

# Convex Optimization

## Machine Learning Summer School

Mark Schmidt

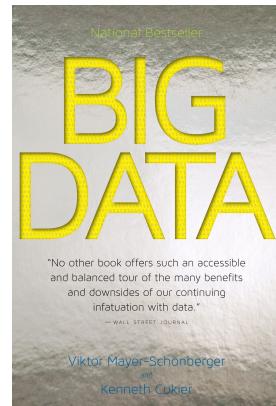
February 2015

## Context: Big Data and Big Models

- We are collecting data at unprecedented rates.
  - Seen across many fields of science and engineering.
  - Not gigabytes, but terabytes or petabytes (and beyond).

## Context: Big Data and Big Models

- We are collecting data at unprecedented rates.
  - Seen across many fields of science and engineering.
  - Not gigabytes, but terabytes or petabytes (and beyond).



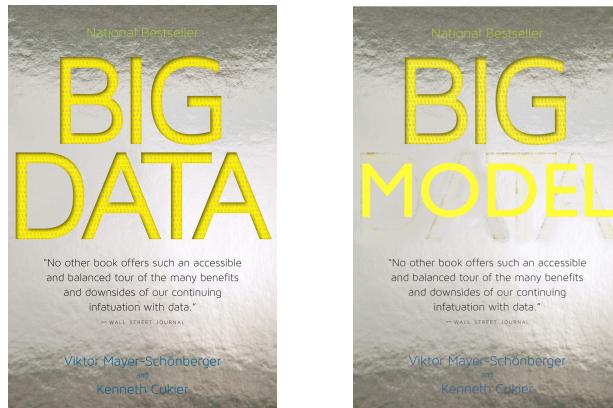
- Many important aspects to the 'big data' puzzle:
  - Distributed data storage and management, parallel computation, software paradigms, data mining, **machine learning**, privacy and security issues, reacting to other agents, power management, summarization and visualization.

## Context: Big Data and Big Models

- Machine learning uses big data to fit richer statistical models:
  - Vision, bioinformatics, speech, natural language, web, social.
  - Developping broadly applicable tools.
  - Output of models can be used for further analysis.

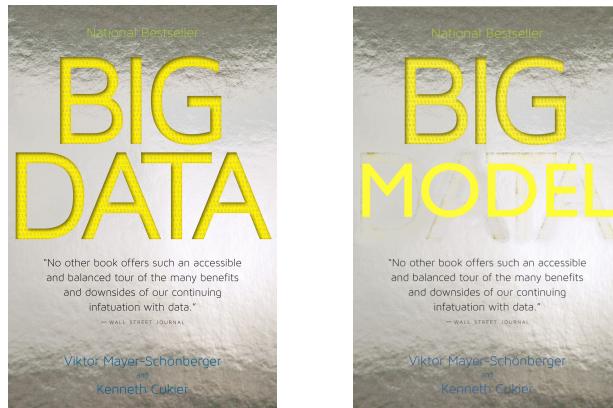
## Context: Big Data and Big Models

- Machine learning uses big data to fit richer statistical models:
  - Vision, bioinformatics, speech, natural language, web, social.
  - Developping broadly applicable tools.
  - Output of models can be used for further analysis.



## Context: Big Data and Big Models

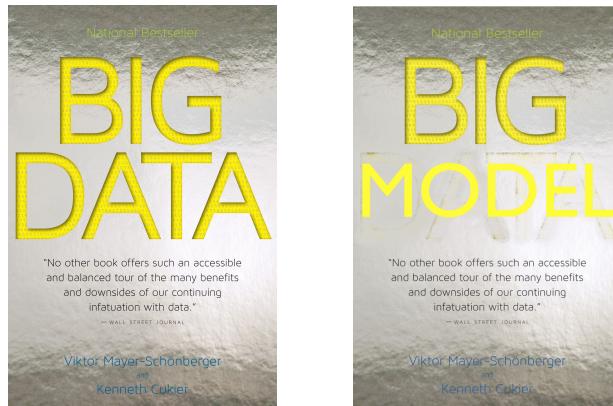
- Machine learning uses big data to fit richer statistical models:
  - Vision, bioinformatics, speech, natural language, web, social.
  - Developping broadly applicable tools.
  - Output of models can be used for further analysis.



- Numerical optimization is at the core of many of these models.

## Context: Big Data and Big Models

- Machine learning uses big data to fit richer statistical models:
  - Vision, bioinformatics, speech, natural language, web, social.
  - Developping broadly applicable tools.
  - Output of models can be used for further analysis.



- Numerical optimization is at the core of many of these models.
- But, traditional ‘black-box’ methods have difficulty with:
  - the **large data sizes**.
  - the **large model complexities**.

## Motivation: Why Learn about Convex Optimization?

Why learn about **optimization**?

- Optimization is at the core of many ML algorithms.
- ML is driving a lot of modern research in optimization.

## Motivation: Why Learn about Convex Optimization?

Why learn about **optimization**?

- Optimization is at the core of many ML algorithms.
- ML is driving a lot of modern research in optimization.

Why in particular learn about **convex** optimization?

- Among only *efficiently-solvable* continuous problems.
  - (least squares, lasso, generalized linear models, SVMs, CRFs)
- Empirically effective non-convex methods are often based on methods with good properties for convex objectives.
  - (functions are locally convex around minimizers)

## Two Components of My Research

- The first component of my research focuses on **computation**:
  - We ‘open up the black box’, by using the structure of machine models to derive faster large-scale optimization algorithms.
  - Can lead to enormous speedups for big data and complex models.

## Two Components of My Research

- The first component of my research focuses on **computation**:
  - We ‘open up the black box’, by using the structure of machine models to derive faster large-scale optimization algorithms.
  - Can lead to enormous speedups for big data and complex models.
- The second component of my research focuses on **modeling**:
  - By expanding the set of tractable problems, we can propose richer classes of statistical models that can be efficiently fit.

## Two Components of My Research

- The first component of my research focuses on **computation**:
  - We ‘open up the black box’, by using the structure of machine models to derive faster large-scale optimization algorithms.
  - Can lead to enormous speedups for big data and complex models.
- The second component of my research focuses on **modeling**:
  - By expanding the set of tractable problems, we can propose richer classes of statistical models that can be efficiently fit.
- We can **alternate between these two**.

## Outline

- 1 Convex Functions
- 2 Smooth Optimization
- 3 Non-Smooth Optimization
- 4 Randomized Algorithms
- 5 Parallel/Distributed Optimization

## Convexity: Zero-order condition

A real-valued function is **convex** if

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y),$$

for all  $x, y \in \mathbb{R}^n$  and all  $0 \leq \theta \leq 1$ .

- Function is *below a linear interpolation* from  $x$  to  $y$ .
- Implies that all local minima are global minima.

(contradiction otherwise)

## Convexity: Zero-order condition

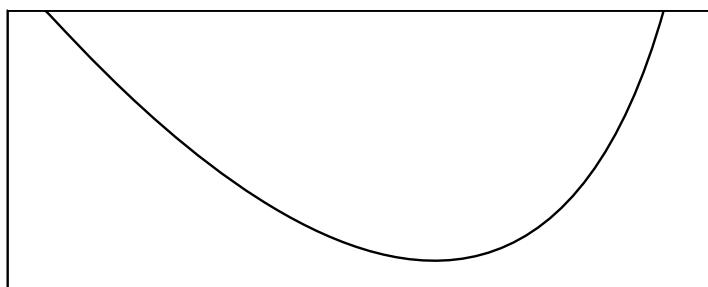
A real-valued function is **convex** if

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y),$$

for all  $x, y \in \mathbb{R}^n$  and all  $0 \leq \theta \leq 1$ .

- Function is *below a linear interpolation* from  $x$  to  $y$ .
- Implies that all local minima are global minima.

(contradiction otherwise)



## Convexity: Zero-order condition

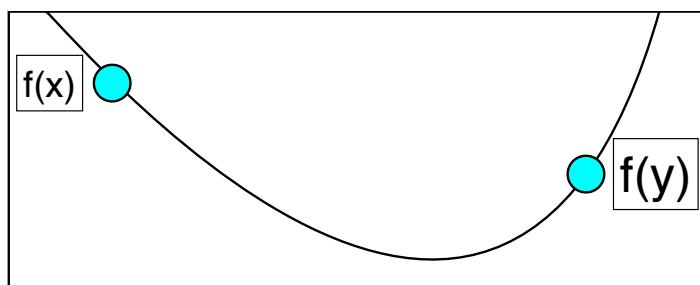
A real-valued function is *convex* if

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y),$$

for all  $x, y \in \mathbb{R}^n$  and all  $0 \leq \theta \leq 1$ .

- Function is *below a linear interpolation* from  $x$  to  $y$ .
- Implies that all local minima are global minima.

(contradiction otherwise)



## Convexity: Zero-order condition

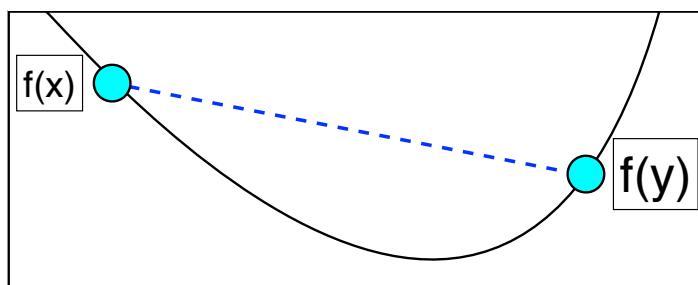
A real-valued function is *convex* if

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y),$$

for all  $x, y \in \mathbb{R}^n$  and all  $0 \leq \theta \leq 1$ .

- Function is *below a linear interpolation* from  $x$  to  $y$ .
- Implies that all local minima are global minima.

(contradiction otherwise)



## Convexity: Zero-order condition

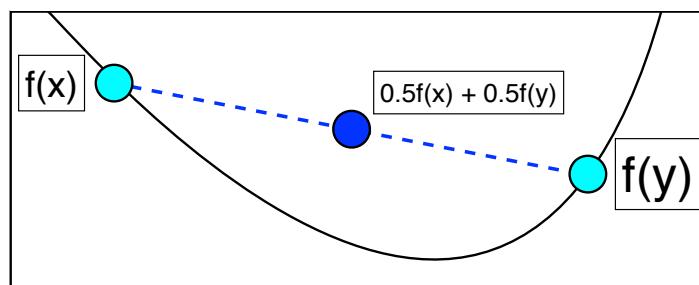
A real-valued function is *convex* if

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y),$$

for all  $x, y \in \mathbb{R}^n$  and all  $0 \leq \theta \leq 1$ .

- Function is *below a linear interpolation* from  $x$  to  $y$ .
- Implies that all local minima are global minima.

(contradiction otherwise)



## Convexity: Zero-order condition

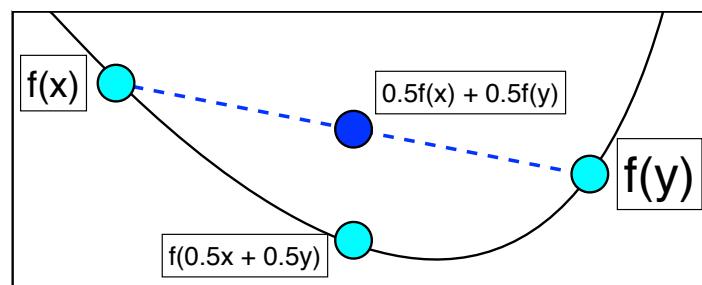
A real-valued function is *convex* if

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y),$$

for all  $x, y \in \mathbb{R}^n$  and all  $0 \leq \theta \leq 1$ .

- Function is *below a linear interpolation* from  $x$  to  $y$ .
- Implies that all local minima are global minima.

(contradiction otherwise)



## Convexity: Zero-order condition

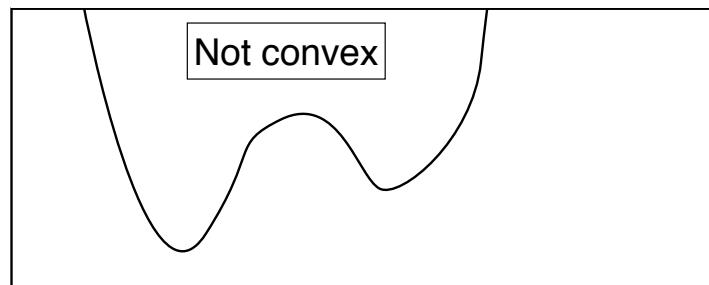
A real-valued function is **convex** if

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y),$$

for all  $x, y \in \mathbb{R}^n$  and all  $0 \leq \theta \leq 1$ .

- Function is *below a linear interpolation* from  $x$  to  $y$ .
- Implies that all local minima are global minima.

(contradiction otherwise)



## Convexity: Zero-order condition

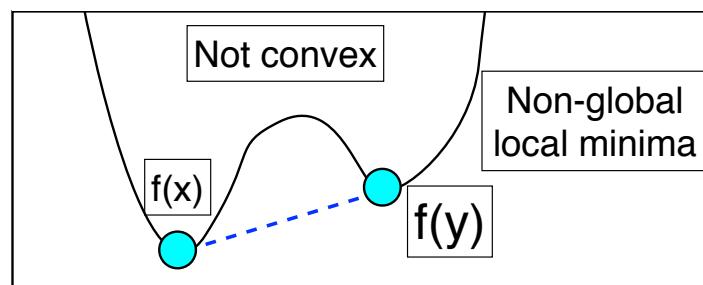
A real-valued function is *convex* if

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y),$$

for all  $x, y \in \mathbb{R}^n$  and all  $0 \leq \theta \leq 1$ .

- Function is *below a linear interpolation* from  $x$  to  $y$ .
- Implies that all local minima are global minima.

(contradiction otherwise)



## Convexity of Norms

We say that a function  $f$  is a **norm** if:

- ①  $f(0) = 0$ .
- ②  $f(\theta x) = |\theta|f(x)$ .
- ③  $f(x + y) \leq f(x) + f(y)$ .

Examples:

$$\|x\|_2 = \sqrt{\sum_i x_i^2} = \sqrt{x^T x}$$

$$\|x\|_1 = \sum_i |x_i|$$

$$\|x\|_H = \sqrt{x^T H x}$$

## Convexity of Norms

We say that a function  $f$  is a **norm** if:

- ①  $f(0) = 0$ .
- ②  $f(\theta x) = |\theta|f(x)$ .
- ③  $f(x + y) \leq f(x) + f(y)$ .

Examples:

$$\|x\|_2 = \sqrt{\sum_i x_i^2} = \sqrt{x^T x}$$

$$\|x\|_1 = \sum_i |x_i|$$

$$\|x\|_H = \sqrt{x^T H x}$$

Norms are convex:

$$f(\theta x + (1 - \theta)y) \leq f(\theta x) + f((1 - \theta)y) \quad (3)$$

$$= \theta f(x) + (1 - \theta)f(y) \quad (2)$$

## Strict Convexity

A real-valued function is *strictly convex* if

$$f(\theta x + (1 - \theta)y) < \theta f(x) + (1 - \theta)f(y),$$

for all  $x \neq y \in \mathbb{R}^n$  and all  $0 < \theta < 1$ .

- Strictly below the linear interpolation from  $x$  to  $y$ .

## Strict Convexity

A real-valued function is *strictly convex* if

$$f(\theta x + (1 - \theta)y) < \theta f(x) + (1 - \theta)f(y),$$

for all  $x \neq y \in \mathbb{R}^n$  and all  $0 < \theta < 1$ .

- Strictly below the linear interpolation from  $x$  to  $y$ .
- Implies at most one global minimum.

(otherwise, could construct lower global minimum)

## Convexity: First-order condition

A real-valued *differentiable function is convex iff*

$$f(y) \geq f(x) + \nabla f(x)^T(y - x),$$

for all  $x, y \in \mathbb{R}^n$ .

- The function is globally *above the tangent* at  $x$ .

(if  $\nabla f(y) = 0$  then  $y$  is a global minimizer)

## Convexity: First-order condition

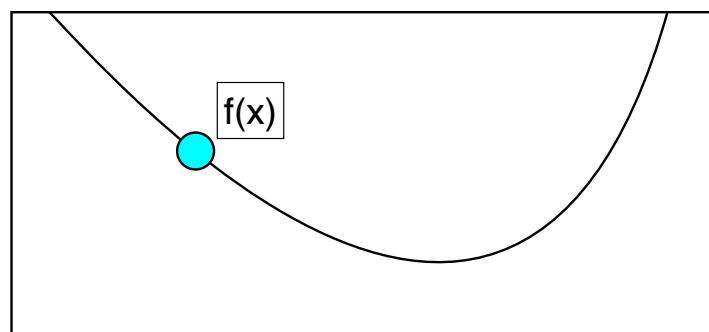
A real-valued *differentiable function is convex iff*

$$f(y) \geq f(x) + \nabla f(x)^T(y - x),$$

for all  $x, y \in \mathbb{R}^n$ .

- The function is globally *above the tangent* at  $x$ .

(if  $\nabla f(y) = 0$  then  $y$  is a global minimizer)



## Convexity: First-order condition

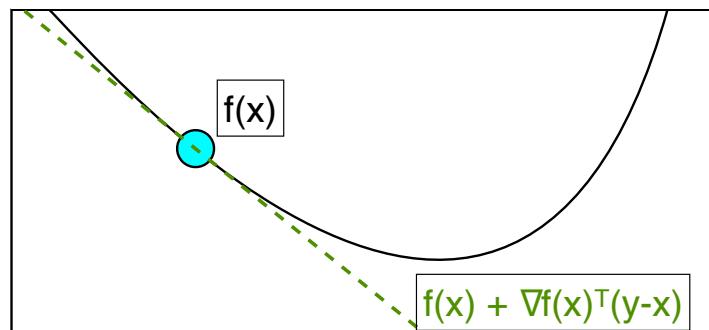
A real-valued *differentiable function is convex iff*

$$f(y) \geq f(x) + \nabla f(x)^T(y - x),$$

for all  $x, y \in \mathbb{R}^n$ .

- The function is globally *above the tangent* at  $x$ .

(if  $\nabla f(y) = 0$  then  $y$  is a global minimizer)



## Convexity: First-order condition

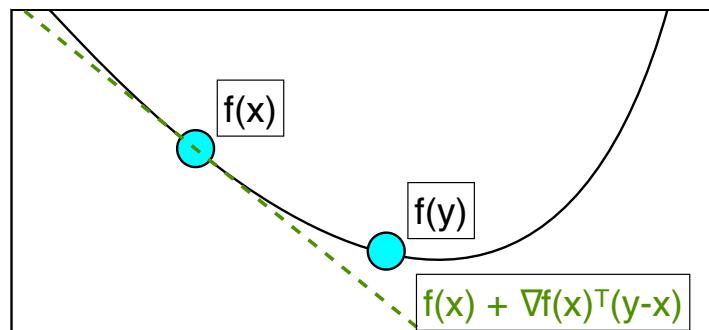
A real-valued *differentiable function is convex iff*

$$f(y) \geq f(x) + \nabla f(x)^T(y - x),$$

for all  $x, y \in \mathbb{R}^n$ .

- The function is globally *above the tangent* at  $x$ .

(if  $\nabla f(y) = 0$  then  $y$  is a global minimizer)



## Convexity: Second-order condition

A real-valued *twice-differentiable function is convex iff*

$$\nabla^2 f(x) \succeq 0$$

*for all  $x \in \mathbb{R}^n$ .*

- The function is *flat or curved upwards* in every direction.

## Convexity: Second-order condition

A real-valued *twice-differentiable* function is convex iff

$$\nabla^2 f(x) \succeq 0$$

for all  $x \in \mathbb{R}^n$ .

- The function is *flat or curved upwards* in every direction.

A real-valued function  $f$  is a *quadratic* if it can be written in the form:

$$f(x) = \frac{1}{2}x^T A x + b^T x + c.$$

Since  $\nabla^2 f(x) = A$ , it is convex if  $A \succeq 0$ .

E.g., least squares has  $\nabla^2 f(x) = A^T A \succeq 0$ .

## Examples of Convex Functions

Some simple convex functions:

- $f(x) = c$
- $f(x) = a^T x$
- $f(x) = ax^2 + b$  (for  $a > 0$ )
- $f(x) = \exp(ax)$
- $f(x) = x \log x$  (for  $x > 0$ )
- $f(x) = \|x\|^2$
- $f(x) = \max_i\{x_i\}$

## Examples of Convex Functions

Some simple convex functions:

- $f(x) = c$
- $f(x) = a^T x$
- $f(x) = ax^2 + b$  (for  $a > 0$ )
- $f(x) = \exp(ax)$
- $f(x) = x \log x$  (for  $x > 0$ )
- $f(x) = \|x\|^2$
- $f(x) = \max_i\{x_i\}$

Some other notable examples:

- $f(x, y) = \log(e^x + e^y)$
- $f(X) = \log \det X$  (for  $X$  positive-definite).
- $f(x, Y) = x^T Y^{-1} x$  (for  $Y$  positive-definite)

## Operations that Preserve Convexity

- ① Non-negative weighted sum:

$$f(x) = \theta_1 f_1(x) + \theta_2 f_2(x).$$

- ② Composition with affine mapping:

$$g(x) = f(Ax + b).$$

- ③ Pointwise maximum:

$$f(x) = \max_i \{f_i(x)\}.$$

## Operations that Preserve Convexity

- ① Non-negative weighted sum:

$$f(x) = \theta_1 f_1(x) + \theta_2 f_2(x).$$

- ② Composition with affine mapping:

$$g(x) = f(Ax + b).$$

- ③ Pointwise maximum:

$$f(x) = \max_i \{f_i(x)\}.$$

Show that least-residual problems are convex for any  $\ell_p$ -norm:

$$f(x) = \|Ax - b\|_p$$

## Operations that Preserve Convexity

- ① Non-negative weighted sum:

$$f(x) = \theta_1 f_1(x) + \theta_2 f_2(x).$$

- ② Composition with affine mapping:

$$g(x) = f(Ax + b).$$

- ③ Pointwise maximum:

$$f(x) = \max_i \{f_i(x)\}.$$

Show that least-residual problems are convex for any  $\ell_p$ -norm:

$$f(x) = \|Ax - b\|_p$$

We know that  $\|\cdot\|_p$  is a norm, so it follows from (2).

## Operations that Preserve Convexity

- ① Non-negative weighted sum:

$$f(x) = \theta_1 f_1(x) + \theta_2 f_2(x).$$

- ② Composition with affine mapping:

$$g(x) = f(Ax + b).$$

- ③ Pointwise maximum:

$$f(x) = \max_i\{f_i(x)\}.$$

Show that SVMs are convex:

$$f(x) = \frac{1}{2}||x||^2 + C \sum_{i=1}^n \max\{0, 1 - b_i a_i^T x\}.$$

## Operations that Preserve Convexity

- ① Non-negative weighted sum:

$$f(x) = \theta_1 f_1(x) + \theta_2 f_2(x).$$

- ② Composition with affine mapping:

$$g(x) = f(Ax + b).$$

- ③ Pointwise maximum:

$$f(x) = \max_i\{f_i(x)\}.$$

Show that SVMs are convex:

$$f(x) = \frac{1}{2}\|x\|^2 + C \sum_{i=1}^n \max\{0, 1 - b_i a_i^T x\}.$$

The first term has Hessian  $I \succ 0$ , for the second term use (3) on the two (convex) arguments, then use (1) to put it all together.

# Outline

- 1 Convex Functions
- 2 Smooth Optimization
- 3 Non-Smooth Optimization
- 4 Randomized Algorithms
- 5 Parallel/Distributed Optimization

## How hard is real-valued optimization?

How long to find an  $\epsilon$ -optimal minimizer of a real-valued function?

$$\min_{x \in \mathbb{R}^n} f(x).$$

## How hard is real-valued optimization?

How long to find an  $\epsilon$ -optimal minimizer of a real-valued function?

$$\min_{x \in \mathbb{R}^n} f(x).$$

- General function: impossible!

(think about arbitrarily small value at some infinite decimal expansion)

## How hard is real-valued optimization?

How long to find an  $\epsilon$ -optimal minimizer of a real-valued function?

$$\min_{x \in \mathbb{R}^n} f(x).$$

- General function: impossible!

(think about arbitrarily small value at some infinite decimal expansion)

We need to make some assumptions about the function:

- Assume  $f$  is **Lipschitz-continuous**: (can not change too quickly)

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

## How hard is real-valued optimization?

How long to find an  $\epsilon$ -optimal minimizer of a real-valued function?

$$\min_{x \in \mathbb{R}^n} f(x).$$

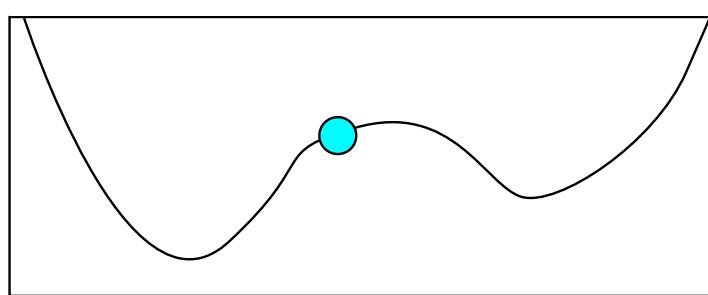
- General function: impossible!

(think about arbitrarily small value at some infinite decimal expansion)

We need to make some assumptions about the function:

- Assume  $f$  is **Lipschitz-continuous**: (can not change too quickly)

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



## How hard is real-valued optimization?

How long to find an  $\epsilon$ -optimal minimizer of a real-valued function?

$$\min_{x \in \mathbb{R}^n} f(x).$$

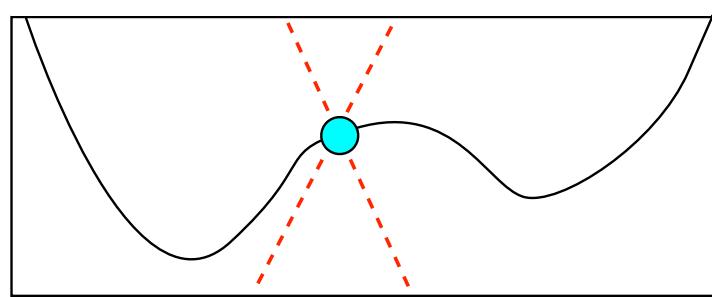
- General function: impossible!

(think about arbitrarily small value at some infinite decimal expansion)

We need to make some assumptions about the function:

- Assume  $f$  is **Lipschitz-continuous**: (can not change too quickly)

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



## How hard is real-valued optimization?

How long to find an  $\epsilon$ -optimal minimizer of a real-valued function?

$$\min_{x \in \mathbb{R}^n} f(x).$$

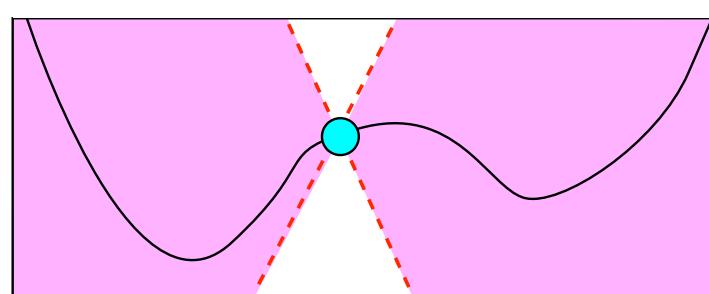
- General function: impossible!

(think about arbitrarily small value at some infinite decimal expansion)

We need to make some assumptions about the function:

- Assume  $f$  is **Lipschitz-continuous**: (can not change too quickly)

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



## How hard is real-valued optimization?

How long to find an  $\epsilon$ -optimal minimizer of a real-valued function?

$$\min_{x \in \mathbb{R}^n} f(x).$$

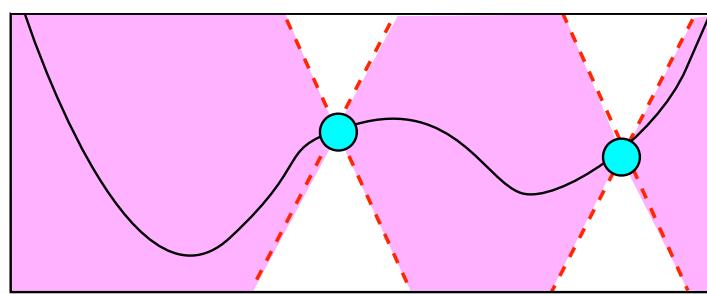
- General function: impossible!

(think about arbitrarily small value at some infinite decimal expansion)

We need to make some assumptions about the function:

- Assume  $f$  is **Lipschitz-continuous**: (can not change too quickly)

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



## How hard is real-valued optimization?

How long to find an  $\epsilon$ -optimal minimizer of a real-valued function?

$$\min_{x \in \mathbb{R}^n} f(x).$$

- General function: impossible!

(think about arbitrarily small value at some infinite decimal expansion)

We need to make some assumptions about the function:

- Assume  $f$  is **Lipschitz-continuous**: (can not change too quickly)

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

- After  $t$  iterations, the error of *any algorithm* is  $\Omega(1/t^{1/n})$ .

(this is in the worst case, and note that grid-search is nearly optimal)

## How hard is real-valued optimization?

How long to find an  $\epsilon$ -optimal minimizer of a real-valued function?

$$\min_{x \in \mathbb{R}^n} f(x).$$

- General function: impossible!

(think about arbitrarily small value at some infinite decimal expansion)

We need to make some assumptions about the function:

- Assume  $f$  is **Lipschitz-continuous**: (can not change too quickly)

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

- After  $t$  iterations, the error of *any algorithm* is  $\Omega(1/t^{1/n})$ .

(this is in the worst case, and note that grid-search is nearly optimal)

- **Optimization is hard, but assumptions make a big difference.**

(we went from impossible to very slow)

## Motivation for First-Order Methods

- Well-known that we can solve **convex** optimization problems in polynomial-time by interior-point methods

## Motivation for First-Order Methods

- Well-known that we can solve convex optimization problems in polynomial-time by interior-point methods
- However, these solvers require  $O(n^2)$  or worse cost per iteration.
  - Infeasible for applications where  $n$  may be in the billions.

## Motivation for First-Order Methods

- Well-known that we can solve **convex** optimization problems in polynomial-time by interior-point methods
- However, these solvers require  $O(n^2)$  or worse cost per iteration.
  - Infeasible for applications where  $n$  may be in the billions.
- Solving big problems has led to re-newed interest in simple **first-order** methods (**gradient methods**):

$$x^+ = x - \alpha \nabla f(x).$$

- These only have  $O(n)$  iteration costs.
- But we must analyze **how many iterations** are needed.

## $\ell_2$ -Regularized Logistic Regression

- Consider  $\ell_2$ -regularized logistic regression:

$$f(x) = \sum_{i=1}^n \log(1 + \exp(-b_i(x^T a_i))) + \frac{\lambda}{2} \|x\|^2.$$

- Objective  $f$  is convex.
- First term is Lipschitz continuous.
- Second term is not Lipschitz continuous.

## $\ell_2$ -Regularized Logistic Regression

- Consider  $\ell_2$ -regularized logistic regression:

$$f(x) = \sum_{i=1}^n \log(1 + \exp(-b_i(x^T a_i))) + \frac{\lambda}{2} \|x\|^2.$$

- Objective  $f$  is convex.
- First term is Lipschitz continuous.
- Second term is not Lipschitz continuous.
- But we have

$$\mu I \preceq \nabla^2 f(x) \preceq L I.$$

$$(L = \frac{1}{4} \|A\|_2^2 + \lambda, \mu = \lambda)$$

- Gradient is Lipschitz-continuous.
- Function is strongly-convex.

(implies strict convexity, and existence of unique solution)

## Properties of Lipschitz-Continuous Gradient

- From Taylor's theorem, for some  $z$  we have:

$$f(y) = f(x) + \nabla f(x)^T(y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(z)(y - x)$$

## Properties of Lipschitz-Continuous Gradient

- From Taylor's theorem, for some  $z$  we have:

$$f(y) = f(x) + \nabla f(x)^T(y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(z)(y - x)$$

- Use that  $\nabla^2 f(z) \preceq L\mathbf{I}$ .

$$f(y) \leq f(x) + \nabla f(x)^T(y - x) + \frac{L}{2} \|y - x\|^2$$

- *Global quadratic upper bound on function value.*

## Properties of Lipschitz-Continuous Gradient

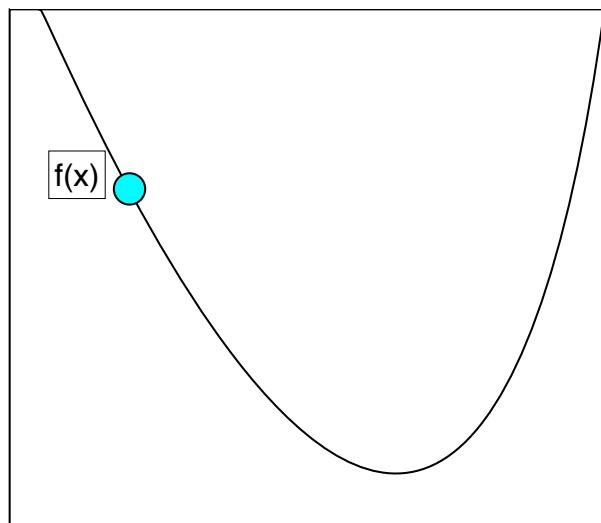
- From Taylor's theorem, for some  $z$  we have:

$$f(y) = f(x) + \nabla f(x)^T(y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(z)(y - x)$$

- Use that  $\nabla^2 f(z) \preceq L I$ .

$$f(y) \leq f(x) + \nabla f(x)^T(y - x) + \frac{L}{2} \|y - x\|^2$$

- Global quadratic upper bound on function value.*



## Properties of Lipschitz-Continuous Gradient

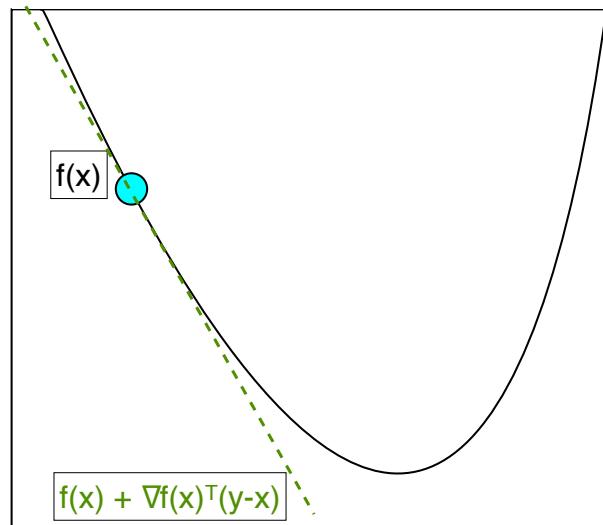
- From Taylor's theorem, for some  $z$  we have:

$$f(y) = f(x) + \nabla f(x)^T(y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(z)(y - x)$$

- Use that  $\nabla^2 f(z) \preceq L I$ .

$$f(y) \leq f(x) + \nabla f(x)^T(y - x) + \frac{L}{2} \|y - x\|^2$$

- Global quadratic upper bound on function value.*



## Properties of Lipschitz-Continuous Gradient

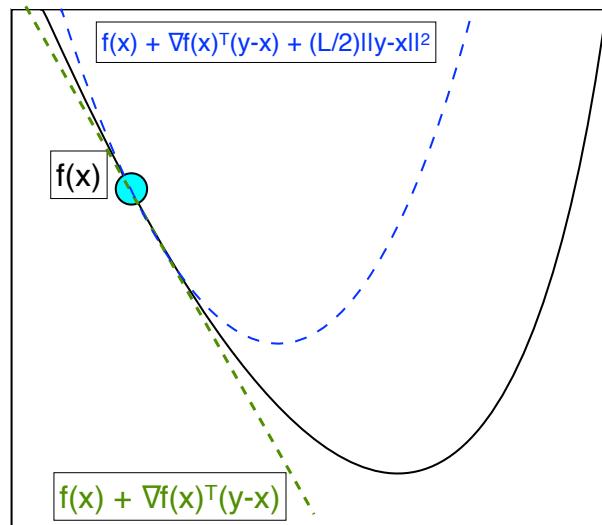
- From Taylor's theorem, for some  $z$  we have:

$$f(y) = f(x) + \nabla f(x)^T(y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(z)(y - x)$$

- Use that  $\nabla^2 f(z) \preceq L\mathbf{I}$ .

$$f(y) \leq f(x) + \nabla f(x)^T(y - x) + \frac{L}{2} \|y - x\|^2$$

- Global quadratic upper bound on function value.*



## Properties of Lipschitz-Continuous Gradient

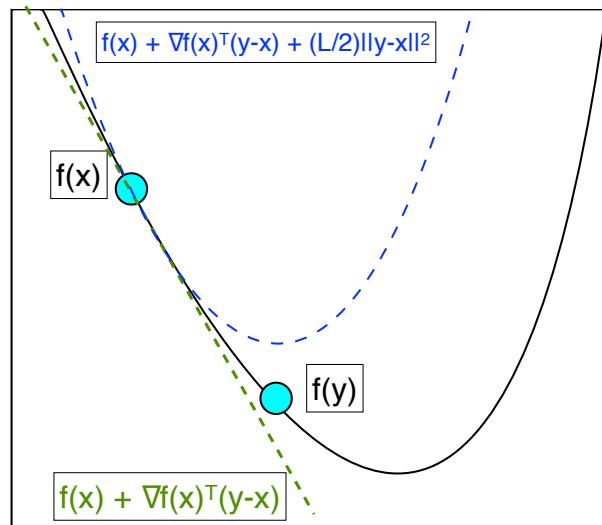
- From Taylor's theorem, for some  $z$  we have:

$$f(y) = f(x) + \nabla f(x)^T(y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(z)(y - x)$$

- Use that  $\nabla^2 f(z) \preceq L\mathbf{I}$ .

$$f(y) \leq f(x) + \nabla f(x)^T(y - x) + \frac{L}{2} \|y - x\|^2$$

- Global quadratic upper bound on function value.*



## Properties of Lipschitz-Continuous Gradient

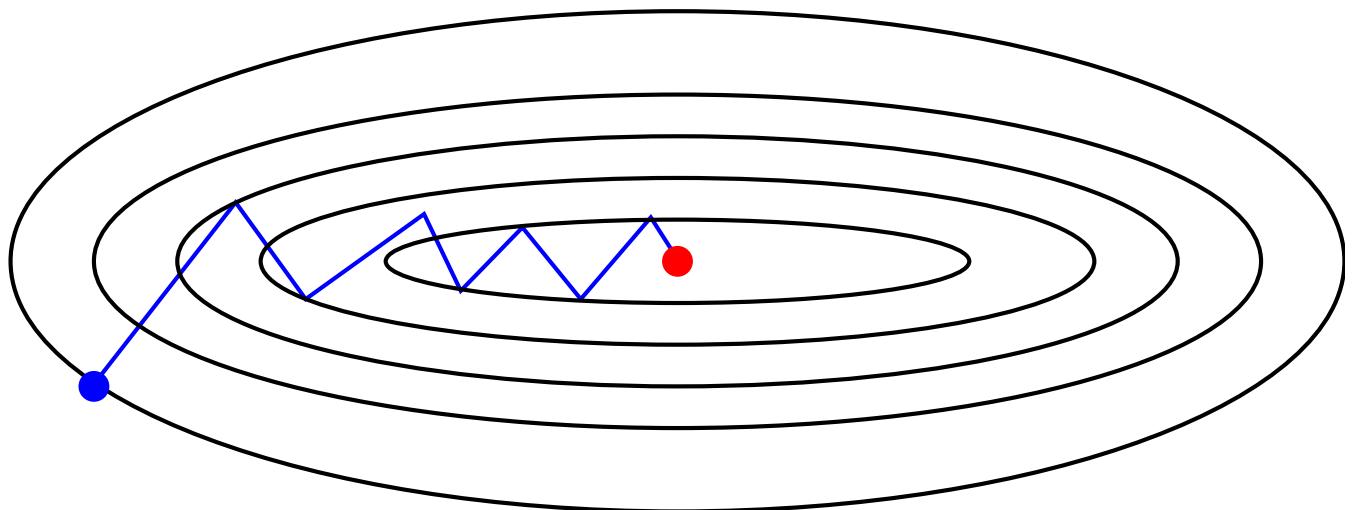
- From Taylor's theorem, for some  $z$  we have:

$$f(y) = f(x) + \nabla f(x)^T(y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(z)(y - x)$$

- Use that  $\nabla^2 f(z) \preceq L I$ .

$$f(y) \leq f(x) + \nabla f(x)^T(y - x) + \frac{L}{2} \|y - x\|^2$$

- Global quadratic upper bound on function value.*



## Properties of Lipschitz-Continuous Gradient

- From Taylor's theorem, for some  $z$  we have:

$$f(y) = f(x) + \nabla f(x)^T(y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(z)(y - x)$$

- Use that  $\nabla^2 f(z) \preceq L\mathbf{I}$ .

$$f(y) \leq f(x) + \nabla f(x)^T(y - x) + \frac{L}{2} \|y - x\|^2$$

- Global quadratic upper bound on function value.*
- Set  $x^+$  to minimize upper bound in terms of  $y$ :

$$x^+ = x - \frac{1}{L} \nabla f(x).$$

(gradient descent with step-size of  $1/L$ )

- Plugging this value in:

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

(decrease of at least  $\frac{1}{2L} \|\nabla f(x)\|^2$ )

## Properties of Strong-Convexity

- From Taylor's theorem, for some  $z$  we have:

$$f(y) = f(x) + \nabla f(x)^T(y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(z)(y - x)$$

## Properties of Strong-Convexity

- From Taylor's theorem, for some  $z$  we have:

$$f(y) = f(x) + \nabla f(x)^T(y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(z)(y - x)$$

- Use that  $\nabla^2 f(z) \succeq \mu I$ .

$$f(y) \geq f(x) + \nabla f(x)^T(y - x) + \frac{\mu}{2} \|y - x\|^2$$

- *Global quadratic lower bound on function value.*

## Properties of Strong-Convexity

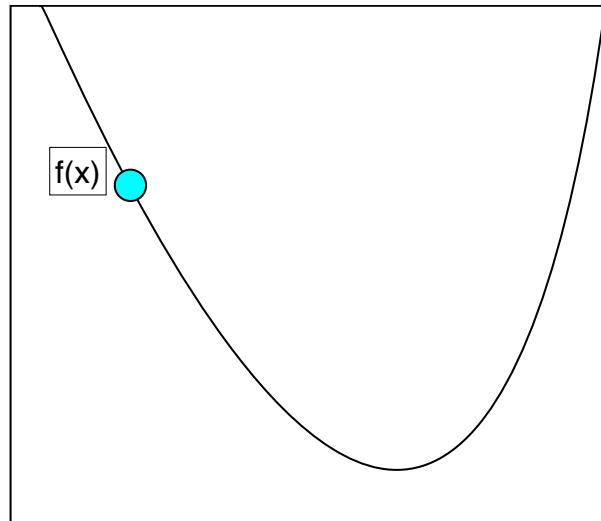
- From Taylor's theorem, for some  $z$  we have:

$$f(y) = f(x) + \nabla f(x)^T(y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(z)(y - x)$$

- Use that  $\nabla^2 f(z) \succeq \mu I$ .

$$f(y) \geq f(x) + \nabla f(x)^T(y - x) + \frac{\mu}{2} \|y - x\|^2$$

- Global quadratic lower bound on function value.*



## Properties of Strong-Convexity

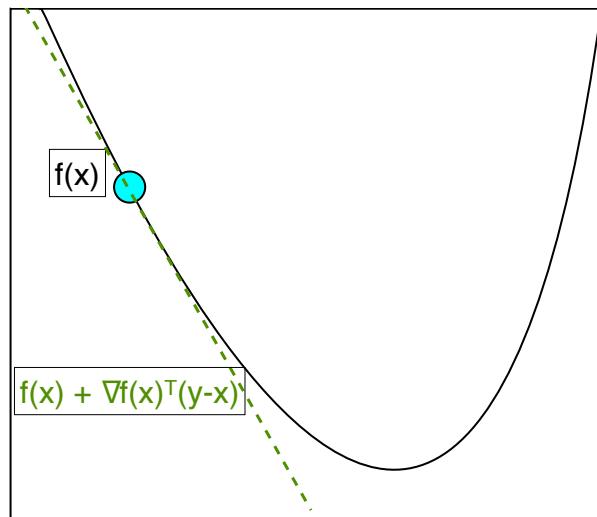
- From Taylor's theorem, for some  $z$  we have:

$$f(y) = f(x) + \nabla f(x)^T(y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(z)(y - x)$$

- Use that  $\nabla^2 f(z) \succeq \mu I$ .

$$f(y) \geq f(x) + \nabla f(x)^T(y - x) + \frac{\mu}{2} \|y - x\|^2$$

- Global quadratic lower bound on function value.*



## Properties of Strong-Convexity

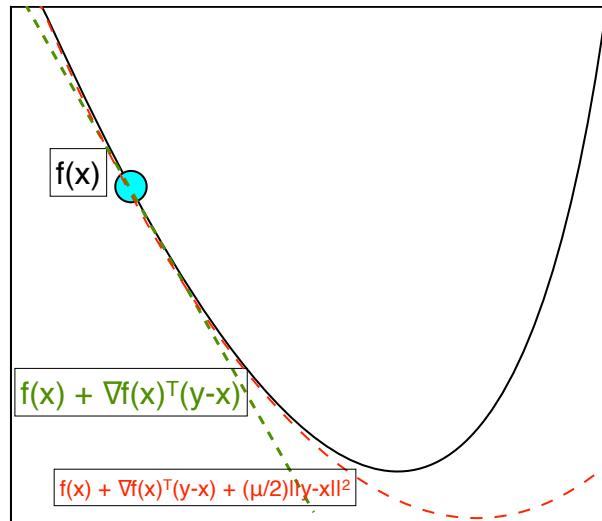
- From Taylor's theorem, for some  $z$  we have:

$$f(y) = f(x) + \nabla f(x)^T(y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(z)(y - x)$$

- Use that  $\nabla^2 f(z) \succeq \mu I$ .

$$f(y) \geq f(x) + \nabla f(x)^T(y - x) + \frac{\mu}{2} \|y - x\|^2$$

- Global quadratic lower bound on function value.*



## Properties of Strong-Convexity

- From Taylor's theorem, for some  $z$  we have:

$$f(y) = f(x) + \nabla f(x)^T(y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(z)(y - x)$$

- Use that  $\nabla^2 f(z) \succeq \mu I$ .

$$f(y) \geq f(x) + \nabla f(x)^T(y - x) + \frac{\mu}{2} \|y - x\|^2$$

- Global quadratic lower bound on function value.*
- Minimize both sides in terms of  $y$ :

$$f(x^*) \geq f(x) - \frac{1}{2\mu} \|\nabla f(x)\|^2.$$

- Upper bound on how far we are from the solution.

## Linear Convergence of Gradient Descent

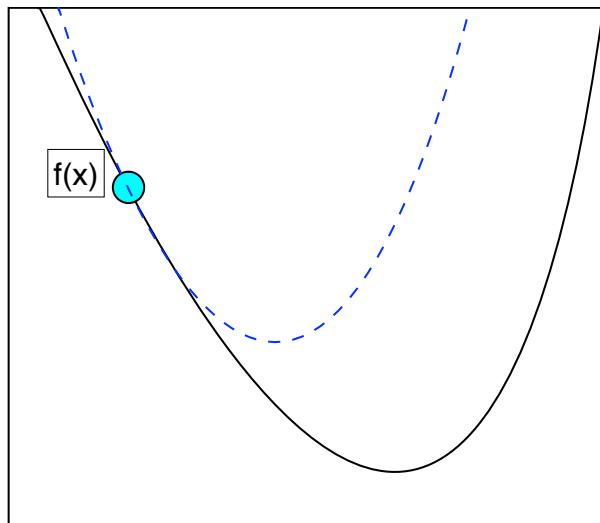
- We have bounds on  $x^+$  and  $x^*$ :

$$f(x^+) \leq f(x) - \frac{1}{2L} \|\nabla f(x)\|^2, \quad f(x^*) \geq f(x) - \frac{1}{2\mu} \|\nabla f(x)\|^2.$$

## Linear Convergence of Gradient Descent

- We have bounds on  $x^+$  and  $x^*$ :

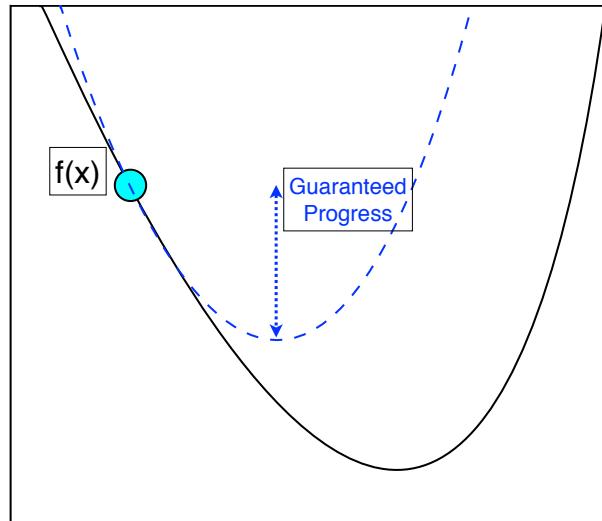
$$f(x^+) \leq f(x) - \frac{1}{2L} \|\nabla f(x)\|^2, \quad f(x^*) \geq f(x) - \frac{1}{2\mu} \|\nabla f(x)\|^2.$$



## Linear Convergence of Gradient Descent

- We have bounds on  $x^+$  and  $x^*$ :

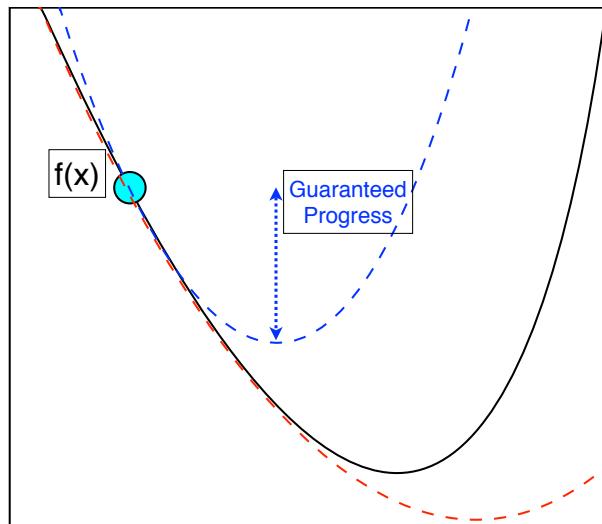
$$f(x^+) \leq f(x) - \frac{1}{2L} \|\nabla f(x)\|^2, \quad f(x^*) \geq f(x) - \frac{1}{2\mu} \|\nabla f(x)\|^2.$$



## Linear Convergence of Gradient Descent

- We have bounds on  $x^+$  and  $x^*$ :

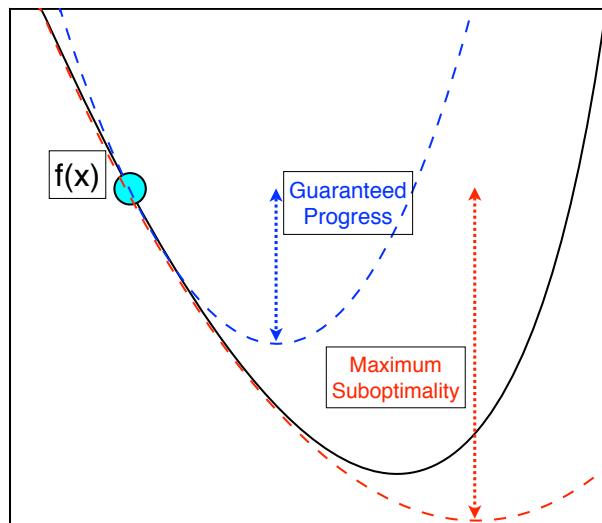
$$f(x^+) \leq f(x) - \frac{1}{2L} \|\nabla f(x)\|^2, \quad f(x^*) \geq f(x) - \frac{1}{2\mu} \|\nabla f(x)\|^2.$$



## Linear Convergence of Gradient Descent

- We have bounds on  $x^+$  and  $x^*$ :

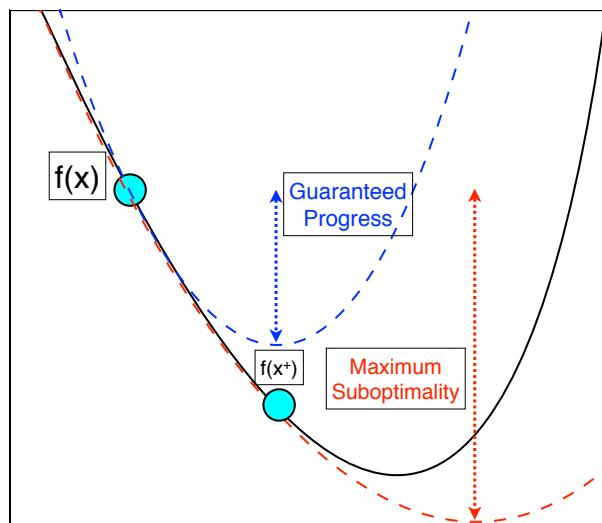
$$f(x^+) \leq f(x) - \frac{1}{2L} \|\nabla f(x)\|^2, \quad f(x^*) \geq f(x) - \frac{1}{2\mu} \|\nabla f(x)\|^2.$$



## Linear Convergence of Gradient Descent

- We have bounds on  $x^+$  and  $x^*$ :

$$f(x^+) \leq f(x) - \frac{1}{2L} \|\nabla f(x)\|^2, \quad f(x^*) \geq f(x) - \frac{1}{2\mu} \|\nabla f(x)\|^2.$$



## Linear Convergence of Gradient Descent

- We have bounds on  $x^+$  and  $x^*$ :

$$f(x^+) \leq f(x) - \frac{1}{2L} \|\nabla f(x)\|^2, \quad f(x^*) \geq f(x) - \frac{1}{2\mu} \|\nabla f(x)\|^2.$$

combine them to get

$$f(x^+) - f(x^*) \leq \left(1 - \frac{\mu}{L}\right) [f(x) - f(x^*)]$$

## Linear Convergence of Gradient Descent

- We have bounds on  $x^+$  and  $x^*$ :

$$f(x^+) \leq f(x) - \frac{1}{2L} \|\nabla f(x)\|^2, \quad f(x^*) \geq f(x) - \frac{1}{2\mu} \|\nabla f(x)\|^2.$$

combine them to get

$$f(x^+) - f(x^*) \leq \left(1 - \frac{\mu}{L}\right) [f(x) - f(x^*)]$$

- This gives a linear convergence rate:

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

- Each iteration multiplies the error by a fixed amount.

(very fast if  $\mu/L$  is not too close to one)

## Maximum Likelihood Logistic Regression

- What about maximum-likelihood logistic regression?

$$f(x) = \sum_{i=1}^n \log(1 + \exp(-b_i(x^T a_i))).$$

## Maximum Likelihood Logistic Regression

- What about maximum-likelihood logistic regression?

$$f(x) = \sum_{i=1}^n \log(1 + \exp(-b_i(x^T a_i))).$$

- We now only have

$$\mathbf{0} \preceq \nabla^2 f(x) \preceq L\mathbf{I}.$$

- Convexity only gives a linear upper bound on  $f(x^*)$ :

$$f(x^*) \leq f(x) + \nabla f(x)^T (x^* - x)$$

## Maximum Likelihood Logistic Regression

- What about maximum-likelihood logistic regression?

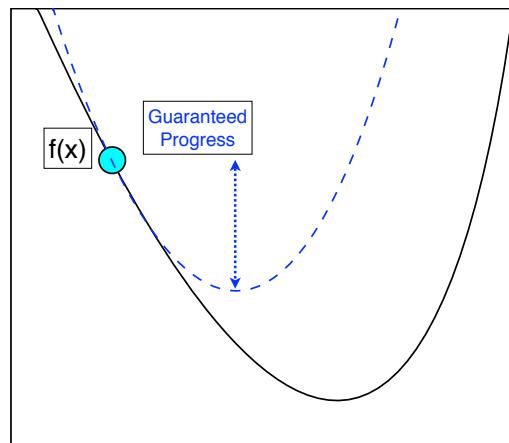
$$f(x) = \sum_{i=1}^n \log(1 + \exp(-b_i(x^T a_i))).$$

- We now only have

$$\mathbf{0} \preceq \nabla^2 f(x) \preceq L\mathbf{I}.$$

- Convexity only gives a linear upper bound on  $f(x^*)$ :

$$f(x^*) \leq f(x) + \nabla f(x)^T (x^* - x)$$



## Maximum Likelihood Logistic Regression

- What about maximum-likelihood logistic regression?

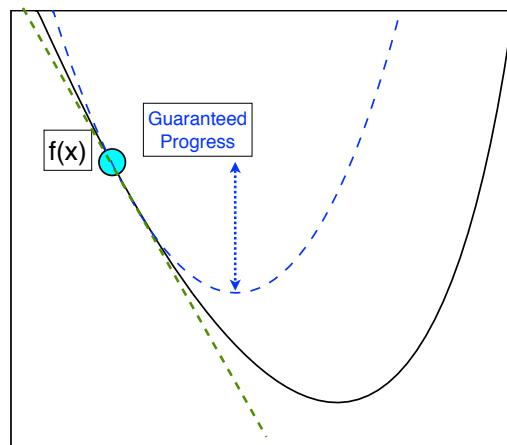
$$f(x) = \sum_{i=1}^n \log(1 + \exp(-b_i(x^T a_i))).$$

- We now only have

$$\mathbf{0} \preceq \nabla^2 f(x) \preceq L\mathbf{I}.$$

- Convexity only gives a linear upper bound on  $f(x^*)$ :

$$f(x^*) \leq f(x) + \nabla f(x)^T (x^* - x)$$



## Maximum Likelihood Logistic Regression

- What about maximum-likelihood logistic regression?

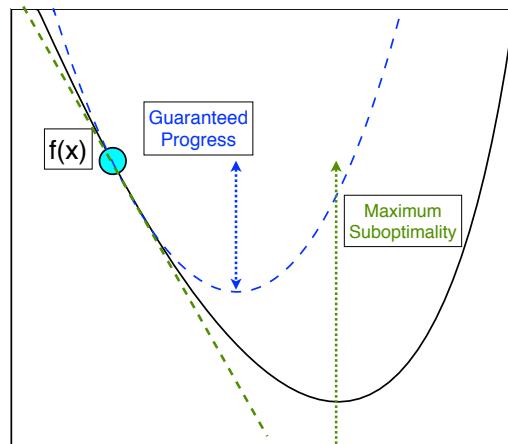
$$f(x) = \sum_{i=1}^n \log(1 + \exp(-b_i(x^T a_i))).$$

- We now only have

$$\mathbf{0} \preceq \nabla^2 f(x) \preceq L\mathbf{I}.$$

- Convexity only gives a linear upper bound on  $f(x^*)$ :

$$f(x^*) \leq f(x) + \nabla f(x)^T (x^* - x)$$



## Maximum Likelihood Logistic Regression

- Consider maximum-likelihood logistic regression:

$$f(x) = \sum_{i=1}^n \log(1 + \exp(-b_i(x^T a_i))).$$

- We now only have

$$\mathbf{0} \preceq \nabla^2 f(x) \preceq L\mathbf{I}.$$

- Convexity only gives a linear upper bound on  $f(x^*)$ :

$$f(x^*) \leq f(x) + \nabla f(x)^T (x^* - x)$$

- If some  $x^*$  exists, we have the sublinear convergence rate:

$$f(x^t) - f(x^*) = O(1/t)$$

(compare to slower  $\Omega(1/t^{-1/N})$  for general Lipschitz functions)

## Maximum Likelihood Logistic Regression

- Consider maximum-likelihood logistic regression:

$$f(x) = \sum_{i=1}^n \log(1 + \exp(-b_i(x^T a_i))).$$

- We now only have

$$\mathbf{0} \preceq \nabla^2 f(x) \preceq L\mathbf{I}.$$

- Convexity only gives a linear upper bound on  $f(x^*)$ :

$$f(x^*) \leq f(x) + \nabla f(x)^T (x^* - x)$$

- If some  $x^*$  exists, we have the sublinear convergence rate:

$$f(x^t) - f(x^*) = O(1/t)$$

(compare to slower  $\Omega(1/t^{-1/N})$  for general Lipschitz functions)

- If  $f$  is convex, then  $f + \lambda\|x\|^2$  is strongly-convex.

## Gradient Method: Practical Issues

- In practice, searching for step size ([line-search](#)) is usually much faster than  $\alpha = 1/L$ .  
(and doesn't require knowledge of  $L$ )

## Gradient Method: Practical Issues

- In practice, searching for step size ([line-search](#)) is usually much faster than  $\alpha = 1/L$ .  
(and doesn't require knowledge of  $L$ )
- Basic [Armijo](#) backtracking line-search:
  - ① Start with a large value of  $\alpha$ .
  - ② Divide  $\alpha$  in half until we satisfy (typically value is  $\gamma = .0001$ )

$$f(x^+) \leq f(x) - \gamma\alpha\|\nabla f(x)\|^2.$$

## Gradient Method: Practical Issues

- In practice, searching for step size ([line-search](#)) is usually much faster than  $\alpha = 1/L$ .  
(and doesn't require knowledge of  $L$ )
- Basic [Armijo](#) backtracking line-search:
  - ① Start with a large value of  $\alpha$ .
  - ② Divide  $\alpha$  in half until we satisfy (typically value is  $\gamma = .0001$ )
$$f(x^+) \leq f(x) - \gamma\alpha\|\nabla f(x)\|^2.$$
- Practical methods may use *Wolfe conditions* (so  $\alpha$  isn't too small), and/or use *interpolation* to propose trial step sizes.  
(with good interpolation,  $\approx 1$  evaluation of  $f$  per iteration)

## Gradient Method: Practical Issues

- In practice, searching for step size ([line-search](#)) is usually much faster than  $\alpha = 1/L$ .  
 (and doesn't require knowledge of  $L$ )
- Basic [Armijo](#) backtracking line-search:
  - ① Start with a large value of  $\alpha$ .
  - ② Divide  $\alpha$  in half until we satisfy (typically value is  $\gamma = .0001$ )
$$f(x^+) \leq f(x) - \gamma\alpha\|\nabla f(x)\|^2.$$
- Practical methods may use *Wolfe conditions* (so  $\alpha$  isn't too small), and/or use *interpolation* to propose trial step sizes.  
 (with good interpolation,  $\approx 1$  evaluation of  $f$  per iteration)
- Also, check your derivative code!

$$\nabla_i f(x) \approx \frac{f(x + \delta e_i) - f(x)}{\delta}$$

- For large-scale problems you can check a random direction  $d$ :

$$\nabla f(x)^T d \approx \frac{f(x + \delta d) - f(x)}{\delta}$$

## Convergence Rate of Gradient Method

We are going to explore the ‘convex optimization zoo’:

- Gradient method for smooth/convex:  $O(1/t)$ .
- Gradient method for smooth/strongly-convex:  $O((1 - \mu/L)^t)$ .

## Convergence Rate of Gradient Method

We are going to explore the ‘convex optimization zoo’:

- Gradient method for smooth/convex:  $O(1/t)$ .
- Gradient method for smooth/strongly-convex:  $O((1 - \mu/L)^t)$ .
- Rates are the same if only once-differentiable.
- Line-search doesn’t change the worst-case rate.

(strongly-convex slightly improved with  $\alpha = 2/(\mu + L)$ )

## Convergence Rate of Gradient Method

We are going to explore the ‘convex optimization zoo’:

- Gradient method for smooth/convex:  $O(1/t)$ .
- Gradient method for smooth/strongly-convex:  $O((1 - \mu/L)^t)$ .
- Rates are the same if only once-differentiable.
- Line-search doesn’t change the worst-case rate.  
(strongly-convex slightly improved with  $\alpha = 2/(\mu + L)$ )
- Is this the best algorithm under these assumptions?

## Accelerated Gradient Method

- Nesterov's accelerated gradient method:

$$\begin{aligned}x_{t+1} &= y_t - \alpha_t \nabla f(y_t), \\y_{t+1} &= x_t + \beta_t(x_{t+1} - x_t),\end{aligned}$$

for appropriate  $\alpha_t, \beta_t$ .

## Accelerated Gradient Method

- Nesterov's accelerated gradient method:

$$\begin{aligned}x_{t+1} &= y_t - \alpha_t \nabla f(y_t), \\y_{t+1} &= x_t + \beta_t(x_{t+1} - x_t),\end{aligned}$$

for appropriate  $\alpha_t, \beta_t$ .

- Motivation: “to make the math work”

(but similar to heavy-ball/momentum and conjugate gradient method)

## Convex Optimization Zoo

Algorithm	Assumptions	Rate
Gradient	Convex	$O(1/t)$
Nesterov	Convex	$O(1/t^2)$
Gradient	Strongly-Convex	$O((1 - \mu/L)^t)$
Nesterov	Strongly-Convex	$O((1 - \sqrt{\mu/L})^t)$

- $O(1/t^2)$  is optimal given only these assumptions.  
(sometimes called the *optimal* gradient method)
- The faster linear convergence rate is close to optimal.
- Also faster in practice, but implementation details matter.

## Newton's Method

- The oldest differentiable optimization method is [Newton's](#).  
(also called IRLS for functions of the form  $f(Ax)$ )
- Modern form uses the update

$$x^+ = x - \alpha d,$$

where  $d$  is a solution to the system

$$\nabla^2 f(x) d = \nabla f(x). \quad (\text{Assumes } \nabla^2 f(x) \succ 0)$$

## Newton's Method

- The oldest differentiable optimization method is **Newton's**.  
(also called IRLS for functions of the form  $f(Ax)$ )
- Modern form uses the update

$$x^+ = x - \alpha d,$$

where  $d$  is a solution to the system

$$\nabla^2 f(x)d = \nabla f(x). \quad (\text{Assumes } \nabla^2 f(x) \succ 0)$$

- Equivalent to minimizing the quadratic approximation:

$$f(y) \approx f(x) + \nabla f(x)^T(y - x) + \frac{1}{2\alpha} \|y - x\|_{\nabla^2 f(x)}^2.$$

(recall that  $\|x\|_H^2 = x^T H x$ )

## Newton's Method

- The oldest differentiable optimization method is **Newton's**.  
(also called IRLS for functions of the form  $f(Ax)$ )
- Modern form uses the update

$$x^+ = x - \alpha d,$$

where  $d$  is a solution to the system

$$\nabla^2 f(x)d = \nabla f(x). \quad (\text{Assumes } \nabla^2 f(x) \succ 0)$$

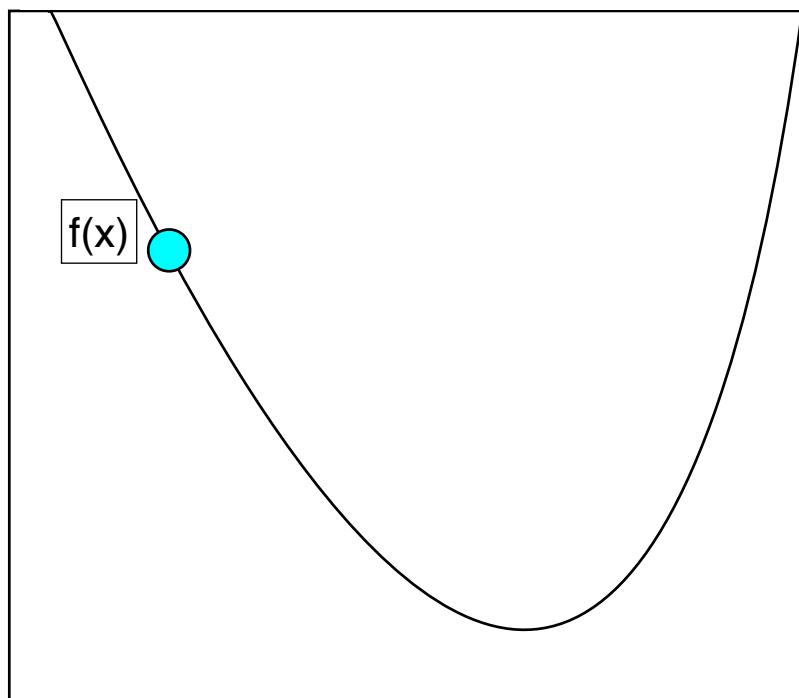
- Equivalent to minimizing the quadratic approximation:
- $$f(y) \approx f(x) + \nabla f(x)^T(y - x) + \frac{1}{2\alpha} \|y - x\|_{\nabla^2 f(x)}^2.$$
- (recall that  $\|x\|_H^2 = x^T H x$ )

- We can generalize the Armijo condition to

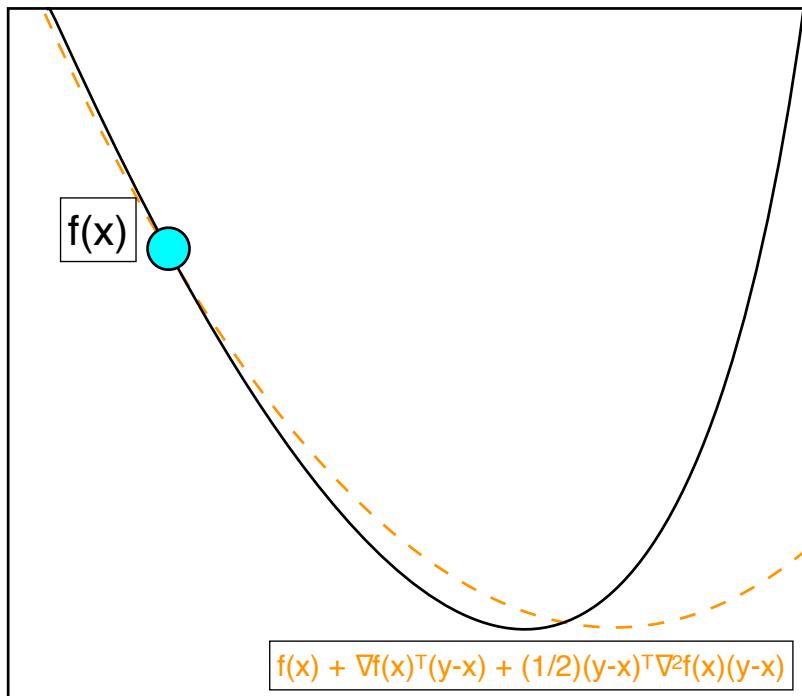
$$f(x^+) \leq f(x) + \gamma \alpha \nabla f(x)^T d.$$

- Has a natural step length of  $\alpha = 1$ .  
(always accepted when close to a minimizer)

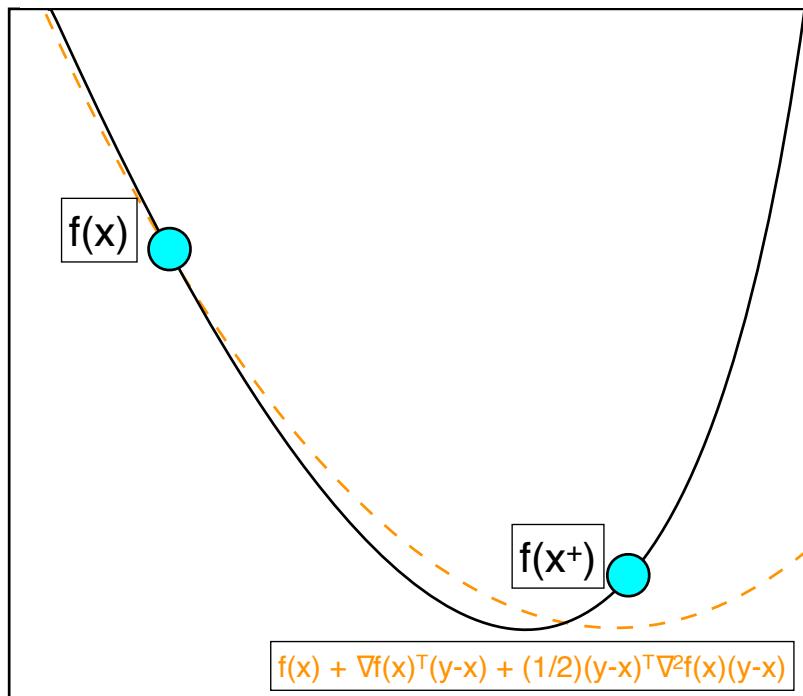
## Newton's Method



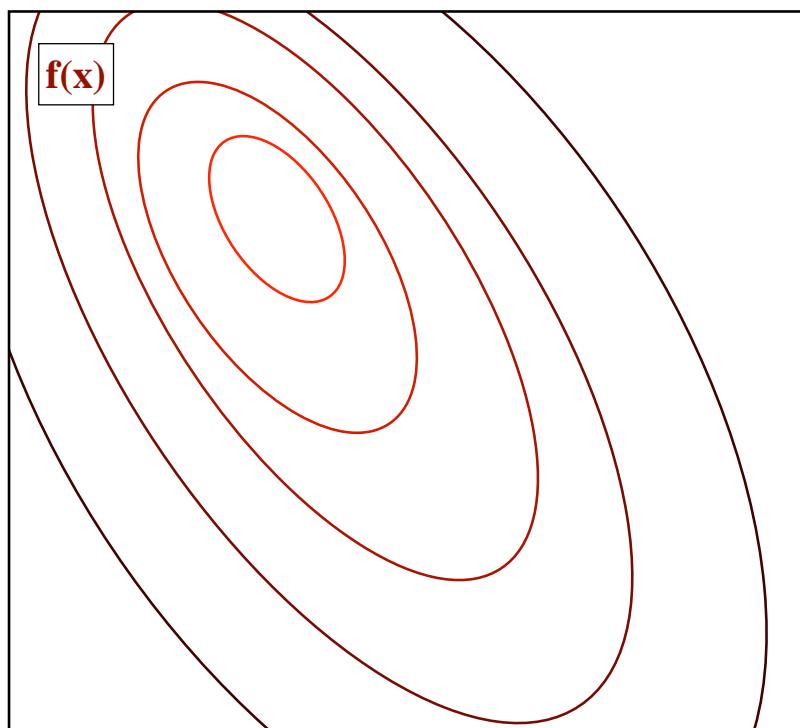
## Newton's Method



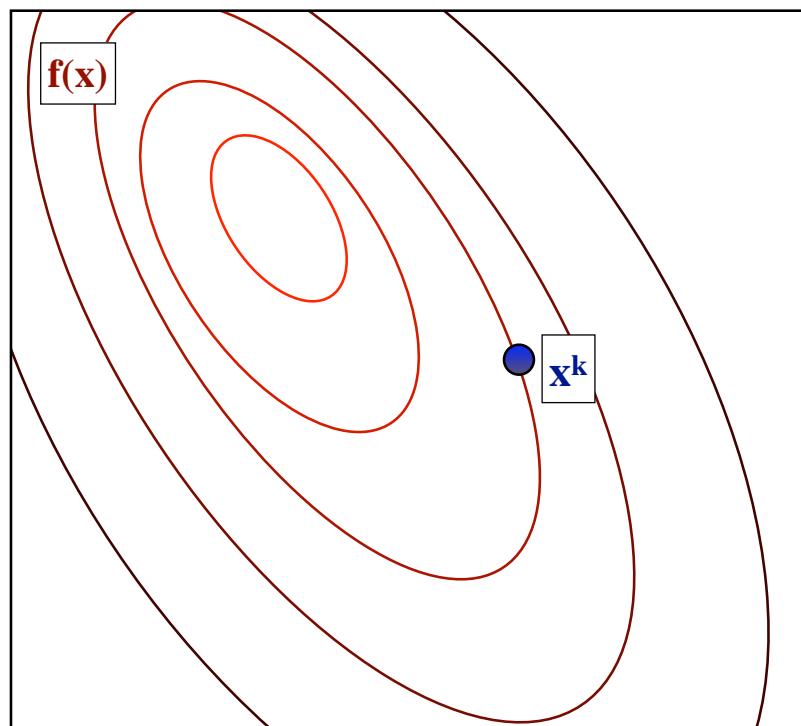
## Newton's Method



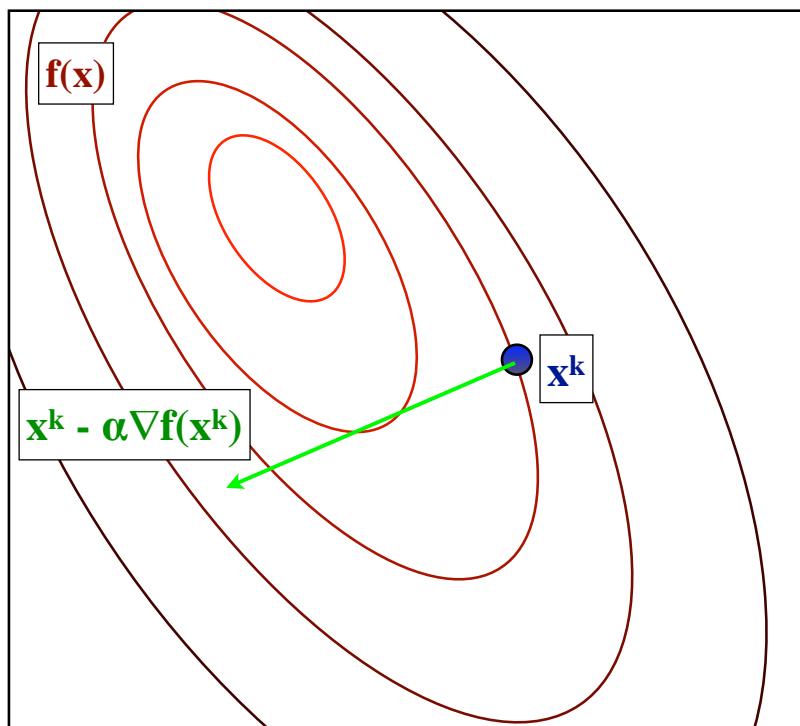
## Newton's Method



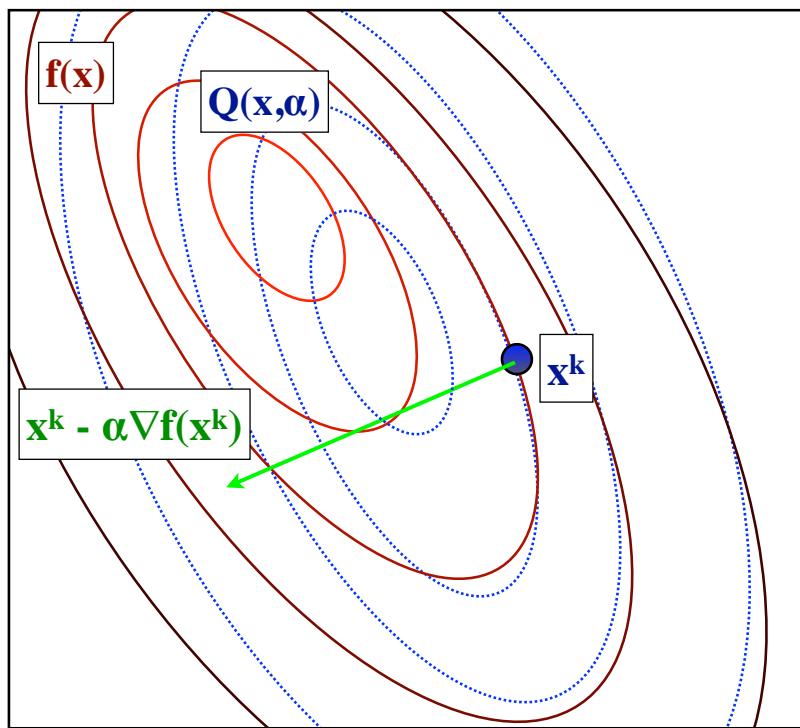
## Newton's Method



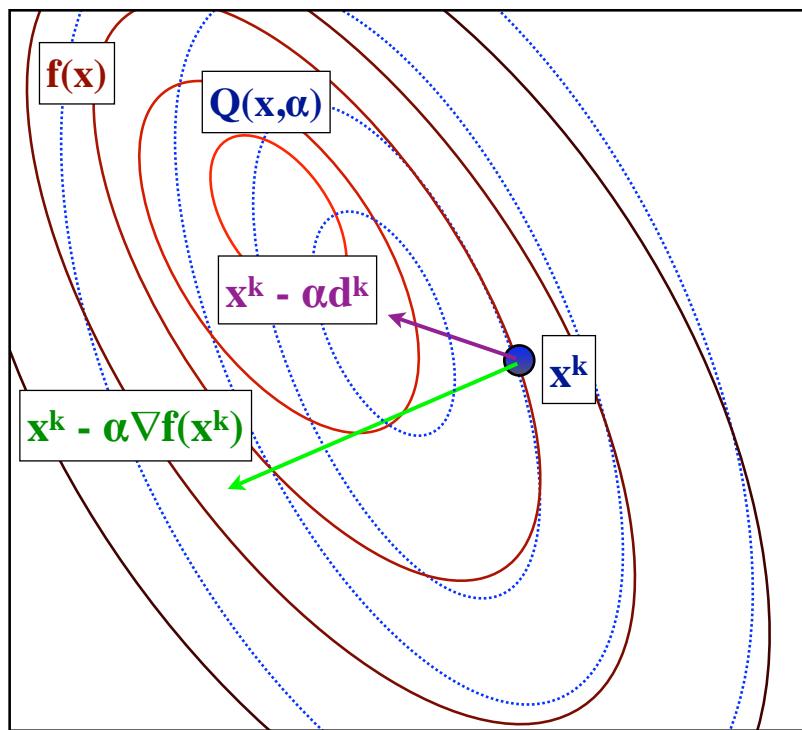
## Newton's Method



## Newton's Method



## Newton's Method



## Convergence Rate of Newton's Method

- If  $\nabla^2 f(x)$  is Lipschitz-continuous and  $\nabla^2 f(x) \succeq \mu$ , then close to  $x^*$  Newton's method has local superlinear convergence:

$$f(x^{t+1}) - f(x^*) \leq \rho_t [f(x^t) - f(x^*)],$$

with  $\lim_{t \rightarrow \infty} \rho_t = 0$ .

- Converges very fast, use it if you can!
- But requires solving  $\nabla^2 f(x)d = \nabla f(x)$ .

## Convergence Rate of Newton's Method

- If  $\nabla^2 f(x)$  is Lipschitz-continuous and  $\nabla^2 f(x) \succeq \mu$ , then close to  $x^*$  Newton's method has **local superlinear convergence**:

$$f(x^{t+1}) - f(x^*) \leq \rho_t [f(x^t) - f(x^*)],$$

with  $\lim_{t \rightarrow \infty} \rho_t = 0$ .

- Converges very fast, use it if you can!
- But **requires solving**  $\nabla^2 f(x)d = \nabla f(x)$ .
- Get global rates under various assumptions  
(cubic-regularization/accelerated/self-concordant).

## Newton's Method: Practical Issues

There are many practical variants of Newton's method:

- Modify the Hessian to be positive-definite.
- Only compute the Hessian every  $m$  iterations.
- Only use the diagonals of the Hessian.
- Quasi-Newton: Update a (diagonal plus low-rank) approximation of the Hessian (BFGS, L-BFGS).

## Newton's Method: Practical Issues

There are many practical variants of Newton's method:

- Modify the Hessian to be positive-definite.
- Only compute the Hessian every  $m$  iterations.
- Only use the diagonals of the Hessian.
- **Quasi-Newton**: Update a (diagonal plus low-rank) approximation of the Hessian (BFGS, [L-BFGS](#)).
- **Hessian-free**: Compute  $d$  inexactly using Hessian-vector products:

$$\nabla^2 f(x)d = \lim_{\delta \rightarrow 0} \frac{\nabla f(x + \delta d) - \nabla f(x)}{\delta}$$

- **Barzilai-Borwein**: Choose a step-size that acts like the Hessian over the last iteration:

$$\alpha = \frac{(x^+ - x)^T (\nabla f(x^+) - \nabla f(x))}{\|\nabla f(x^+) - \nabla f(x)\|^2}$$

Another related method is [nonlinear conjugate gradient](#).

# Outline

- 1 Convex Functions
- 2 Smooth Optimization
- 3 Non-Smooth Optimization
- 4 Randomized Algorithms
- 5 Parallel/Distributed Optimization

## Motivation: Sparse Regularization

- Consider  $\ell_1$ -regularized optimization problems,

$$\min_x f(x) + \lambda \|x\|_1,$$

where  $f$  is differentiable.

- For example,  $\ell_1$ -regularized least squares,

$$\min_x \|Ax - b\|^2 + \lambda \|x\|_1$$

- Regularizes and encourages sparsity in  $x$

## Motivation: Sparse Regularization

- Consider  $\ell_1$ -regularized optimization problems,

$$\min_x f(x) + \lambda \|x\|_1,$$

where  $f$  is differentiable.

- For example,  $\ell_1$ -regularized least squares,

$$\min_x \|Ax - b\|^2 + \lambda \|x\|_1$$

- Regularizes and encourages sparsity in  $x$
- The objective is non-differentiable when any  $x_i = 0$ .
- How can we solve non-smooth convex optimization problems?

## Sub-Gradients and Sub-Differentials

Recall that for *differentiable* convex functions we have

$$f(y) \geq f(x) + \nabla f(x)^T(y - x), \forall x, y.$$

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

$$f(y) \geq f(x) + d^T(y - x), \forall y.$$

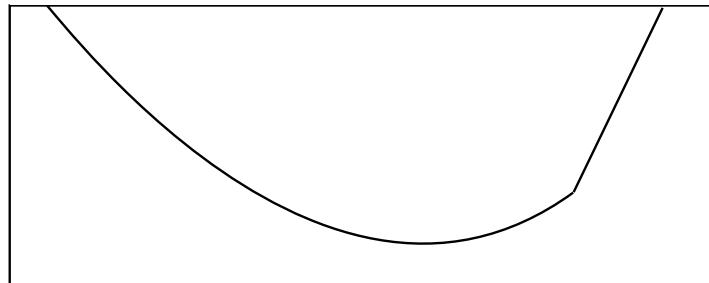
## Sub-Gradients and Sub-Differentials

Recall that for *differentiable* convex functions we have

$$f(y) \geq f(x) + \nabla f(x)^T(y - x), \forall x, y.$$

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

$$f(y) \geq f(x) + d^T(y - x), \forall y.$$



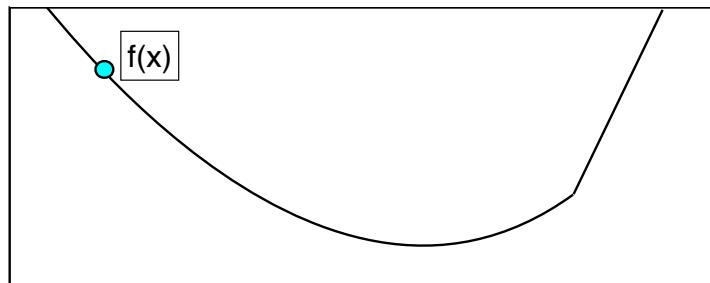
## Sub-Gradients and Sub-Differentials

Recall that for *differentiable* convex functions we have

$$f(y) \geq f(x) + \nabla f(x)^T(y - x), \forall x, y.$$

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

$$f(y) \geq f(x) + d^T(y - x), \forall y.$$



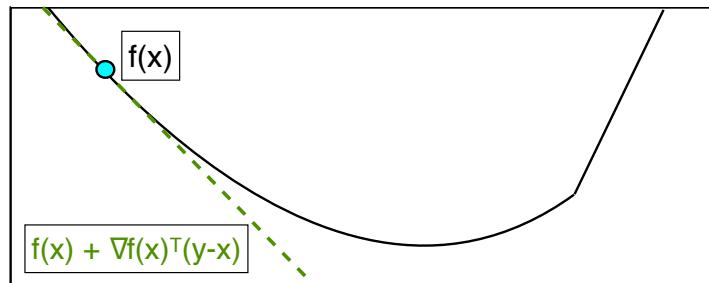
## Sub-Gradients and Sub-Differentials

Recall that for *differentiable* convex functions we have

$$f(y) \geq f(x) + \nabla f(x)^T(y - x), \forall x, y.$$

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

$$f(y) \geq f(x) + d^T(y - x), \forall y.$$



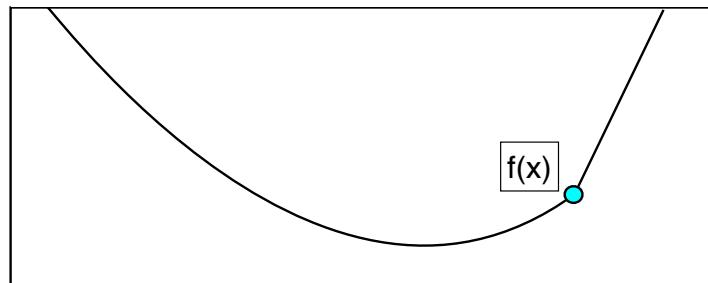
## Sub-Gradients and Sub-Differentials

Recall that for *differentiable* convex functions we have

$$f(y) \geq f(x) + \nabla f(x)^T(y - x), \forall x, y.$$

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

$$f(y) \geq f(x) + d^T(y - x), \forall y.$$



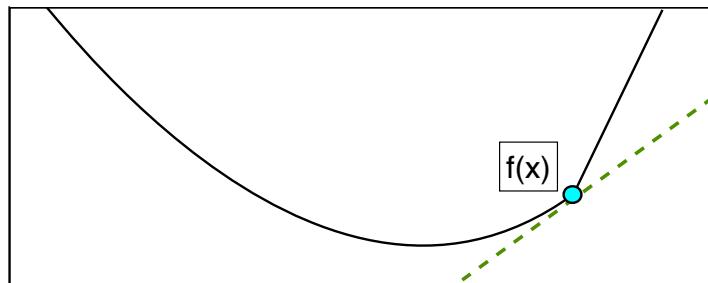
## Sub-Gradients and Sub-Differentials

Recall that for *differentiable* convex functions we have

$$f(y) \geq f(x) + \nabla f(x)^T(y - x), \forall x, y.$$

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

$$f(y) \geq f(x) + d^T(y - x), \forall y.$$



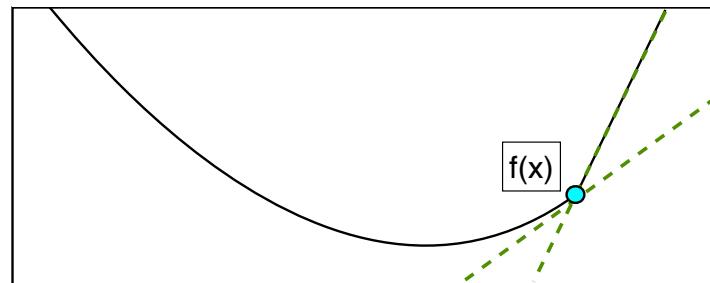
## Sub-Gradients and Sub-Differentials

Recall that for *differentiable* convex functions we have

$$f(y) \geq f(x) + \nabla f(x)^T(y - x), \forall x, y.$$

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

$$f(y) \geq f(x) + d^T(y - x), \forall y.$$



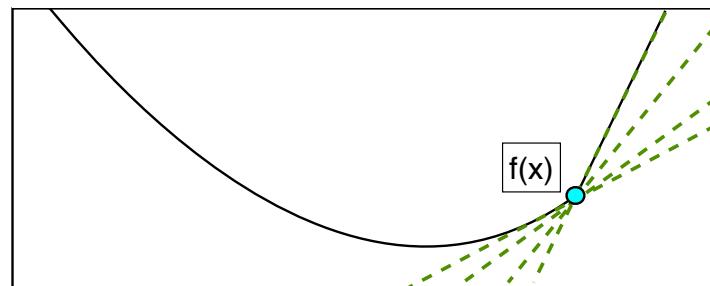
## Sub-Gradients and Sub-Differentials

Recall that for *differentiable* convex functions we have

$$f(y) \geq f(x) + \nabla f(x)^T(y - x), \forall x, y.$$

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

$$f(y) \geq f(x) + d^T(y - x), \forall y.$$



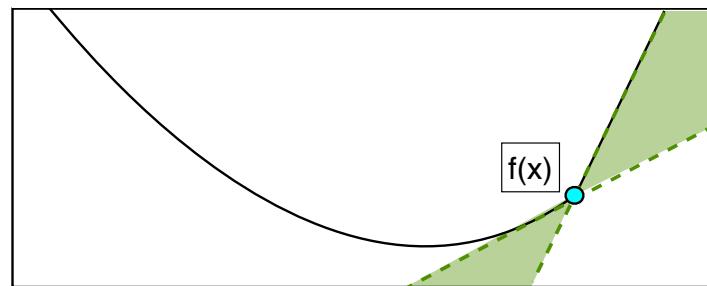
## Sub-Gradients and Sub-Differentials

Recall that for *differentiable* convex functions we have

$$f(y) \geq f(x) + \nabla f(x)^T(y - x), \forall x, y.$$

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

$$f(y) \geq f(x) + d^T(y - x), \forall y.$$



## Sub-Gradients and Sub-Differentials

Recall that for *differentiable* convex functions we have

$$f(y) \geq f(x) + \nabla f(x)^T(y - x), \forall x, y.$$

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

$$f(y) \geq f(x) + d^T(y - x), \forall y.$$

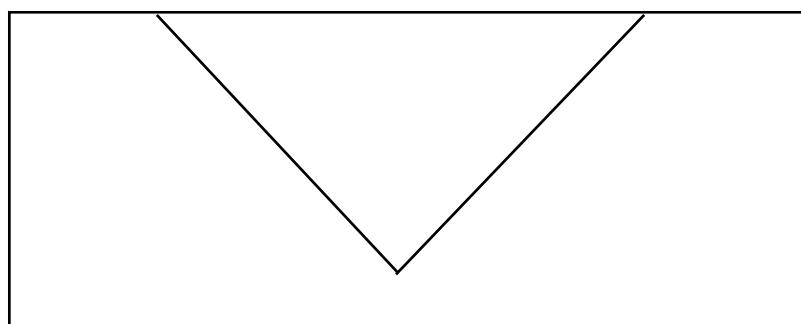
- $f$  is differentiable at  $x$  iff  $\nabla f(x)$  is the only subgradient.
- At non-differentiable  $x$ , we have a set of subgradients.
- Set of subgradients is the *sub-differential*  $\partial f(x)$ .
- Note that  $0 \in \partial f(x)$  iff  $x$  is a global minimum.

## Sub-Differential of Absolute Value and Max Functions

- The sub-differential of the absolute value function:

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

(sign of the variable if non-zero, anything in  $[-1, 1]$  at 0)

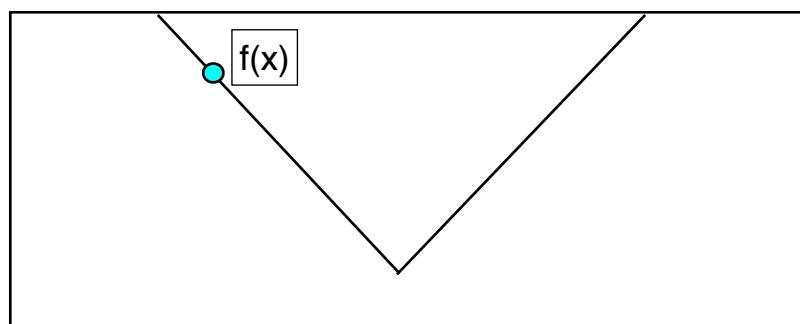


## Sub-Differential of Absolute Value and Max Functions

- The sub-differential of the absolute value function:

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

(sign of the variable if non-zero, anything in  $[-1, 1]$  at 0)

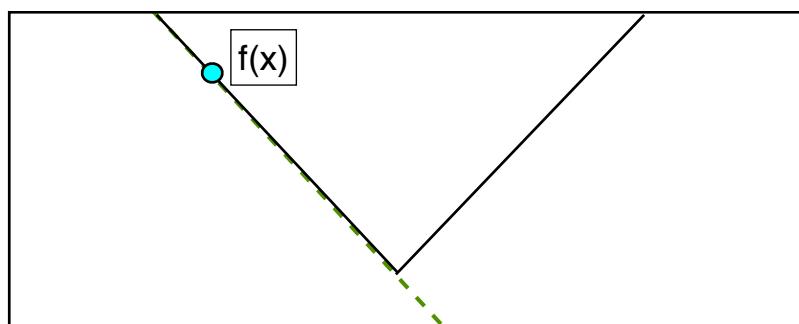


## Sub-Differential of Absolute Value and Max Functions

- The sub-differential of the absolute value function:

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

(sign of the variable if non-zero, anything in  $[-1, 1]$  at 0)

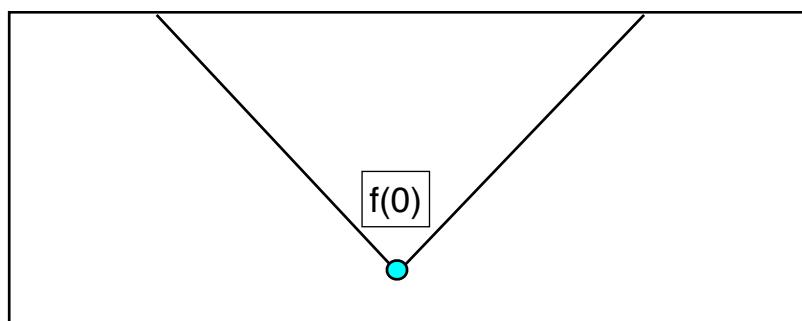


## Sub-Differential of Absolute Value and Max Functions

- The sub-differential of the absolute value function:

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

(sign of the variable if non-zero, anything in  $[-1, 1]$  at 0)

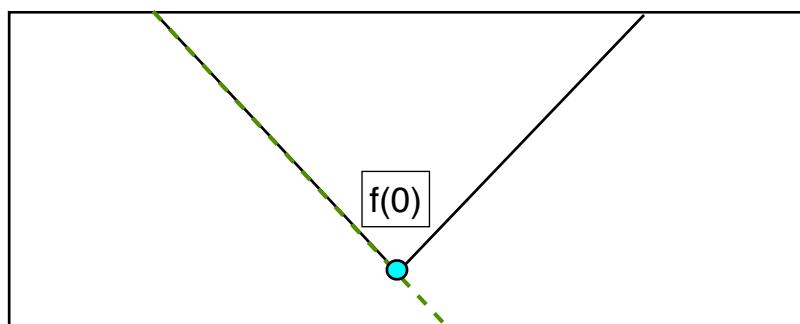


## Sub-Differential of Absolute Value and Max Functions

- The sub-differential of the absolute value function:

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

(sign of the variable if non-zero, anything in  $[-1, 1]$  at 0)

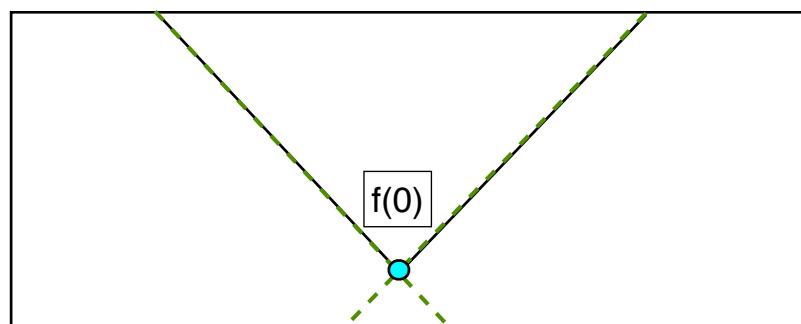


## Sub-Differential of Absolute Value and Max Functions

- The sub-differential of the absolute value function:

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

(sign of the variable if non-zero, anything in  $[-1, 1]$  at 0)

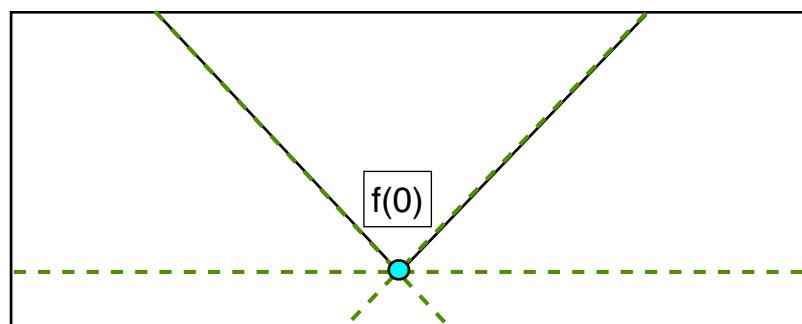


## Sub-Differential of Absolute Value and Max Functions

- The sub-differential of the absolute value function:

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

(sign of the variable if non-zero, anything in  $[-1, 1]$  at 0)

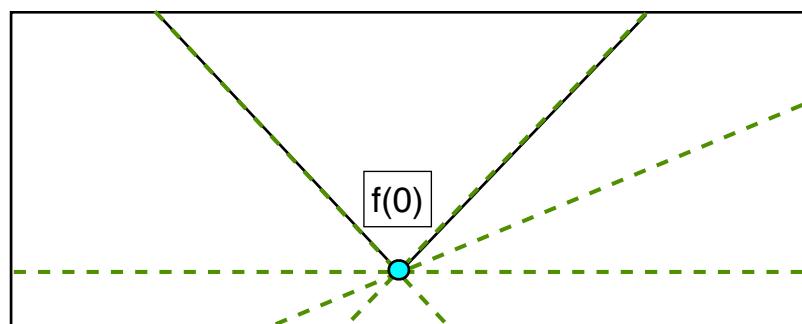


## Sub-Differential of Absolute Value and Max Functions

- The sub-differential of the absolute value function:

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

(sign of the variable if non-zero, anything in  $[-1, 1]$  at 0)

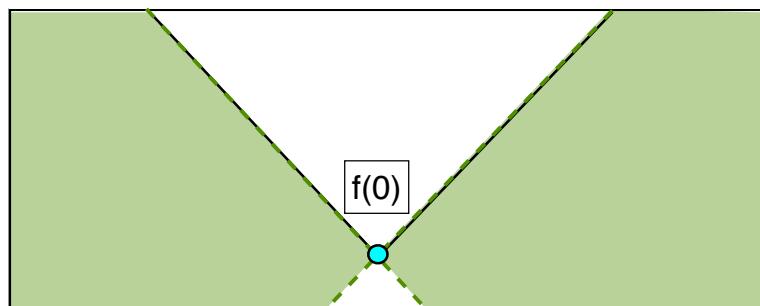


## Sub-Differential of Absolute Value and Max Functions

- The sub-differential of the absolute value function:

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

(sign of the variable if non-zero, anything in  $[-1, 1]$  at 0)



## Sub-Differential of Absolute Value and Max Functions

- The sub-differential of the absolute value function:

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

(sign of the variable if non-zero, anything in  $[-1, 1]$  at 0)

- The sub-differential of the maximum of differentiable  $f_i$ :

$$\partial \max\{f_1(x), f_2(x)\} = \begin{cases} \nabla f_1(x) & f_1(x) > f_2(x) \\ \nabla f_2(x) & f_2(x) > f_1(x) \\ \theta \nabla f_1(x) + (1 - \theta) \nabla f_2(x) & f_1(x) = f_2(x) \end{cases}$$

(any convex combination of the gradients of the argmax)

## Sub-gradient method

- The **sub-gradient method**:

$$x^+ = x - \alpha d,$$

for some  $d \in \partial f(x)$ .

## Sub-gradient method

- The **sub-gradient method**:

$$x^+ = x - \alpha d,$$

for some  $d \in \partial f(x)$ .

- The *steepest descent* step is given by  $\arg \min_{d \in \partial f(x)} \{\|d\|\}$ .  
(often hard to compute, but **easy for  $\ell_1$ -regularization**)

## Sub-gradient method

- The **sub-gradient method**:

$$x^+ = x - \alpha d,$$

for some  $d \in \partial f(x)$ .

- The *steepest descent* step is given by  $\arg \min_{d \in \partial f(x)} \{\|d\|\}$ .  
(often hard to compute, but **easy for  $\ell_1$ -regularization**)
- Otherwise, may **increase** the objective even for small  $\alpha$ .
- But  $\|x^+ - x^*\| \leq \|x - x^*\|$  for small enough  $\alpha$ .
- For convergence, we require  $\alpha \rightarrow 0$ .

## Sub-gradient method

- The **sub-gradient method**:

$$x^+ = x - \alpha d,$$

for some  $d \in \partial f(x)$ .

- The *steepest descent* step is given by  $\arg \min_{d \in \partial f(x)} \{\|d\|\}$ .  
(often hard to compute, but **easy for  $\ell_1$ -regularization**)
- Otherwise, may **increase** the objective even for small  $\alpha$ .
- But  $\|x^+ - x^*\| \leq \|x - x^*\|$  for small enough  $\alpha$ .
- For convergence, we require  $\alpha \rightarrow 0$ .
- Many variants average the iterations:

$$\bar{x}^k = \sum_{i=0}^{k-1} w_i x^i.$$

- Many variants average the gradients ('dual averaging'):

$$\bar{d}^k = \sum_{i=0}^{k-1} w_i d^i.$$

## Convex Optimization Zoo

Algorithm	Assumptions	Rate
Subgradient	Convex	$O(1/\sqrt{t})$
Gradient	Convex	$O(1/t)$

## Convex Optimization Zoo

Algorithm	Assumptions	Rate
Subgradient	Convex	$O(1/\sqrt{t})$
Gradient	Convex	$O(1/t)$
Subgradient	Strongly-Convex	$O(1/t)$
Gradient	Strongly-Convex	$O((1 - \mu/L)^t)$

## Convex Optimization Zoo

Algorithm	Assumptions	Rate
Subgradient	Convex	$O(1/\sqrt{t})$
Gradient	Convex	$O(1/t)$
Subgradient	Strongly-Convex	$O(1/t)$
Gradient	Strongly-Convex	$O((1 - \mu/L)^t)$

- Alternative is cutting-plane/bundle methods:
  - Minimize an approximation based on *all* subgradients  $\{d_t\}$ .
  - But have the *same rates as the subgradient method*.

(tend to be better in practice)

## Convex Optimization Zoo

Algorithm	Assumptions	Rate
Subgradient	Convex	$O(1/\sqrt{t})$
Gradient	Convex	$O(1/t)$
Subgradient	Strongly-Convex	$O(1/t)$
Gradient	Strongly-Convex	$O((1 - \mu/L)^t)$

- Alternative is cutting-plane/bundle methods:
  - Minimize an approximation based on *all* subgradients  $\{d_t\}$ .
  - But have the *same rates as the subgradient method*.  
(tend to be better in practice)
- Bad news: **Rates are optimal for black-box methods.**

## Convex Optimization Zoo

Algorithm	Assumptions	Rate
Subgradient	Convex	$O(1/\sqrt{t})$
Gradient	Convex	$O(1/t)$
Subgradient	Strongly-Convex	$O(1/t)$
Gradient	Strongly-Convex	$O((1 - \mu/L)^t)$

- Alternative is cutting-plane/bundle methods:
  - Minimize an approximation based on *all* subgradients  $\{d_t\}$ .
  - But have the *same rates as the subgradient method*.  
(tend to be better in practice)
- Bad news: **Rates are optimal for black-box methods.**
- But, we often have more than a black-box.

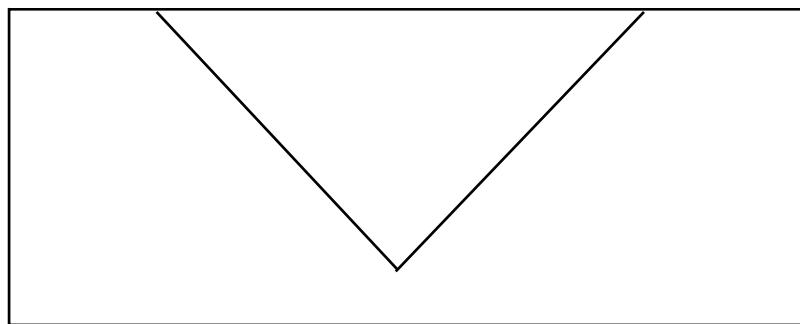
## Smoothing Approximations of Non-Smooth Functions

- Smoothing: replace non-smooth  $f$  with smooth  $f_\epsilon$ .
- Apply a fast method for smooth optimization.

## Smoothing Approximations of Non-Smooth Functions

- Smoothing: replace non-smooth  $f$  with smooth  $f_\epsilon$ .
- Apply a fast method for smooth optimization.
- Smooth approximation to the absolute value:

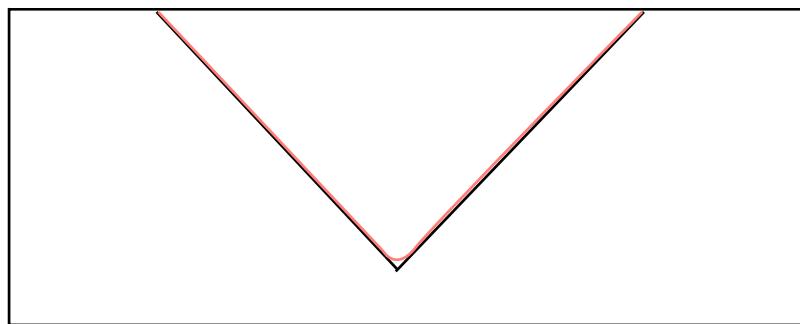
$$|x| \approx \sqrt{x^2 + \nu}.$$



## Smoothing Approximations of Non-Smooth Functions

- Smoothing: replace non-smooth  $f$  with smooth  $f_\epsilon$ .
- Apply a fast method for smooth optimization.
- Smooth approximation to the absolute value:

$$|x| \approx \sqrt{x^2 + \nu}.$$



## Smoothing Approximations of Non-Smooth Functions

- Smoothing: replace non-smooth  $f$  with smooth  $f_\epsilon$ .
- Apply a fast method for smooth optimization.
- Smooth approximation to the absolute value:

$$|x| \approx \sqrt{x^2 + \nu}.$$

- Smooth approximation to the max function:

$$\max\{a, b\} \approx \log(\exp(a) + \exp(b))$$

- Smooth approximation to the hinge loss:

$$\max\{0, x\} \approx \begin{cases} 0 & x \geq 1 \\ 1 - x^2 & t < x < 1 \\ (1 - t)^2 + 2(1 - t)(t - x) & x \leq t \end{cases}$$

## Smoothing Approximations of Non-Smooth Functions

- Smoothing: replace non-smooth  $f$  with smooth  $f_\epsilon$ .
- Apply a fast method for smooth optimization.
- Smooth approximation to the absolute value:

$$|x| \approx \sqrt{x^2 + \nu}.$$

- Smooth approximation to the max function:

$$\max\{a, b\} \approx \log(\exp(a) + \exp(b))$$

- Smooth approximation to the hinge loss:

$$\max\{0, x\} \approx \begin{cases} 0 & x \geq 1 \\ 1 - x^2 & t < x < 1 \\ (1 - t)^2 + 2(1 - t)(t - x) & x \leq t \end{cases}$$

- Generic smoothing strategy: strongly-convex regularization of convex conjugate.[Nesterov, 2005]

## Convex Optimization Zoo

Algorithm	Assumptions	Rate
Subgradient	Convex	$O(1/\sqrt{t})$
Gradient	Convex	$O(1/t)$
Nesterov	Convex	$O(1/t^2)$

## Convex Optimization Zoo

Algorithm	Assumptions	Rate
Subgradient	Convex	$O(1/\sqrt{t})$
Gradient	Convex	$O(1/t)$
Nesterov	Convex	$O(1/t^2)$
Gradient	Smoothed to $1/\epsilon$ , Convex	$O(1/\sqrt{t})$

## Convex Optimization Zoo

Algorithm	Assumptions	Rate
Subgradient	Convex	$O(1/\sqrt{t})$
Gradient	Convex	$O(1/t)$
Nesterov	Convex	$O(1/t^2)$
Gradient Nesterov	Smoothed to $1/\epsilon$ , Convex	$O(1/\sqrt{t})$
Nesterov	Smoothed to $1/\epsilon$ , Convex	$O(1/t)$

## Convex Optimization Zoo

Algorithm	Assumptions	Rate
Subgradient	Convex	$O(1/\sqrt{t})$
Gradient	Convex	$O(1/t)$
Nesterov	Convex	$O(1/t^2)$
Gradient Nesterov	Smoothed to $1/\epsilon$ , Convex	$O(1/\sqrt{t})$
Nesterov	Smoothed to $1/\epsilon$ , Convex	$O(1/t)$

- Smoothing is only faster if you use Nesterov's method.
- In practice, faster to slowly decrease smoothing level.
- You can get the  $O(1/t)$  rate for  $\min_x \max_i \{f_i(x)\}$  for  $f_i$  convex and smooth using *mirror-prox* method.[Nemirovski, 2004]

## Converting to Constrained Optimization

- Re-write non-smooth problem as constrained problem.

## Converting to Constrained Optimization

- Re-write non-smooth problem as constrained problem.
- The problem

$$\min_x f(x) + \lambda \|x\|_1,$$

is equivalent to the problem

$$\min_{x^+ \geq 0, x^- \geq 0} f(x^+ - x^-) + \lambda \sum_i (x_i^+ + x_i^-),$$

or the problems

$$\min_{-y \leq x \leq y} f(x) + \lambda \sum_i y_i, \quad \min_{\|x\|_1 \leq \gamma} f(x) + \lambda \gamma$$

## Converting to Constrained Optimization

- Re-write non-smooth problem as constrained problem.
- The problem

$$\min_x f(x) + \lambda \|x\|_1,$$

is equivalent to the problem

$$\min_{x^+ \geq 0, x^- \geq 0} f(x^+ - x^-) + \lambda \sum_i (x_i^+ + x_i^-),$$

or the problems

$$\min_{-y \leq x \leq y} f(x) + \lambda \sum_i y_i, \quad \min_{\|x\|_1 \leq \gamma} f(x) + \lambda \gamma$$

- These are **smooth objective with ‘simple’ constraints**.

$$\min_{x \in \mathcal{C}} f(x).$$

## Optimization with Simple Constraints

- Recall: gradient descent minimizes quadratic approximation:

$$x^+ = \arg \min_y \left\{ f(x) + \nabla f(x)^T (y - x) + \frac{1}{2\alpha} \|y - x\|^2 \right\}.$$

## Optimization with Simple Constraints

- Recall: gradient descent minimizes quadratic approximation:

$$x^+ = \arg \min_y \left\{ f(x) + \nabla f(x)^T (y - x) + \frac{1}{2\alpha} \|y - x\|^2 \right\}.$$

- Consider minimizing subject to simple constraints:

$$x^+ = \arg \min_{y \in \mathcal{C}} \left\{ f(x) + \nabla f(x)^T (y - x) + \frac{1}{2\alpha} \|y - x\|^2 \right\}.$$

## Optimization with Simple Constraints

- Recall: gradient descent minimizes quadratic approximation:

$$x^+ = \arg \min_y \left\{ f(x) + \nabla f(x)^T(y - x) + \frac{1}{2\alpha} \|y - x\|^2 \right\}.$$

- Consider minimizing subject to simple constraints:

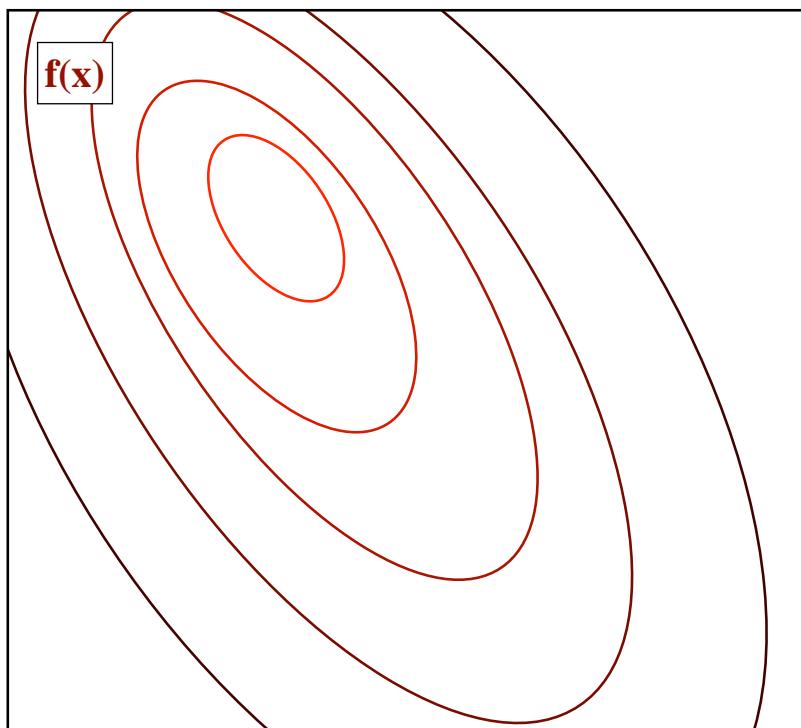
$$x^+ = \arg \min_{y \in \mathcal{C}} \left\{ f(x) + \nabla f(x)^T(y - x) + \frac{1}{2\alpha} \|y - x\|^2 \right\}.$$

- Equivalent to **projection** of gradient descent:

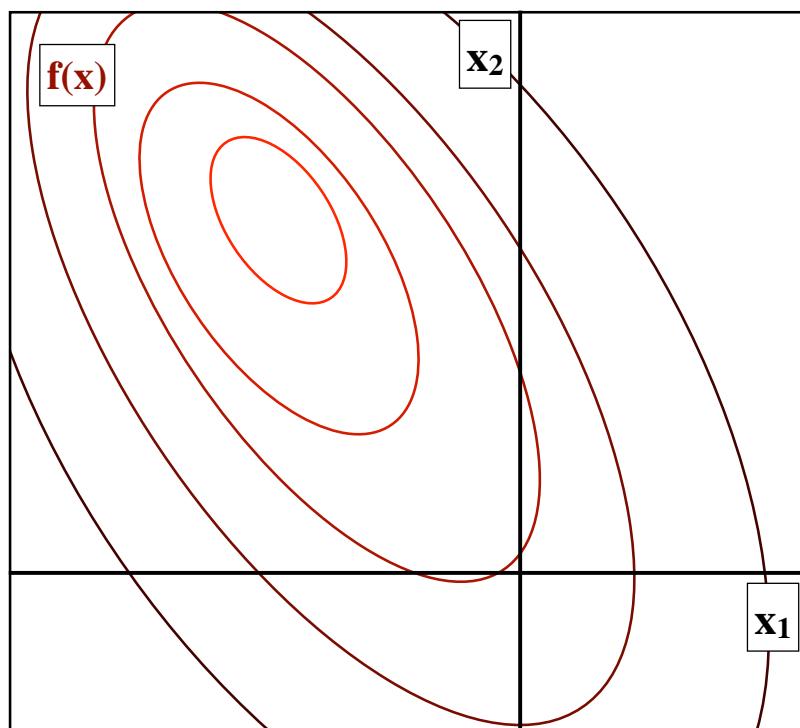
$$x^{GD} = x - \alpha \nabla f(x),$$

$$x^+ = \arg \min_{y \in \mathcal{C}} \left\{ \|y - x^{GD}\| \right\},$$

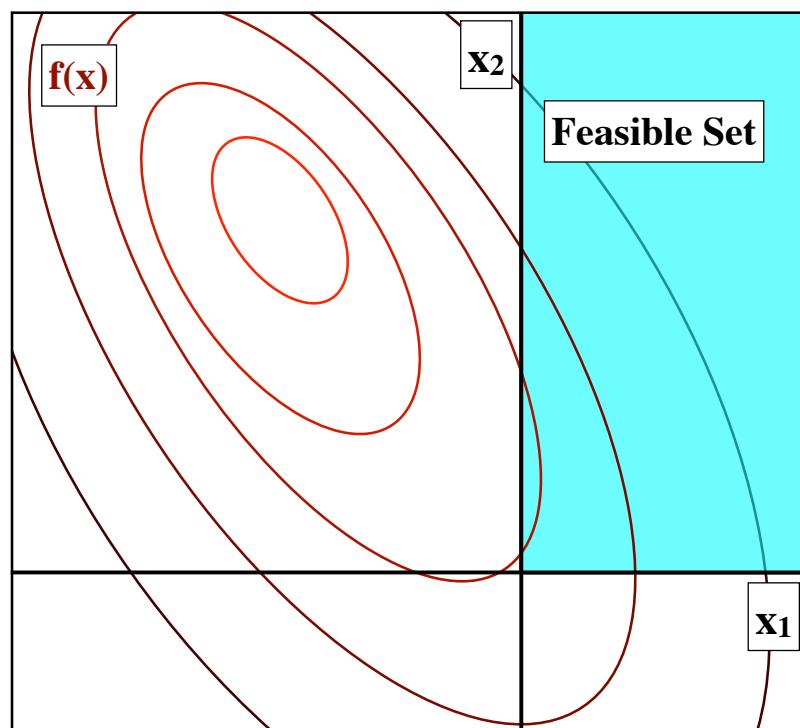
## Gradient Projection



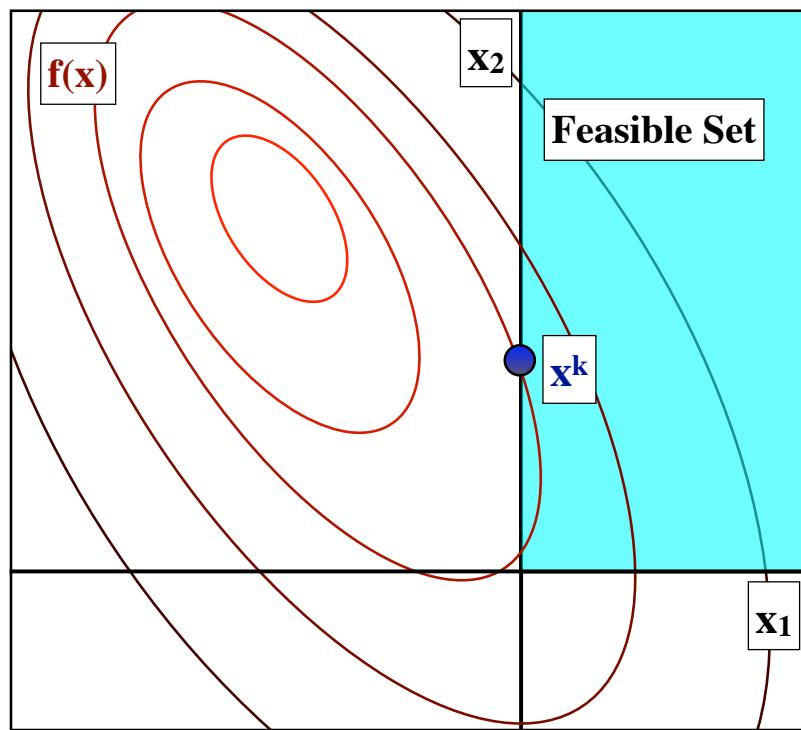
## Gradient Projection



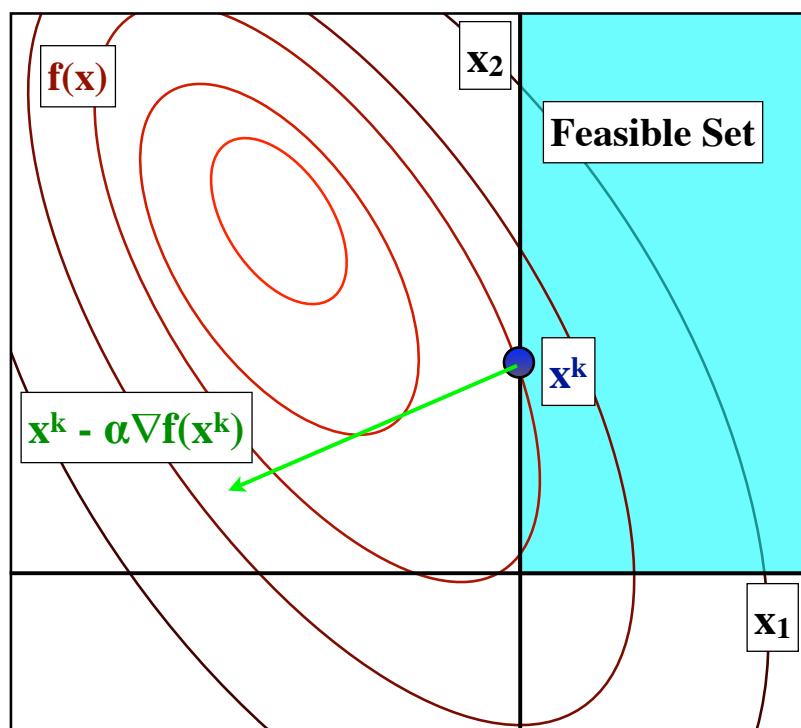
## Gradient Projection



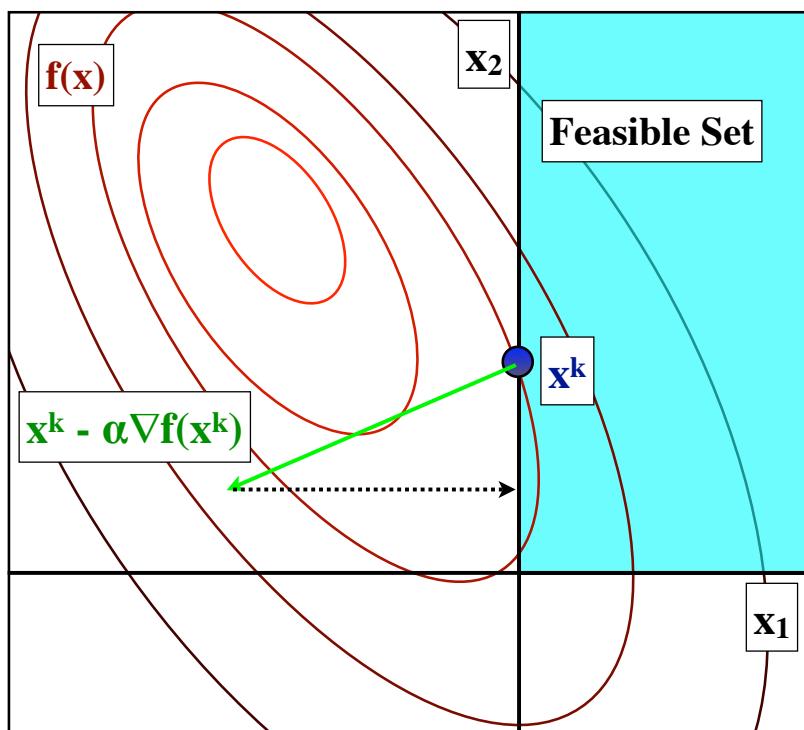
## Gradient Projection



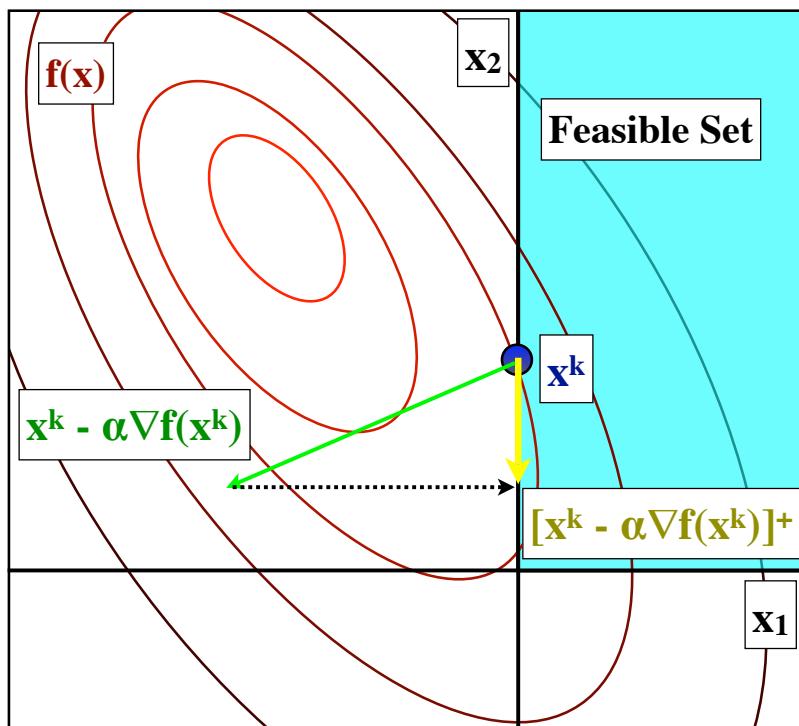
## Gradient Projection



## Gradient Projection



## Gradient Projection



## Projection Onto Simple Sets

Projections onto simple sets:

- $\arg \min_{y \geq 0} \|y - x\| = \max\{x, 0\}$

## Projection Onto Simple Sets

Projections onto simple sets:

- $\arg \min_{y \geq 0} \|y - x\| = \max\{x, 0\}$
- $\arg \min_{l \leq y \leq u} \|y - x\| = \max\{l, \min\{x, u\}\}$

## Projection Onto Simple Sets

Projections onto simple sets:

- $\arg \min_{y \geq 0} \|y - x\| = \max\{x, 0\}$
- $\arg \min_{l \leq y \leq u} \|y - x\| = \max\{l, \min\{x, u\}\}$
- $\arg \min_{a^T y = b} \|y - x\| = x + (b - a^T x)a/\|a\|^2.$

## Projection Onto Simple Sets

Projections onto simple sets:

- $\arg \min_{y \geq 0} \|y - x\| = \max\{x, 0\}$
- $\arg \min_{l \leq y \leq u} \|y - x\| = \max\{l, \min\{x, u\}\}$
- $\arg \min_{a^T y = b} \|y - x\| = x + (b - a^T x)a/\|a\|^2.$
- $\arg \min_{a^T y \geq b} \|y - x\| = \begin{cases} x & a^T x \geq b \\ x + (b - a^T x)a/\|a\|^2 & a^T x < b \end{cases}$

## Projection Onto Simple Sets

Projections onto simple sets:

- $\arg \min_{y \geq 0} \|y - x\| = \max\{x, 0\}$
- $\arg \min_{l \leq y \leq u} \|y - x\| = \max\{l, \min\{x, u\}\}$
- $\arg \min_{a^T y = b} \|y - x\| = x + (b - a^T x)a/\|a\|^2.$
- $\arg \min_{a^T y \geq b} \|y - x\| = \begin{cases} x & a^T x \geq b \\ x + (b - a^T x)a/\|a\|^2 & a^T x < b \end{cases}$
- $\arg \min_{\|y\| \leq \tau} \|y - x\| = \tau x / \|x\|.$

## Projection Onto Simple Sets

Projections onto simple sets:

- $\arg \min_{y \geq 0} \|y - x\| = \max\{x, 0\}$
- $\arg \min_{l \leq y \leq u} \|y - x\| = \max\{l, \min\{x, u\}\}$
- $\arg \min_{a^T y = b} \|y - x\| = x + (b - a^T x)a/\|a\|^2.$
- $\arg \min_{a^T y \geq b} \|y - x\| = \begin{cases} x & a^T x \geq b \\ x + (b - a^T x)a/\|a\|^2 & a^T x < b \end{cases}$
- $\arg \min_{\|y\| \leq \tau} \|y - x\| = \tau x / \|x\|.$
- Linear-time algorithm for  $\ell_1$ -norm  $\|y\|_1 \leq \tau$ .

## Projection Onto Simple Sets

Projections onto simple sets:

- $\arg \min_{y \geq 0} \|y - x\| = \max\{x, 0\}$
- $\arg \min_{l \leq y \leq u} \|y - x\| = \max\{l, \min\{x, u\}\}$
- $\arg \min_{a^T y = b} \|y - x\| = x + (b - a^T x)a/\|a\|^2.$
- $\arg \min_{a^T y \geq b} \|y - x\| = \begin{cases} x & a^T x \geq b \\ x + (b - a^T x)a/\|a\|^2 & a^T x < b \end{cases}$
- $\arg \min_{\|y\| \leq \tau} \|y - x\| = \tau x / \|x\|.$
- Linear-time algorithm for  $\ell_1$ -norm  $\|y\|_1 \leq \tau$ .
- Linear-time algorithm for probability simplex  $y \geq 0, \sum y = 1$ .

## Projection Onto Simple Sets

Projections onto simple sets:

- $\arg \min_{y \geq 0} \|y - x\| = \max\{x, 0\}$
- $\arg \min_{l \leq y \leq u} \|y - x\| = \max\{l, \min\{x, u\}\}$
- $\arg \min_{a^T y = b} \|y - x\| = x + (b - a^T x)a/\|a\|^2.$
- $\arg \min_{a^T y \geq b} \|y - x\| = \begin{cases} x & a^T x \geq b \\ x + (b - a^T x)a/\|a\|^2 & a^T x < b \end{cases}$
- $\arg \min_{\|y\| \leq \tau} \|y - x\| = \tau x / \|x\|.$
- Linear-time algorithm for  $\ell_1$ -norm  $\|y\|_1 \leq \tau$ .
- Linear-time algorithm for probability simplex  $y \geq 0, \sum y = 1$ .
- Intersection of simple sets: Dykstra's algorithm.

## Convex Optimization Zoo

Algorithm	Assumptions	Rate
P(Subgradient)	Convex	$O(1/\sqrt{t})$
P(Subgradient)	Strongly	$O(1/t)$
P(Nesterov)	Smoothed to $1/\epsilon$ , Convex	$O(1/t)$
P(Gradient)	Convex	$O(1/t)$
P(Nesterov)	Convex	$O(1/t^2)$
P(Gradient)	Strongly	$O((1 - \mu/L)^t)$
P(Nesterov)	Strongly	$O((1 - \sqrt{\mu/L})^t)$
P(Newton)	Strongly	$O(\prod_{i=1}^t \rho_t), \rho_t \rightarrow 0$

- Convergence rates are the same for projected versions!

## Convex Optimization Zoo

Algorithm	Assumptions	Rate
P(Subgradient)	Convex	$O(1/\sqrt{t})$
P(Subgradient)	Strongly	$O(1/t)$
P(Nesterov)	Smoothed to $1/\epsilon$ , Convex	$O(1/t)$
P(Gradient)	Convex	$O(1/t)$
P(Nesterov)	Convex	$O(1/t^2)$
P(Gradient)	Strongly	$O((1 - \mu/L)^t)$
P(Nesterov)	Strongly	$O((1 - \sqrt{\mu/L})^t)$
P(Newton)	Strongly	$O(\prod_{i=1}^t \rho_t), \rho_t \rightarrow 0$

- Convergence rates are the same for projected versions!
- Can do many of the same tricks (i.e. Armijo line-search, polynomial interpolation, Barzilai-Borwein, quasi-Newton).

## Convex Optimization Zoo

Algorithm	Assumptions	Rate
P(Subgradient)	Convex	$O(1/\sqrt{t})$
P(Subgradient)	Strongly	$O(1/t)$
P(Nesterov)	Smoothed to $1/\epsilon$ , Convex	$O(1/t)$
P(Gradient)	Convex	$O(1/t)$
P(Nesterov)	Convex	$O(1/t^2)$
P(Gradient)	Strongly	$O((1 - \mu/L)^t)$
P(Nesterov)	Strongly	$O((1 - \sqrt{\mu/L})^t)$
P(Newton)	Strongly	$O(\prod_{i=1}^t \rho_t), \rho_t \rightarrow 0$

- Convergence rates are the same for projected versions!
- Can do many of the same tricks (i.e. Armijo line-search, polynomial interpolation, Barzilai-Borwein, quasi-Newton).
- For Newton, you need to project under  $\|\cdot\|_{\nabla^2 f(x)}$   
(expensive, but special tricks for the case of simplex or lower/upper bounds)
- You don't need to compute the projection exactly.

## Proximal-Gradient Method

- A generalization of projected-gradient is [Proximal-gradient](#).
- The proximal-gradient method addresses problem of the form

$$\min_x f(x) + r(x),$$

where  $f$  is smooth but  $r$  is a general convex function.

## Proximal-Gradient Method

- A generalization of projected-gradient is **Proximal-gradient**.
- The proximal-gradient method addresses problem of the form

$$\min_x f(x) + r(x),$$

where  $f$  is smooth but  $r$  is a general convex function.

- Applies **proximity** operator of  $r$  to gradient descent on  $f$ :

$$x^{GD} = x - \alpha \nabla f(x),$$

$$x^+ = \arg \min_y \left\{ \frac{1}{2} \|y - x^{GD}\|^2 + \alpha r(y) \right\},$$

## Proximal-Gradient Method

- A generalization of projected-gradient is **Proximal-gradient**.
- The proximal-gradient method addresses problem of the form

$$\min_x f(x) + r(x),$$

where  $f$  is smooth but  $r$  is a general convex function.

- Applies **proximity** operator of  $r$  to gradient descent on  $f$ :

$$x^{GD} = x - \alpha \nabla f(x),$$

$$x^+ = \arg \min_y \left\{ \frac{1}{2} \|y - x^{GD}\|^2 + \alpha r(y) \right\},$$

- Equivalent to using the approximation

$$x^+ = \arg \min_y \left\{ f(x) + \nabla f(x)^T (y - x) + \frac{1}{2\alpha} \|y - x\|^2 + r(y) \right\}.$$

- Convergence rates are still the same as for minimizing  $f$ .

## Proximal Operator, Iterative Soft Thresholding

- The proximal operator is the solution to

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

## Proximal Operator, Iterative Soft Thresholding

- The proximal operator is the solution to

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

- For L1-regularization, we obtain iterative soft-thresholding:

$$x^+ = \text{softThresh}_{\alpha\lambda}[x - \alpha \nabla f(x)].$$

## Proximal Operator, Iterative Soft Thresholding

- The proximal operator is the solution to

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

- For L1-regularization, we obtain iterative soft-thresholding:

$$x^+ = \text{softThresh}_{\alpha\lambda}[x - \alpha \nabla f(x)].$$

- Example with  $\lambda = 1$ :

Input	Threshold	Soft-Threshold
$\begin{bmatrix} 0.6715 \\ -1.2075 \\ 0.7172 \\ 1.6302 \\ 0.4889 \end{bmatrix}$		

## Proximal Operator, Iterative Soft Thresholding

- The proximal operator is the solution to

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

- For L1-regularization, we obtain iterative soft-thresholding:

$$x^+ = \text{softThresh}_{\alpha\lambda}[x - \alpha \nabla f(x)].$$

- Example with  $\lambda = 1$ :

Input	Threshold	Soft-Threshold
$\begin{bmatrix} 0.6715 \\ -1.2075 \\ 0.7172 \\ 1.6302 \\ 0.4889 \end{bmatrix}$	$\begin{bmatrix} 0 \\ -1.2075 \\ 0 \\ 1.6302 \\ 0 \end{bmatrix}$	

## Proximal Operator, Iterative Soft Thresholding

- The proximal operator is the solution to

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

- For L1-regularization, we obtain iterative soft-thresholding:

$$x^+ = \text{softThresh}_{\alpha\lambda}[x - \alpha \nabla f(x)].$$

- Example with  $\lambda = 1$ :

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

## Special case of Projected-Gradient Methods

- Projected-gradient methods are another special case:

$$r(x) = \begin{cases} 0 & \text{if } x \in \mathcal{C} \\ \infty & \text{if } x \notin \mathcal{C} \end{cases},$$

## Special case of Projected-Gradient Methods

- Projected-gradient methods are another special case:

$$r(x) = \begin{cases} 0 & \text{if } x \in \mathcal{C} \\ \infty & \text{if } x \notin \mathcal{C} \end{cases},$$

gives

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

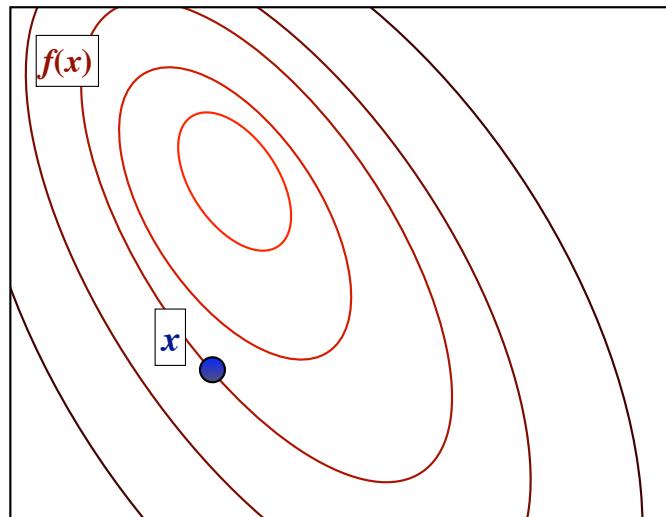
## Special case of Projected-Gradient Methods

- Projected-gradient methods are another special case:

$$r(x) = \begin{cases} 0 & \text{if } x \in \mathcal{C} \\ \infty & \text{if } x \notin \mathcal{C} \end{cases},$$

gives

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



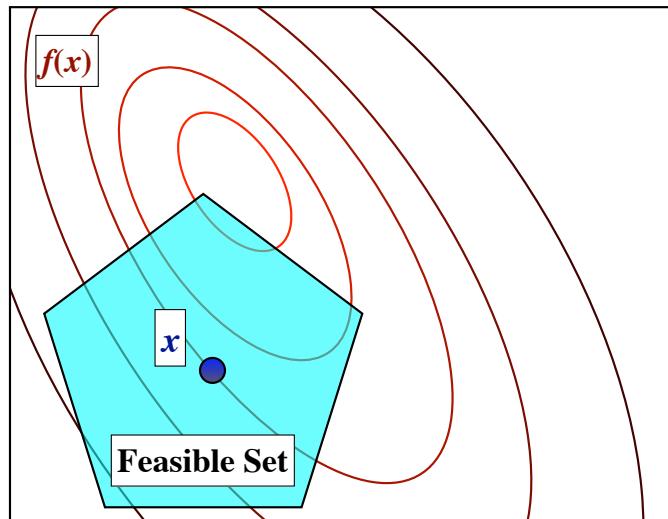
## Special case of Projected-Gradient Methods

- Projected-gradient methods are another special case:

$$r(x) = \begin{cases} 0 & \text{if } x \in \mathcal{C} \\ \infty & \text{if } x \notin \mathcal{C} \end{cases},$$

gives

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



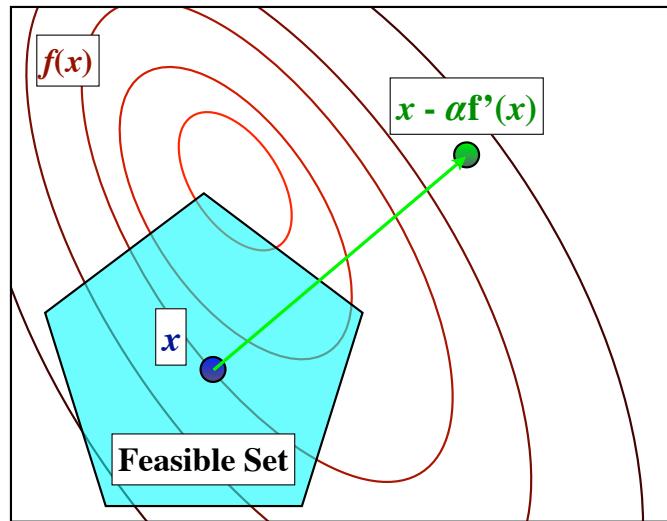
## Special case of Projected-Gradient Methods

- Projected-gradient methods are another special case:

$$r(x) = \begin{cases} 0 & \text{if } x \in \mathcal{C} \\ \infty & \text{if } x \notin \mathcal{C} \end{cases},$$

gives

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



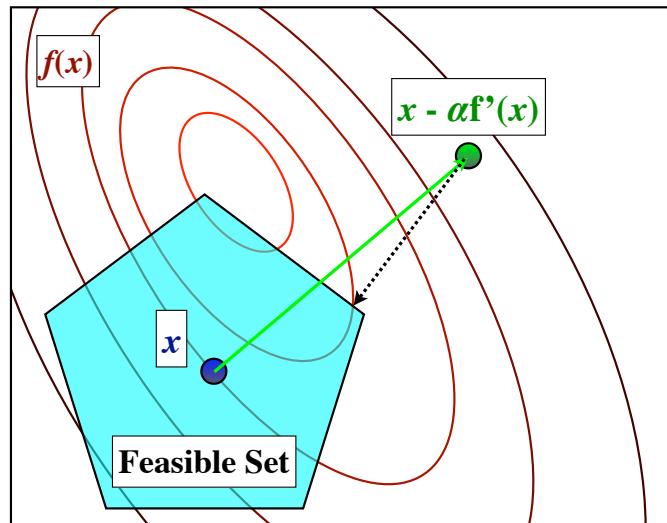
## Special case of Projected-Gradient Methods

- Projected-gradient methods are another special case:

$$r(x) = \begin{cases} 0 & \text{if } x \in \mathcal{C} \\ \infty & \text{if } x \notin \mathcal{C} \end{cases},$$

gives

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



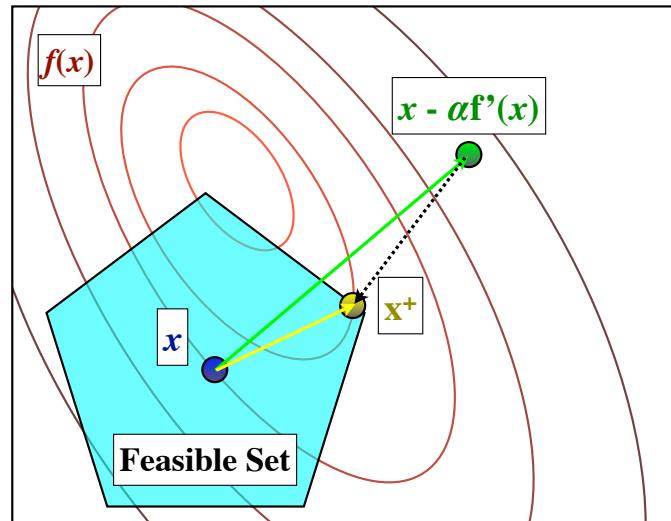
## Special case of Projected-Gradient Methods

- Projected-gradient methods are another special case:

$$r(x) = \begin{cases} 0 & \text{if } x \in \mathcal{C} \\ \infty & \text{if } x \notin \mathcal{C} \end{cases},$$

gives

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



## Exact Proximal-Gradient Methods

- For what problems can we apply these methods?

## Exact Proximal-Gradient Methods

- For what problems can we apply these methods?
- We can efficiently compute the proximity operator for:
  - ① L1-Regularization.

## Exact Proximal-Gradient Methods

- For what problems can we apply these methods?
- We can efficiently compute the proximity operator for:
  - 1 L1-Regularization.
  - 2 Group  $\ell_1$ -Regularization.

## Exact Proximal-Gradient Methods

- For what problems can we apply these methods?
- We can efficiently compute the proximity operator for:
  - 1 L1-Regularization.
  - 2 Group  $\ell_1$ -Regularization.
  - 3 Lower and upper bounds.

## Exact Proximal-Gradient Methods

- For what problems can we apply these methods?
- We can efficiently compute the proximity operator for:
  - ① L1-Regularization.
  - ② Group  $\ell_1$ -Regularization.
  - ③ Lower and upper bounds.
  - ④ Small number of linear constraint.

## Exact Proximal-Gradient Methods

- For what problems can we apply these methods?
- We can efficiently compute the proximity operator for:
  - 1 L1-Regularization.
  - 2 Group  $\ell_1$ -Regularization.
  - 3 Lower and upper bounds.
  - 4 Small number of linear constraint.
  - 5 Probability constraints.

## Exact Proximal-Gradient Methods

- For what problems can we apply these methods?
- We can efficiently compute the proximity operator for:
  - 1 L1-Regularization.
  - 2 Group  $\ell_1$ -Regularization.
  - 3 Lower and upper bounds.
  - 4 Small number of linear constraint.
  - 5 Probability constraints.
  - 6 A few other simple regularizers/constraints.

## Exact Proximal-Gradient Methods

- For what problems can we apply these methods?
- We can efficiently compute the proximity operator for:
  - 1 L1-Regularization.
  - 2 Group  $\ell_1$ -Regularization.
  - 3 Lower and upper bounds.
  - 4 Small number of linear constraint.
  - 5 Probability constraints.
  - 6 A few other simple regularizers/constraints.
- Can solve these non-smooth/constrained problems as fast as smooth/unconstrained problems!

## Exact Proximal-Gradient Methods

- For what problems can we apply these methods?
- We can efficiently compute the proximity operator for:
  - 1 L1-Regularization.
  - 2 Group  $\ell_1$ -Regularization.
  - 3 Lower and upper bounds.
  - 4 Small number of linear constraint.
  - 5 Probability constraints.
  - 6 A few other simple regularizers/constraints.
- Can solve these non-smooth/constrained problems as fast as smooth/unconstrained problems!
- But for many problems we **can not efficiently compute this operator.**

## Inexact Proximal-Gradient Methods

- We can efficiently **approximate** the proximity operator for:

## Inexact Proximal-Gradient Methods

- We can efficiently **approximate** the proximity operator for:
  - ① Structured sparsity.

## Inexact Proximal-Gradient Methods

- We can efficiently **approximate** the proximity operator for:
  - ① **Structured sparsity.**
  - ② Penalties on the differences between variables.

## Inexact Proximal-Gradient Methods

- We can efficiently **approximate** the proximity operator for:
  - ① Structured sparsity.
  - ② Penalties on the differences between variables.
  - ③ Regularizers and constraints on the singular values of matrices.

## Inexact Proximal-Gradient Methods

- We can efficiently **approximate** the proximity operator for:
  - ① Structured sparsity.
  - ② Penalties on the differences between variables.
  - ③ Regularizers and constraints on the singular values of matrices.
  - ④ Sums of simple functions.

## Inexact Proximal-Gradient Methods

- We can efficiently **approximate** the proximity operator for:
  - ① Structured sparsity.
  - ② Penalties on the differences between variables.
  - ③ Regularizers and constraints on the singular values of matrices.
  - ④ Sums of simple functions.
- Many recent works use **inexact proximal-gradient** methods:  
Cai et al. [2010], Liu & Ye [2010], Barbero & Sra [2011], Fadili & Peyré [2011],  
Ma et al. [2011]

## Inexact Proximal-Gradient Methods

- We can efficiently **approximate** the proximity operator for:
  - ① Structured sparsity.
  - ② Penalties on the differences between variables.
  - ③ Regularizers and constraints on the singular values of matrices.
  - ④ Sums of simple functions.
- Many recent works use **inexact proximal-gradient** methods:  
Cai et al. [2010], Liu & Ye [2010], Barbero & Sra [2011], Fadili & Peyré [2011],  
Ma et al. [2011]
- Do **inexact methods have the same rates?**

## Inexact Proximal-Gradient Methods

- We can efficiently **approximate** the proximity operator for:
  - ① Structured sparsity.
  - ② Penalties on the differences between variables.
  - ③ Regularizers and constraints on the singular values of matrices.
  - ④ Sums of simple functions.
- Many recent works use **inexact proximal-gradient** methods:  
Cai et al. [2010], Liu & Ye [2010], Barbero & Sra [2011], Fadili & Peyré [2011],  
Ma et al. [2011]
- Do **inexact methods have the same rates?**
  - *Yes, if the errors are appropriately controlled.* [Schmidt et al., 2011]

## Convergence Rate of Inexact Proximal-Gradient

**Proposition** [Schmidt et al., 2011] If the sequences of gradient errors  $\{||e_t||\}$  and proximal errors  $\{\sqrt{\varepsilon_t}\}$  are in  $\{O((1 - \mu/L)^t)\}$ , then the inexact proximal-gradient method has an error of  $O((1 - \mu/L)^t)$ .

## Convergence Rate of Inexact Proximal-Gradient

**Proposition** [Schmidt et al., 2011] If the sequences of gradient errors  $\{\|e_t\|\}$  and proximal errors  $\{\sqrt{\varepsilon_t}\}$  are in  $\{O((1 - \mu/L)^t)\}$ , then the inexact proximal-gradient method has an error of  $O((1 - \mu/L)^t)$ .

- Classic result as a special case (constants are good).
- The rates degrades gracefully if the errors are larger.
- Similar analyses in convex case.
- Huge improvement in practice over black-box methods.
- Also exist accelerated and spectral proximal-gradient methods.

## Discussion of Proximal-Gradient

- Theoretical justification for what works in practice.
- Significantly extends class of tractable problems.
- Many applications with inexact proximal operators:
  - Genomic expression, model predictive control, neuroimaging, satellite image fusion, simulating flow fields.

## Discussion of Proximal-Gradient

- Theoretical justification for what works in practice.
- Significantly extends class of tractable problems.
- Many applications with inexact proximal operators:
  - Genomic expression, model predictive control, neuroimaging, satellite image fusion, simulating flow fields.
- But, it assumes computing  $\nabla f(x)$  and  $\text{prox}_h[x]$  have similar cost.

## Discussion of Proximal-Gradient

- Theoretical justification for what works in practice.
- Significantly extends class of tractable problems.
- Many applications with inexact proximal operators:
  - Genomic expression, model predictive control, neuroimaging, satellite image fusion, simulating flow fields.
- But, it assumes computing  $\nabla f(x)$  and  $\text{prox}_h[x]$  have similar cost.
- Often  $\nabla f(x)$  is much more expensive:
  - We may have a large dataset.
  - Data-fitting term might be complex.
- Particularly true for structured output prediction:
  - Text, biological sequences, speech, images, matchings, graphs.

## Costly Data-Fitting Term, Simple Regularizer

- Consider fitting a **conditional random field** with  $\ell_1$ -regularization:

$$\min_{x \in \mathbb{R}^P} \quad \frac{1}{N} \sum_{i=1}^N f_i(x) + r(x)$$

costly smooth + simple

## Costly Data-Fitting Term, Simple Regularizer

- Consider fitting a **conditional random field** with  $\ell_1$ -regularization:

$$\min_{x \in \mathbb{R}^P} \quad \frac{1}{N} \sum_{i=1}^N f_i(x) + r(x)$$

costly smooth + simple

- Different than classic optimization (like linear programming).  
(cheap smooth plus complex non-smooth)

## Costly Data-Fitting Term, Simple Regularizer

- Consider fitting a **conditional random field** with  $\ell_1$ -regularization:

$$\min_{x \in \mathbb{R}^P} \quad \frac{1}{N} \sum_{i=1}^N f_i(x) + r(x)$$

costly smooth + simple

- Different than classic optimization (like linear programming).  
(cheap smooth plus complex non-smooth)
- Inspiration from the smooth case:
  - For smooth high-dimensional problems, **L-BFGS** quasi-Newton algorithm outperforms accelerated/spectral gradient methods.

## Quasi-Newton Methods

- Gradient method for optimizing a smooth  $f$ :

$$x^+ = x - \alpha \nabla f(x).$$

## Quasi-Newton Methods

- Gradient method for optimizing a smooth  $f$ :

$$x^+ = x - \alpha \nabla f(x).$$

- Newton-like methods alternatively use:

$$x^+ = x - \alpha \mathcal{H}^{-1} \nabla f(x).$$

- $\mathcal{H}$  approximates the second-derivative matrix.

## Quasi-Newton Methods

- Gradient method for optimizing a smooth  $f$ :

$$x^+ = x - \alpha \nabla f(x).$$

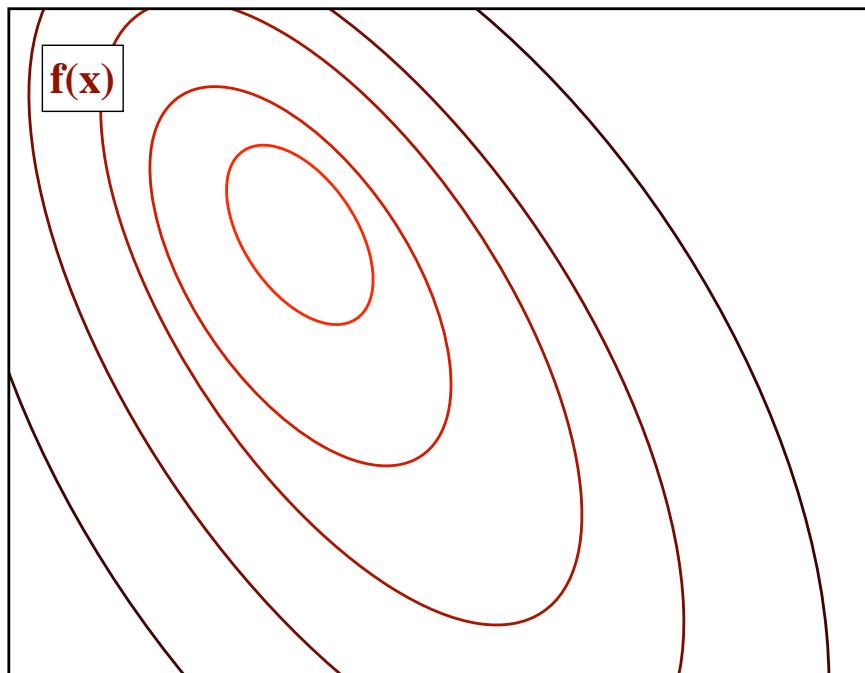
- Newton-like methods alternatively use:

$$x^+ = x - \alpha H^{-1} \nabla f(x).$$

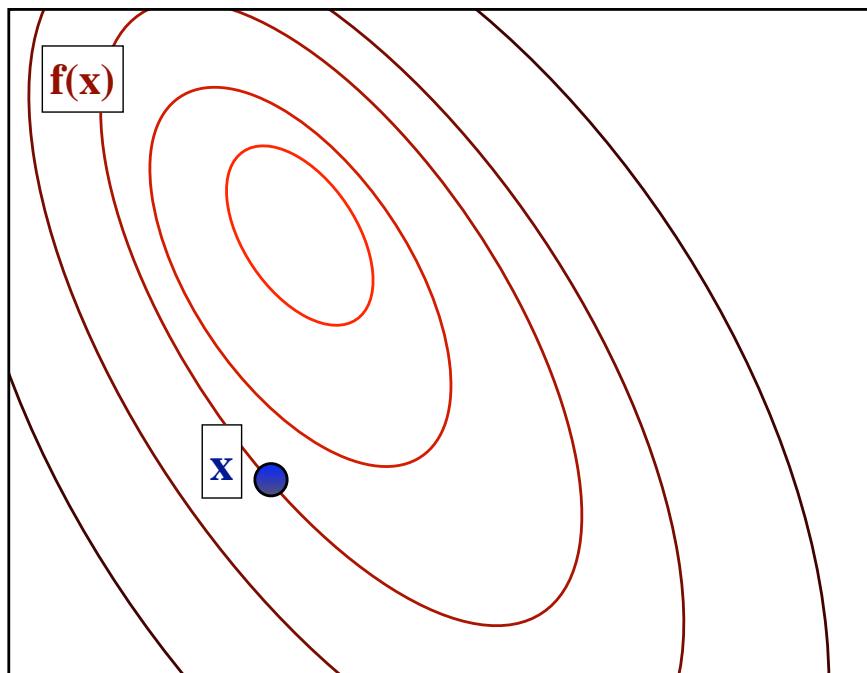
- $H$  approximates the second-derivative matrix.
- L-BFGS is a particular strategy to choose the  $H$  values:
  - Based on gradient differences.
  - Linear storage and linear time.

<http://www.di.ens.fr/~mschmidt/Software/minFunc.html>

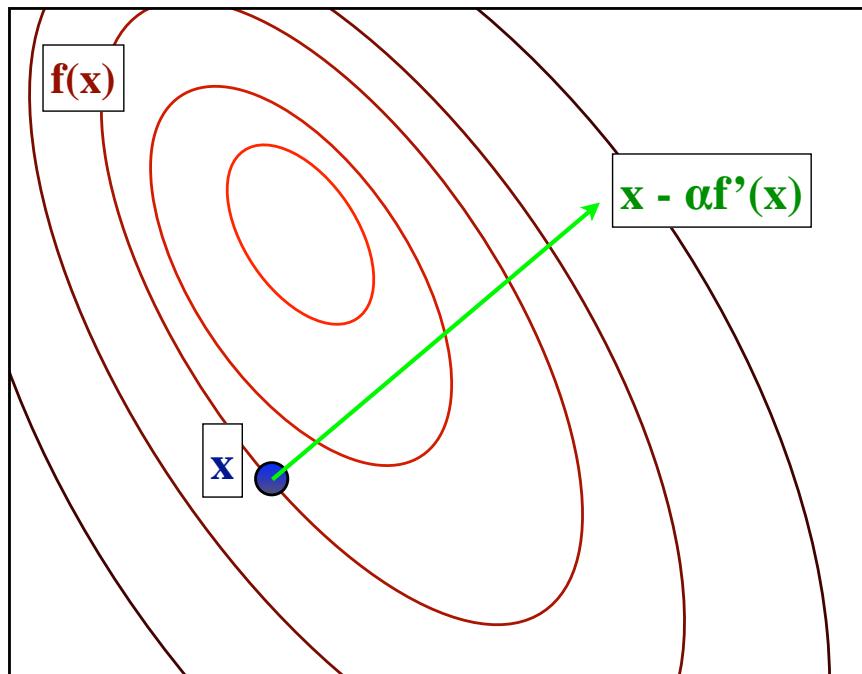
## Gradient Method and Newton's Method



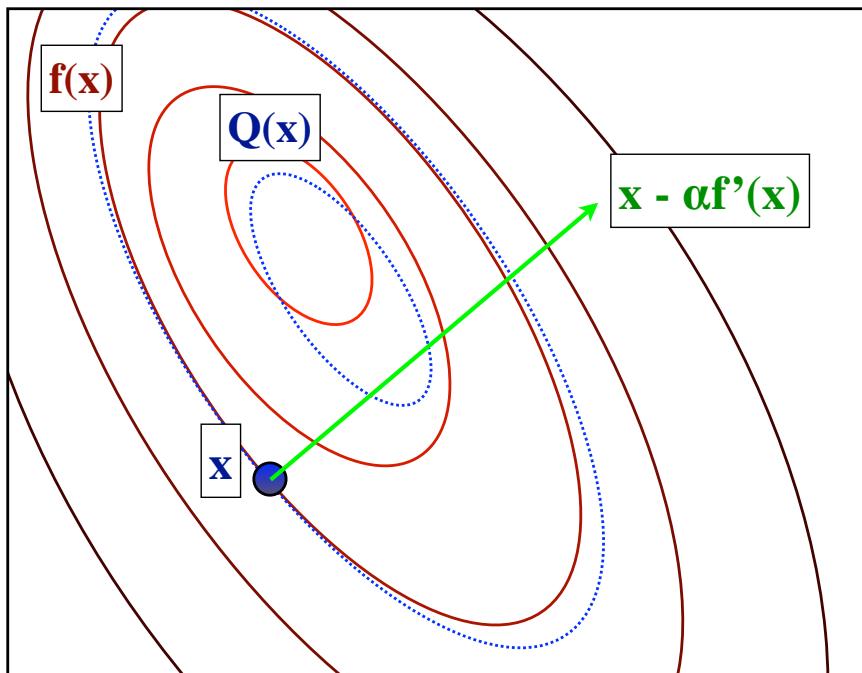
## Gradient Method and Newton's Method



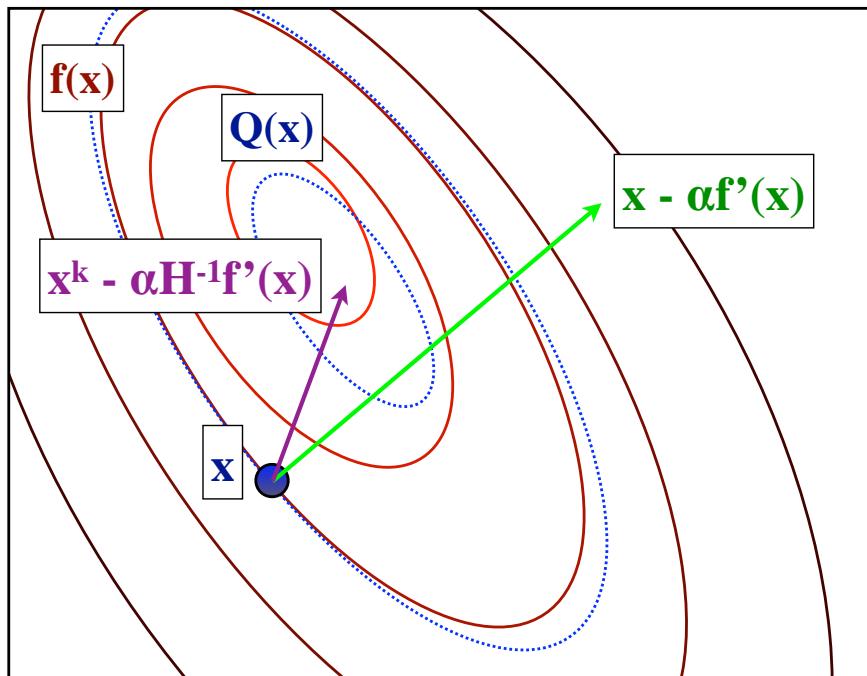
## Gradient Method and Newton's Method



## Gradient Method and Newton's Method



## Gradient Method and Newton's Method



## Naive Proximal Quasi-Newton Method

- Proximal-gradient method:

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

## Naive Proximal Quasi-Newton Method

- Proximal-gradient method:

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

- Can we just plug in the **Newton**-like step?

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

## Naive Proximal Quasi-Newton Method

- Proximal-gradient method:

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

- Can we just plug in the **Newton**-like step?

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

- NO!

## Naive Proximal Quasi-Newton Method

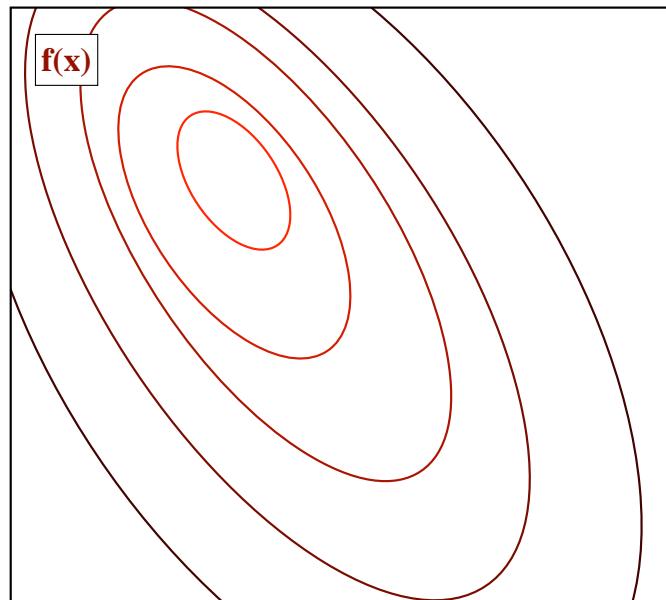
- Proximal-gradient method:

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

- Can we just plug in the **Newton**-like step?

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

- NO!



## Naive Proximal Quasi-Newton Method

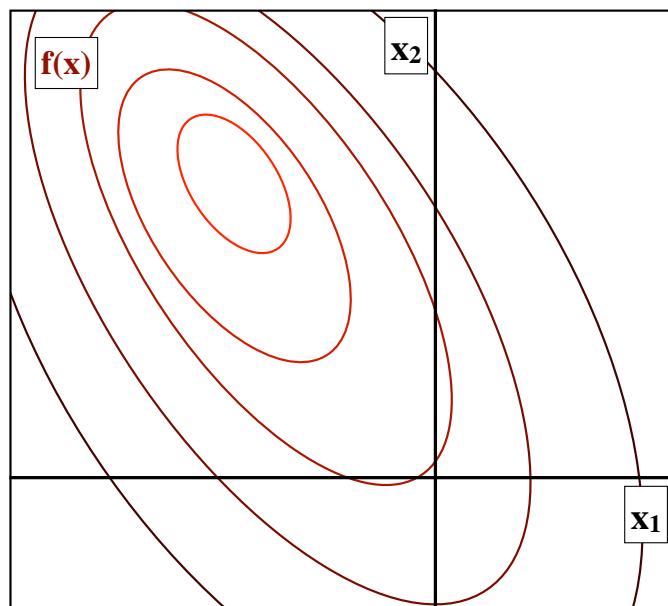
- Proximal-gradient method:

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

- Can we just plug in the **Newton**-like step?

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

- NO!



## Naive Proximal Quasi-Newton Method

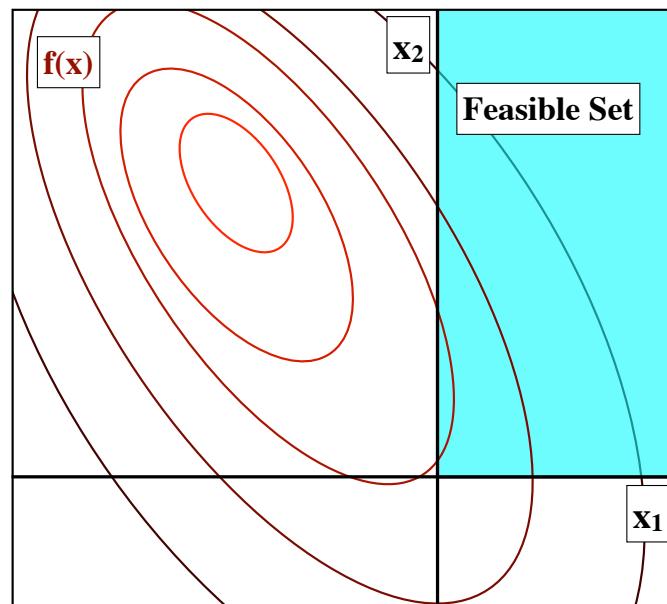
- Proximal-gradient method:

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

- Can we just plug in the **Newton**-like step?

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

- NO!



## Naive Proximal Quasi-Newton Method

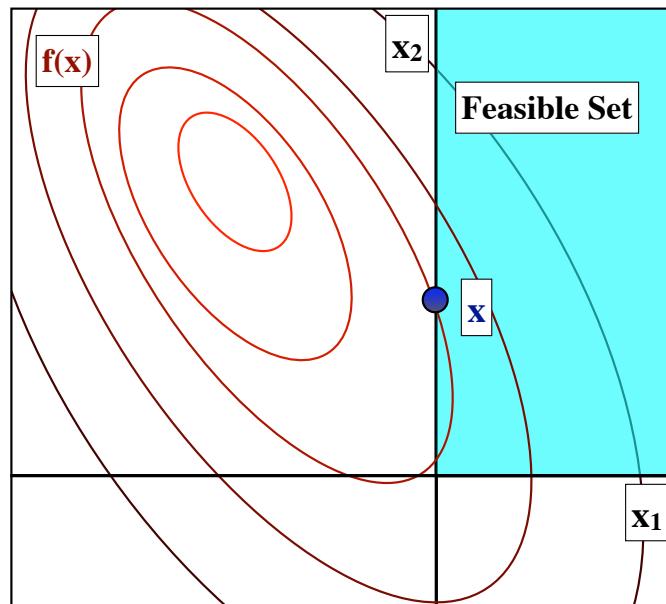
- Proximal-gradient method:

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

- Can we just plug in the **Newton**-like step?

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

- NO!



## Naive Proximal Quasi-Newton Method

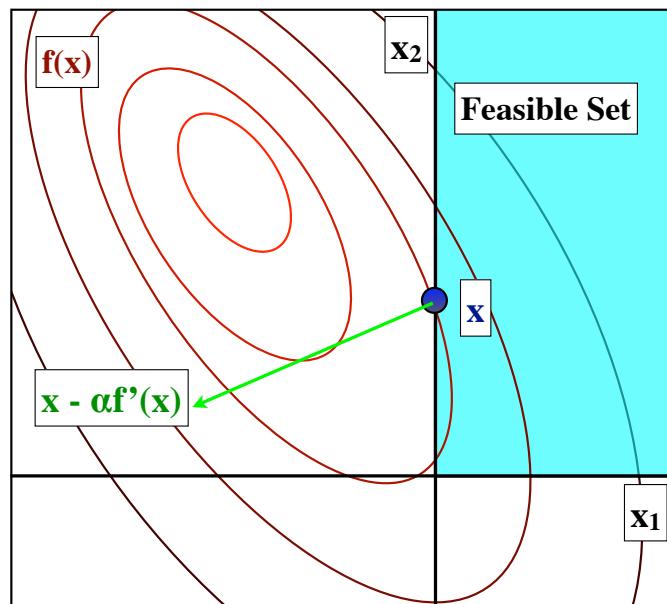
- Proximal-gradient method:

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

- Can we just plug in the **Newton**-like step?

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

- NO!



## Naive Proximal Quasi-Newton Method

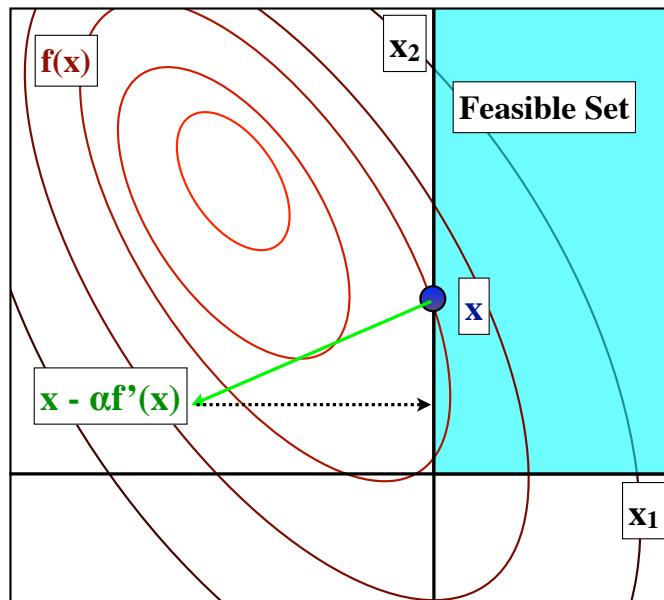
- Proximal-gradient method:

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

- Can we just plug in the **Newton**-like step?

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

- NO!



## Naive Proximal Quasi-Newton Method

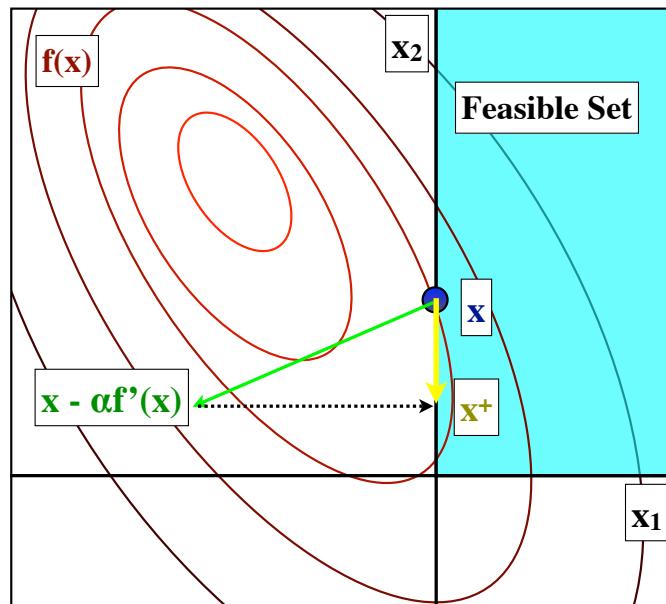
- Proximal-gradient method:

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

- Can we just plug in the **Newton**-like step?

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

- NO!



## Naive Proximal Quasi-Newton Method

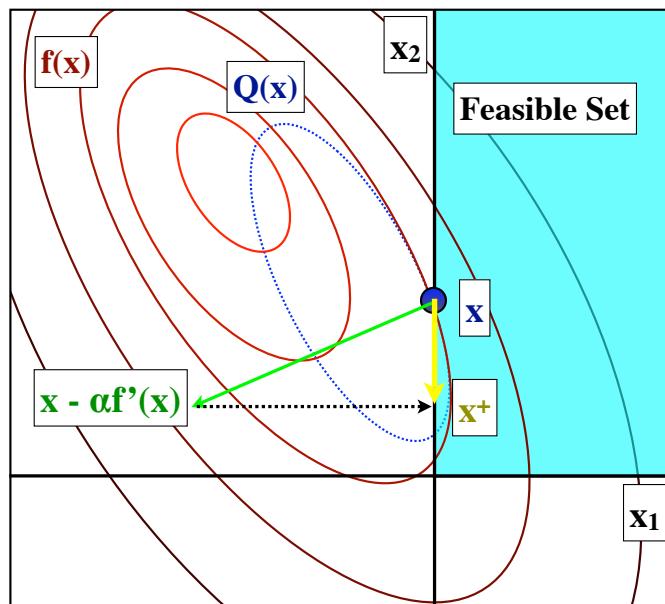
- Proximal-gradient method:

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

- Can we just plug in the **Newton**-like step?

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

- NO!



## Naive Proximal Quasi-Newton Method

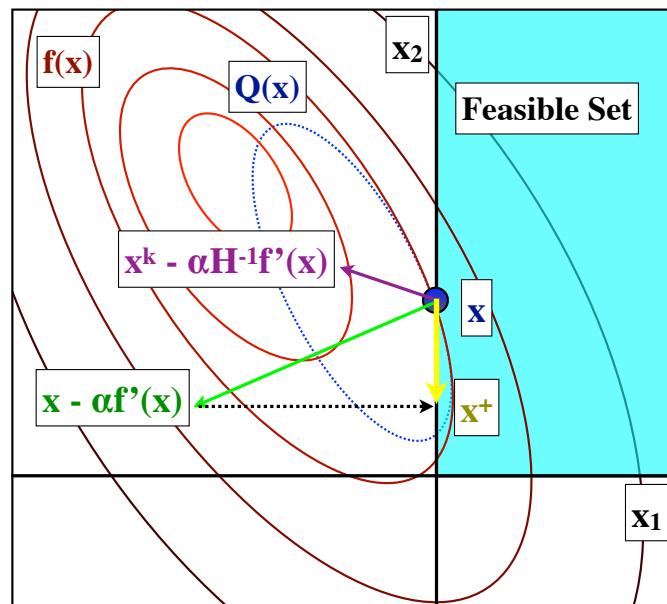
- Proximal-gradient method:

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

- Can we just plug in the **Newton**-like step?

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

- NO!



## Naive Proximal Quasi-Newton Method

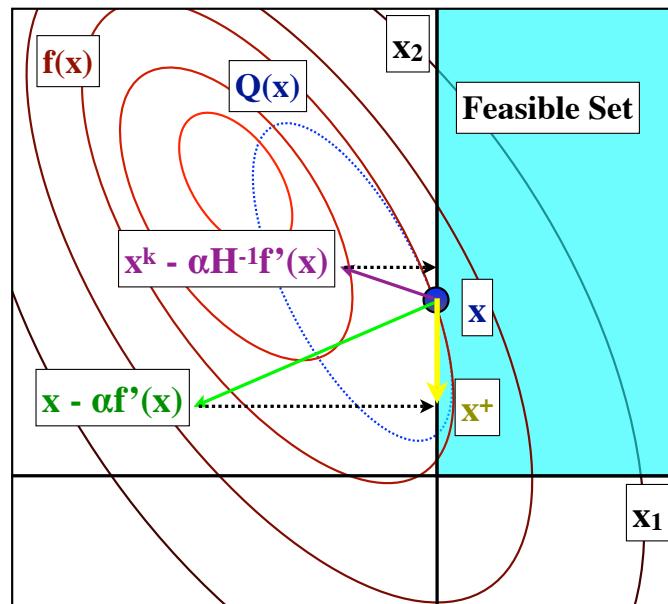
- Proximal-gradient method:

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

- Can we just plug in the **Newton**-like step?

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

- NO!



## Naive Proximal Quasi-Newton Method

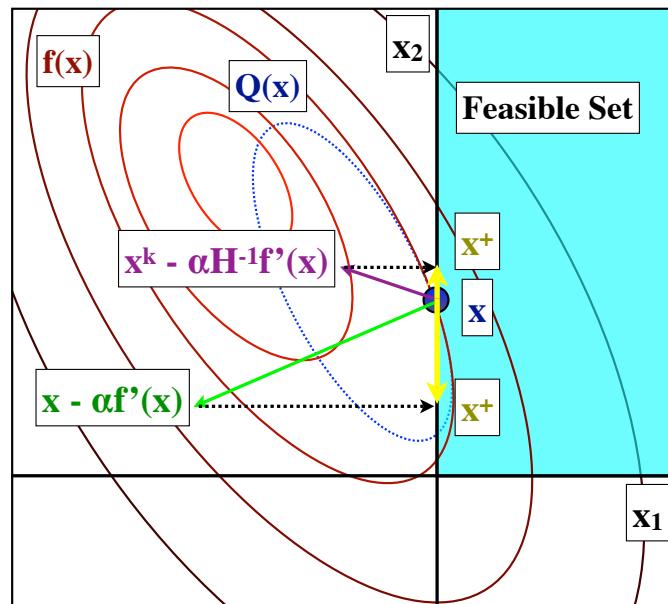
- Proximal-gradient method:

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

- Can we just plug in the **Newton**-like step?

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

- NO!

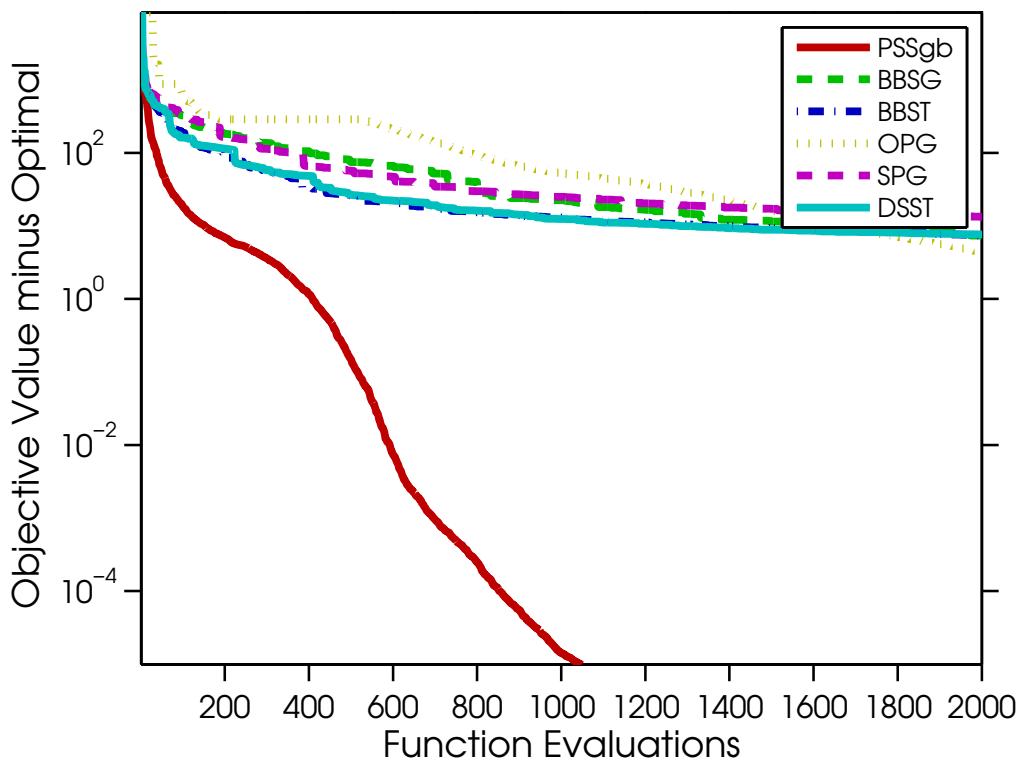


## Two-Metric (Sub)Gradient Projection

- In some cases, we can modify  $H$  to make this work:
  - Bound constraints.
  - Probability constraints.
  - L1-regularization.
- Two-metric (sub)gradient projection.  
[Gafni & Bertsekas, 1984, Schmidt, 2010].
- Key idea: make  $H$  diagonal with respect to coordinates near non-differentiability.

## Comparing to accelerated/spectral/diagonal gradient

Comparing to methods that do not use L-BFGS (sido data):



<http://www.di.ens.fr/~mschmidt/Software/L1General.html>

## Inexact Proximal-Newton

- The **broken** proximal-Newton method:

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

with the Euclidean proximal operator:

$$\text{prox}_r[y] = \arg \min_{x \in \mathbb{R}^P} r(x) + \frac{1}{2} \|x - y\|^2,$$

## Inexact Proximal-Newton

- The **fixed** proximal-Newton method:

$$x^+ = \text{prox}_{\alpha r}[x - \alpha H^{-1} \nabla f(x)]_H,$$

with the Euclidean proximal operator:

$$\text{prox}_r[y] = \arg \min_{x \in \mathbb{R}^P} r(x) + \frac{1}{2} \|x - y\|^2,$$

## Inexact Proximal-Newton

- The **fixed** proximal-Newton method:

$$x^+ = \text{prox}_{\alpha r}[x - \alpha \mathbf{H}^{-1} \nabla f(x)]_{\mathbf{H}},$$

with the **non-Euclidean** proximal operator:

$$\text{prox}_r[y]_{\mathbf{H}} = \arg \min_{x \in \mathbb{R}^P} r(x) + \frac{1}{2} \|x - y\|_{\mathbf{H}}^2,$$

where  $\|x\|_{\mathbf{H}}^2 = x^T \mathbf{H} x$ .

## Inexact Proximal-Newton

- The **fixed** proximal-Newton method:

$$x^+ = \text{prox}_{\alpha r}[x - \alpha \mathbf{H}^{-1} \nabla f(x)]_{\mathbf{H}},$$

with the **non-Euclidean** proximal operator:

$$\text{prox}_r[y]_{\mathbf{H}} = \arg \min_{x \in \mathbb{R}^P} r(x) + \frac{1}{2} \|x - y\|_{\mathbf{H}}^2,$$

where  $\|x\|_{\mathbf{H}}^2 = x^T \mathbf{H} x$ .

- **Non-smooth Newton-like** method
- **Same convergence properties as smooth case.**

## Inexact Proximal-Newton

- The **fixed** proximal-Newton method:

$$x^+ = \text{prox}_{\alpha r}[x - \alpha \mathbf{H}^{-1} \nabla f(x)]_{\mathbf{H}},$$

with the **non-Euclidean** proximal operator:

$$\text{prox}_r[y]_{\mathbf{H}} = \arg \min_{x \in \mathbb{R}^P} r(x) + \frac{1}{2} \|x - y\|_{\mathbf{H}}^2,$$

where  $\|x\|_{\mathbf{H}}^2 = x^T \mathbf{H} x$ .

- **Non-smooth Newton-like** method
- **Same convergence properties as smooth case.**
- But, **the prox is expensive** even with a simple regularizer.

## Inexact Proximal-Newton

- The **fixed** proximal-Newton method:

$$x^+ = \text{prox}_{\alpha r}[x - \alpha \mathbf{H}^{-1} \nabla f(x)]_{\mathbf{H}},$$

with the **non-Euclidean** proximal operator:

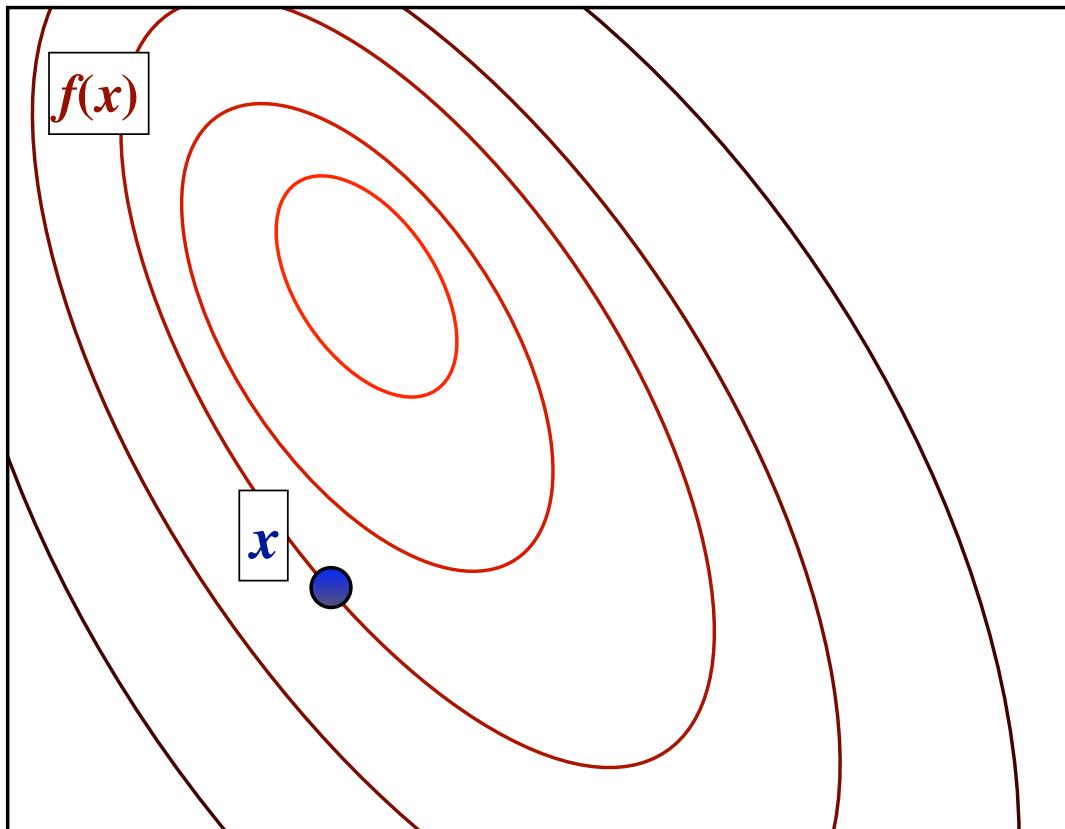
$$\text{prox}_r[y]_{\mathbf{H}} = \arg \min_{x \in \mathbb{R}^P} r(x) + \frac{1}{2} \|x - y\|_{\mathbf{H}}^2,$$

where  $\|x\|_{\mathbf{H}}^2 = x^T \mathbf{H} x$ .

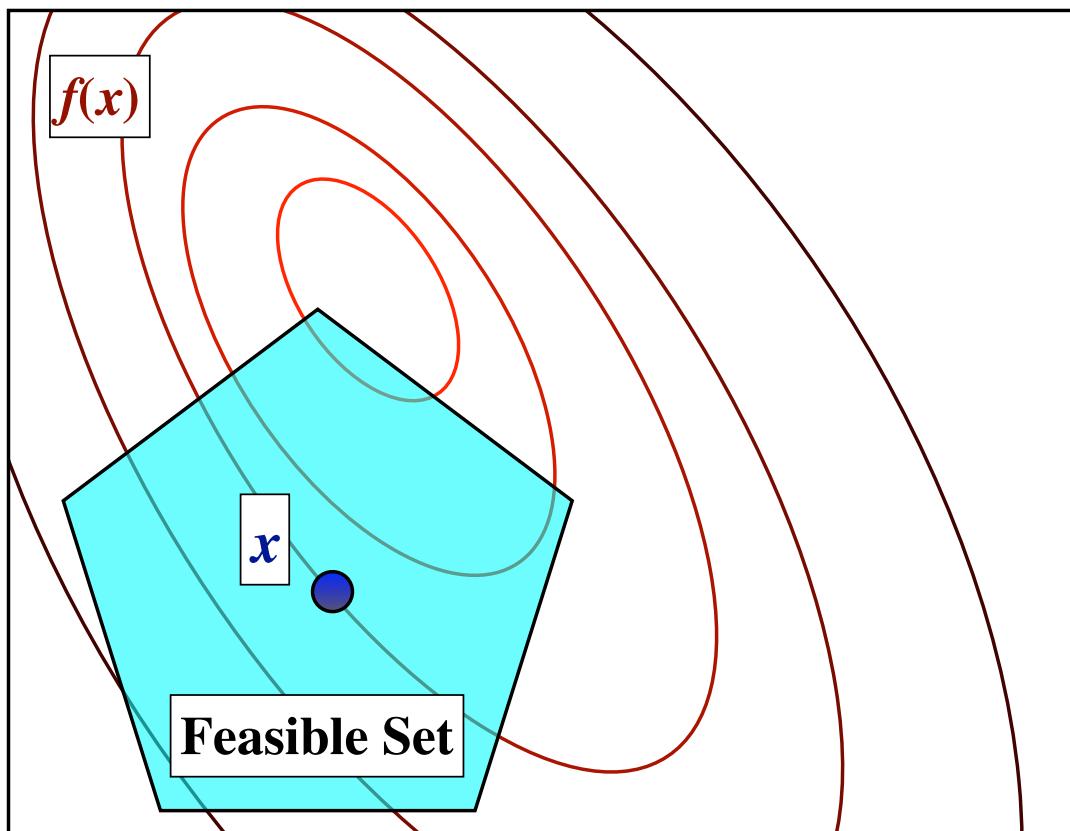
- **Non-smooth Newton-like** method
- **Same convergence properties as smooth case.**
- But, **the prox is expensive** even with a simple regularizer.
- Solution: **use a cheap approximate solution.**

(e.g., spectral proximal-gradient)

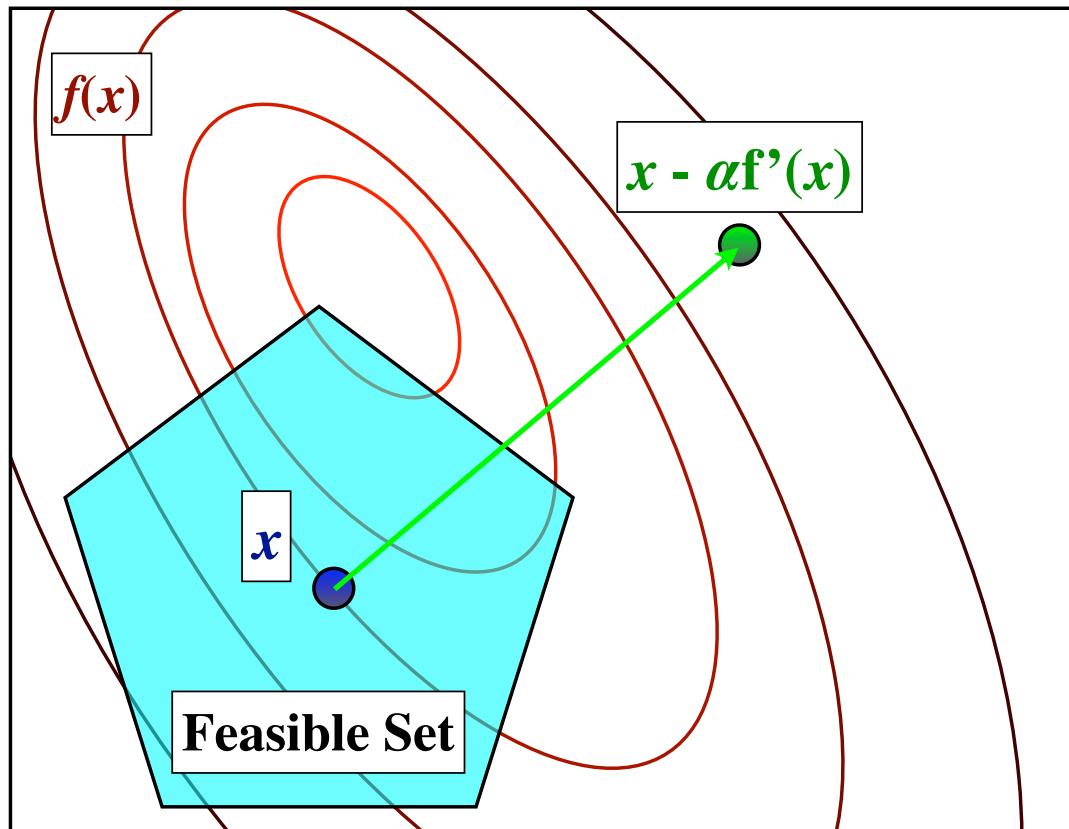
## Inexact Projected Newton



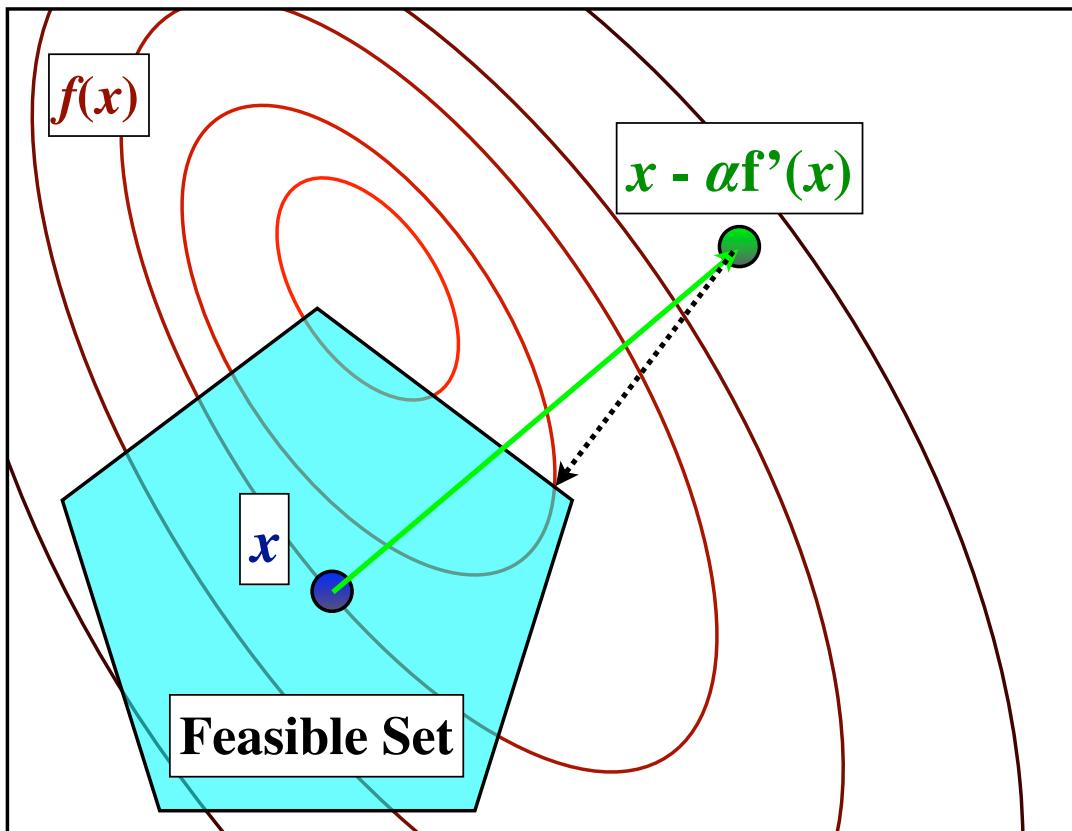
## Inexact Projected Newton



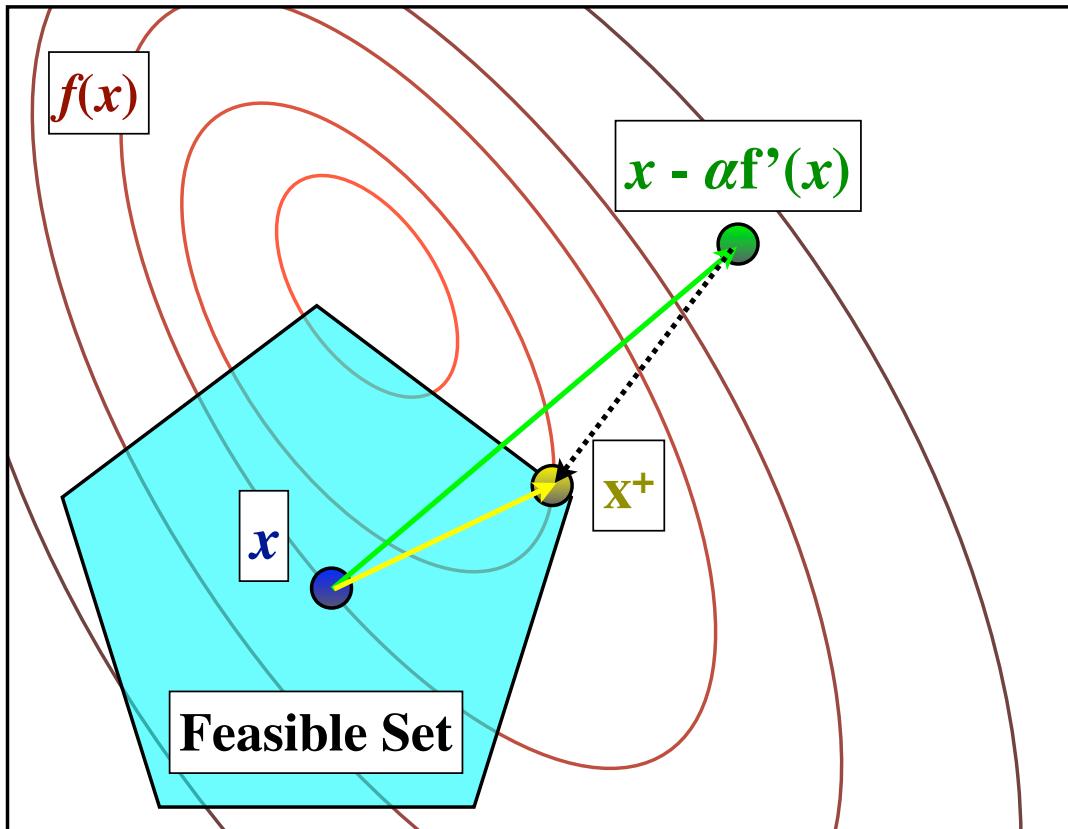
## Inexact Projected Newton



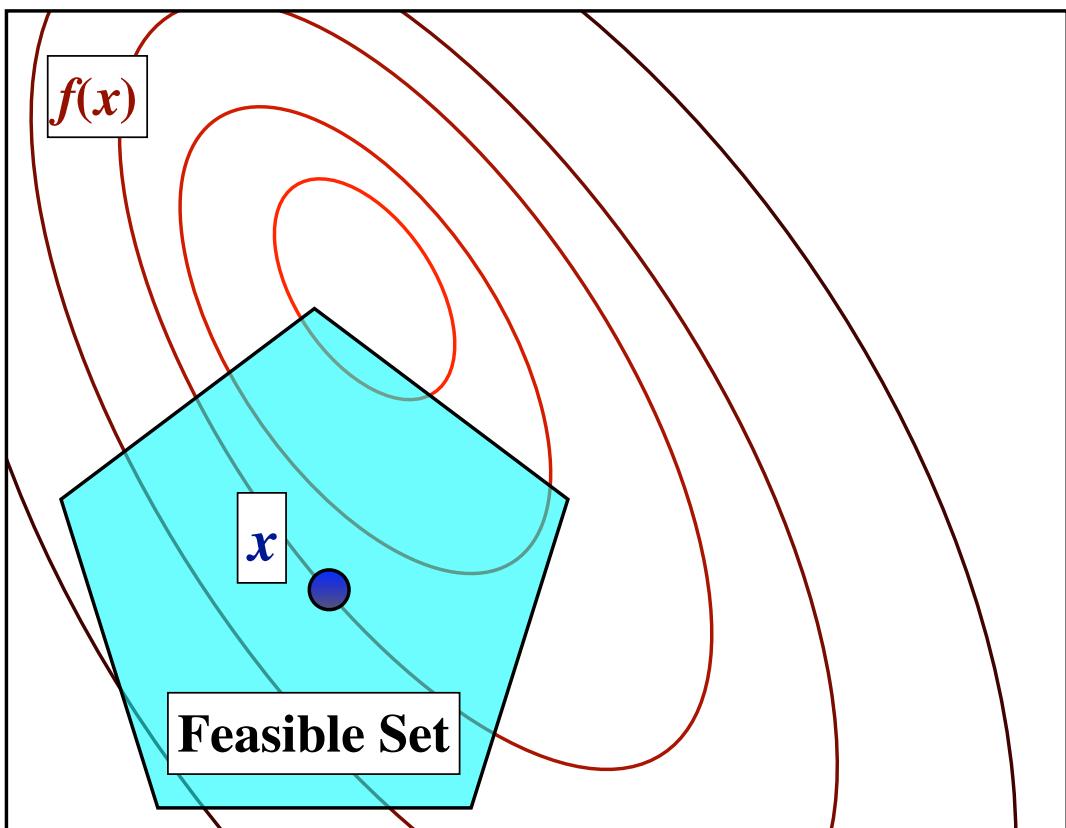
## Inexact Projected Newton



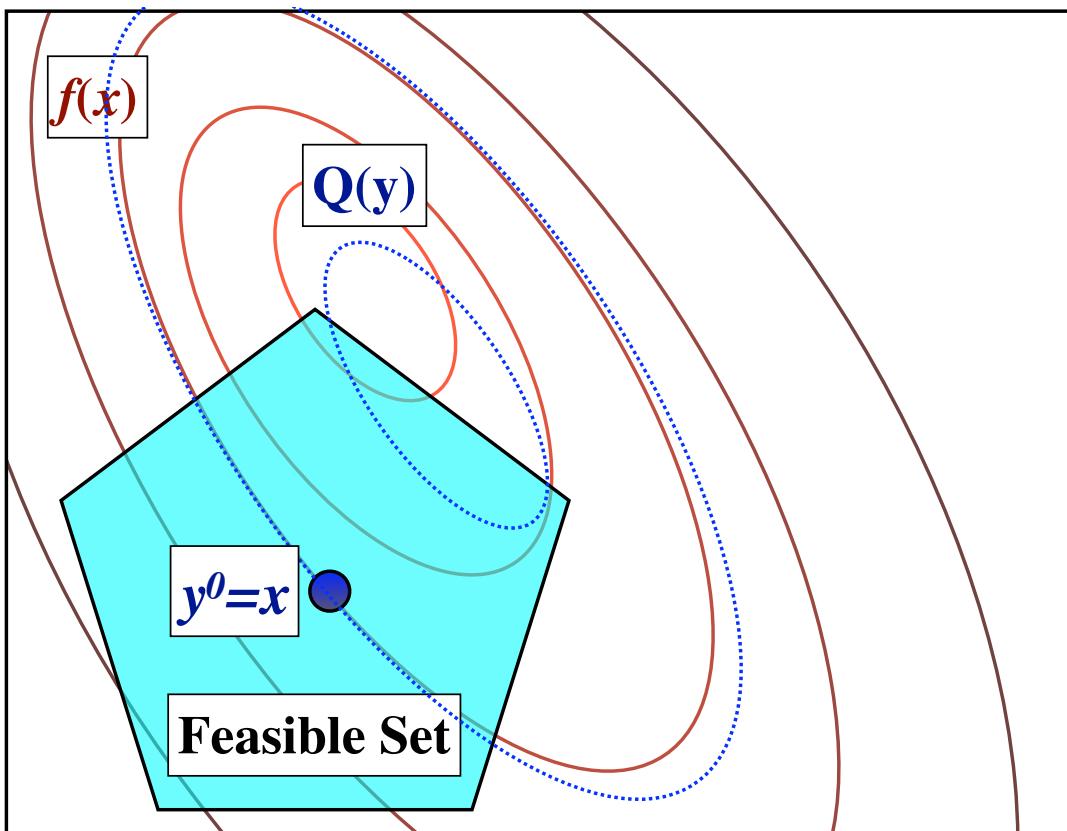
## Inexact Projected Newton



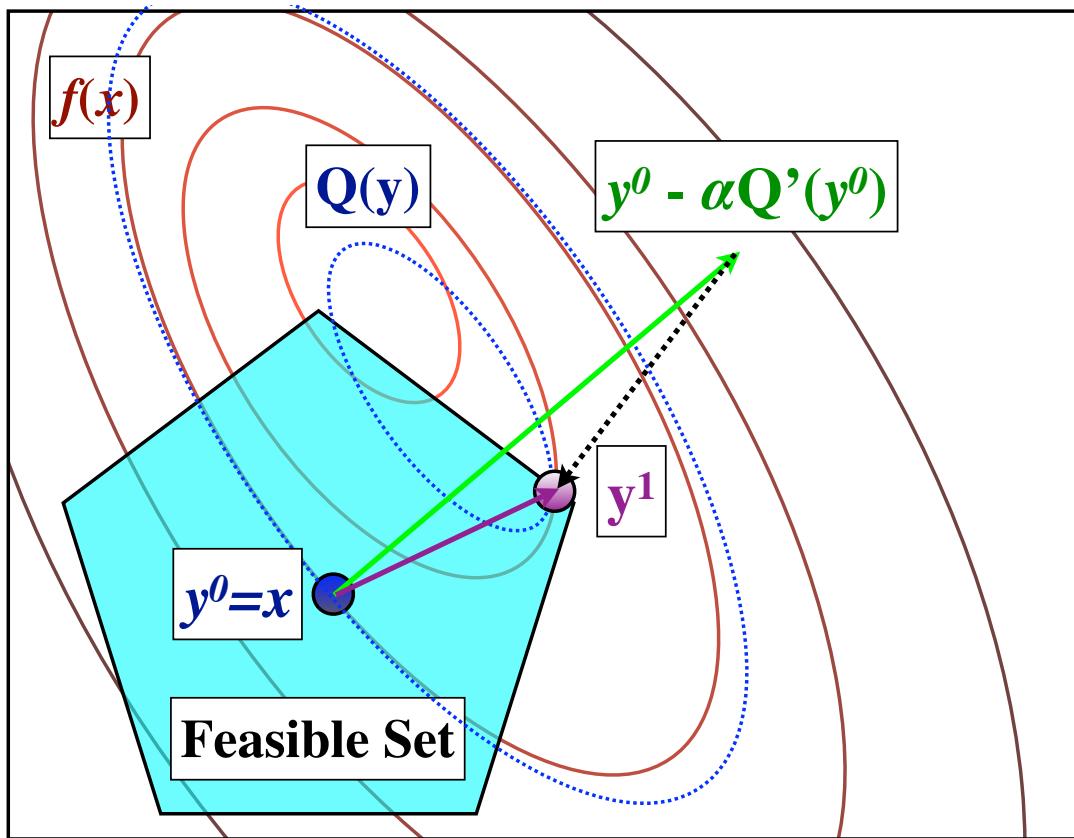
## Inexact Projected Newton



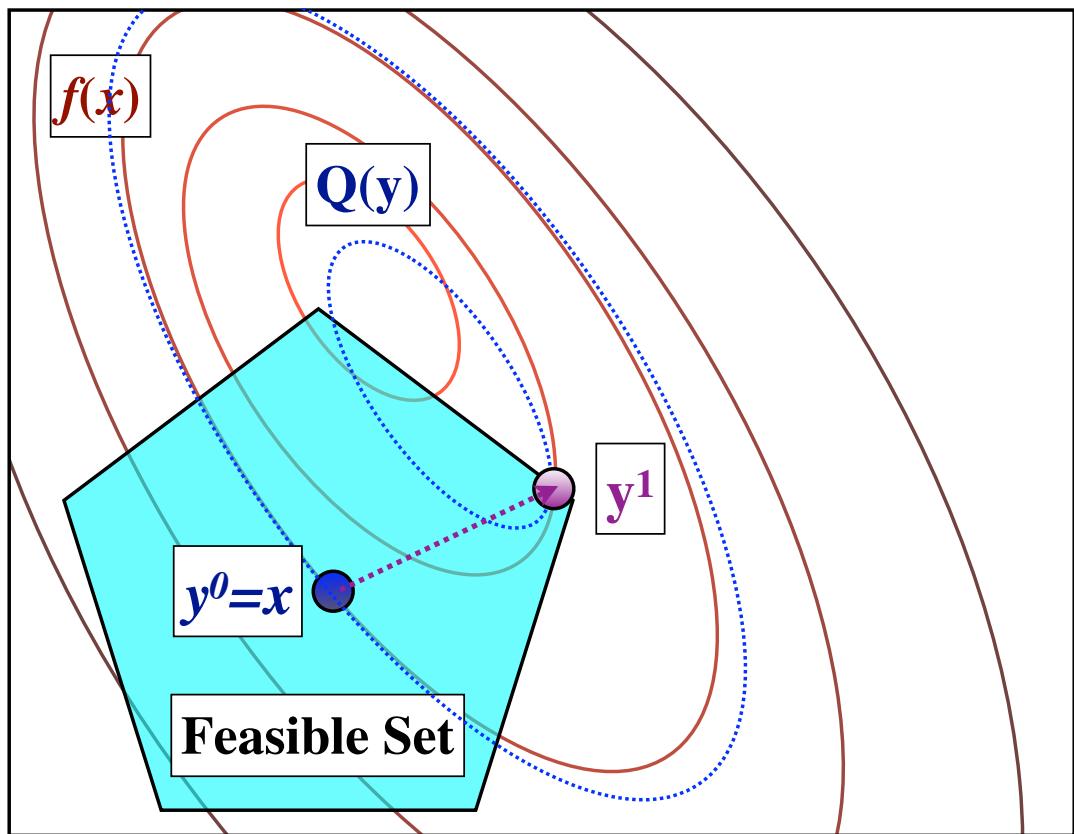
## Inexact Projected Newton



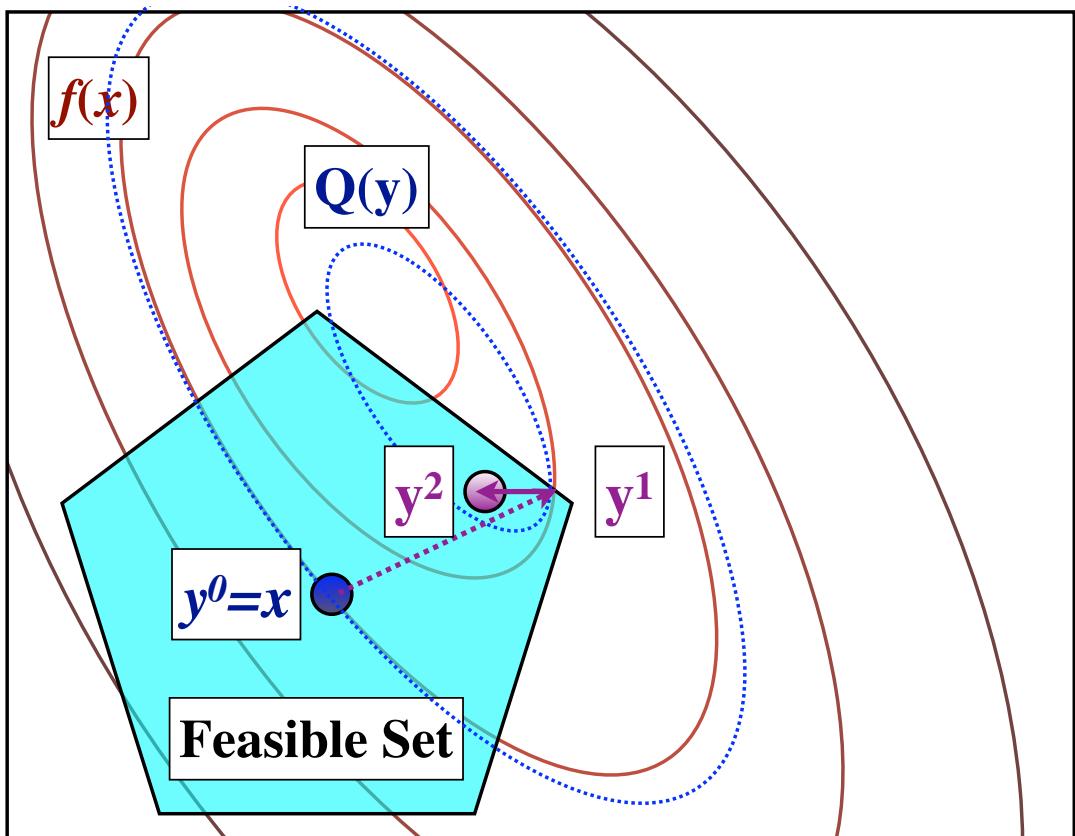
## Inexact Projected Newton



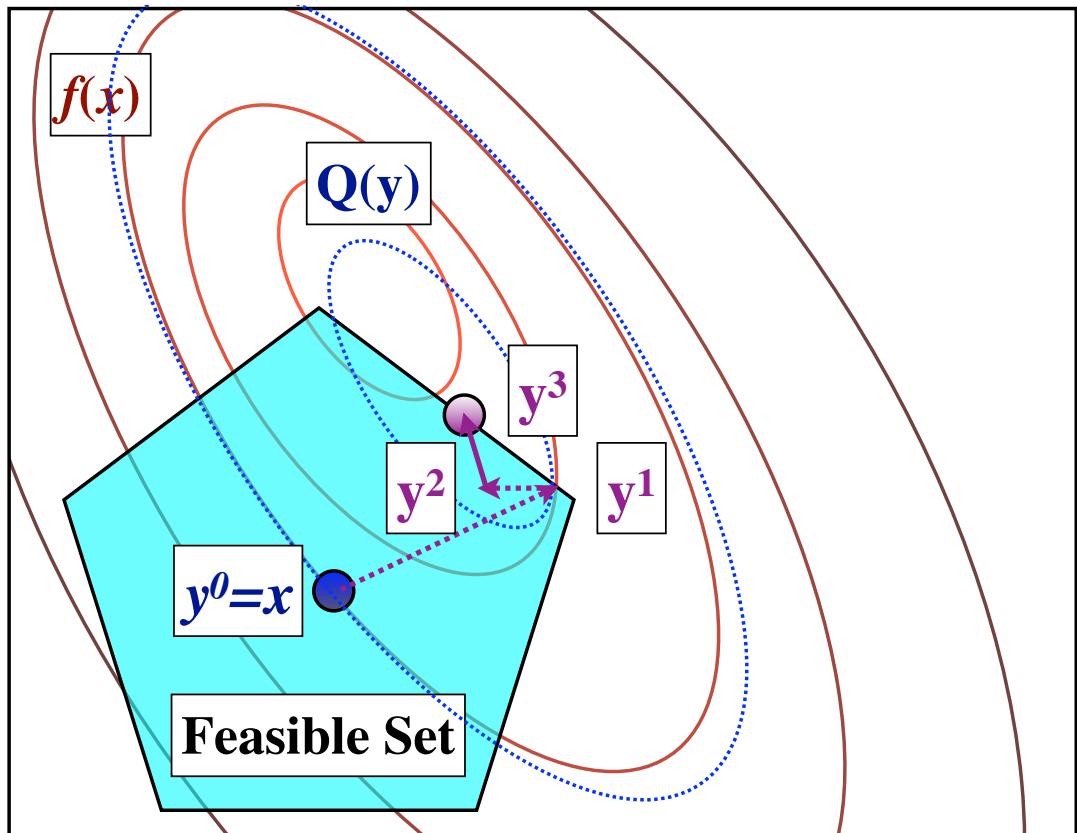
## Inexact Projected Newton



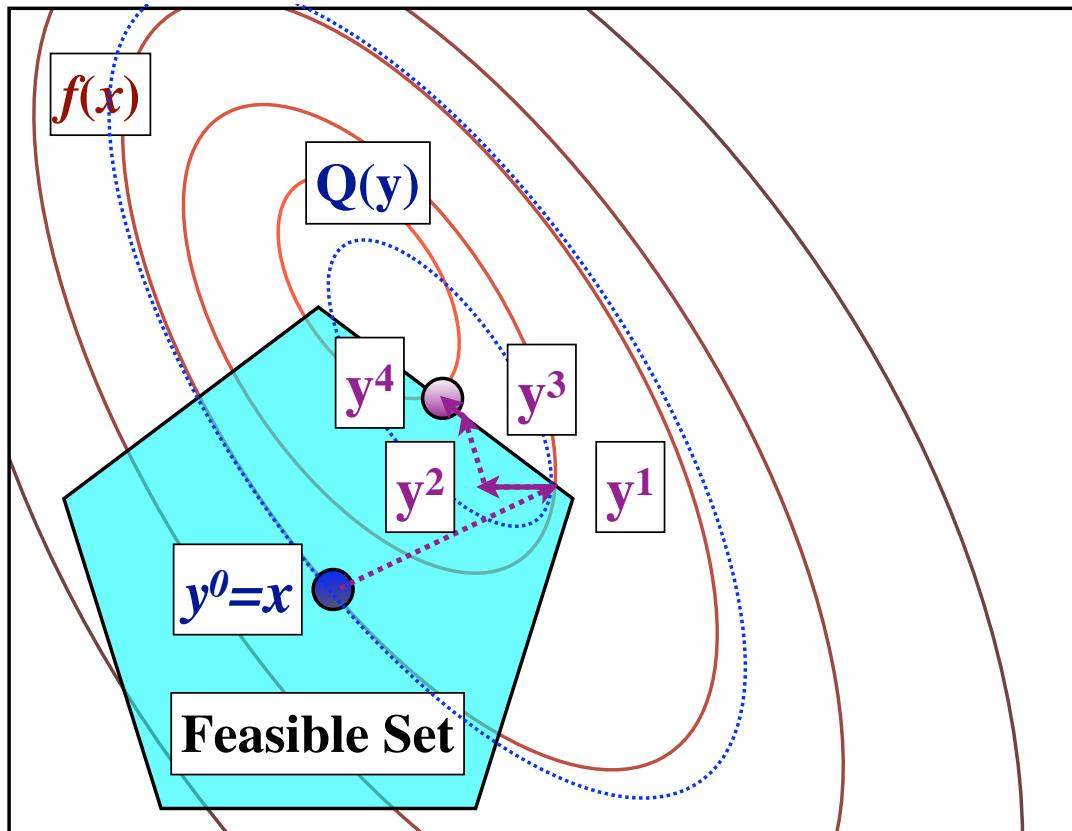
## Inexact Projected Newton



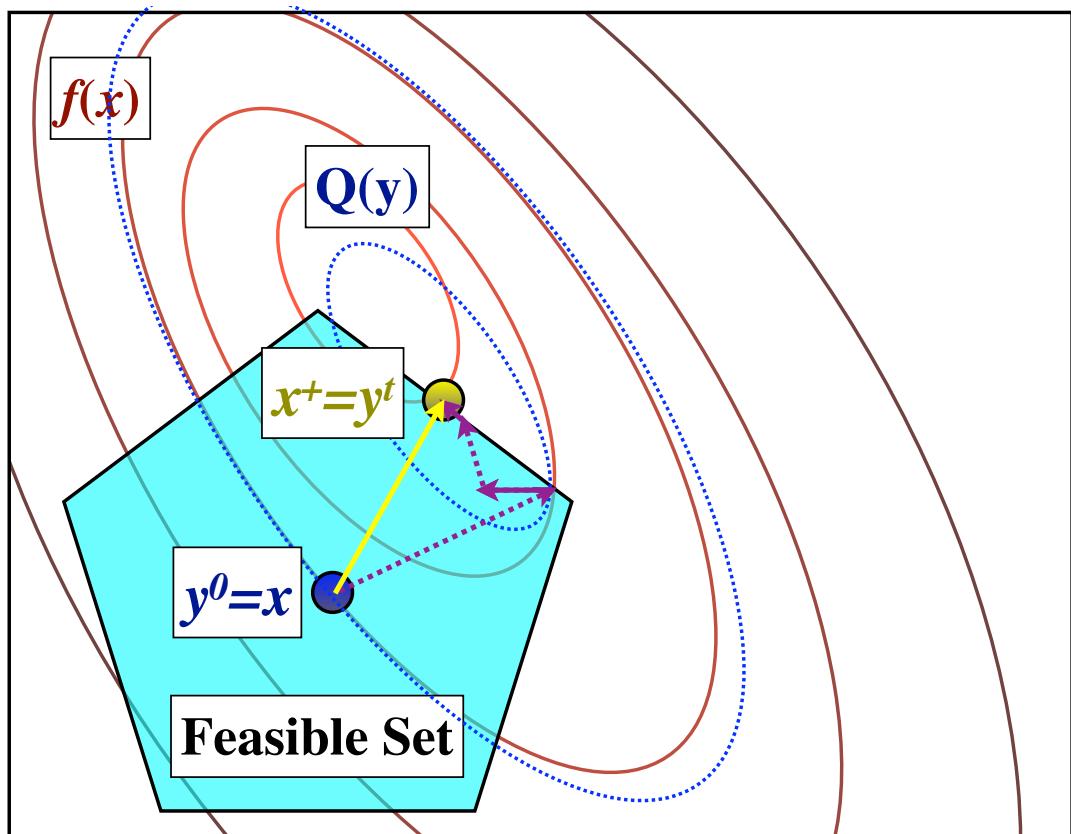
## Inexact Projected Newton



## Inexact Projected Newton



## Inexact Projected Newton



## Projected Quasi-Newton (PQN) Algorithm

- A proximal quasi-Newton (PQN) algorithm:  
[Schmidt et al., 2009, Schmidt, 2010]

## Projected Quasi-Newton (PQN) Algorithm

- A proximal quasi-Newton (PQN) algorithm:  
[Schmidt et al., 2009, Schmidt, 2010]
  - Outer: evaluate  $f(x)$  and  $\nabla f(x)$ , use L-BFGS to update  $H$ .

## Projected Quasi-Newton (PQN) Algorithm

- A proximal quasi-Newton (PQN) algorithm:  
[Schmidt et al., 2009, Schmidt, 2010]
  - Outer: evaluate  $f(x)$  and  $\nabla f(x)$ , use L-BFGS to update  $H$ .
  - Inner: spectral proximal-gradient to approximate proximal operator:
    - Requires multiplication by  $H$  (linear-time for L-BFGS).
    - Requires proximal operator of  $r$  (cheap for simple constraints).

## Projected Quasi-Newton (PQN) Algorithm

- A proximal quasi-Newton (PQN) algorithm:  
[Schmidt et al., 2009, Schmidt, 2010]
  - Outer: evaluate  $f(x)$  and  $\nabla f(x)$ , use L-BFGS to update  $H$ .
  - Inner: spectral proximal-gradient to approximate proximal operator:
    - Requires multiplication by  $H$  (linear-time for L-BFGS).
    - Requires proximal operator of  $r$  (cheap for simple constraints).
  - For small  $\alpha$ , one iteration is sufficient to give descent.

## Projected Quasi-Newton (PQN) Algorithm

- A proximal quasi-Newton (PQN) algorithm:  
[Schmidt et al., 2009, Schmidt, 2010]
  - Outer: evaluate  $f(x)$  and  $\nabla f(x)$ , use L-BFGS to update  $H$ .
  - Inner: spectral proximal-gradient to approximate proximal operator:
    - Requires multiplication by  $H$  (linear-time for L-BFGS).
    - Requires proximal operator of  $r$  (cheap for simple constraints).
  - For small  $\alpha$ , one iteration is sufficient to give descent.
- Cheap inner iterations lead to fewer expensive outer iterations.

## Projected Quasi-Newton (PQN) Algorithm

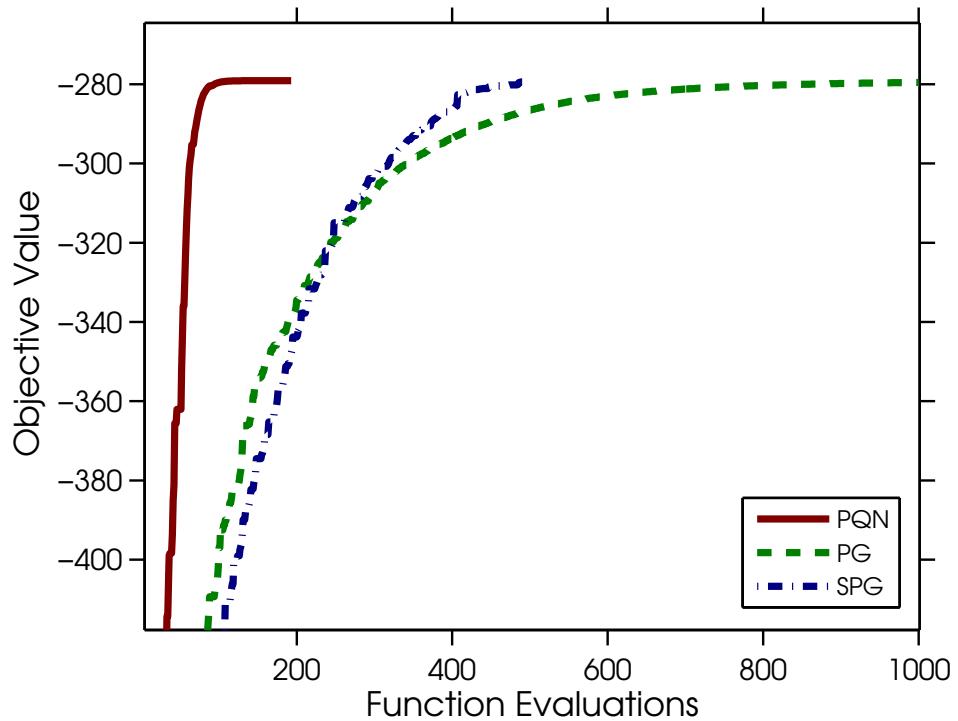
- A proximal quasi-Newton (PQN) algorithm:  
[Schmidt et al., 2009, Schmidt, 2010]
  - Outer: evaluate  $f(x)$  and  $\nabla f(x)$ , use L-BFGS to update  $H$ .
  - Inner: spectral proximal-gradient to approximate proximal operator:
    - Requires multiplication by  $H$  (linear-time for L-BFGS).
    - Requires proximal operator of  $r$  (cheap for simple constraints).
  - For small  $\alpha$ , one iteration is sufficient to give descent.
- Cheap inner iterations lead to fewer expensive outer iterations.
- “Optimizing costly functions with simple constraints” .

## Projected Quasi-Newton (PQN) Algorithm

- A proximal quasi-Newton (PQN) algorithm:  
[Schmidt et al., 2009, Schmidt, 2010]
  - Outer: evaluate  $f(x)$  and  $\nabla f(x)$ , use L-BFGS to update  $H$ .
  - Inner: spectral proximal-gradient to approximate proximal operator:
    - Requires multiplication by  $H$  (linear-time for L-BFGS).
    - Requires proximal operator of  $r$  (cheap for simple constraints).
  - For small  $\alpha$ , one iteration is sufficient to give descent.
- Cheap inner iterations lead to fewer expensive outer iterations.
- “Optimizing costly functions with simple constraints” .
- “Optimizing costly functions with simple regularizers” .

## Graphical Model Structure Learning with Groups

Comparing PQN to first-order methods on a graphical model structure learning problem. [Gasch et al., 2000, Duchi et al., 2008].



## Alternating Direction Method of Multipliers

- Alternating direction method of multipliers (ADMM) solves:

$$\min_{Ax+By=c} f(x) + r(y).$$

- Alternate between prox-like operators with respect to  $f$  and  $r$ .

## Alternating Direction Method of Multipliers

- Alternating direction method of multipliers (ADMM) solves:

$$\min_{Ax+By=c} f(x) + r(y).$$

- Alternate between prox-like operators with respect to  $f$  and  $r$ .
- Can introduce constraints to convert to this form:

$$\min_x f(Ax) + r(x) \Leftrightarrow \min_{x=Ay} f(x) + r(y),$$

## Alternating Direction Method of Multipliers

- Alternating direction method of multipliers (ADMM) solves:

$$\min_{Ax+By=c} f(x) + r(y).$$

- Alternate between prox-like operators with respect to  $f$  and  $r$ .
- Can introduce constraints to convert to this form:

$$\min_x f(Ax) + r(x) \Leftrightarrow \min_{x=Ay} f(x) + r(y),$$

$$\min_x f(x) + r(Bx) \Leftrightarrow \min_{y=Bx} f(x) + r(y).$$

## Alternating Direction Method of Multipliers

- Alternating direction method of multipliers (ADMM) solves:

$$\min_{Ax+By=c} f(x) + r(y).$$

- Alternate between prox-like operators with respect to  $f$  and  $r$ .
- Can introduce constraints to convert to this form:

$$\min_x f(Ax) + r(x) \Leftrightarrow \min_{x=Ay} f(x) + r(y),$$

$$\min_x f(x) + r(Bx) \Leftrightarrow \min_{y=Bx} f(x) + r(y).$$

- If prox can not be computed exactly: Linearized ADMM.

## Dual Methods

- Strongly-convex problems have smooth duals.
- **Solve the dual instead of the primal.**

## Dual Methods

- Strongly-convex problems have smooth duals.
- Solve the dual instead of the primal.
- SVM non-smooth strongly-convex primal:

$$\min_x C \sum_{i=1}^N \max\{0, 1 - b_i a_i^T x\} + \frac{1}{2} \|x\|^2.$$

- SVM smooth dual:

$$\min_{0 \leq \alpha \leq C} \frac{1}{2} \alpha^T A A^T \alpha - \sum_{i=1}^N \alpha_i$$

- Smooth bound constrained problem:
  - Two-metric projection (efficient Newton-liked method).
  - Randomized coordinate descent (next section).

## Discussion

- State of the art methods consider several other issues:
  - **Shrinking**: Identify variables likely to stay zero.  
[El Ghaoui et al., 2010].
  - **Continuation**: Start with a large  $\lambda$  and slowly decrease it.  
[Xiao and Zhang, 2012]
  - **Frank-Wolfe**: Using linear approximations to obtain efficient/sparse updates.

## Frank-Wolfe Method

- In some cases the projected gradient step

$$x^+ = \arg \min_{y \in \mathcal{C}} \left\{ f(x) + \nabla f(x)^T (y - x) + \frac{1}{2\alpha} \|y - x\|^2 \right\},$$

may be hard to compute (e.g., dual of max-margin Markov networks).

## Frank-Wolfe Method

- In some cases the projected gradient step

$$x^+ = \arg \min_{y \in \mathcal{C}} \left\{ f(x) + \nabla f(x)^T (y - x) + \frac{1}{2\alpha} \|y - x\|^2 \right\},$$

may be hard to compute (e.g., dual of max-margin Markov networks).

- Frank-Wolfe method simply uses:

$$x^+ = \arg \min_{y \in \mathcal{C}} \left\{ f(x) + \nabla f(x)^T (y - x) \right\},$$

requires compact  $\mathcal{C}$ , takes convex combination of  $x$  and  $x^+$ .

## Frank-Wolfe Method

- In some cases the projected gradient step

$$x^+ = \arg \min_{y \in \mathcal{C}} \left\{ f(x) + \nabla f(x)^T (y - x) + \frac{1}{2\alpha} \|y - x\|^2 \right\},$$

may be hard to compute (e.g., dual of max-margin Markov networks).

- Frank-Wolfe method simply uses:

$$x^+ = \arg \min_{y \in \mathcal{C}} \left\{ f(x) + \nabla f(x)^T (y - x) \right\},$$

requires compact  $\mathcal{C}$ , takes convex combination of  $x$  and  $x^+$ .

- Iterate can be written as convex combination of vertices of  $\mathcal{C}$ .
- $O(1/t)$  rate for smooth convex objectives, some linear convergence results for smooth and strongly-convex.[Jaggi, 2013]

## Outline

- 1 Convex Functions
- 2 Smooth Optimization
- 3 Non-Smooth Optimization
- 4 Randomized Algorithms
- 5 Parallel/Distributed Optimization

## Big-N Problems

- We want to minimize the sum of a **finite** set of smooth functions:

$$\min_{x \in \mathbb{R}^P} f(x) := \frac{1}{N} \sum_{i=1}^N f_i(x).$$

## Big-N Problems

- We want to minimize the sum of a **finite** set of smooth functions:

$$\min_{x \in \mathbb{R}^P} f(x) := \frac{1}{N} \sum_{i=1}^N f_i(x).$$

- We are interested in cases where ***N* is very large.**

## Big-N Problems

- We want to minimize the sum of a **finite** set of smooth functions:

$$\min_{x \in \mathbb{R}^P} f(x) := \frac{1}{N} \sum_{i=1}^N f_i(x).$$

- We are interested in cases where  **$N$  is very large**.
- Simple example is least-squares,

$$f_i(x) := (a_i^T x - b_i)^2.$$

- Other examples:
  - logistic regression, Huber regression, smooth SVMs, CRFs, etc.

## Stochastic vs. Deterministic Gradient Methods

- We consider minimizing  $f(x) = \frac{1}{N} \sum_{i=1}^N f_i(x)$ .

## Stochastic vs. Deterministic Gradient Methods

- We consider minimizing  $f(x) = \frac{1}{N} \sum_{i=1}^N f_i(x)$ .
- Deterministic gradient method [Cauchy, 1847]:

$$x_{t+1} = x_t - \alpha_t \nabla f(x_t) = x_t - \frac{\alpha_t}{N} \sum_{i=1}^N \nabla f_i(x_t).$$

- Iteration cost is linear in  $N$ .
- Quasi-Newton methods still require  $O(N)$ .
- Convergence with constant  $\alpha_t$  or line-search.

## Stochastic vs. Deterministic Gradient Methods

- We consider minimizing  $f(x) = \frac{1}{N} \sum_{i=1}^N f_i(x)$ .
- **Deterministic** gradient method [Cauchy, 1847]:

$$x_{t+1} = x_t - \alpha_t \nabla f(x_t) = x_t - \frac{\alpha_t}{N} \sum_{i=1}^N \nabla f_i(x_t).$$

- Iteration cost is **linear in  $N$** .
- Quasi-Newton methods still require  $O(N)$ .
- Convergence with constant  $\alpha_t$  or line-search.
- **Stochastic** gradient method [Robbins & Monro, 1951]:
- Random selection of  $i(t)$  from  $\{1, 2, \dots, N\}$ .

$$x_{t+1} = x_t - \alpha_t f'_{i(t)}(x_t).$$

## Stochastic vs. Deterministic Gradient Methods

- We consider minimizing  $f(x) = \frac{1}{N} \sum_{i=1}^N f_i(x)$ .
- **Deterministic** gradient method [Cauchy, 1847]:

$$x_{t+1} = x_t - \alpha_t \nabla f(x_t) = x_t - \frac{\alpha_t}{N} \sum_{i=1}^N \nabla f_i(x_t).$$

- Iteration cost is **linear in  $N$** .
- Quasi-Newton methods still require  $O(N)$ .
- Convergence with constant  $\alpha_t$  or line-search.
- **Stochastic** gradient method [Robbins & Monro, 1951]:
- Random selection of  $i(t)$  from  $\{1, 2, \dots, N\}$ .

$$x_{t+1} = x_t - \alpha_t f'_{i(t)}(x_t).$$

- Gives unbiased estimate of true gradient,

$$\mathbb{E}[f'_{i(t)}(x)] = \frac{1}{N} \sum_{i=1}^N \nabla f_i(x) = \nabla f(x).$$

- Iteration cost is **independent of  $N$** .

## Stochastic vs. Deterministic Gradient Methods

- We consider minimizing  $f(x) = \frac{1}{N} \sum_{i=1}^N f_i(x)$ .
- **Deterministic** gradient method [Cauchy, 1847]:

$$x_{t+1} = x_t - \alpha_t \nabla f(x_t) = x_t - \frac{\alpha_t}{N} \sum_{i=1}^N \nabla f_i(x_t).$$

- Iteration cost is **linear in  $N$** .
- Quasi-Newton methods still require  $O(N)$ .
- Convergence with constant  $\alpha_t$  or line-search.
- **Stochastic** gradient method [Robbins & Monro, 1951]:
- Random selection of  $i(t)$  from  $\{1, 2, \dots, N\}$ .

$$x_{t+1} = x_t - \alpha_t f'_{i(t)}(x_t).$$

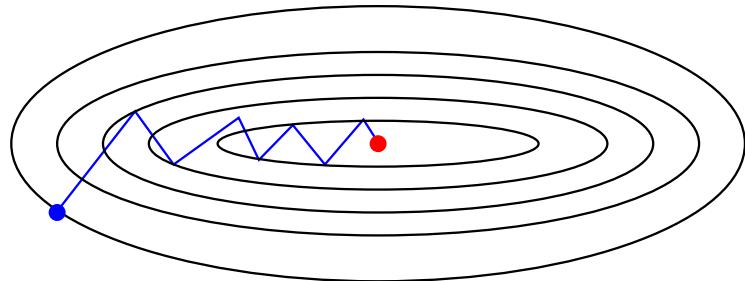
- Gives unbiased estimate of true gradient,

$$\mathbb{E}[f'_{i(t)}(x)] = \frac{1}{N} \sum_{i=1}^N \nabla f_i(x) = \nabla f(x).$$

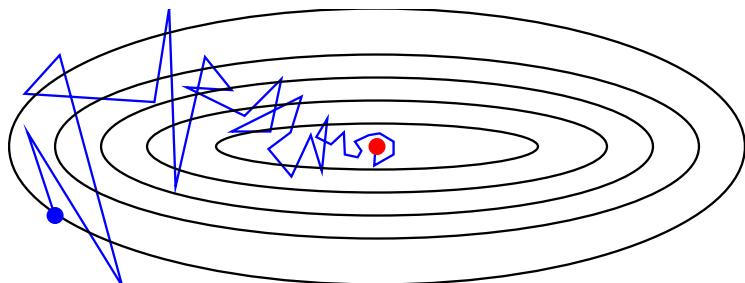
- Iteration cost is **independent of  $N$** .
- As in subgradient method, **we require  $\alpha_t \rightarrow 0$** .
- Classical choice is  $\alpha_t = O(1/t)$ .

## Stochastic vs. Deterministic Gradient Methods

- We consider minimizing  $g(x) = \frac{1}{N} \sum_{i=1}^n f_i(x)$ .
- **Deterministic** gradient method [Cauchy, 1847]:



- **Stochastic** gradient method [Robbins & Monro, 1951]:



## Convex Optimization Zoo

Stochastic iterations are  $N$  times faster, but how many iterations?

## Convex Optimization Zoo

Stochastic iterations are  $N$  times faster, but how many iterations?

Algorithm	Assumptions	Exact	Stochastic
Subgradient	Convex	$O(1/\sqrt{t})$	$O(1/\sqrt{t})$
Subgradient	Strongly	$O(1/t)$	$O(1/t)$

## Convex Optimization Zoo

Stochastic iterations are  $N$  times faster, but how many iterations?

Algorithm	Assumptions	Exact	Stochastic
Subgradient	Convex	$O(1/\sqrt{t})$	$O(1/\sqrt{t})$
Subgradient	Strongly	$O(1/t)$	$O(1/t)$

- Good news for non-smooth problems:
  - stochastic is as fast as deterministic
- We can solve non-smooth problems  $N$  times faster!

## Convex Optimization Zoo

Stochastic iterations are  $N$  times faster, but how many iterations?

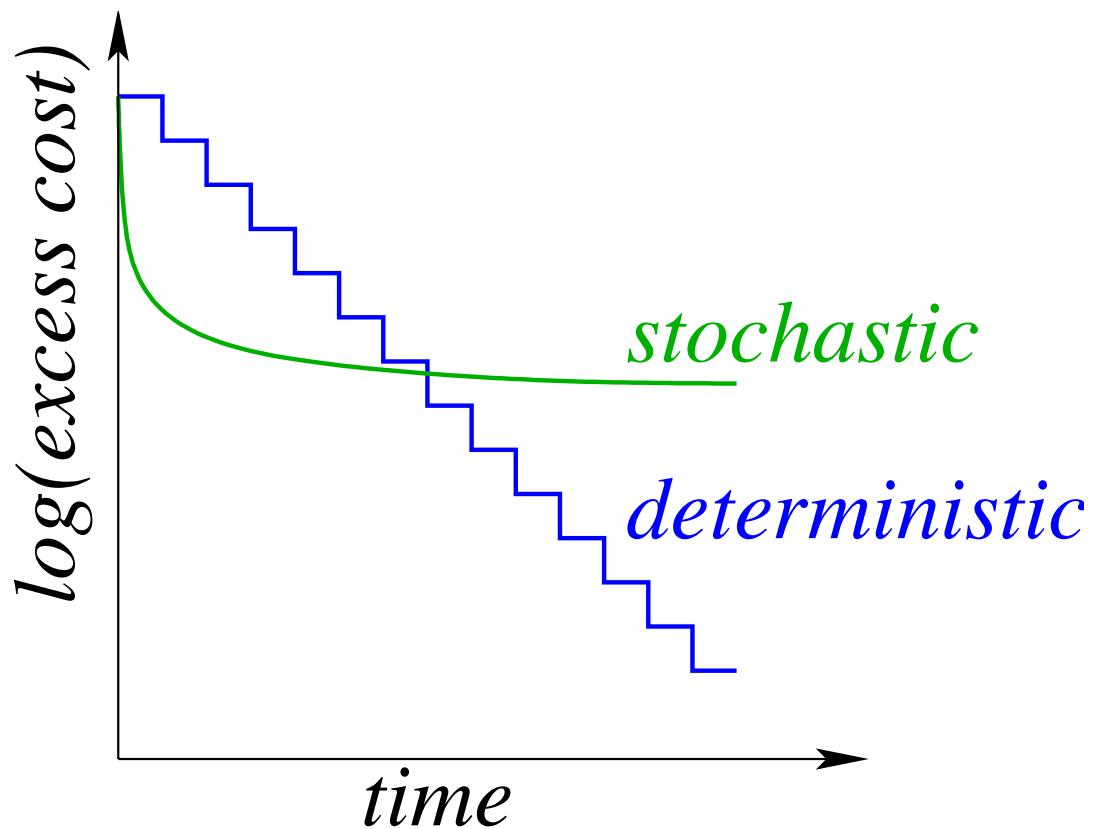
Algorithm	Assumptions	Exact	Stochastic
Subgradient	Convex	$O(1/\sqrt{t})$	$O(1/\sqrt{t})$
Subgradient	Strongly	$O(1/t)$	$O(1/t)$

- Good news for non-smooth problems:
  - stochastic is as fast as deterministic
- We can solve non-smooth problems  $N$  times faster!
- Bad news for smooth problems:
  - smoothness does not help stochastic methods.

Algorithm	Assumptions	Exact	Stochastic
Gradient	Convex	$O(1/t)$	$O(1/\sqrt{t})$
Gradient	Strongly	$O((1 - \mu/L)^t)$	$O(1/t)$

## Deterministic vs. Stochastic Convergence Rates

Plot of convergence rates in smooth/strongly-convex case:



## Speeding up Stochastic Gradient Methods

- We can try accelerated/Newton-like stochastic methods:
  - These **do not** improve on the convergence rates.
  - But may improve performance at start if noise is small.

## Speeding up Stochastic Gradient Methods

- We can try accelerated/Newton-like stochastic methods:
  - These **do not** improve on the convergence rates.
  - But may improve performance at start if noise is small.
- Ideas that do improve convergence: **averaging** and **large steps**:

## Speeding up Stochastic Gradient Methods

- We can try accelerated/Newton-like stochastic methods:
  - These **do not** improve on the convergence rates.
  - But may improve performance at start if noise is small.
- Ideas that do improve convergence: **averaging** and **large steps**:
  - $\alpha_t = O(1/t^\alpha)$  for  $\alpha \in (0.5, 1)$  more robust than  $O(1/t)$ .  
[Bach & Moulines, 2011]

## Speeding up Stochastic Gradient Methods

- We can try accelerated/Newton-like stochastic methods:
  - These **do not** improve on the convergence rates.
  - But may improve performance at start if noise is small.
- Ideas that do improve convergence: **averaging** and **large steps**:
  - $\alpha_t = O(1/t^\alpha)$  for  $\alpha \in (0.5, 1)$  more robust than  $O(1/t)$ .  
[Bach & Moulines, 2011]
  - Gradient averaging improves constants ('dual averaging').  
[Nesterov, 2007]

## Speeding up Stochastic Gradient Methods

- We can try accelerated/Newton-like stochastic methods:
  - These **do not** improve on the convergence rates.
  - But may improve performance at start if noise is small.
- Ideas that do improve convergence: **averaging** and **large steps**:
  - $\alpha_t = O(1/t^\alpha)$  for  $\alpha \in (0.5, 1)$  more robust than  $O(1/t)$ .  
[Bach & Moulines, 2011]
  - Gradient averaging improves constants ('dual averaging').  
[Nesterov, 2007]
  - Averaging later iterations gives rates without smoothness.  
[Rakhlin et al., 2011]

## Speeding up Stochastic Gradient Methods

- We can try accelerated/Newton-like stochastic methods:
  - These **do not** improve on the convergence rates.
  - But may improve performance at start if noise is small.
- Ideas that do improve convergence: **averaging** and **large steps**:
  - $\alpha_t = O(1/t^\alpha)$  for  $\alpha \in (0.5, 1)$  more robust than  $O(1/t)$ .  
[Bach & Moulines, 2011]
  - Gradient averaging improves constants ('dual averaging').  
[Nesterov, 2007]
  - Averaging later iterations gives rates without smoothness.  
[Rakhlin et al., 2011]
  - Averaging in smooth case achieves **same asymptotic rate as optimal stochastic Newton** method.[Polyak & Juditsky, 1992]]

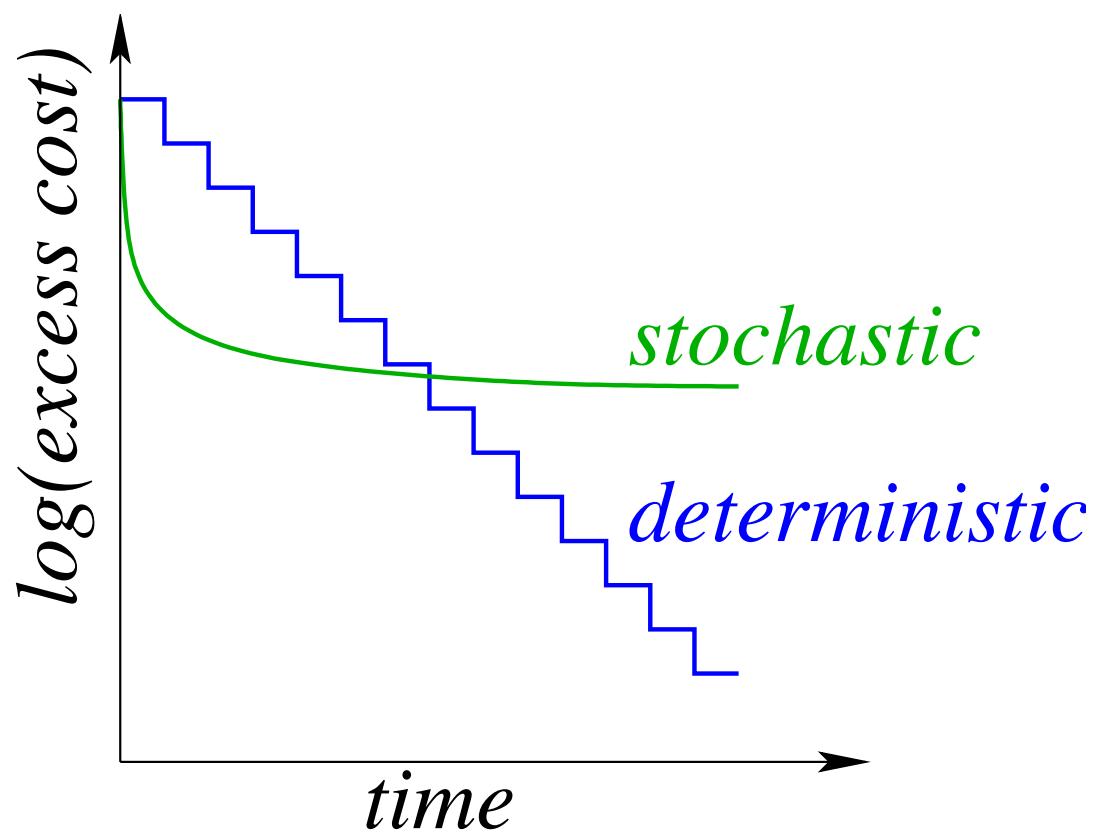
## Speeding up Stochastic Gradient Methods

- We can try accelerated/Newton-like stochastic methods:
  - These **do not** improve on the convergence rates.
  - But may improve performance at start if noise is small.
- Ideas that do improve convergence: **averaging** and **large steps**:
  - $\alpha_t = O(1/t^\alpha)$  for  $\alpha \in (0.5, 1)$  more robust than  $O(1/t)$ .  
[Bach & Moulines, 2011]
  - Gradient averaging improves constants ('dual averaging').  
[Nesterov, 2007]
  - Averaging later iterations gives rates without smoothness.  
[Rakhlin et al., 2011]
  - Averaging in smooth case achieves **same asymptotic rate as optimal stochastic Newton** method.[Polyak & Juditsky, 1992]]
  - **Constant** step-size  $\alpha$  achieves rate of  $O(\rho^t) + O(\alpha)$ .  
[Nedic & Bertsekas, 2000]

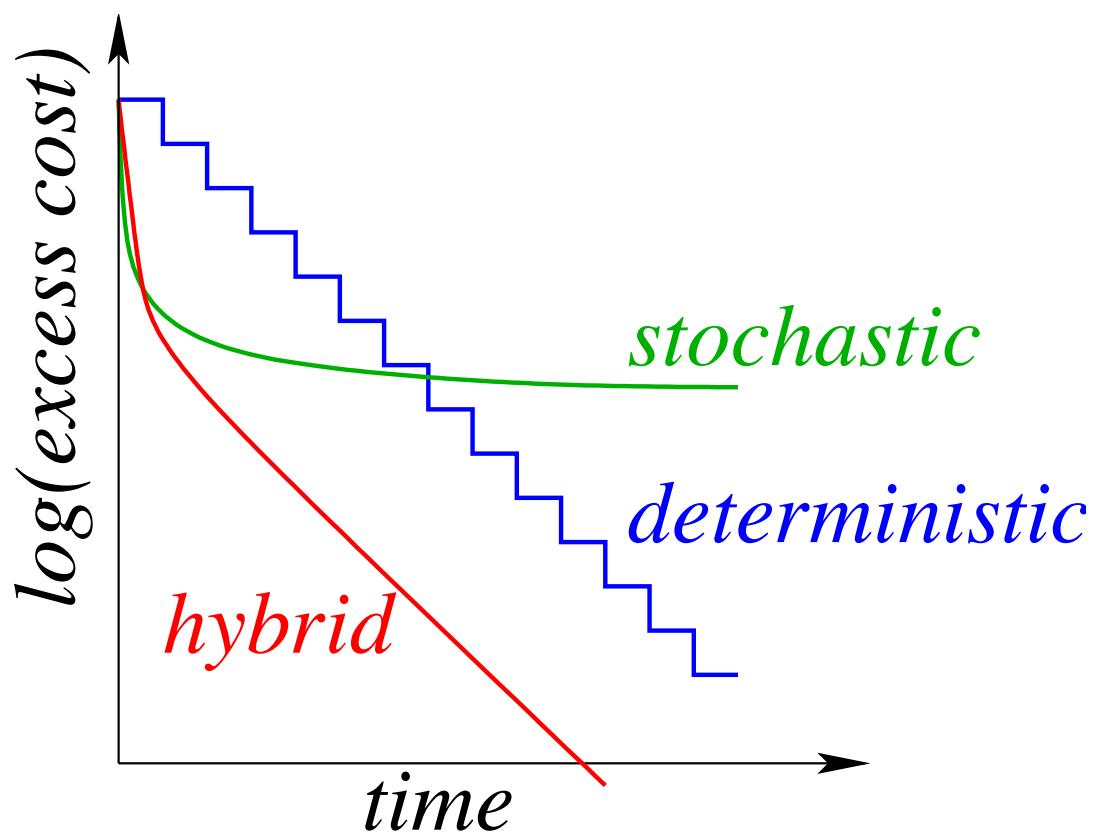
## Speeding up Stochastic Gradient Methods

- We can try accelerated/Newton-like stochastic methods:
  - These **do not** improve on the convergence rates.
  - But may improve performance at start if noise is small.
- Ideas that do improve convergence: **averaging** and **large steps**:
  - $\alpha_t = O(1/t^\alpha)$  for  $\alpha \in (0.5, 1)$  more robust than  $O(1/t)$ .  
[Bach & Moulines, 2011]
  - Gradient averaging improves constants ('dual averaging').  
[Nesterov, 2007]
  - Averaging later iterations gives rates without smoothness.  
[Rakhlin et al., 2011]
  - Averaging in smooth case achieves **same asymptotic rate as optimal stochastic Newton** method.[Polyak & Juditsky, 1992]]
  - **Constant** step-size  $\alpha$  achieves rate of  $O(\rho^t) + O(\alpha)$ .  
[Nedic & Bertsekas, 2000]
  - Averaging and constant step-size achieves  $O(1/t)$  rate for stochastic Newton-like methods without strong convexity.  
[Bach & Moulines, 2013]

## Motivation for Hybrid Methods for Smooth Problems



## Motivation for Hybrid Methods for Smooth Problems



## Stochastic Average Gradient

- Should we use stochastic methods for smooth problems?
- **Can we have a rate of  $O(\rho^t)$  with only 1 gradient evaluation per iteration?**

## Stochastic Average Gradient

- Should we use stochastic methods for smooth problems?
- **Can we have a rate of  $O(\rho^t)$  with only 1 gradient evaluation per iteration?**
  - YES!

## Stochastic Average Gradient

- Should we use stochastic methods for smooth problems?
- **Can we have a rate of  $O(\rho^t)$  with only 1 gradient evaluation per iteration?**
  - YES! The **stochastic average gradient (SAG)** algorithm:
    - Randomly select  $i(t)$  from  $\{1, 2, \dots, n\}$  and compute  $f'_{i(t)}(x^t)$ .

$$x^{t+1} = x^t - \frac{\alpha^t}{N} \sum_{i=1}^N \nabla f_i(x^t)$$

## Stochastic Average Gradient

- Should we use stochastic methods for smooth problems?
- **Can we have a rate of  $O(\rho^t)$  with only 1 gradient evaluation per iteration?**
  - YES! The **stochastic average gradient (SAG)** algorithm:
    - Randomly select  $i(t)$  from  $\{1, 2, \dots, n\}$  and compute  $f'_{i(t)}(x^t)$ .

$$x^{t+1} = x^t - \frac{\alpha^t}{N} \sum_{i=1}^N \nabla f_i(x^t)$$

## Stochastic Average Gradient

- Should we use stochastic methods for smooth problems?
- **Can we have a rate of  $O(\rho^t)$  with only 1 gradient evaluation per iteration?**
  - YES! The **stochastic average gradient (SAG)** algorithm:
    - Randomly select  $i(t)$  from  $\{1, 2, \dots, n\}$  and compute  $f'_{i(t)}(x^t)$ .

$$x^{t+1} = x^t - \frac{\alpha^t}{N} \sum_{i=1}^N y_i^t$$

- **Memory:**  $y_i^t = \nabla f_i(x^t)$  from the **last  $t$**  where  $i$  was selected.  
[Le Roux et al., 2012]

## Stochastic Average Gradient

- Should we use stochastic methods for smooth problems?
- **Can we have a rate of  $O(\rho^t)$  with only 1 gradient evaluation per iteration?**
  - YES! The **stochastic average gradient (SAG)** algorithm:
    - Randomly select  $i(t)$  from  $\{1, 2, \dots, n\}$  and compute  $f'_{i(t)}(x^t)$ .
  - **Memory:**  $y_i^t = \nabla f_i(x^t)$  from the **last  $t$**  where  $i$  was selected.  
[Le Roux et al., 2012]
  - **Stochastic** variant of increment average gradient (IAG).  
[Blatt et al., 2007]

## Stochastic Average Gradient

- Should we use stochastic methods for smooth problems?
- **Can we have a rate of  $O(\rho^t)$  with only 1 gradient evaluation per iteration?**
  - YES! The **stochastic average gradient (SAG)** algorithm:
    - Randomly select  $i(t)$  from  $\{1, 2, \dots, n\}$  and compute  $f'_{i(t)}(x^t)$ .
  - **Memory:**  $y_i^t = \nabla f_i(x^t)$  from the **last  $t$**  where  $i$  was selected.  
[Le Roux et al., 2012]
  - **Stochastic** variant of increment average gradient (IAG).  
[Blatt et al., 2007]
    - Assumes gradients of non-selected examples don't change.
    - Assumption becomes accurate as  $\|x^{t+1} - x^t\| \rightarrow 0$ .
    - Memory requirements reduced to  $O(N)$  for many problems.

## Convex Optimization Zoo

Algorithm	Rate	Grads
Stochastic Gradient Gradient	$O(1/t)$ $O((1 - \mu/L)^t)$	1 N

## Convex Optimization Zoo

Algorithm	Rate	Grads
Stochastic Gradient	$O(1/t)$	1
Gradient	$O((1 - \mu/L)^t)$	N
SAG	$O((1 - \min\{\frac{\mu}{16L_i}, \frac{1}{8N}\})^t)$	1

## Convex Optimization Zoo

Algorithm	Rate	Grads
Stochastic Gradient	$O(1/t)$	1
Gradient	$O((1 - \mu/L)^t)$	N
SAG	$O((1 - \min\{\frac{\mu}{16L_i}, \frac{1}{8N}\})^t)$	1

- $L_i$  is the Lipschitz constant over all  $\nabla f_i$  ( $L_i \geq L$ ).
- SAG has a similar speed to the gradient method, but only looks at one training example per iteration.

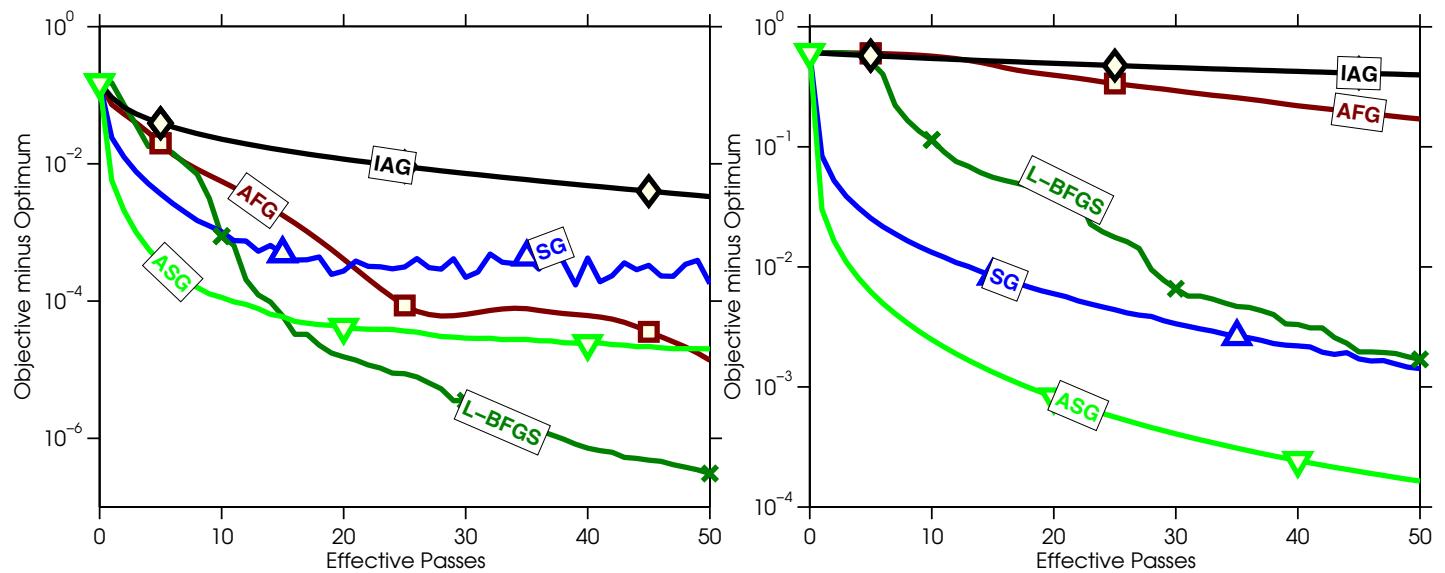
## Convex Optimization Zoo

Algorithm	Rate	Grads
Stochastic Gradient	$O(1/t)$	1
Gradient	$O((1 - \mu/L)^t)$	N
SAG	$O((1 - \min\{\frac{\mu}{16L_i}, \frac{1}{8N}\})^t)$	1

- $L_i$  is the Lipschitz constant over all  $\nabla f_i$  ( $L_i \geq L$ ).
- SAG has a similar speed to the gradient method, but only looks at one training example per iteration.
- Recent work extends this result in various ways:
  - Similar rates for stochastic dual coordinate ascent.  
[Shalev-Schwartz & Zhang, 2013]
  - Memory-free variants. [Johnson & Zhang, 2013, Madani et al., 2013]
  - Proximal-gradient variants. [Mairal, 2013]
  - ADMM variants. [Wong et al., 2013]
  - Improved constants. [Defazio et al., 2014]
  - Non-uniform sampling. [Schmidt et al., 2013]

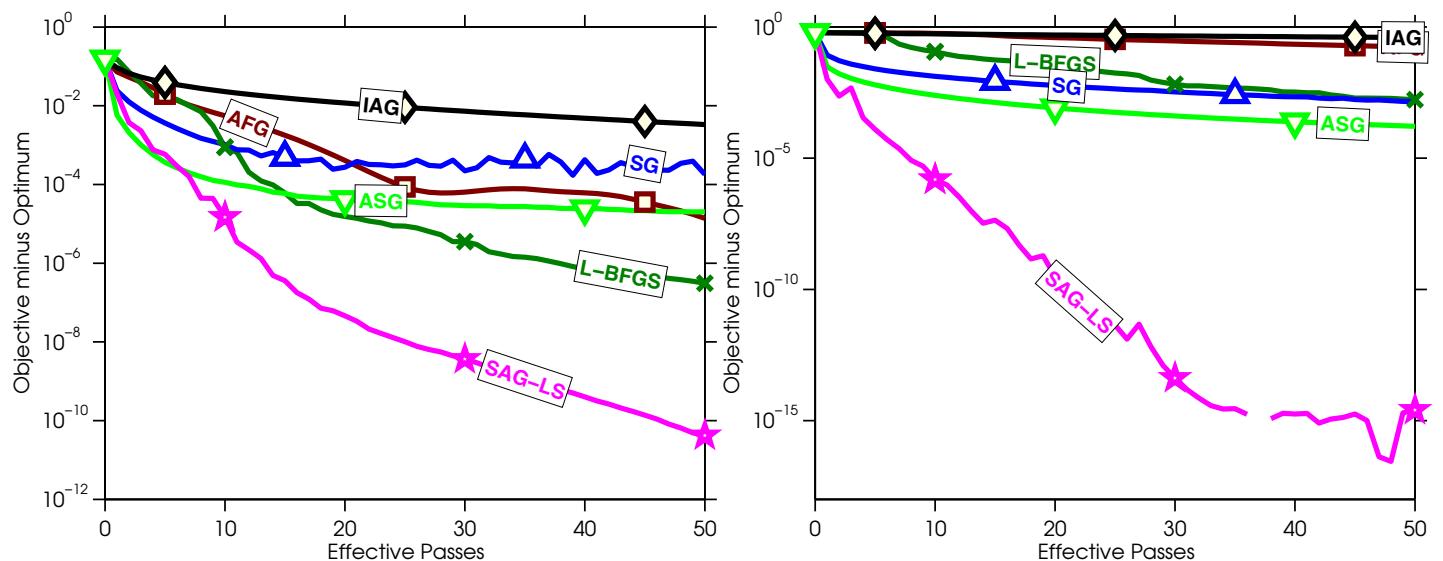
## Comparing FG and SG Methods

- quantum ( $n = 50000$ ,  $p = 78$ ) and rcv1 ( $n = 697641$ ,  $p = 47236$ )



## SAG Compared to FG and SG Methods

- quantum ( $n = 50000$ ,  $p = 78$ ) and rcv1 ( $n = 697641$ ,  $p = 47236$ )



## Coordinate Descent Methods

- Consider problems of the form

$$\min_x f(Ax) + \sum_{i=1}^n h_i(x_i), \quad \min_x \sum_{i \in \mathcal{V}} f_i(x_i) + \sum_{(i,j) \in \mathcal{E}} f_{ij}(x_i, x_j),$$

where  $f$  and  $f_{ij}$  are smooth and  $G = \{\mathcal{V}, \mathcal{E}\}$  is a graph.

## Coordinate Descent Methods

- Consider problems of the form

$$\min_x f(Ax) + \sum_{i=1}^n h_i(x_i), \quad \min_x \sum_{i \in \mathcal{V}} f_i(x_i) + \sum_{(i,j) \in \mathcal{E}} f_{ij}(x_i, x_j),$$

where  $f$  and  $f_{ij}$  are smooth and  $G = \{\mathcal{V}, \mathcal{E}\}$  is a graph.

- Appealing strategy for these problems is [coordinate descent](#):

$$x_{\mathbf{j}}^+ = x_{\mathbf{j}} - \alpha \nabla_{\mathbf{j}} f(x).$$

(i.e., update one variable at a time)

- We can typically perform a cheap and precise [line-search](#) for  $\alpha$ .

## Convergence Rate of Coordinate Descent

- The *steepest descent* choice is  $j = \arg \max_j \{|\nabla_j f(x)|\}$ .  
(but only efficient to calculate in some special cases)

## Convergence Rate of Coordinate Descent

- The *steepest descent* choice is  $j = \arg \max_j \{|\nabla_j f(x)|\}$ .  
(but only efficient to calculate in some special cases)
- Convergence rate (strongly-convex, gradient is  $L$ -Lipschitz):

$$O((1 - \mu/LD)^t)$$

## Convergence Rate of Coordinate Descent

- The *steepest descent* choice is  $j = \arg \max_j \{|\nabla_j f(x)|\}$ .  
(but only efficient to calculate in some special cases)
- Convergence rate (strongly-convex, gradient is  $L$ -Lipschitz):  
$$O((1 - \mu/LD)^t)$$
- This  $L$  is typically much smaller than  $L$  across all coordinates:

## Convergence Rate of Coordinate Descent

- The *steepest descent* choice is  $j = \arg \max_j \{|\nabla_j f(x)|\}$ .  
(but only efficient to calculate in some special cases)
- Convergence rate (strongly-convex, gradient is  $L$ -Lipschitz):
$$O((1 - \mu/LD)^t)$$
- This  $L$  is typically much smaller than  $L$  across all coordinates:
  - Coordinate descent if we can do  $D$  coordinate descent steps for cost of one gradient step.

## Convergence Rate of Coordinate Descent

- The *steepest descent* choice is  $j = \arg \max_j \{|\nabla_j f(x)|\}$ .  
(but only efficient to calculate in some special cases)
- Convergence rate (strongly-convex, gradient is  $L$ -Lipschitz):

$$O((1 - \mu/LD)^t)$$

- This  $L$  is typically much smaller than  $L$  across all coordinates:
  - Coordinate descent if we can do  $D$  coordinate descent steps for cost of one gradient step.
- Choosing a random coordinate  $j$  has same rate as steepest coordinate descent.[Nesterov, 2010]

## Convergence Rate of Coordinate Descent

- The *steepest descent* choice is  $j = \arg \max_j \{|\nabla_j f(x)|\}$ .  
(but only efficient to calculate in some special cases)
- Convergence rate (strongly-convex, gradient is  $L$ -Lipschitz):

$$O((1 - \mu/LD)^t)$$

- This  $L$  is typically much smaller than  $L$  across all coordinates:
  - Coordinate descent if we can do  $D$  coordinate descent steps for cost of one gradient step.
- Choosing a random coordinate  $j$  has same rate as steepest coordinate descent.[Nesterov, 2010]
- Various extensions:
  - Non-uniform sampling (Lipschitz sampling) [Nesterov, 2010]

## Convergence Rate of Coordinate Descent

- The *steepest descent* choice is  $j = \arg \max_j \{|\nabla_j f(x)|\}$ .  
(but only efficient to calculate in some special cases)
- Convergence rate (strongly-convex, gradient is  $L$ -Lipschitz):

$$O((1 - \mu/LD)^t)$$

- This  $L$  is typically much smaller than  $L$  across all coordinates:
  - Coordinate descent if we can do  $D$  coordinate descent steps for cost of one gradient step.
- Choosing a random coordinate  $j$  has same rate as steepest coordinate descent.[Nesterov, 2010]
- Various extensions:
  - Non-uniform sampling (Lipschitz sampling) [Nesterov, 2010]
  - Projected coordinate descent (product constraints)  
[Nesterov, 2010]

## Convergence Rate of Coordinate Descent

- The *steepest descent* choice is  $j = \arg \max_j \{|\nabla_j f(x)|\}$ .  
(but only efficient to calculate in some special cases)
- Convergence rate (strongly-convex, gradient is  $L$ -Lipschitz):

$$O((1 - \mu/LD)^t)$$

- This  $L$  is typically much smaller than  $L$  across all coordinates:
  - Coordinate descent if we can do  $D$  coordinate descent steps for cost of one gradient step.
- Choosing a random coordinate  $j$  has same rate as steepest coordinate descent.[Nesterov, 2010]
- Various extensions:
  - Non-uniform sampling (Lipschitz sampling) [Nesterov, 2010]
  - Projected coordinate descent (product constraints)  
[Nesterov, 2010]
  - Proximal coordinate descent (separable non-smooth term)  
Richtarik & Takac, 2011]

## Convergence Rate of Coordinate Descent

- The *steepest descent* choice is  $j = \arg \max_j \{|\nabla_j f(x)|\}$ .  
(but only efficient to calculate in some special cases)
- Convergence rate (strongly-convex, gradient is  $L$ -Lipschitz):

$$O((1 - \mu/LD)^t)$$

- This  $L$  is typically much smaller than  $L$  across all coordinates:
  - Coordinate descent if we can do  $D$  coordinate descent steps for cost of one gradient step.
- Choosing a random coordinate  $j$  has same rate as steepest coordinate descent.[Nesterov, 2010]
- Various extensions:
  - Non-uniform sampling (Lipschitz sampling) [Nesterov, 2010]
  - Projected coordinate descent (product constraints)  
[Nesterov, 2010]
  - Proximal coordinate descent (separable non-smooth term)  
Richtarik & Takac, 2011]
  - Frank-Wolfe coordinate descent (product constraints)  
[LaCoste-Julien et al., 2013]

## Convergence Rate of Coordinate Descent

- The *steepest descent* choice is  $j = \arg \max_j \{|\nabla_j f(x)|\}$ .  
(but only efficient to calculate in some special cases)
- Convergence rate (strongly-convex, gradient is  $L$ -Lipschitz):

$$O((1 - \mu/LD)^t)$$

- This  $L$  is typically much smaller than  $L$  across all coordinates:
  - Coordinate descent if we can do  $D$  coordinate descent steps for cost of one gradient step.
- Choosing a random coordinate  $j$  has same rate as steepest coordinate descent.[Nesterov, 2010]
- Various extensions:
  - Non-uniform sampling (Lipschitz sampling) [Nesterov, 2010]
  - Projected coordinate descent (product constraints)  
[Nesterov, 2010]
  - Proximal coordinate descent (separable non-smooth term)  
Richtarik & Takac, 2011]
  - Frank-Wolfe coordinate descent (product constraints)  
[LaCoste-Julien et al., 2013]
  - Accelerated version [Fercoq & Richtarik, 2013]

## Randomized Linear Algebra

- Consider problems of the form

$$\min_x f(Ax),$$

where **bottleneck is matrix multiplication** and  $A$  is low-rank.

## Randomized Linear Algebra

- Consider problems of the form

$$\min_x f(Ax),$$

where **bottleneck is matrix multiplication** and  $A$  is low-rank.

- Randomized linear algebra approaches uses

$$A \approx Q(Q^T A),$$

or chooses random row/columns subsets.

## Randomized Linear Algebra

- Consider problems of the form

$$\min_x f(Ax),$$

where **bottleneck is matrix multiplication** and  $A$  is low-rank.

- Randomized linear algebra approaches uses

$$A \approx Q(Q^T A),$$

or chooses random row/columns subsets.

- Now work with  $f(Q(Q^T A))$ .

## Randomized Linear Algebra

- Consider problems of the form

$$\min_x f(Ax),$$

where **bottleneck is matrix multiplication** and  **$A$  is low-rank**.

- Randomized linear algebra approaches uses

$$A \approx Q(Q^T A),$$

or chooses random row/columns subsets.

- Now work with  $f(Q(Q^T A))$ .
- $Q$  formed from Gram-Schmidt and **matrix multiplication with random vectors** gives very good approximation bounds, if singular values decay quickly.[Halko et al., 2011, Mahoney, 2011]

## Randomized Linear Algebra

- Consider problems of the form

$$\min_x f(Ax),$$

where **bottleneck is matrix multiplication** and  **$A$  is low-rank**.

- Randomized linear algebra approaches uses

$$A \approx Q(Q^T A),$$

or chooses random row/columns subsets.

- Now work with  $f(Q(Q^T A))$ .
- $Q$  formed from Gram-Schmidt and **matrix multiplication with random vectors** gives very good approximation bounds, if singular values decay quickly.[Halko et al., 2011, Mahoney, 2011]
- May work quite badly if singular values decay slowly.

# Outline

- 1 Convex Functions
- 2 Smooth Optimization
- 3 Non-Smooth Optimization
- 4 Randomized Algorithms
- 5 Parallel/Distributed Optimization

## Motivation for Parallel and Distributed

- Two recent trends:
  - We aren't making large gains in serial computation speed.
  - Datasets no longer fit on a single machine.

## Motivation for Parallel and Distributed

- Two recent trends:
  - We aren't making large gains in serial computation speed.
  - Datasets no longer fit on a single machine.
- Result: we must use **parallel and distributed** computation.
- Two major issues:
  - **Synchronization**: we can't wait for the slowest machine.
  - **Communication**: we can't transfer all information.

## Embarassing Parallelism in Machine Learning

- A lot of machine learning problems are **embarrassingly parallel**:
  - Split task across  $M$  machines, solve independently, combine.

## Embarassing Parallelism in Machine Learning

- A lot of machine learning problems are **embarrassingly parallel**:
  - Split task across  $M$  machines, solve independently, combine.
- E.g., computing the gradient in deterministic gradient method,

$$\frac{1}{N} \sum_{i=1}^N \nabla f_i(x) = \frac{1}{N} \left( \sum_{i=1}^{N/M} \nabla f_i(x) + \sum_{i=(N/M)+1}^{2N/M} \nabla f_i(x) + \dots \right).$$

## Embarassing Parallelism in Machine Learning

- A lot of machine learning problems are **embarrassingly parallel**:
  - Split task across  $M$  machines, solve independently, combine.
- E.g., computing the gradient in deterministic gradient method,

$$\frac{1}{N} \sum_{i=1}^N \nabla f_i(x) = \frac{1}{N} \left( \sum_{i=1}^{N/M} \nabla f_i(x) + \sum_{i=(N/M)+1}^{2N/M} \nabla f_i(x) + \dots \right).$$

- These allow optimal **linear** speedups.
  - You should always consider this first!

## Asynchronous Computation

- Do we have to wait for the last computer to finish?

## Asynchronous Computation

- Do we have to wait for the last computer to finish?
- No!
- Updating asynchronously saves a lot of time.

## Asynchronous Computation

- Do we have to wait for the last computer to finish?
- No!
- Updating asynchronously saves a lot of time.
- E.g., stochastic gradient method on shared memory:

$$x^{k+1} = x^k - \alpha \nabla f_{i_k}(x^{k-\textcolor{blue}{m}}).$$

## Asynchronous Computation

- Do we have to wait for the last computer to finish?
- No!
- Updating asynchronously saves a lot of time.
- E.g., stochastic gradient method on shared memory:

$$x^{k+1} = x^k - \alpha \nabla f_{i_k}(x^{k-\textcolor{blue}{m}}).$$

- You need to decrease step-size in proportion to asynchrony.
- Convergence rate decays elegantly with delay  $\textcolor{blue}{m}$ . [Niu et al., 2011]

## Reduced Communication: Parallel Coordinate Descent

- It may be expensive to communicate parameters  $x$ .

## Reduced Communication: Parallel Coordinate Descent

- It may be expensive to communicate parameters  $x$ .
- One solution: use parallel coordinate descent:

$$\begin{aligned}x_{j_1} &= x_{j_1} - \alpha_{j_1} \nabla_{j_1} f(x) \\x_{j_2} &= x_{j_2} - \alpha_{j_2} \nabla_{j_2} f(x) \\x_{j_3} &= x_{j_3} - \alpha_{j_3} \nabla_{j_3} f(x)\end{aligned}$$

- Only needs to communicate single coordinates.

## Reduced Communication: Parallel Coordinate Descent

- It may be expensive to communicate parameters  $x$ .
- One solution: use [parallel coordinate descent](#):

$$\begin{aligned}x_{j_1} &= x_{j_1} - \alpha_{j_1} \nabla_{j_1} f(x) \\x_{j_2} &= x_{j_2} - \alpha_{j_2} \nabla_{j_2} f(x) \\x_{j_3} &= x_{j_3} - \alpha_{j_3} \nabla_{j_3} f(x)\end{aligned}$$

- Only needs to communicate single coordinates.
- Again need to decrease step-size for convergence.
- Speedup is based on density of graph.[Richtarik & Takac, 2013]

## Reduced Communication: Decentralized Gradient

- We may need to distribute the data across machines.
- We may not want to update a ‘centralized’ vector  $x$ .

## Reduced Communication: Decentralized Gradient

- We may need to distribute the data across machines.
- We may not want to update a ‘centralized’ vector  $x$ .
- One solution: **decentralized gradient method**:
  - Each processor has its own data samples  $f_1, f_2, \dots, f_m$ .
  - Each processor has its own parameter vector  $x_c$ .
  - Each processor only communicates with a limited number of neighbours  $\text{nei}(c)$ .

## Reduced Communication: Decentralized Gradient

- We may need to distribute the data across machines.
- We may not want to update a ‘centralized’ vector  $x$ .
- One solution: **decentralized gradient method**:
  - Each processor has its own data samples  $f_1, f_2, \dots, f_m$ .
  - Each processor has its own parameter vector  $x_c$ .
  - Each processor only communicates with a limited number of neighbours  $\text{nei}(c)$ .

$$x_c = \frac{1}{|\text{nei}(c)|} \sum_{c' \in \text{nei}(c)} x_c - \frac{\alpha_c}{M} \sum_{i=1}^M \nabla f_i(x_c).$$

- Gradient descent is special case where all neighbours communicate.
- With modified update, rate decays gracefully as graph becomes sparse.[Shi et al., 2014]
- Can also consider communication failures.[Agarwal & Duchi, 2011]

## Summary

Summary:

- Part 1: Convex functions have special properties that allow us to efficiently minimize them.
- Part 2: Gradient-based methods allow elegant scaling with problems sizes for smooth problems.
- Part 3: Tricks like proximal-gradient methods allow the same scaling for many non-smooth problems.
- Part 4: Randomized algorithms allow even further scaling for problem structures that commonly arise in machine learning.
- Part 5: The future will require parallel and distributed that are asynchronous and are careful about communication costs.

Thank you for coming and staying until the end!