex functions are always above tangent,

$$f(v) \geq f(w) + \nabla f(w)^T(v - w), \forall w, v.$$

*A vector $d$ is a subgradient of a convex function $f$ at $w$ if*

$$f(v) \geq f(w) + d^T(v - w), \forall v.$$



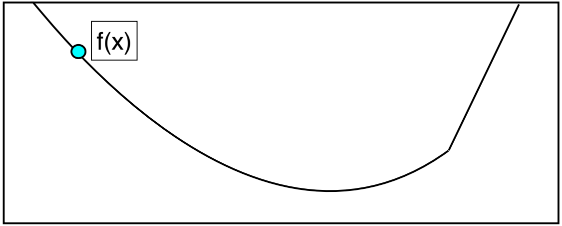
# Sub-Gradients and Sub-Differentials

*Differentiable* convex functions are always above tangent,

$$f(v) \geq f(w) + \nabla f(w)^T (v - w), \forall w, v.$$

*A vector $d$ is a subgradient of a convex function $f$ at $w$ if*

$$f(v) \geq f(w) + d^T (v - w), \forall v.$$



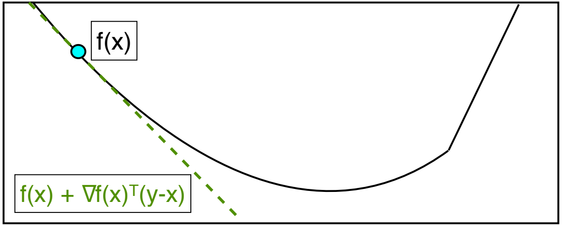
# Sub-Gradients and Sub-Differentials

*Differentiable* convex functions are always above tangent,

$$f(v) \geq f(w) + \nabla f(w)^T (v - w), \forall w, v.$$

*A vector $d$ is a subgradient of a convex function $f$ at $w$ if*

$$f(v) \geq f(w) + d^T (v - w), \forall v.$$



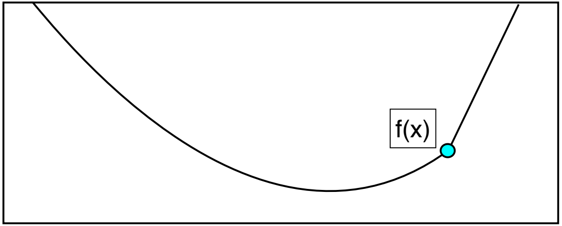
# Sub-Gradients and Sub-Differentials

*Differentiable* convex functions are always above tangent,

$$f(v) \geq f(w) + \nabla f(w)^T (v - w), \forall w, v.$$

*A vector $d$ is a subgradient of a convex function $f$ at $w$ if*

$$f(v) \geq f(w) + d^T (v - w), \forall v.$$



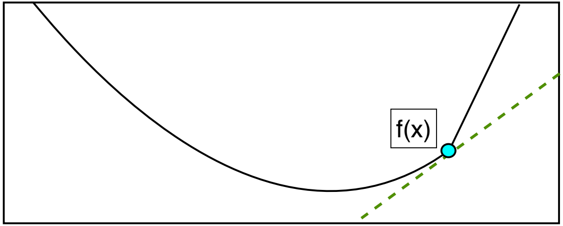
# Sub-Gradients and Sub-Differentials

*Differentiable* convex functions are always above tangent,

$$f(v) \geq f(w) + \nabla f(w)^T (v - w), \forall w, v.$$

*A vector $d$ is a subgradient of a convex function $f$ at $w$ if*

$$f(v) \geq f(w) + d^T (v - w), \forall v.$$



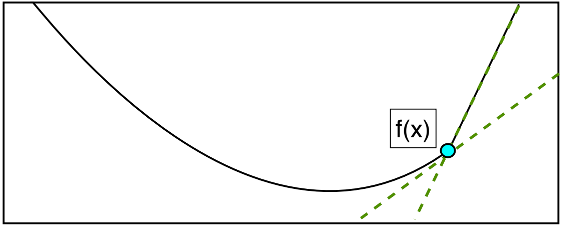
# Sub-Gradients and Sub-Differentials

*Differentiable* convex functions are always above tangent,

$$f(v) \geq f(w) + \nabla f(w)^T (v - w), \forall w, v.$$

*A vector $d$ is a subgradient of a convex function $f$ at $w$ if*

$$f(v) \geq f(w) + d^T (v - w), \forall v.$$



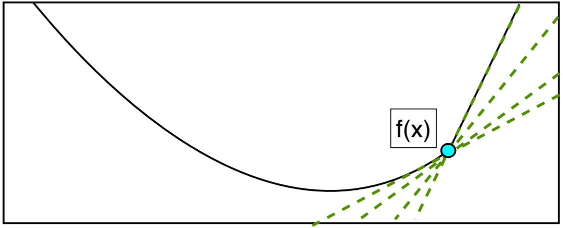
# Sub-Gradients and Sub-Differentials

*Differentiable* convex functions are always above tangent,

$$f(v) \geq f(w) + \nabla f(w)^T(v - w), \forall w, v.$$

*A vector $d$ is a subgradient of a convex function $f$ at $w$ if*

$$f(v) \geq f(w) + d^T(v - w), \forall v.$$



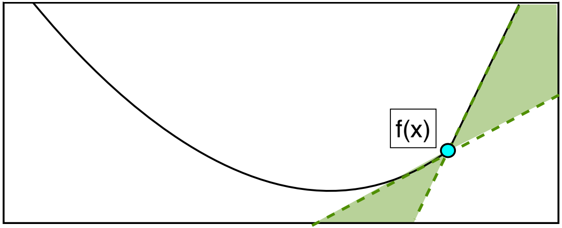
# Sub-Gradients and Sub-Differentials

*Differentiable* convex functions are always above tangent,

$$f(v) \geq f(w) + \nabla f(w)^T (v - w), \forall w, v.$$

*A vector $d$ is a subgradient of a convex function $f$ at $w$ if*

$$f(v) \geq f(w) + d^T (v - w), \forall v.$$



# Sub-Gradients and Sub-Differentials Properties

- We can have a set of subgradients called the sub-differential, $\partial f(w)$.
  - "Subdifferential is all the possible 'tangent' lines".
- For convex functions:
  - Sub-differential is always non-empty (except some weird degenerate cases).
  - At differentiable $w$, the only subgradient is the gradient.
  - At non-differentiable $w$, there will be a convex set of subgradients.
  - We have $0 \in \partial f(w)$ iff $w$ is a global minimum.
    - This generalizes the condition that $\nabla f(w) = 0$ for differentiable functions.
- For non-convex functions:
  - "Global" subgradients may not exist for every $w$.
  - Instead, we define subgradients "locally" around current $w$.
    - This is how you define "gradient" of ReLU function in neural networks.

# Example: Sub-Differential of Absolute Function

- Sub-differential of absolute value function:

$$\partial|w| = \begin{cases} 1 & w > 0 \\ -1 & w < 0 \\ [-1, 1] & w = 0 \end{cases}$$

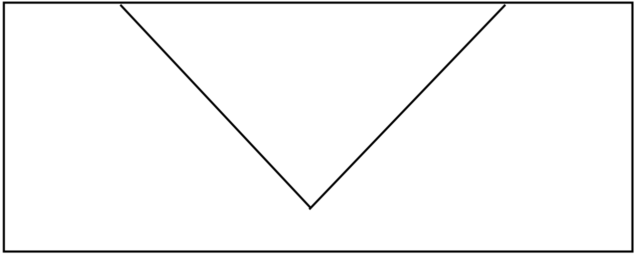
  - "Sign of the variable if it's non-zero, anything in $[-1, 1]$ if it's zero."

# Example: Sub-Differential of Absolute Function

- Sub-differential of absolute value function:

$$\partial|w| = \begin{cases} 1 & w > 0 \\ -1 & w < 0 \\ [-1,1] & w = 0 \end{cases}$$

  - "Sign of the variable if it's non-zero, anything in $[-1,1]$ if it's zero."

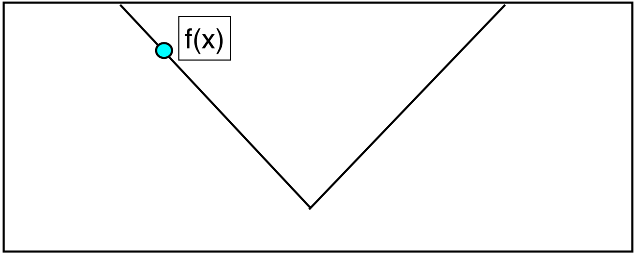


# Example: Sub-Differential of Absolute Function

- Sub-differential of absolute value function:

$$\partial|w| = \begin{cases} 1 & w > 0 \\ -1 & w < 0 \\ [-1, 1] & w = 0 \end{cases}$$

  - "Sign of the variable if it's non-zero, anything in $[-1, 1]$ if it's zero."

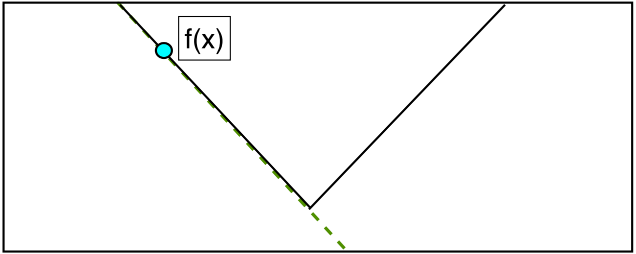


# Example: Sub-Differential of Absolute Function

- Sub-differential of absolute value function:

$$\partial|w| = \begin{cases} 1 & w > 0 \\ -1 & w < 0 \\ [-1, 1] & w = 0 \end{cases}$$

  - "Sign of the variable if it's non-zero, anything in $[-1, 1]$ if it's zero."

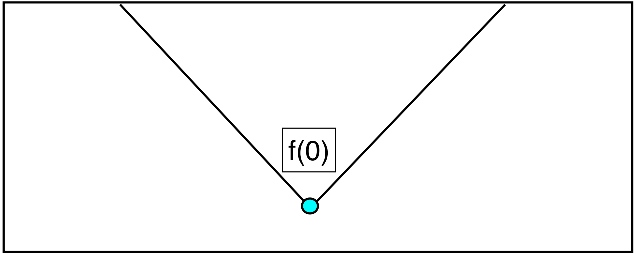


# Example: Sub-Differential of Absolute Function

- Sub-differential of absolute value function:

$$\partial|w| = \begin{cases} 1 & w > 0 \\ -1 & w < 0 \\ [-1, 1] & w = 0 \end{cases}$$

  - "Sign of the variable if it's non-zero, anything in $[-1, 1]$ if it's zero."

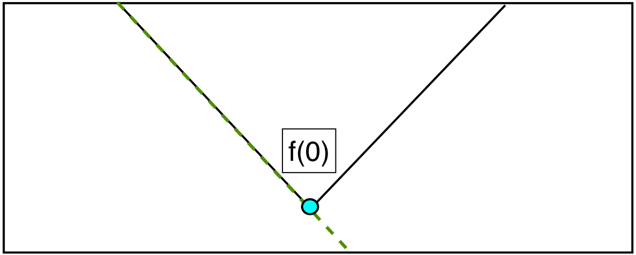


# Example: Sub-Differential of Absolute Function

- Sub-differential of absolute value function:

$$\partial|w| = \begin{cases} 1 & w > 0 \\ -1 & w < 0 \\ [-1,1] & w = 0 \end{cases}$$

  - "Sign of the variable if it's non-zero, anything in $[-1,1]$ if it's zero."

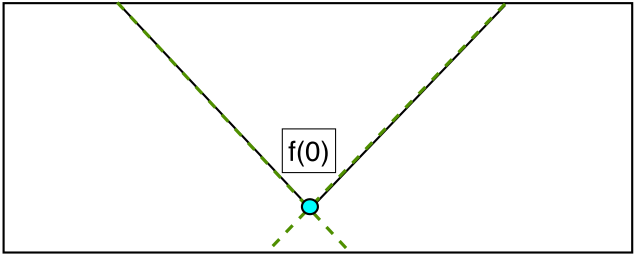


# Example: Sub-Differential of Absolute Function

- Sub-differential of absolute value function:

$$\partial|w| = \begin{cases} 1 & w > 0 \\ -1 & w < 0 \\ [-1,1] & w = 0 \end{cases}$$

  - "Sign of the variable if it's non-zero, anything in $[-1, 1]$ if it's zero."

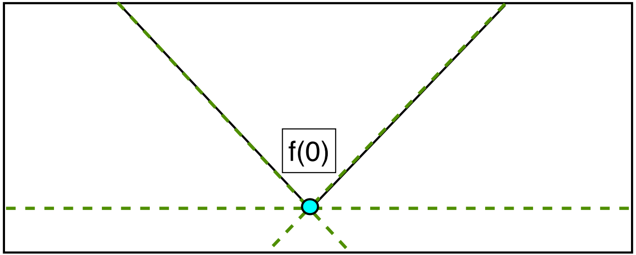


# Example: Sub-Differential of Absolute Function

- Sub-differential of absolute value function:

$$\partial|w| = \begin{cases} 1 & w > 0 \\ -1 & w < 0 \\ [-1, 1] & w = 0 \end{cases}$$

  - "Sign of the variable if it's non-zero, anything in $[-1, 1]$ if it's zero."

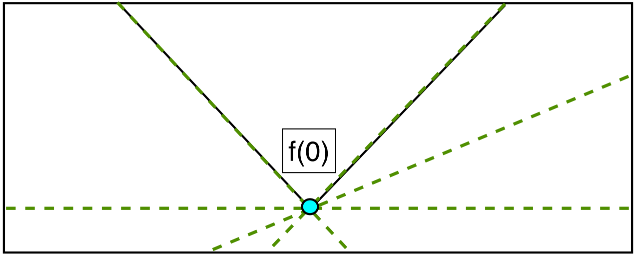


# Example: Sub-Differential of Absolute Function

- Sub-differential of absolute value function:

$$\partial|w| = \begin{cases} 1 & w > 0 \\ -1 & w < 0 \\ [-1,1] & w = 0 \end{cases}$$

  - "Sign of the variable if it's non-zero, anything in $[-1,1]$ if it's zero."

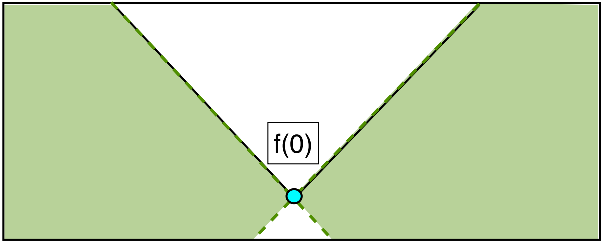


# Example: Sub-Differential of Absolute Function

- Sub-differential of absolute value function:

$$\partial|w| = \begin{cases} 1 & w > 0 \\ -1 & w < 0 \\ [-1, 1] & w = 0 \end{cases}$$

  - "Sign of the variable if it's non-zero, anything in $[-1, 1]$ if it's zero."



# Sub-Differential of Common Operations

- Two convenient rules for calculating subgradients of convex functions:
  - Sub-differential of max is all convex combinations of argmax gradients:

$$\partial \max\{f_1(x), f_2(x)\} = \begin{cases} \nabla f_1(x) & f_1(x) > f_2(x) \\ \nabla f_2(x) & f_2(x) > f_1(x) \\ \underbrace{\theta \nabla f_1(x) + (1-\theta)\nabla f_2(x)}_{\text{for all } 0 \leq \theta \leq 1} & f_1(x) = f_2(x) \end{cases}$$

    - This rules gives sub-differential of absolute value, using that $|\alpha| = \max\{\alpha, -\alpha\}$.
  - Sub-differential of sum is all sum of subgradients of individual functions:

$$\partial(f_1(x) + f_2(x)) = d_1 + d_2 \quad \text{for any} \quad d_1 \in \partial f_1(x), d_2 \in \partial f_2(x).$$

  - Sub-differential of composition with affine function works like the chain rule:

$$\partial f_1(Aw) = A^T \partial f_1(z), \quad \text{where} \quad z = Aw,$$

    and we also have $\partial \alpha f(w) = \alpha \partial f(w)$ for $\alpha > 0$.

# Why does L1-Regularization but not L2-Regularization give Sparsity?

- Consider L2-regularized least squares,

$$f(w) = \frac{1}{2}\|Xw - y\|^2 + \frac{\lambda}{2}\|w\|^2.$$

- Element $j$ of the gradient at $w_j = 0$ is given by

$$\nabla_j f(w) = x_j^T \underbrace{(Xw - y)}_{r} + \lambda 0.$$

- For $w_j = 0$ to be a solution, we need $0 = \nabla_j f(w)$ or that

$$x_j^T r = 0,$$

  that column $j$ is orthogonal to the final residual.
  - This is possible, but it is very unlikely (probability 0 for random data).
  - Increasing $\lambda$ doesn't help.

# Why does L1-Regularization but not L2-Regularization give Sparsity?

- Consider L1-regularized least squares,

$$f(w) = \frac{1}{2}\|Xw - y\|^2 + \frac{\lambda}{2}\|w\|_1.$$

- Element $j$ of the subdifferential at $w_j = 0$ is given by

$$\partial_j f(w) \equiv x_j^T \underbrace{(Xw - y)}_{r} + \lambda \underbrace{[-1, 1]}_{\partial|w_j|}.$$

- For $w_j = 0$ to be a solution, we need $0 \in \partial_j f(w)$ or that

$$|x_j^T r| \leq \lambda,$$

that column $j$ is "close to" orthogonal to the final residual.
  - So features $j$ that have little to do with $y$ will often lead to $w_j = 0$.
  - Increasing $\lambda$ makes this more likely to happen.

# Outline

# Solving L1-Regularization Problems

- How can we minimize non-smooth L1-regularized objectives?

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

- Use our trick to formulate as a quadratic program?
  - $O(d^2)$ or worse.
- Make a smooth approximation to the L1-norm?
  - Destroys sparsity (we'll again just have one subgradient at zero).

- Use a subgradient method?

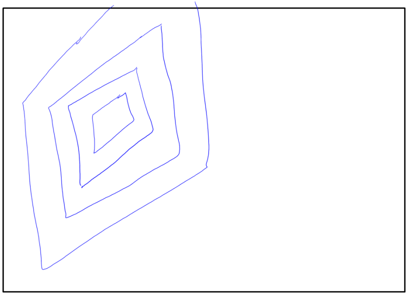
# Subgradient Method

- The basic subgradient method:

$$w^{k+1} = w^k - \alpha_k g_k,$$

for some $g_k \in \partial f(w^k)$.

- This can increase the objective even for small $\alpha_k$.
  - Though for convex $f$ the distance to solutions decreases:
    - $\|w^{k+1} - w^*\| < \|w^k - w^*\|$ for small enough $\alpha_k$.



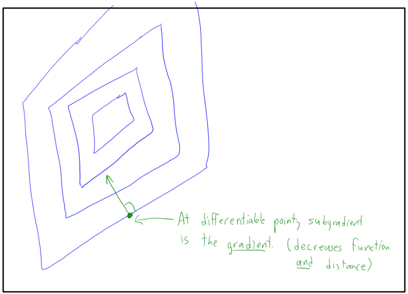
# Subgradient Method

- The basic subgradient method:

$$w^{k+1} = w^k - \alpha_k g_k,$$

for some $g_k \in \partial f(w^k)$.

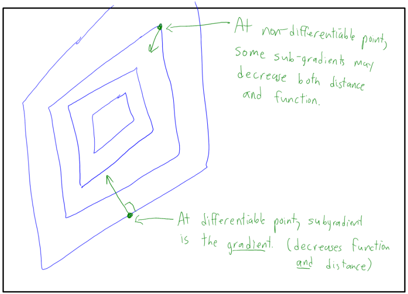
- This can increase the objective even for small $\alpha_k$.
  - Though for convex $f$ the distance to solutions decreases:
    - $\|w^{k+1} - w^*\| < \|w^k - w^*\|$ for small enough $\alpha_k$.



# Subgradient Method

- The basic subgradient method:
$$w^{k+1} = w^k - \alpha_k g_k,$$
for some $g_k \in \partial f(w^k)$.

- This can increase the objective even for small $\alpha_k$.
    - Though for convex $f$ the distance to solutions decreases:
        - $\|w^{k+1} - w^*\| < \|w^k - w^*\|$ for small enough $\alpha_k$.



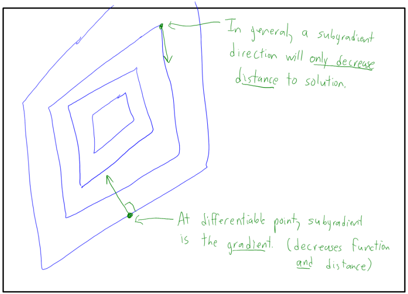
# Subgradient Method

- The basic subgradient method:

$$w^{k+1} = w^k - \alpha_k g_k,$$

  for some $g_k \in \partial f(w^k)$.

- This can increase the objective even for small $\alpha_k$.
  - Though for convex $f$ the distance to solutions decreases:
    - $\|w^{k+1} - w^*\| < \|w^k - w^*\|$ for small enough $\alpha_k$.



# Subgradient Method

- The basic subgradient method:

$$w^{k+1} = w^k - \alpha_k g_k,$$

for some $g_k \in \partial f(w^k)$.

- This can increase the objective even for small $\alpha_k$.
  - Though for convex $f$ the distance to solutions decreases:
    - $\|w^{k+1} - w^*\| < \|w^k - w^*\|$ for small enough $\alpha_k$.
- The subgradients $g_k$ don't necessarily converge to 0 as we approach a $w^*$.
  - If we are at a solution $w^*$, we might move away from it.
  - So as in stochastic gradient, we need decreasing step-sizes like

$$\alpha_k = O(1/k), \quad \text{or} \quad \alpha_k = O(1/\sqrt{k}) \text{ (and averaging the } w^k\text{)},$$

in order to converge.
    - This destroys performance.

# Convergence Rate of Subgradient Methods

- Subgradient methods are slower than gradient descent:

| Assumption | Gradient | Subgradient | Quantity |
|---|---|---|---|
| Convex | $O(1/\epsilon)$ | $O(1/\epsilon^2)$ | $f(w^t) - f^* \leq \epsilon$ |
| Strongly-Convex | $O(\log(1/\epsilon))$ | $O(1/\epsilon)$ | $f(w^t) - f^* \leq \epsilon$ |

- Other subgradient-based methods are not faster.
  - There are matching lower bounds in dimension-independent setting.
  - Includes cutting plane and bundle methods.
- In particular, acceleration doesn't improve subgradient rates.
  - We do NOT go from $O(1/\epsilon^2)$ to $O(1/\epsilon)$ by adding momentum.
- Smoothing $f$ and applying gradient descent doesn't help.
  - May need to have $L = 1/\epsilon$ in a sufficiently-accurate smooth approximation.
  - However, if you smooth and accelerate you can close the gaps a bit (bonus).

# The Key to Faster Methods

- How can we achieve the speed of gradient descent on non-smooth problems?
  - Make extra assumptions about the function/algorithm $f$.

- For L1-regularized least squares, we'll use that the objective has the form

$$F(w) = \underbrace{f(w)}_{\text{smooth}} + \underbrace{r(w)}_{\text{``simple''}},$$

  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.

- Proximal-gradient methods have rates of gradient descent for such problems.
  - 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,

$$\underset{w \geq 0}{\text{argmin}}\, f(w).$$

- In this case the algorithm has a simple form,

$$w^{k+1} = \max\{0, \underbrace{w^k - \alpha_k \nabla f(w^k)}_{\text{gradient descent}}\},$$

  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.

# A Broken "Projected-Gradient" Algorithms

- Projected-gradient addresses problem of minimizing smooth $f$ over a convex set $\mathcal{C}$,

$$\mathop{\text{argmin}}_{w \in \mathcal{C}} f(w).$$

- As another example, we often want $w$ to be a probability,

$$\mathop{\text{argmin}}_{w \geq 0,\, 1^T w = 1} f(w),$$

- Based on our "set negative values to 0" intuition, we might consider this algorithm:
  1. Perform an unconstrained gradient descent step.
  2. Set negative values to 0 and divide by the sum.

- This algorithms does NOT work.
  - But it can be fixed if we use the projection onto the set in Step 2...

# Projected-Gradient

- We can view the projected-gradient algorithm as having two steps:
  1. Perform an unconstrained gradient descent step,

$$w^{k+\frac{1}{2}} = w^k - \alpha_k \nabla f(w^k).$$

  2. Compute the projection onto the set $\mathcal{C}$,

$$w^{k+1} \in \operatorname*{argmin}_{v \in \mathcal{C}} \|v - w^{k+\frac{1}{2}}\|.$$

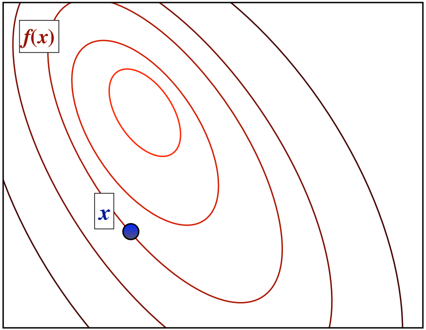
- Projection is the closest point that satisfies the constraints.
  - Generalizes "projection onto subspace" from linear algebra.
  - We'll also write projection of $w$ onto $\mathcal{C}$ as

$$\text{proj}_{\mathcal{C}}[w] = \operatorname*{argmin}_{v \in \mathcal{C}} \|v - w\|,$$

and for convex $\mathcal{C}$ it's unique.

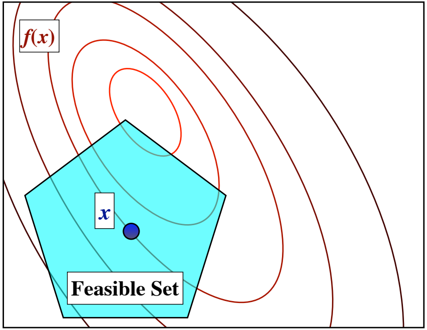
# Projected-Gradient

$$w^{k+\frac{1}{2}} = \underbrace{w^k - \alpha_k \nabla f(w^k)}_{\text{gradient step}}, \quad w^{k+1} \in \underbrace{\operatorname*{argmin}_{v \in \mathcal{C}} \left\| v - w^{k+\frac{1}{2}} \right\|}_{\text{projection step}}.$$



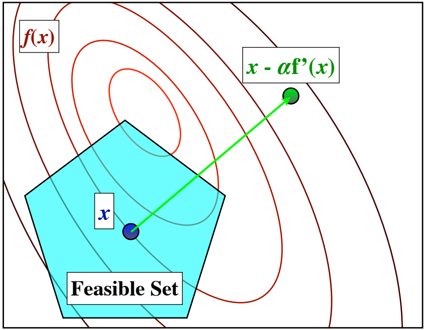
# Projected-Gradient

$$w^{k+\frac{1}{2}} = \underbrace{w^k - \alpha_k \nabla f(w^k)}_{\text{gradient step}}, \quad w^{k+1} \in \underbrace{\operatorname*{argmin}_{v \in \mathcal{C}} \|v - w^{k+\frac{1}{2}}\|}_{\text{projection step}}.$$



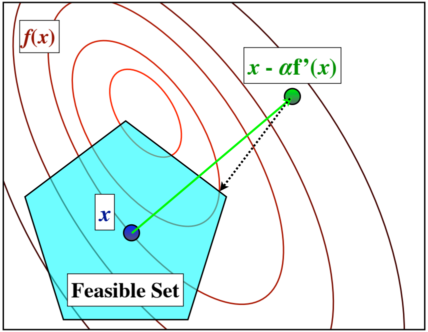
# Projected-Gradient

$$w^{k+\frac{1}{2}} = \underbrace{w^k - \alpha_k \nabla f(w^k)}_{\text{gradient step}}, \quad w^{k+1} \in \underbrace{\operatorname*{argmin}_{v \in \mathcal{C}} \|v - w^{k+\frac{1}{2}}\|}_{\text{projection step}}.$$



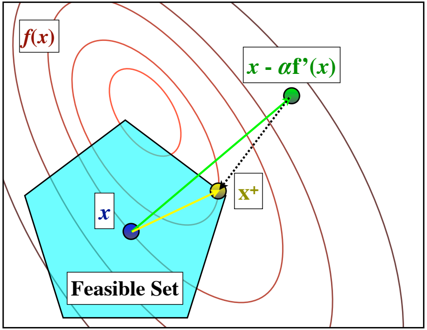
# Projected-Gradient

$$w^{k+\frac{1}{2}} = \underbrace{w^k - \alpha_k \nabla f(w^k)}_{\text{gradient step}}, \quad w^{k+1} \in \underbrace{\operatorname*{argmin}_{v \in \mathcal{C}} \|v - w^{k+\frac{1}{2}}\|}_{\text{projection step}}.$$



# Projected-Gradient

$$w^{k+\frac{1}{2}} = \underbrace{w^k - \alpha_k \nabla f(w^k)}_{\text{gradient step}}, \quad w^{k+1} \in \underbrace{\operatorname*{argmin}_{v \in \mathcal{C}} \|v - w^{k+\frac{1}{2}}\|}_{\text{projection step}}.$$



# Convergence Rate of Projected Gradient

- Projected versions have same complexity as unconstrained versions:

| Assumption | Proj(Grad) | Proj(Subgrad) | Quantity |
|---|---|---|---|
| Convex | $O(1/\epsilon)$ | $O(1/\epsilon^2)$ | $f(w^t) - f^* \leq \epsilon$ |
| Strongly-Convex | $O(\log(1/\epsilon))$ | $O(1/\epsilon)$ | $f(w^t) - f^* \leq \epsilon$ |

- Nice properties in the smooth case:
  - With $\alpha_t < 2/L$, guaranteed to decrease objective.
  - There exist practical step-size strategies as with gradient descent (bonus).
  - For convex $f$ a $w^*$ is optimal iff it's a "fixed point" of the update,

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

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

- There exist accelerated versions and Newton-like versions (bonus slides).
  - Acceleration is an obvious modification, Newton is more complicated.

# Summary

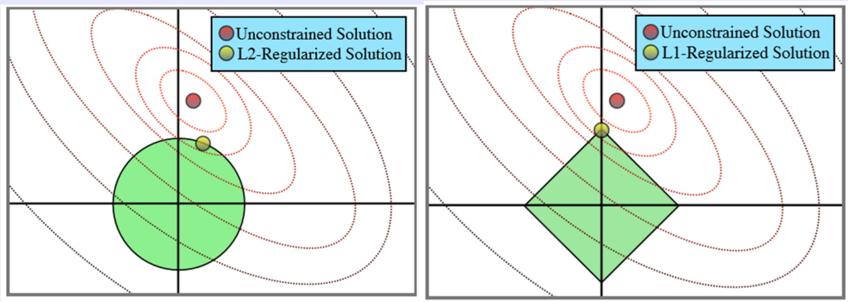
- L1-regularization: feature selection as convex optimization.
- Subgradients: generalize gradients for non-smooth convex functions.
- Subgradient method: optimal but very-slow general non-smooth method.
- Projected-gradient allows optimization with simple constraints.

- Next time: going beyond L1-regularization to “structured sparsity”.

# L1-Regularization vs. L2-Regularization

- Another view on sparsity of L2- vs. L1-regularization using our constraint trick:

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




- Notice that L2-regularization has a rotataional invariance.
  - This actually makes it more sensitive to irrelevant features.

# Does Smoothing Help?

- Nesterov's smoothing paper gives a way to take a non-smooth convex $f$ and number $\epsilon$, then it constructs a new function $f_\epsilon$ such that

$$f(w) \leq f_\epsilon(w) \leq f(w) + \epsilon,$$

  so that minimizing $f_\epsilon(w)$ gets us within $\epsilon$ of the optimal solution.
  - And further that $f_\epsilon(w)$ is differentiable with $L = O(1/\epsilon)$.

- If we apply gradient descent to the smooth function, we get

$$t = \underbrace{O(L/\epsilon)}_{\text{smoothed problem}} = \underbrace{O(1/\epsilon^2)}_{\text{original problem}},$$

  for convex functions (same speed as subgradient).
- For strongly-convex functions we get

$$t = O(L \log(1/\epsilon)) = O((1/\epsilon) \log(1/\epsilon)),$$

  which is actually worse than the best subgradient methods by a log factor.

# Does Smoothing Help?

- Nesterov's smoothing paper gives a way to take a non-smooth convex $f$ and number $\epsilon$, then it constructs a new function $f_\epsilon$ such that

$$f(w) \leq f_\epsilon(w) \leq f(w) + \epsilon,$$

so that minimizing $f_\epsilon(w)$ gets us within $\epsilon$ of the optimal solution.
    - And further that $f_\epsilon(w)$ is differentiable with $L = O(1/\epsilon)$.

- If we apply accelerated gradient descent to the smooth function, we get

$$t = O(\sqrt{L/\epsilon}) = O(1/\epsilon),$$

which is faster than subgradient methods.
(same speed as unaccelerated gradient descent)

- For strongly-convex functions the accelerated method gets

$$t = O(\sqrt{L}\log(1/\epsilon)) = O((1/\sqrt{\epsilon})\log(1/\epsilon)),$$

which is faster than subgradient methods (but not linear converence).

# What is the best subgradient?

- We considered the deterministic subgradient method,

$$x^{t+1} = x^t - \alpha_t g_t, \text{ where } g_t \in \partial f(x^t),$$

under any choice of subgradient.

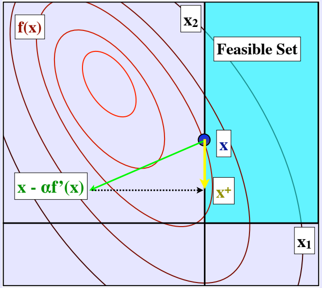
- But what is the "best" subgradient to use?
    - Convex functions have directional derivatives everywhere.
    - Direction $-g_t$ that minimizes directional derivative is minimum-norm subgradient,

$$g^t = \operatorname*{argmin}_{g \in \partial f(x^t)} ||g||$$

    - This is the steepest descent direction for non-smooth convex optimization problems.
    - You can compute this for L1-regularization, but not many other problems.
    - Used in best deterministic L1-regularization methods, combined with Newton.

# Line-Search for Projected Gradient

- There are two ways to do line-search for this algorithm:
  - Backtrack along the line between $x^+$ and $x$ (search interior).
    - "Backtracking along the feasible direction", costs 1 projection per iteration.




  - Backtrack by decreasing $\alpha$ and re-projecting (search boundary).
    - "Backtracking along the projection arc", costs 1 projection per backtrack.
    - More expensive but (under weak conditions) we reach boundary in finite time.

# 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.

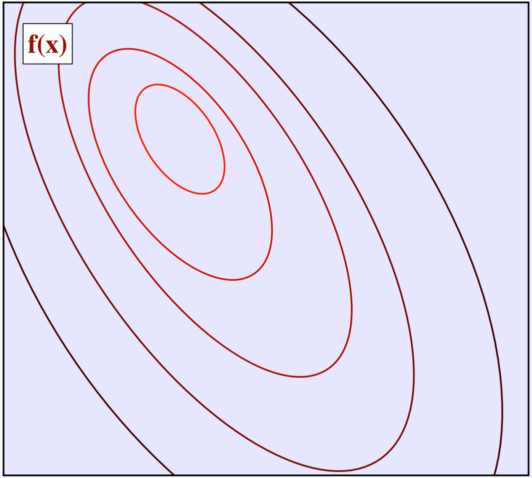
- The naive Newton-like methods with Hessian approximation $H_t$,

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

does NOT work.

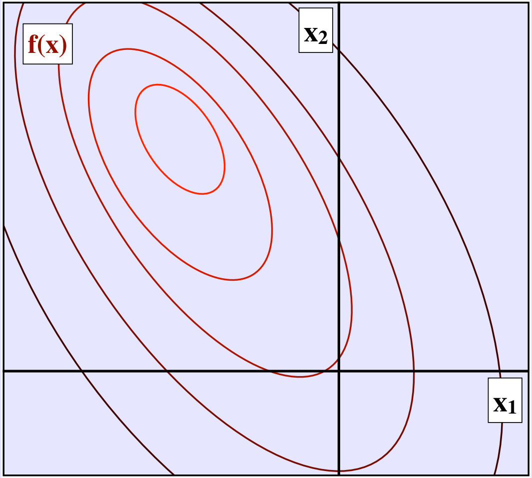
# Naive Projected-Newton

Naive projected Newton method can point in the wrong direction.



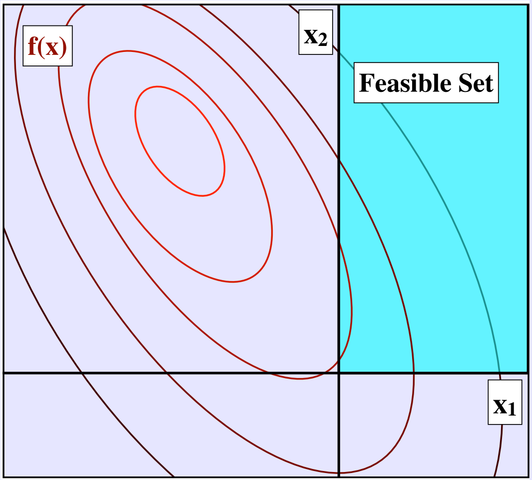
# Naive Projected-Newton

Naive projected Newton method can point in the wrong direction.



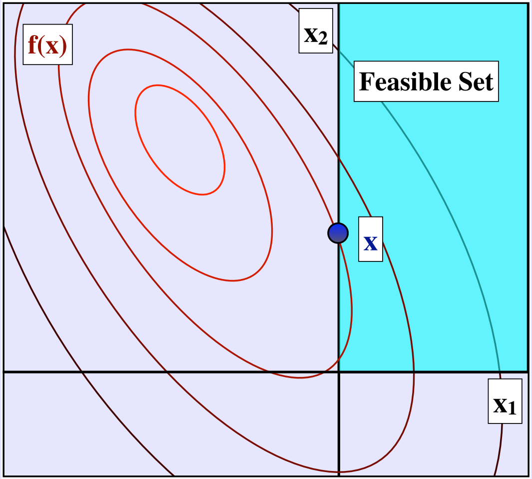
# Naive Projected-Newton

Naive projected Newton method can point in the wrong direction.



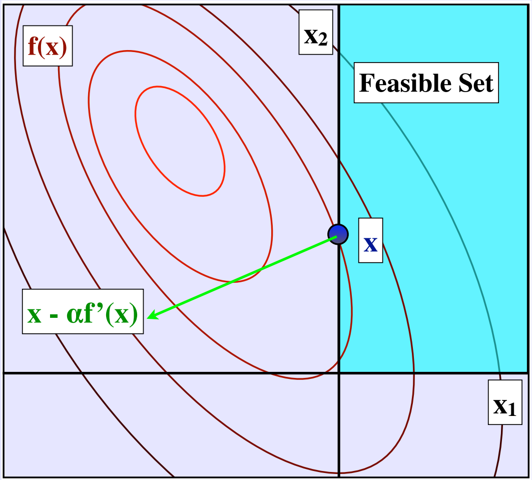
# Naive Projected-Newton

Naive projected Newton method can point in the wrong direction.



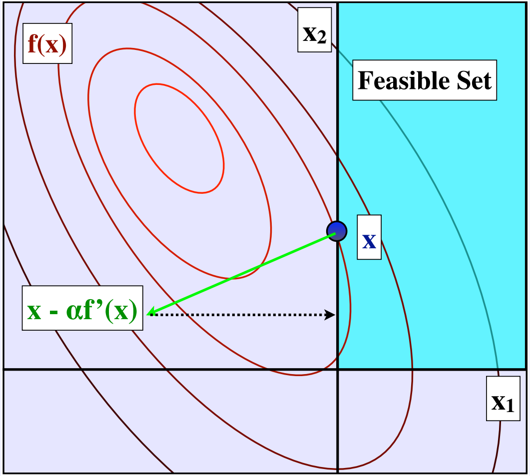
# Naive Projected-Newton

Naive projected Newton method can point in the wrong direction.



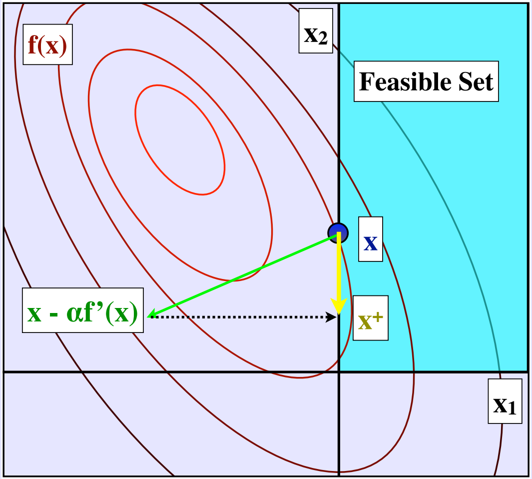
# Naive Projected-Newton

Naive projected Newton method can point in the wrong direction.



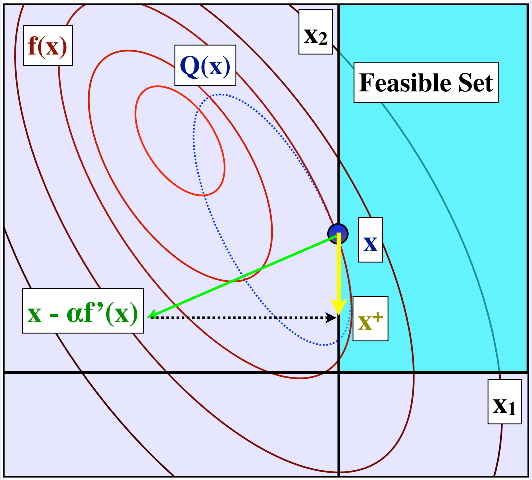
# Naive Projected-Newton

Naive projected Newton method can point in the wrong direction.



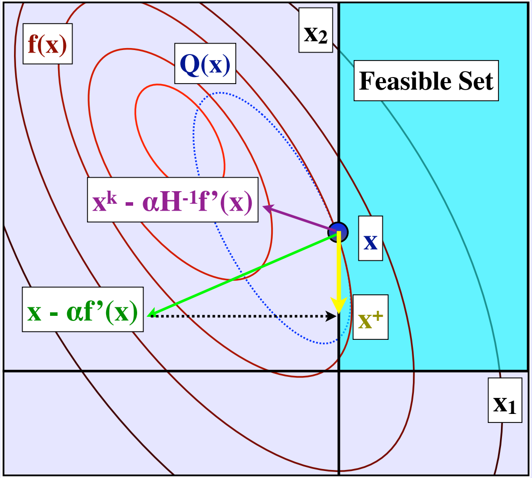
# Naive Projected-Newton

Naive projected Newton method can point in the wrong direction.



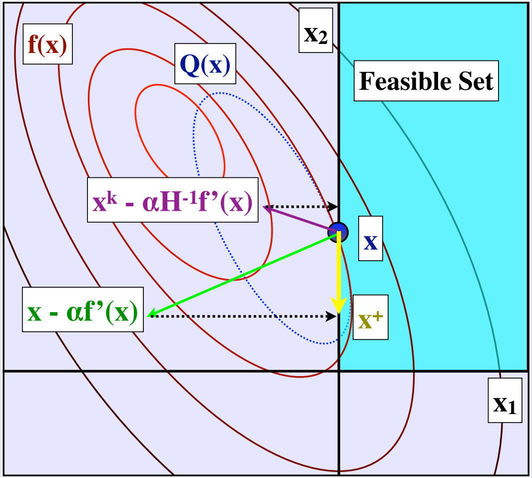
# Naive Projected-Newton

Naive projected Newton method can point in the wrong direction.



# Naive Projected-Newton

Naive projected Newton method can point in the wrong direction.



# Naive Projected-Newton

Naive projected Newton method can point in the wrong direction.