Optimization for Machine Learning

CS 406

Mark Schmidt

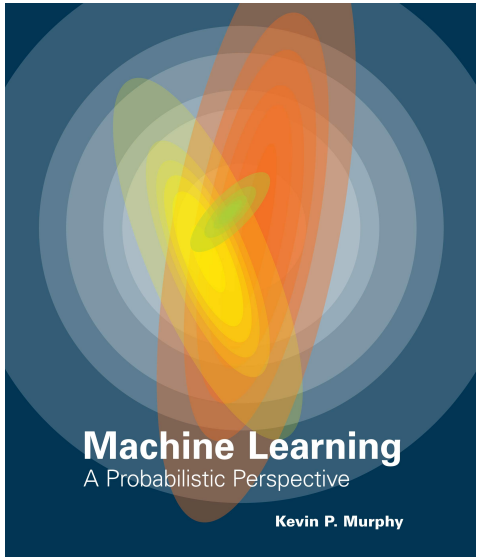
UBC Computer Science

Term 2, 2014-15

Goals of this Lecture

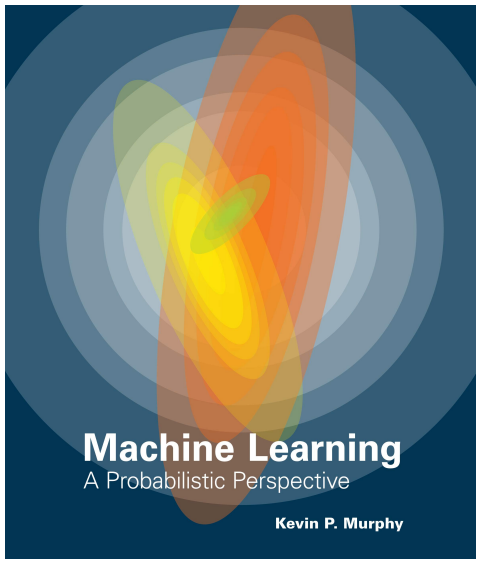
1. Give an overview and motivation for the machine learning technique of supervised learning.
2. Generalize convergence rates of gradient methods for solving linear systems to general smooth convex optimization problems.
3. Introduce the proximal-gradient algorithm, one of the most efficient algorithms for solving special classes of non-smooth convex optimization problems.
4. Introduce the stochastic-gradient algorithm, for solving data-fitting problems when the size of the data is very large.

Machine Learning



- *Study of using computers to automatically detect patterns in data, and use these to make predictions or decisions.*

Machine Learning



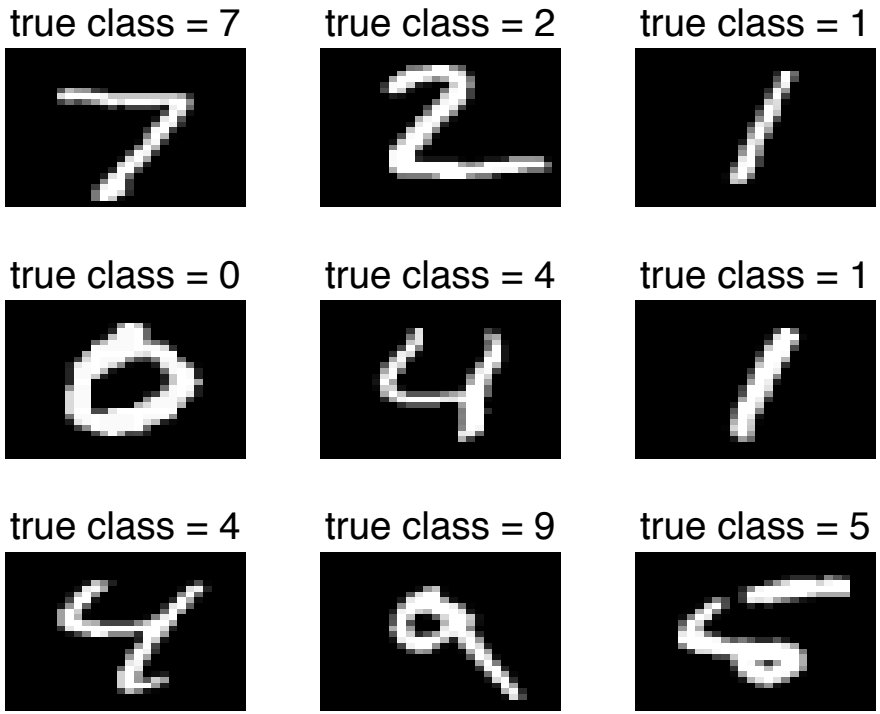
- *Study of using computers to automatically detect patterns in data, and use these to make predictions or decisions*.
- One of the fastest-growing areas of science/engineering.
- Recent successes: Kinect, book/movie recommendation, spam detection, credit card fraud detection, face recognition, speech recognition, object recognition, self-driving cars.

Supervised learning

- Supervised learning:
  - Given input and output examples.
  - Build a model that predicts the output from the inputs.
  - You can use the model to predict the output on new inputs.

Supervised learning

- Supervised learning:
  - Given input and output examples.
  - Build a model that predicts the output from the inputs.
  - You can use the model to predict the output on new inputs.
- Canonical example: hand-written digit recognition:



Supervised Learning

Supervised Learning

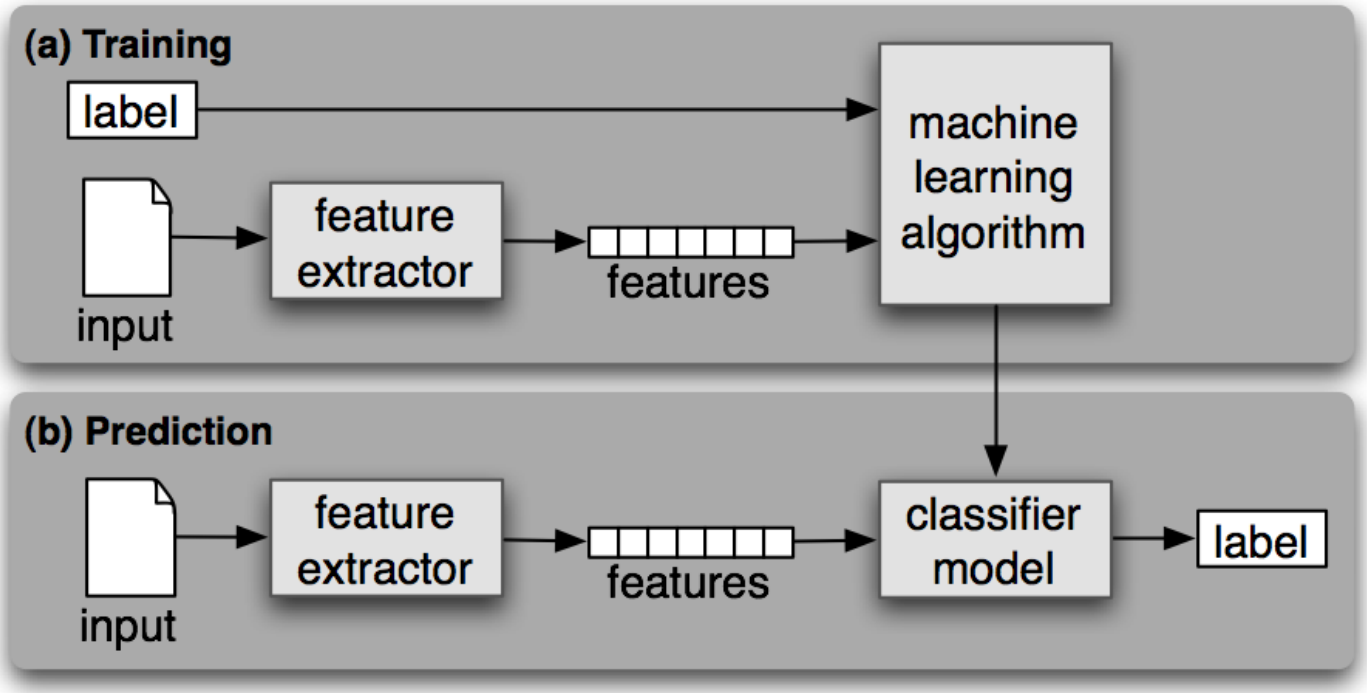
- You have a well-defined pattern recognition problem.
- But don't know how to write a program to solve it.

Supervised Learning

- You have a well-defined pattern recognition problem.
- But don't know how to write a program to solve it.
- And you have lots of labeled data.
- Key reason for machine learning's popularity and success.

Training and Testing

- Steps for supervised learning:
  1. Training phase: build model that maps from input features to labels. (based on many examples of the correct behaviour)
  2. Testing phase: model is used to label new inputs.



Connection to Numerical Optimization

- Typically, the training phase is formulated as an optimization problem,

$$\min_{\mathbf{x}} \quad \sum_{i=1}^{m} f_i(\mathbf{x}) \quad + \quad \lambda r(\mathbf{x}).$$

$$\text{data fitting term} \quad + \quad \text{regularizer}$$

Connection to Numerical Optimization

- Typically, the training phase is formulated as an optimization problem,

$$\min_{\mathbf{x}} \quad \sum_{i=1}^{m} f_i(\mathbf{x}) \quad + \quad \lambda r(\mathbf{x}).$$

$$\text{data fitting term} \quad + \quad \text{regularizer}$$

- Data-fitting term: how well does $\mathbf{x}$ fit data sample i?
- Regularizer: how simple is $\mathbf{x}$?
  (simple models are more likely to do well at test time)

Connection to Numerical Optimization

- Typically, the training phase is formulated as an optimization problem,

$$\min_{\mathbf{x}} \quad \sum_{i=1}^{m} f_i(\mathbf{x}) \quad + \quad \lambda r(\mathbf{x}).$$

$$\text{data fitting term} \quad + \quad \text{regularizer}$$

- Data-fitting term: how well does $\mathbf{x}$ fit data sample i?
- Regularizer: how simple is $\mathbf{x}$?
  (simple models are more likely to do well at test time)
- Example is least squares,

$$f_i(\mathbf{x}) = \frac{1}{2}(b_i - \sum_{j=1}^{n} a_{ij}x_j)^2.$$

Connection to Numerical Optimization

- Typically, the training phase is formulated as an optimization problem,

$$\min_{\mathbf{x}} \quad \sum_{i=1}^{m} f_i(\mathbf{x}) \quad + \quad \lambda r(\mathbf{x}).$$

$$\text{data fitting term} \quad + \quad \text{regularizer}$$

- Data-fitting term: how well does $\mathbf{x}$ fit data sample i?
- Regularizer: how simple is $\mathbf{x}$?
  (simple models are more likely to do well at test time)
- Example is least squares,

$$f_i(\mathbf{x}) = \frac{1}{2}(b_i - \sum_{j=1}^{n} a_{ij}x_j)^2.$$

- Squared ℓ_2-norm regularization:

$$r(\mathbf{x}) = \|\mathbf{x}\|^2.$$

Outline

Motivation for First-Order Methods

- We first consider the unconstrained optimization problem,

$$\min_{x} f(x).$$

- In typical ML models the dimension, dimension n is very large.
- We will focus on matrix-free methods, as in the previous lecture:
  - Allows n to be in the billions or more.
  - We can show dimension-independent convergence rates.

Motivation for First-Order Methods

- We first consider the unconstrained optimization problem,

$$\min_{\mathbf{x}} f(x).$$

- In typical ML models the dimension, dimension n is very large.
- We will focus on matrix-free methods, as in the previous lecture:
  - Allows n to be in the billions or more.
  - We can show dimension-independent convergence rates.
- As before, the simplest case is gradient descent,

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k).$$

- How many iterations are needed?

Strongly-Convex and Strongly-Smooth

- Consider special case of least squares:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \frac{1}{2}\|\mathbf{b} - A\mathbf{x}\|^2.$$

- Recall that

$$\nabla^2 f(\mathbf{x}) = A^T A,$$

so the eigenvalues of $\nabla^2 f(\mathbf{x})$ are between λ_1 and λ_n for all $\mathbf{x}$.

Strongly-Convex and Strongly-Smooth

- Consider special case of least squares:

$$\min_{\mathbf{x}} f(\mathbf{x}) = \frac{1}{2}\|\mathbf{b} - A\mathbf{x}\|^2.$$

- Recall that

$$\nabla^2 f(\mathbf{x}) = A^T A,$$

  so the eigenvalues of $\nabla^2 f(\mathbf{x})$ are between λ_1 and λ_n for all $\mathbf{x}$.
- Functions f with eigenvalues of Hessian bounded between positive constants for all f are called 'strongly smooth' and 'strongly convex'.
- These assumptions are sufficient to show a linear convergence rate.

Implication of Strong-Smoothness

- From Taylor's theorem, for some $\mathbf{z}$ we have:

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

Implication of Strong-Smoothness

- From Taylor's theorem, for some $\mathbf{z}$ we have:

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

- Use that $\mathbf{v}^T\nabla^2 f(\mathbf{z})\mathbf{v} \leq \lambda_1 \mathbf{v}^T\mathbf{v} = \lambda_1\|v\|^2$ for any $\mathbf{v}$ and $\mathbf{z}$.

$$f(\mathbf{y}) \leq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y}-\mathbf{x}) + \frac{\lambda_1}{2}\|\mathbf{y}-\mathbf{x}\|^2$$

- *Global quadratic upper bound on function value*.

Implication of Strong-Smoothness

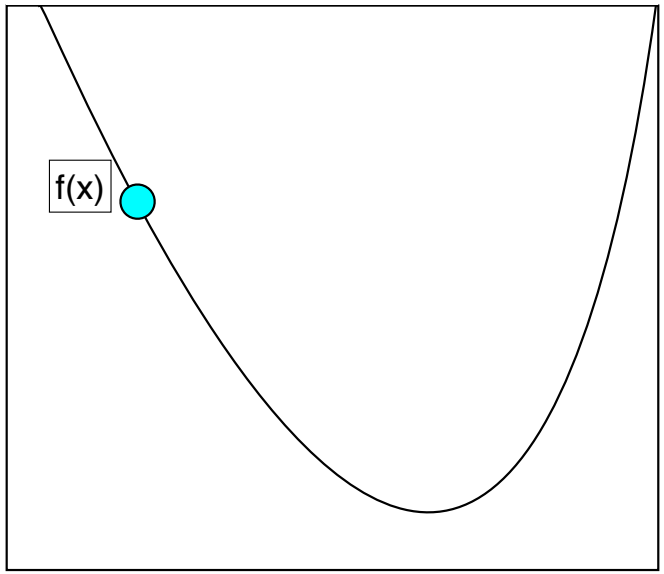
- From Taylor's theorem, for some $\mathbf{z}$ we have:

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

- Use that $\mathbf{v}^T\nabla^2 f(\mathbf{z})\mathbf{v} \leq \lambda_1 \mathbf{v}^T\mathbf{v} = \lambda_1\|v\|^2$ for any $\mathbf{v}$ and $\mathbf{z}$.

$$f(\mathbf{y}) \leq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y}-\mathbf{x}) + \frac{\lambda_1}{2}\|\mathbf{y}-\mathbf{x}\|^2$$

- *Global quadratic upper bound on function value.*



Implication of Strong-Smoothness

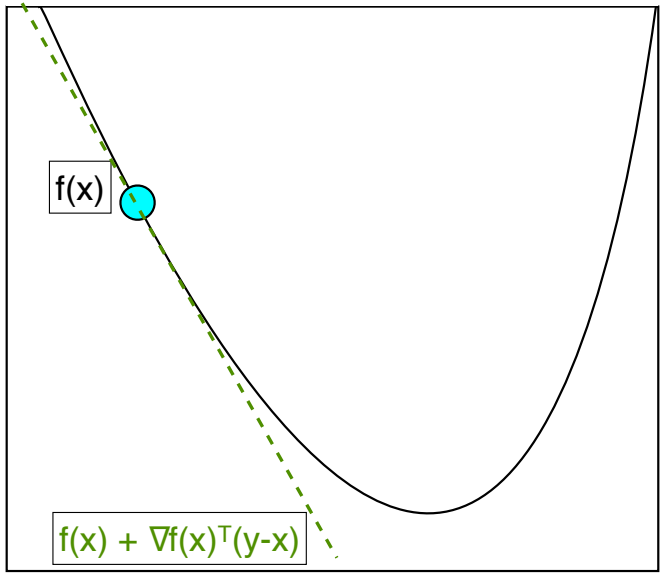
- From Taylor's theorem, for some $\mathbf{z}$ we have:

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

- Use that $\mathbf{v}^T\nabla^2 f(\mathbf{z})\mathbf{v} \leq \lambda_1 \mathbf{v}^T\mathbf{v} = \lambda_1\|v\|^2$ for any $\mathbf{v}$ and $\mathbf{z}$.

$$f(\mathbf{y}) \leq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y}-\mathbf{x}) + \frac{\lambda_1}{2}\|\mathbf{y}-\mathbf{x}\|^2$$

- *Global quadratic upper bound on function value.*



Implication of Strong-Smoothness

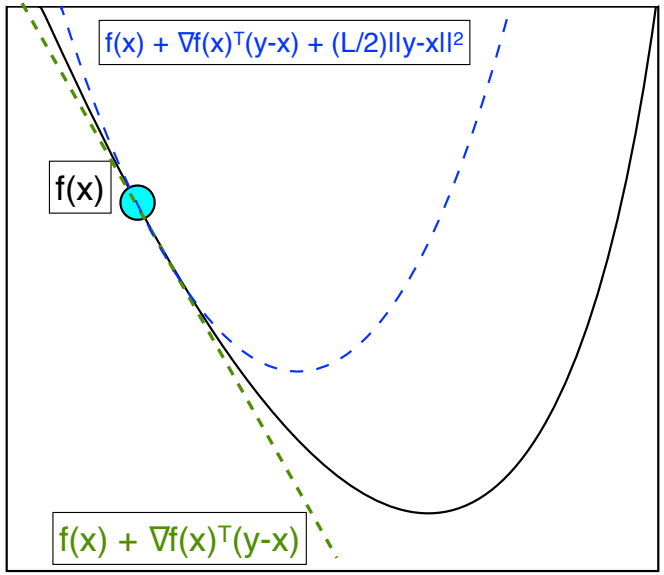
- From Taylor's theorem, for some $\mathbf{z}$ we have:

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

- Use that $\mathbf{v}^T\nabla^2 f(\mathbf{z})\mathbf{v} \leq \lambda_1 \mathbf{v}^T\mathbf{v} = \lambda_1\|v\|^2$ for any $\mathbf{v}$ and $\mathbf{z}$.

$$f(\mathbf{y}) \leq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y}-\mathbf{x}) + \frac{\lambda_1}{2}\|\mathbf{y}-\mathbf{x}\|^2$$

- *Global quadratic upper bound on function value.*



Implication of Strong-Convexity

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

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

Implication of Strong-Convexity

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

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

- Use that $\mathbf{v}^T\nabla^2 f(\mathbf{z})\mathbf{v} \geq \lambda_n\|\mathbf{v}\|^2$ for any $\mathbf{v}$ and $\mathbf{z}$.

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

- *Global quadratic lower bound on function value.*

Implication of Strong-Convexity

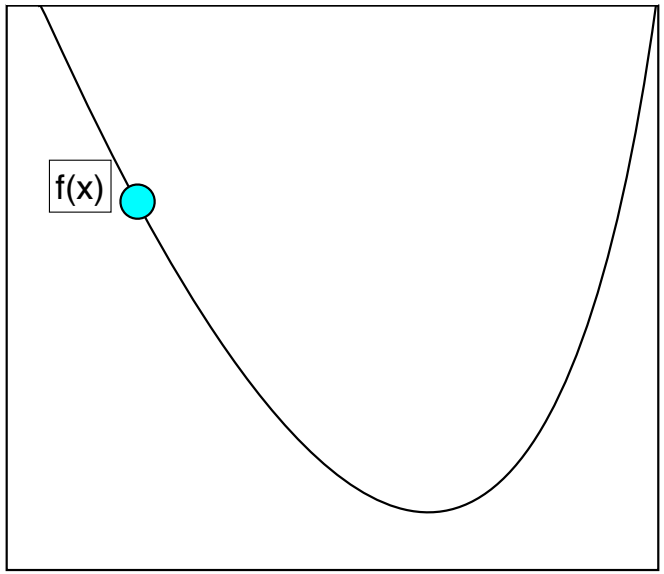
- From Taylor's theorem, for some z we have:

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

- Use that $\mathbf{v}^T\nabla^2 f(\mathbf{z})\mathbf{v} \geq \lambda_n\|\mathbf{v}\|^2$ for any $\mathbf{v}$ and $\mathbf{z}$.

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

- *Global quadratic lower bound on function value.*



Implication of Strong-Convexity

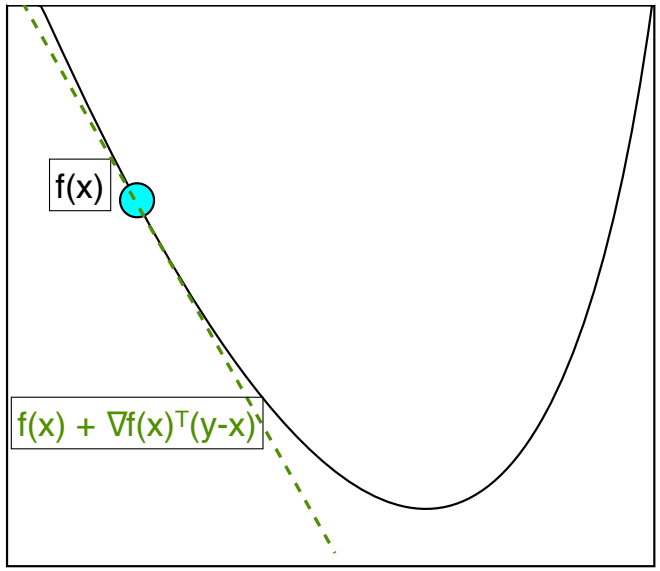
- From Taylor's theorem, for some z we have:

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

- Use that $\mathbf{v}^T\nabla^2 f(\mathbf{z})\mathbf{v} \geq \lambda_n\|\mathbf{v}\|^2$ for any $\mathbf{v}$ and $\mathbf{z}$.

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

- *Global quadratic lower bound on function value.*



Implication of Strong-Convexity

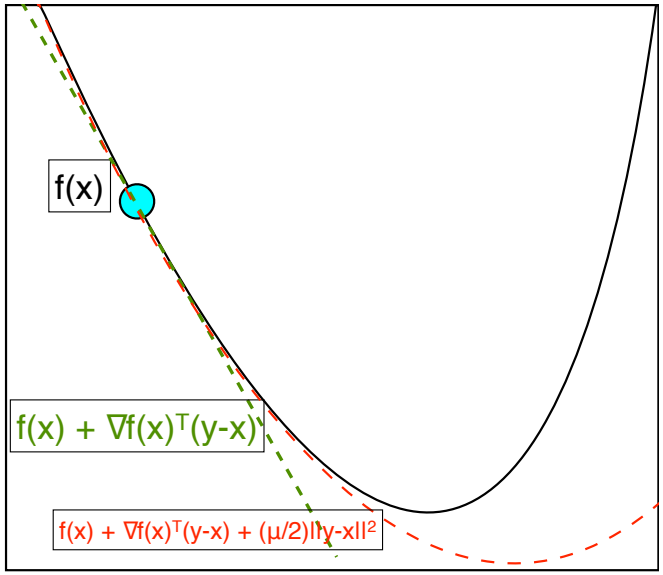
- From Taylor's theorem, for some z we have:

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

- Use that $\mathbf{v}^T\nabla^2 f(\mathbf{z})\mathbf{v} \geq \lambda_n\|\mathbf{v}\|^2$ for any $\mathbf{v}$ and $\mathbf{z}$.

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

- *Global quadratic lower bound on function value.*



Bounds on Progress and Sub-Optimality

- We have the upper bound

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T(\mathbf{x}_{k+1} - \mathbf{x}_k) + \frac{\lambda_1}{2}\|\mathbf{x}_{k+1} - \mathbf{x}_k\|^2,$$

Bounds on Progress and Sub-Optimality

- We have the upper bound

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T(\mathbf{x}_{k+1} - \mathbf{x}_k) + \frac{\lambda_1}{2}\|\mathbf{x}_{k+1} - \mathbf{x}_k\|^2,$$

treating $\mathbf{x}_{k+1}$ as a variable and minimizing the right side gives

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{1}{\lambda_1}\nabla f(\mathbf{x}_k), \quad f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) - \frac{1}{2\lambda_1}\|\nabla f(\mathbf{x}_k)\|^2,$$

Bounds on Progress and Sub-Optimality

- We have the upper bound

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T(\mathbf{x}_{k+1} - \mathbf{x}_k) + \frac{\lambda_1}{2}\|\mathbf{x}_{k+1} - \mathbf{x}_k\|^2,$$

treating $\mathbf{x}_{k+1}$ as a variable and minimizing the right side gives

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{1}{\lambda_1}\nabla f(\mathbf{x}_k), \quad f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) - \frac{1}{2\lambda_1}\|\nabla f(\mathbf{x}_k)\|^2,$$

which is gradient descent with a particular step-size.

Bounds on Progress and Sub-Optimality

- We have the upper bound

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T(\mathbf{x}_{k+1} - \mathbf{x}_k) + \frac{\lambda_1}{2}\|\mathbf{x}_{k+1} - \mathbf{x}_k\|^2,$$

  treating $\mathbf{x}_{k+1}$ as a variable and minimizing the right side gives

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{1}{\lambda_1}\nabla f(\mathbf{x}_k), \quad f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) - \frac{1}{2\lambda_1}\|\nabla f(\mathbf{x}_k)\|^2,$$

  which is gradient descent with a particular step-size.
- We have the lower bound

$$f(\mathbf{y}) \geq f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T(\mathbf{y} - \mathbf{x}_k) + \frac{\lambda_n}{2}\|\mathbf{y} - \mathbf{x}_k\|^2,$$

Bounds on Progress and Sub-Optimality

- We have the upper bound

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T(\mathbf{x}_{k+1} - \mathbf{x}_k) + \frac{\lambda_1}{2}\|\mathbf{x}_{k+1} - \mathbf{x}_k\|^2,$$

  treating $\mathbf{x}_{k+1}$ as a variable and minimizing the right side gives

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{1}{\lambda_1}\nabla f(\mathbf{x}_k), \quad f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) - \frac{1}{2\lambda_1}\|\nabla f(\mathbf{x}_k)\|^2,$$

  which is gradient descent with a particular step-size.
- We have the lower bound

$$f(\mathbf{y}) \geq f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T(\mathbf{y} - \mathbf{x}_k) + \frac{\lambda_n}{2}\|\mathbf{y} - \mathbf{x}_k\|^2,$$

  and minimizing both sides in terms of y gives

$$f(\mathbf{x}^*) \geq f(\mathbf{x}_k) - \frac{1}{2\lambda_n}\|\nabla f(\mathbf{x}_k)\|^2,$$

  which bounds how far $\mathbf{x}_k$ is from the solution $\mathbf{x}^*$.

Linear Convergence of Gradient Descent

- We have bounds on $\mathbf{x}_{k+1}$ and $\mathbf{x}^*$:

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) - \frac{1}{2\lambda_1}\|\nabla f(\mathbf{x}_k)\|^2, \quad f(\mathbf{x}^*) \geq f(\mathbf{x}_k) - \frac{1}{2\lambda_n}\|\nabla f(\mathbf{x}_k)\|^2.$$

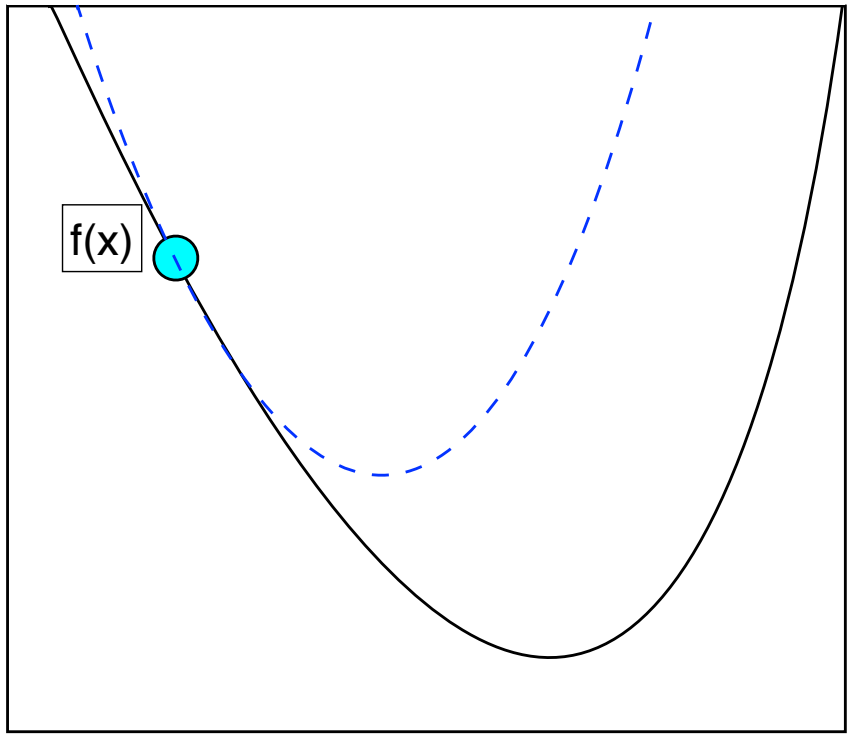
- The bound guaranteed progress and maximum sub-optimality.

Linear Convergence of Gradient Descent

- We have bounds on $\mathbf{x}_{k+1}$ and $\mathbf{x}^*$:

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) - \frac{1}{2\lambda_1}\|\nabla f(\mathbf{x}_k)\|^2, \quad f(\mathbf{x}^*) \geq f(\mathbf{x}_k) - \frac{1}{2\lambda_n}\|\nabla f(\mathbf{x}_k)\|^2.$$

- The bound guaranteed progress and maximum sub-optimality.

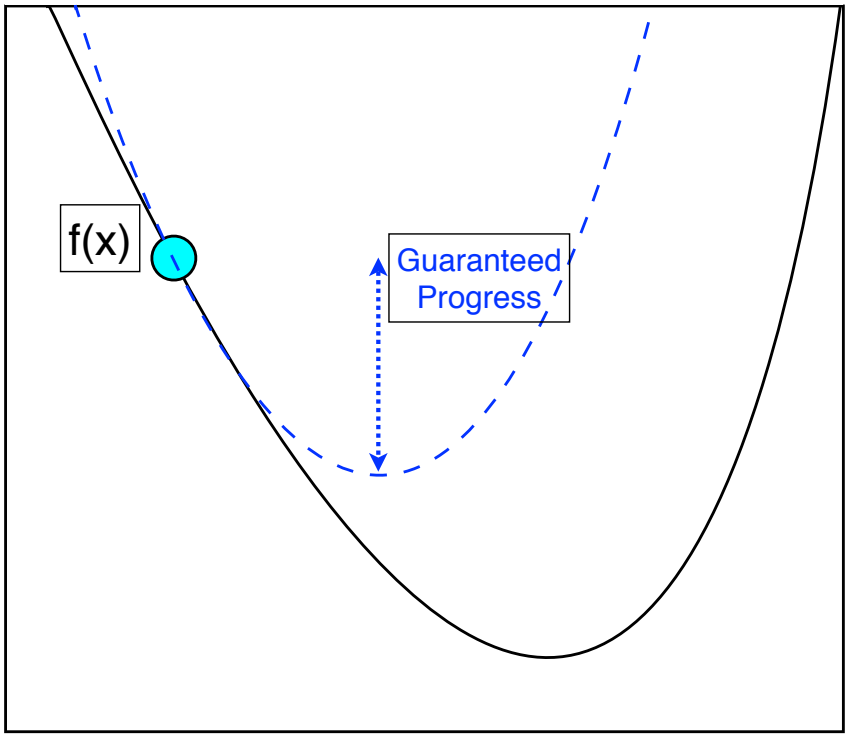


Linear Convergence of Gradient Descent

- We have bounds on $\mathbf{x}_{k+1}$ and $\mathbf{x}^*$:

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) - \frac{1}{2\lambda_1}\|\nabla f(\mathbf{x}_k)\|^2, \quad f(\mathbf{x}^*) \geq f(\mathbf{x}_k) - \frac{1}{2\lambda_n}\|\nabla f(\mathbf{x}_k)\|^2.$$

- The bound guaranteed progress and maximum sub-optimality.

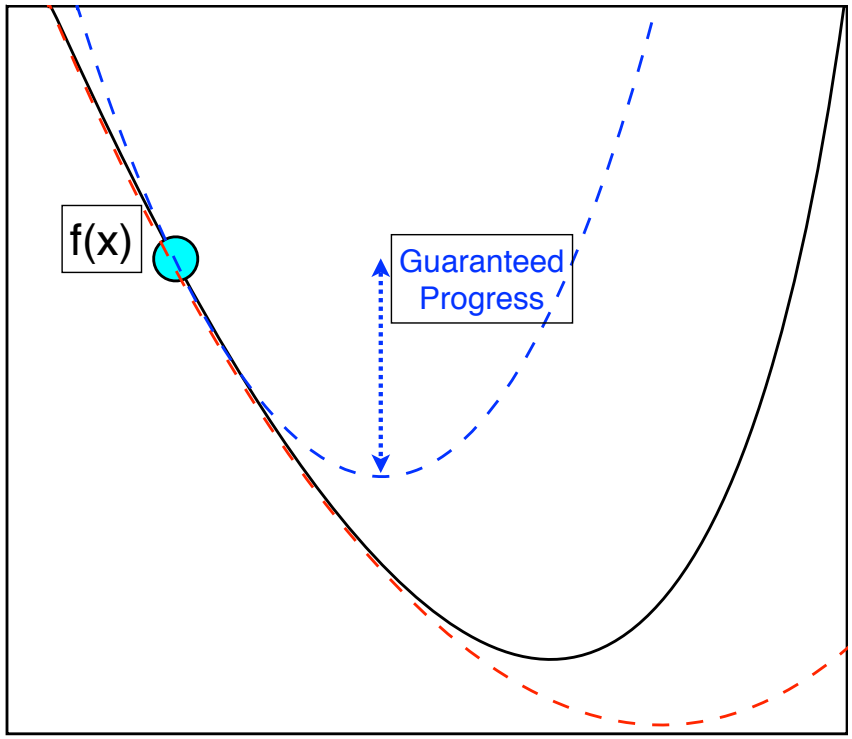


Linear Convergence of Gradient Descent

- We have bounds on $\mathbf{x}_{k+1}$ and $\mathbf{x}^*$:

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) - \frac{1}{2\lambda_1}\|\nabla f(\mathbf{x}_k)\|^2, \quad f(\mathbf{x}^*) \geq f(\mathbf{x}_k) - \frac{1}{2\lambda_n}\|\nabla f(\mathbf{x}_k)\|^2.$$

- The bound guaranteed progress and maximum sub-optimality.

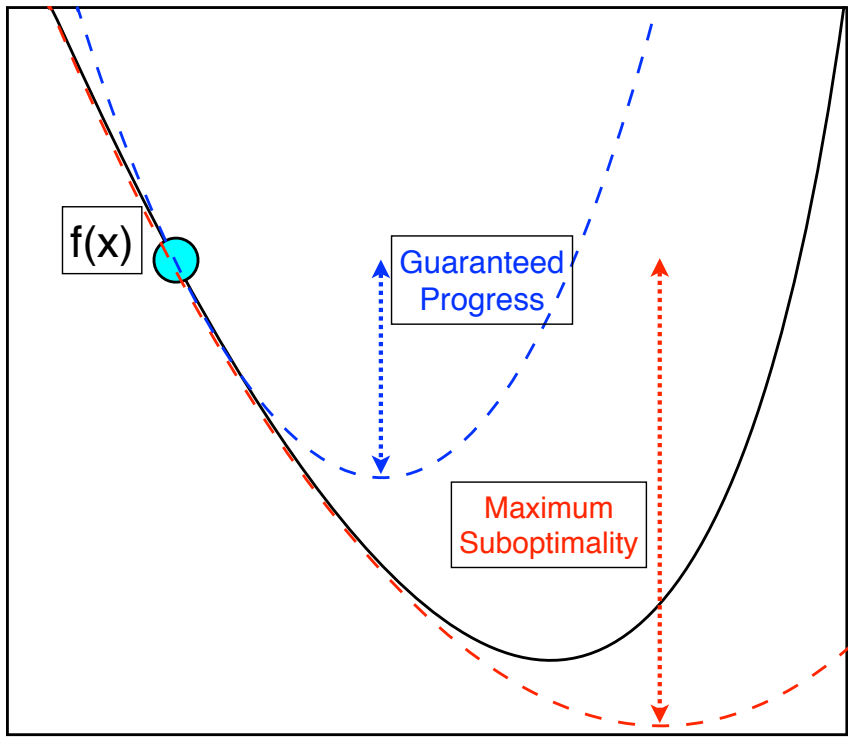


Linear Convergence of Gradient Descent

- We have bounds on $\mathbf{x}_{k+1}$ and $\mathbf{x}^*$:

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) - \frac{1}{2\lambda_1}\|\nabla f(\mathbf{x}_k)\|^2, \quad f(\mathbf{x}^*) \geq f(\mathbf{x}_k) - \frac{1}{2\lambda_n}\|\nabla f(\mathbf{x}_k)\|^2.$$

- The bound guaranteed progress and maximum sub-optimality.

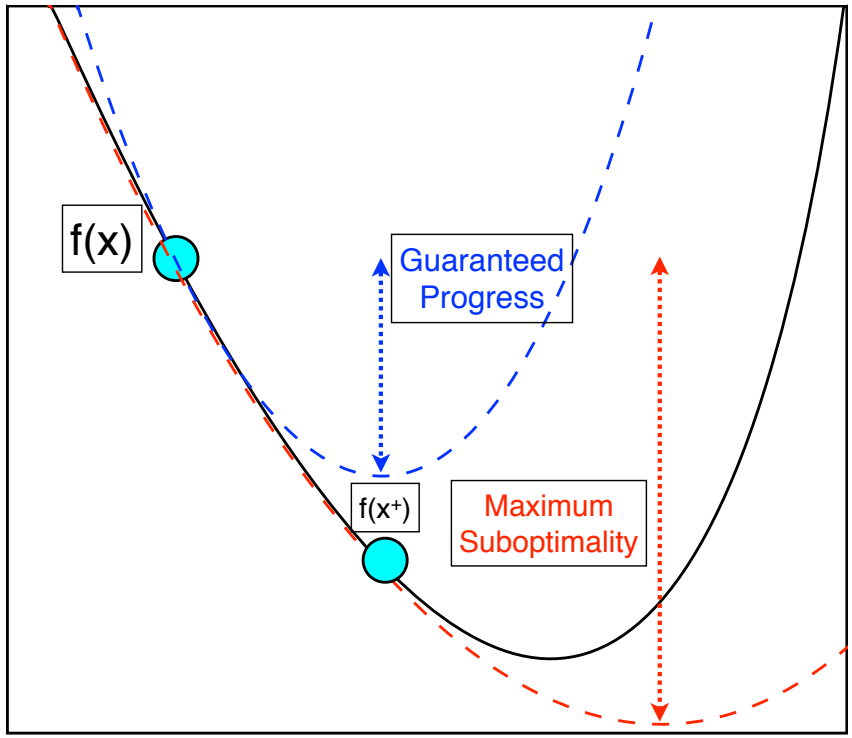


Linear Convergence of Gradient Descent

- We have bounds on $\mathbf{x}_{k+1}$ and $\mathbf{x}^*$:

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) - \frac{1}{2\lambda_1}\|\nabla f(\mathbf{x}_k)\|^2, \quad f(\mathbf{x}^*) \geq f(\mathbf{x}_k) - \frac{1}{2\lambda_n}\|\nabla f(\mathbf{x}_k)\|^2.$$

- The bound guaranteed progress and maximum sub-optimality.



Linear Convergence of Gradient Descent

- We have bounds on $\mathbf{x}_{k+1}$ and $\mathbf{x}^*$:

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) - \frac{1}{2\lambda_1}\|\nabla f(\mathbf{x}_k)\|^2, \quad f(\mathbf{x}^*) \geq f(\mathbf{x}_k) - \frac{1}{2\lambda_n}\|\nabla f(\mathbf{x}_k)\|^2.$$

combine them to get

$$f(\mathbf{x}_{k+1}) - f(\mathbf{x}^*) \leq \left(1 - \frac{\lambda_n}{\lambda_1}\right)[f(\mathbf{x}_k) - f(\mathbf{x}^*)]$$

Linear Convergence of Gradient Descent

- We have bounds on $\mathbf{x}_{k+1}$ and $\mathbf{x}^*$:

$$f(\mathbf{x}_{k+1}) \leq f(\mathbf{x}_k) - \frac{1}{2\lambda_1}\|\nabla f(\mathbf{x}_k)\|^2, \quad f(\mathbf{x}^*) \geq f(\mathbf{x}_k) - \frac{1}{2\lambda_n}\|\nabla f(\mathbf{x}_k)\|^2.$$

  combine them to get

$$f(\mathbf{x}_{k+1}) - f(\mathbf{x}^*) \leq \left(1 - \frac{\lambda_n}{\lambda_1}\right)[f(\mathbf{x}_k) - f(\mathbf{x}^*)]$$

- Applying recursively gives a linear convergence rate:

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \leq \left(1 - \frac{\lambda_n}{\lambda_1}\right)^k [f(\mathbf{x}_0) - f(\mathbf{x}^*)]$$

- We say that the condition number for f is given by $\kappa_f = \frac{\lambda_1}{\lambda_n}$.

Convergence Rate of Gradient Descent

- What about line-search?
  - Exact line-search has the same rate (using $\alpha_k = 1/\lambda_1$ is a special case).
  - Backtracking line-search has a slightly slower rate (but don't need λ_1).

Convergence Rate of Gradient Descent

- What about line-search?
  - Exact line-search has the same rate (using $\alpha_k = 1/\lambda_1$ is a special case).
  - Backtracking line-search has a slightly slower rate (but don't need λ_1).
- It's also possible to show that a different step-size gives

$$\|\mathbf{x}_k - \mathbf{x}^*\| \leq \left(\frac{\kappa_f - 1}{\kappa_f + 1}\right)^k \|\mathbf{x}_0 - \mathbf{x}^*\|.$$

Convergence Rate of Gradient Descent

- What about line-search?
  - Exact line-search has the same rate (using $\alpha_k = 1/\lambda_1$ is a special case).
  - Backtracking line-search has a slightly slower rate (but don't need λ_1).
- It's also possible to show that a different step-size gives

$$\|\mathbf{x}_k - \mathbf{x}^*\| \leq \left(\frac{\kappa_f - 1}{\kappa_f + 1}\right)^k \|\mathbf{x}_0 - \mathbf{x}^*\|.$$

- Similar to the rate for solving linear systems (last lecture).
- Can we derive a method with the faster rate of conjugate gradient?

  ('Non-linear' conjugate gradient methods don't actually have a faster rate.)

Nesterov's accelerated gradient method

- There is a method similar to conjugate gradient,

$$\mathbf{x}_{k+1} = \mathbf{y}_k - \alpha_k \nabla f(\mathbf{y}_k),$$
$$\mathbf{y}_{k+1} = \mathbf{x}_k + \beta_k(\mathbf{x}_{k+1} - \mathbf{x}_k),$$

  called Nesterov's accelerated gradient method.

Nesterov's accelerated gradient method

- There is a method similar to conjugate gradient,

$$\mathbf{x}_{k+1} = \mathbf{y}_k - \alpha_k \nabla f(\mathbf{y}_k),$$
$$\mathbf{y}_{k+1} = \mathbf{x}_k + \beta_k(\mathbf{x}_{k+1} - \mathbf{x}_k),$$

  called Nesterov's accelerated gradient method.

- It has a faster rate of

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \leq \left(1 - \frac{1}{\sqrt{\kappa_f}}\right)^t [f(\mathbf{x}_0) - f(\mathbf{x}^*)].$$

Nesterov's accelerated gradient method

- There is a method similar to conjugate gradient,

$$\mathbf{x}_{k+1} = \mathbf{y}_k - \alpha_k \nabla f(\mathbf{y}_k),$$
$$\mathbf{y}_{k+1} = \mathbf{x}_k + \beta_k(\mathbf{x}_{k+1} - \mathbf{x}_k),$$

  called Nesterov's accelerated gradient method.

- It has a faster rate of

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \leq \left(1 - \frac{1}{\sqrt{\kappa_f}}\right)^t [f(\mathbf{x}_0) - f(\mathbf{x}^*)].$$

- Slower in practice than non-linear conjugate gradient and quasi-Newton methods, but does not depend on dimension and generalizes to non-smooth problems...

Outline

Motivation: Spam Filtering

- We want to recognize e-mail spam.

Motivation: Spam Filtering

- We want to recognize e-mail spam.
- We will look at phrases in the e-mail messages:
  - "CPSC 406".
  - "Meet singles in your area now'

Motivation: Spam Filtering

- We want to recognize e-mail spam.
- We will look at phrases in the e-mail messages:
  - "CPSC 406".
  - "Meet singles in your area now'
- There are too many possible phrases (model would be huge).
- But some are more helpful than others: feature selection.

LASSO: Sparse Regularization

- Consider the ℓ_1-regularized least squares problem (LASSO),

$$\min_{\mathbf{x}} \frac{1}{2}\|\mathbf{b} - A\mathbf{x}\|^2 + \lambda\|\mathbf{x}\|_1.$$

- Recall the definition of the ℓ_1-norm,

$$\|\mathbf{x}\|_1 = \sum_j |x_j|.$$

LASSO: Sparse Regularization

- Consider the ℓ_1-regularized least squares problem (LASSO),

$$\min_{\mathbf{x}} \frac{1}{2}\|\mathbf{b} - A\mathbf{x}\|^2 + \lambda\|\mathbf{x}\|_1.$$

- Recall the definition of the ℓ_1-norm,

$$\|\mathbf{x}\|_1 = \sum_j |x_j|.$$

- The ℓ_1-norm shrinks $\mathbf{x}$, and encourages x_j to be exactly zero:
  - Weight x_j for "meet singles" now should be hi (relevant).
  - Weight x_j for "Hello" should be 0 (not relevant).
- Each column of A contains the values of one feature, so setting $x_j = 0$ means we ignore the feature.

LASSO: Sparse Regularization

- Consider the ℓ_1-regularized least squares problem (LASSO),

$$\min_{\mathbf{x}} \frac{1}{2}\|\mathbf{b} - A\mathbf{x}\|^2 + \lambda\|\mathbf{x}\|_1.$$

- Recall the definition of the ℓ_1-norm,

$$\|\mathbf{x}\|_1 = \sum_j |x_j|.$$

- The ℓ_1-norm shrinks $\mathbf{x}$, and encourages x_j to be exactly zero:
  - Weight x_j for "meet singles" now should be hi (relevant).
  - Weight x_j for "Hello" should be 0 (not relevant).
- Each column of A contains the values of one feature, so setting $x_j = 0$ means we ignore the feature.
- The challenge is that $|x_j|$ is non-differentiable.

LASSO: Sparse Regularization

- How can we solve non-differentiable problems like the LASSO?

LASSO: Sparse Regularization

- How can we solve non-differentiable problems like the LASSO?
- Try to convert it into a smooth problem?
  - We can write the LASSO as a quadratic program (QP).
  - But can't solve general huge-dimensional QPs.

LASSO: Sparse Regularization

- How can we solve non-differentiable problems like the LASSO?
- Try to convert it into a smooth problem?
  - We can write the LASSO as a quadratic program (QP).
  - But can't solve general huge-dimensional QPs.
- Use an off-the-shelf non-smooth solver?
  - These methods have sub-linear convergence rates.
  - They are very slow!

LASSO: Sparse Regularization

- How can we solve non-differentiable problems like the LASSO?
- Try to convert it into a smooth problem?
  - We can write the LASSO as a quadratic program (QP).
  - But can't solve general huge-dimensional QPs.
- Use an off-the-shelf non-smooth solver?
  - These methods have sub-linear convergence rates.
  - They are very slow!
- Use a special class of methods called proximal-gradient methods.

Example: Non-Negative Least Squares

- Consider non-negative least squares,

$$\min_{\mathbf{x} \geq 0} \sum_{i=1}^{m} (b_i - \sum_{j=1}^{n} a_{ij} x_j)^2,$$

- Should this be easier than with general constraints?

Example: Non-Negative Least Squares

- Consider non-negative least squares,

$$\min_{\mathbf{x} \geq 0} \sum_{i=1}^{m} (b_i - \sum_{j=1}^{n} a_{ij} x_j)^2,$$

- Should this be easier than with general constraints?
- The constraints are simple:
  - Given y, we can efficiently find closest x satisfying constraints.
    (just set negative y_i to zero)

Example: Non-Negative Least Squares

- Consider non-negative least squares,

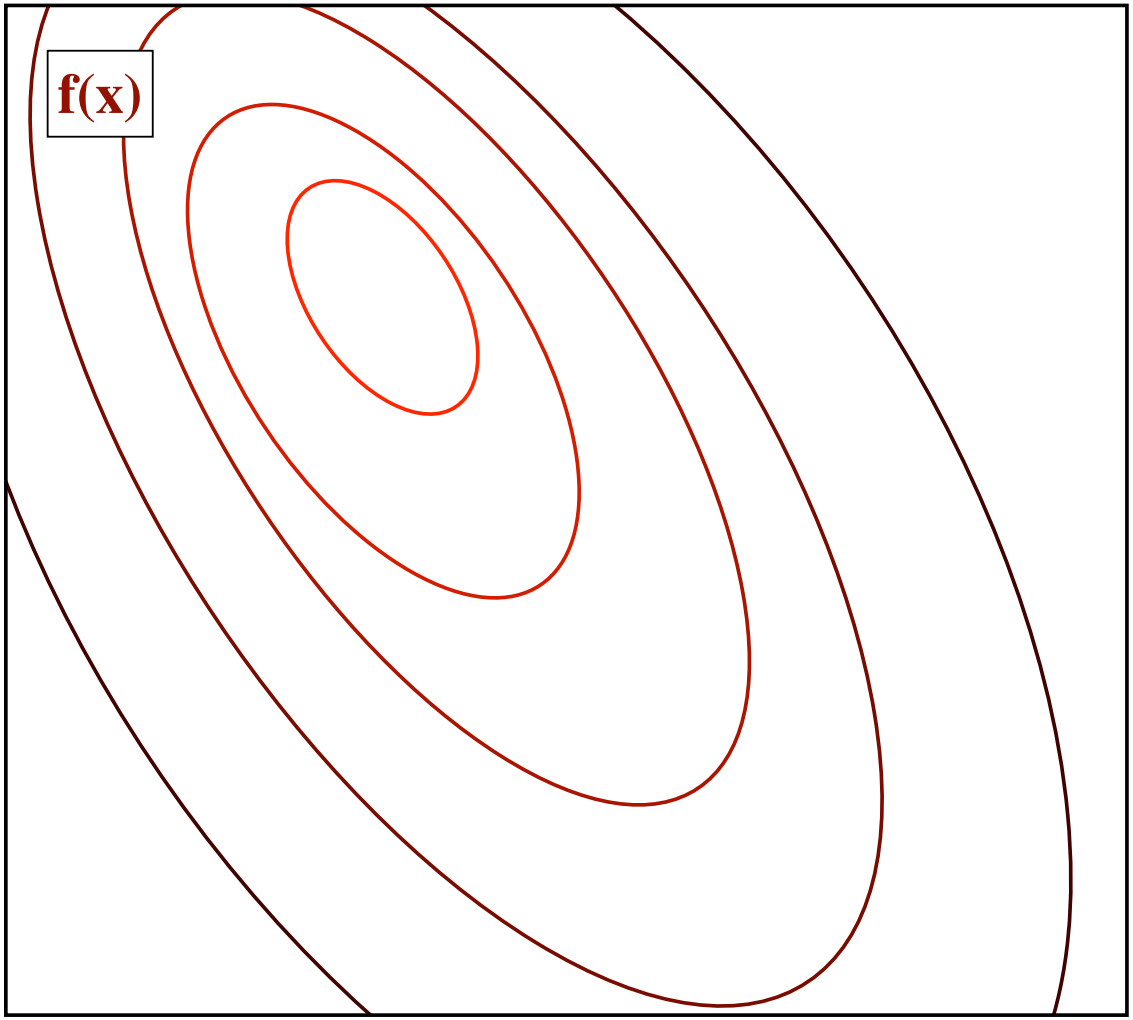
$$\min_{\mathbf{x} \geq 0} \sum_{i=1}^{m} (b_i - \sum_{j=1}^{n} a_{ij} x_j)^2,$$

- Should this be easier than with general constraints?
- The constraints are simple:
  - Given y, we can efficiently find closest x satisfying constraints.
    (just set negative y_i to zero)
- Gradient projection:
  - Alternates between gradient step and projection step:

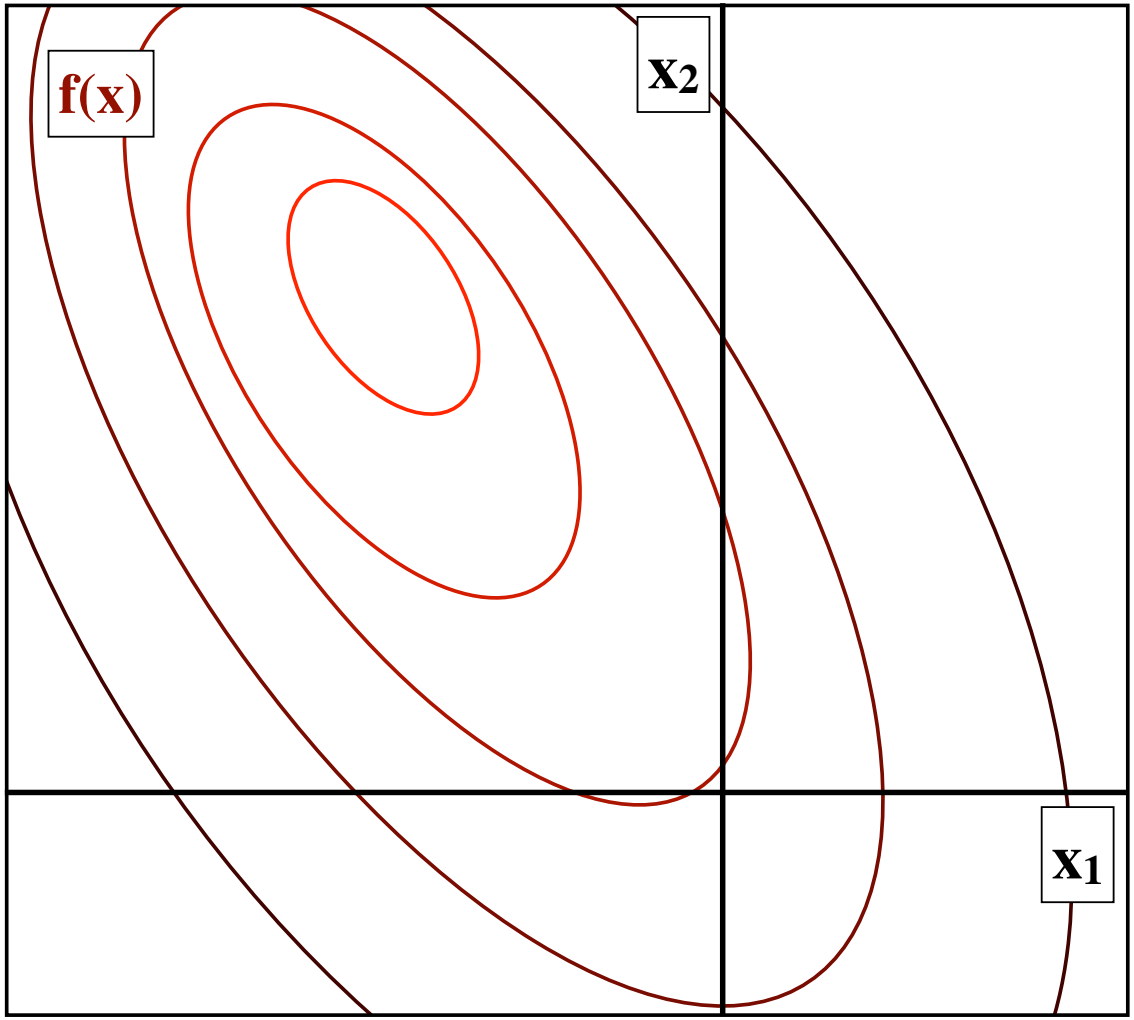
$$\mathbf{x}_{k+1} = \text{project}[\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)],$$

$$\text{project}[\mathbf{y}] = \arg\min_{\mathbf{x}} \frac{1}{2} \|\mathbf{x} - \mathbf{y}\|^2, \quad \text{s.t.} \quad \mathbf{x} \geq 0.$$

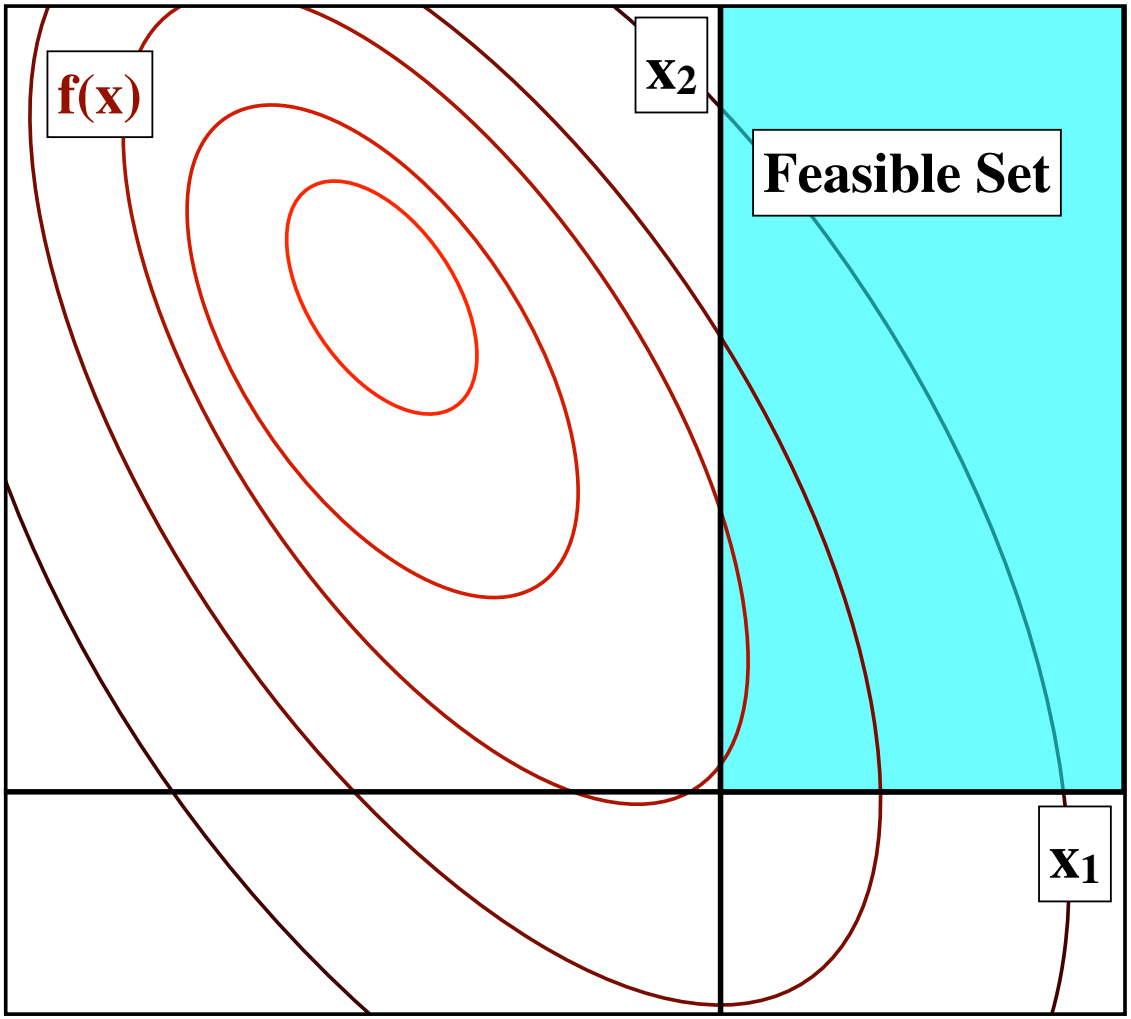
Gradient Projection



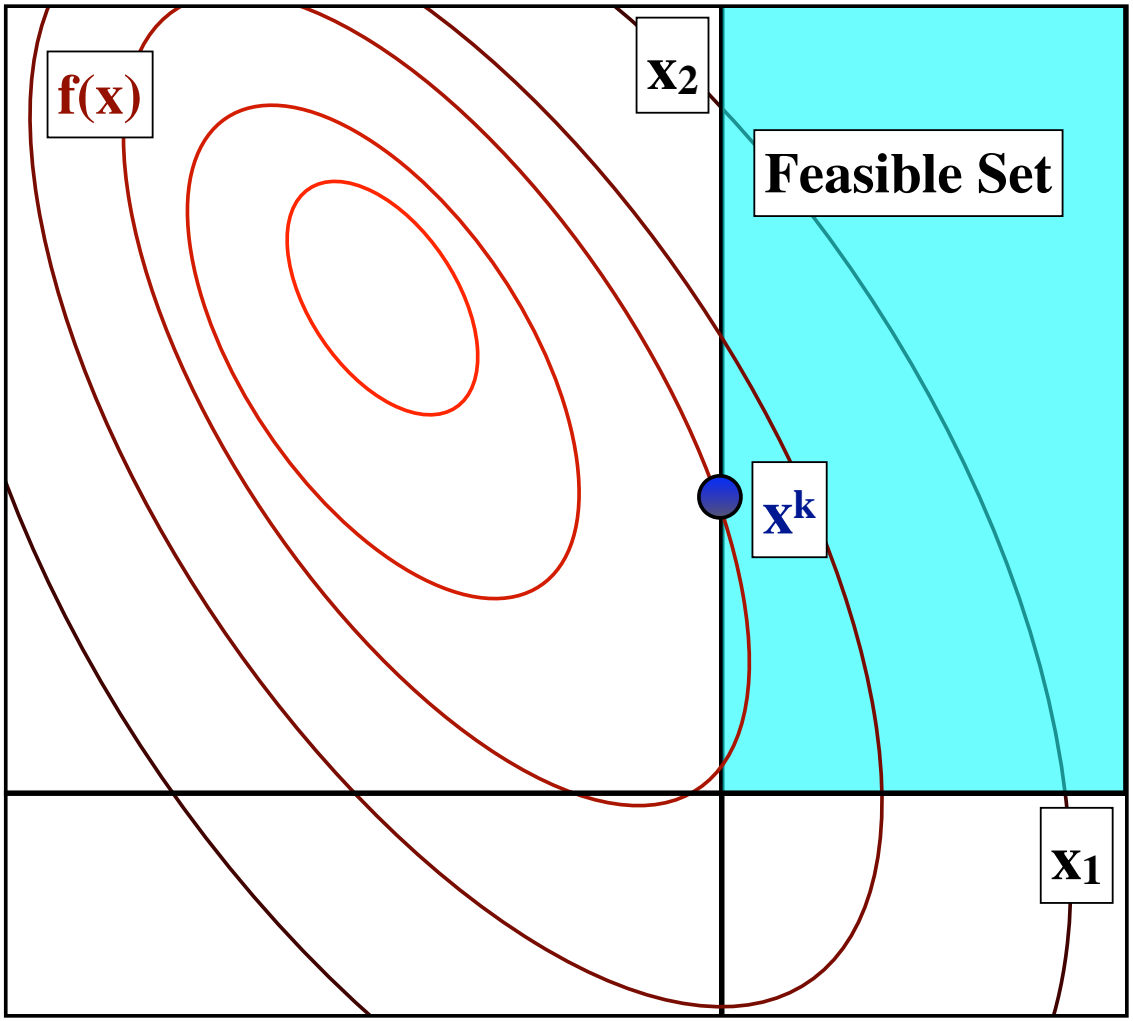
Gradient Projection



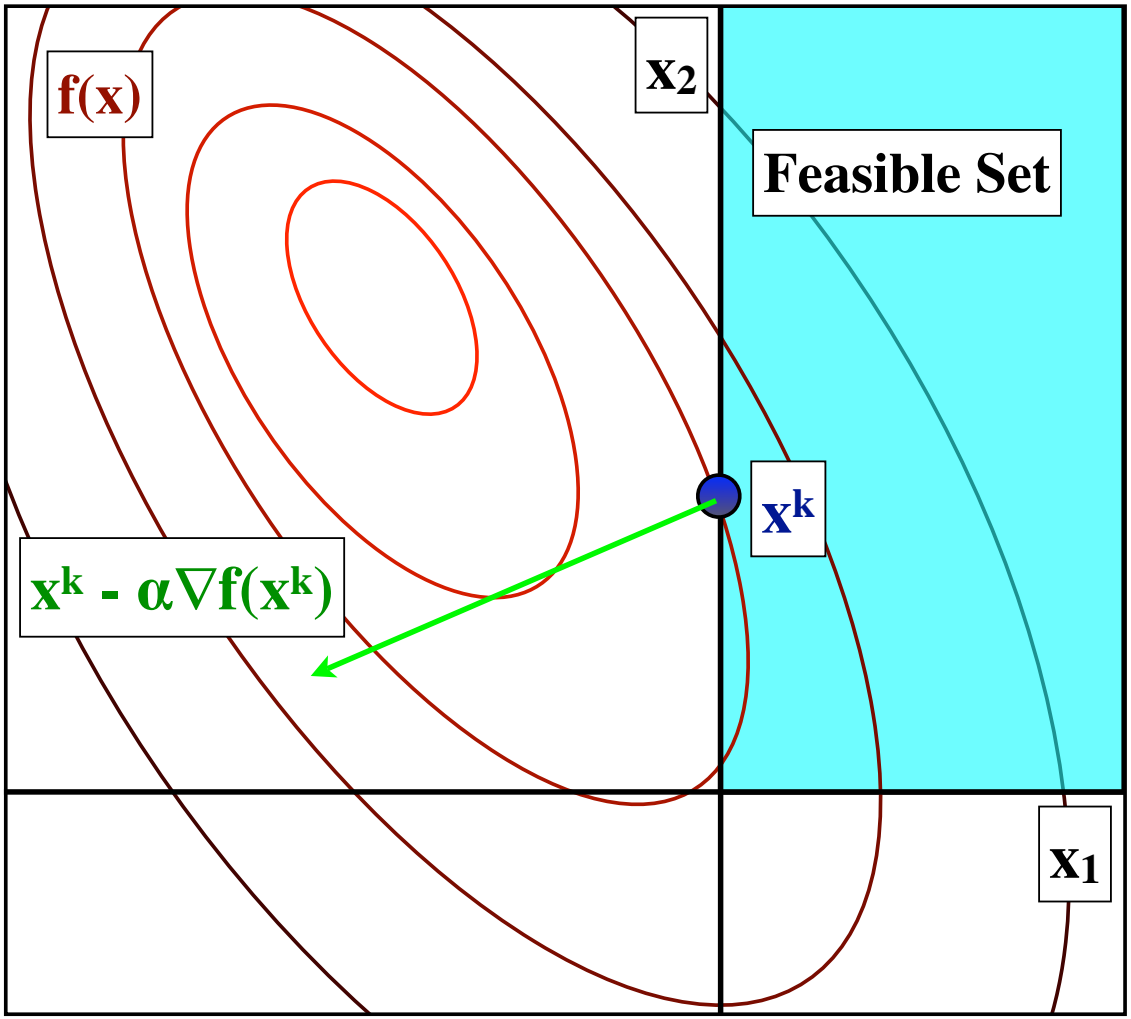
Gradient Projection



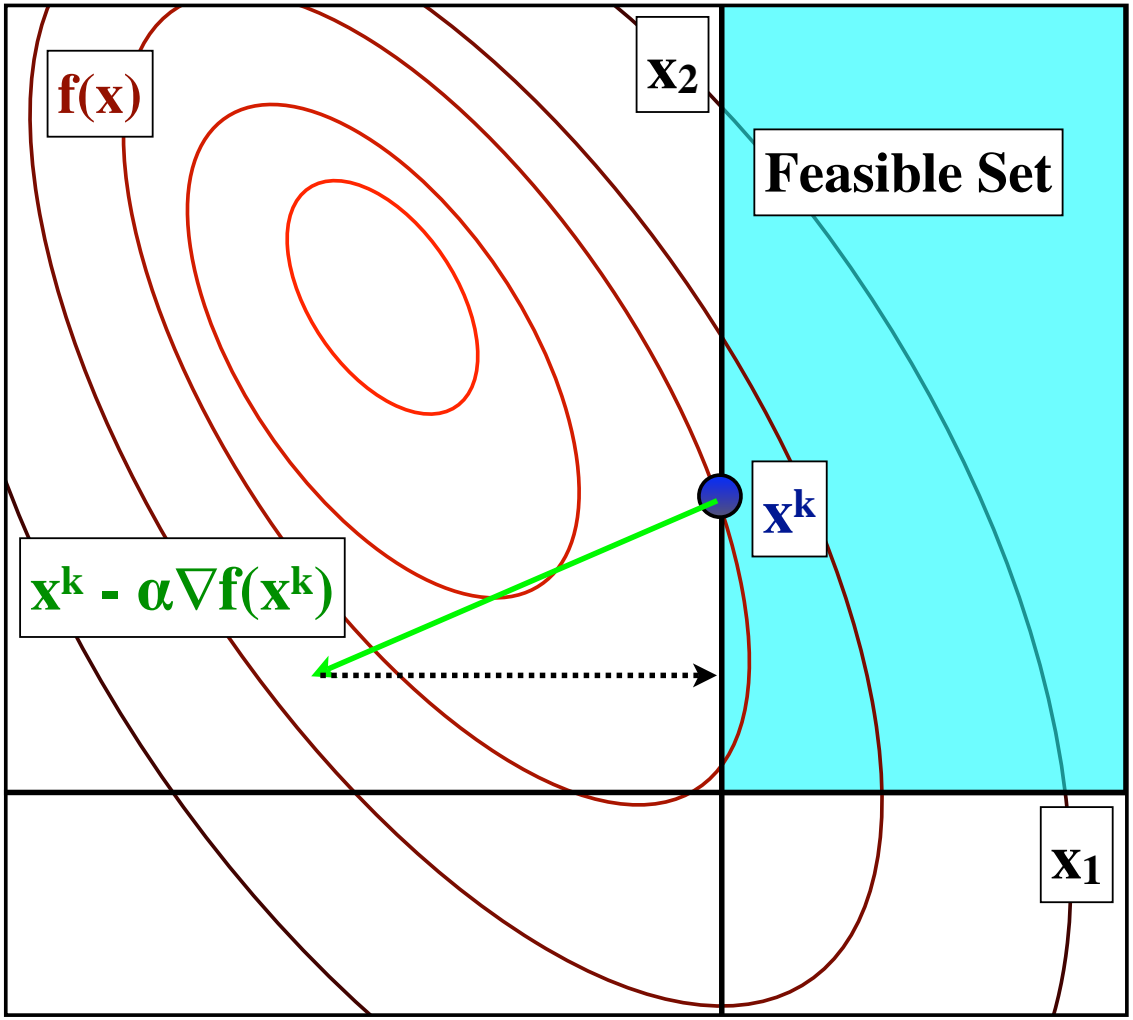
Gradient Projection



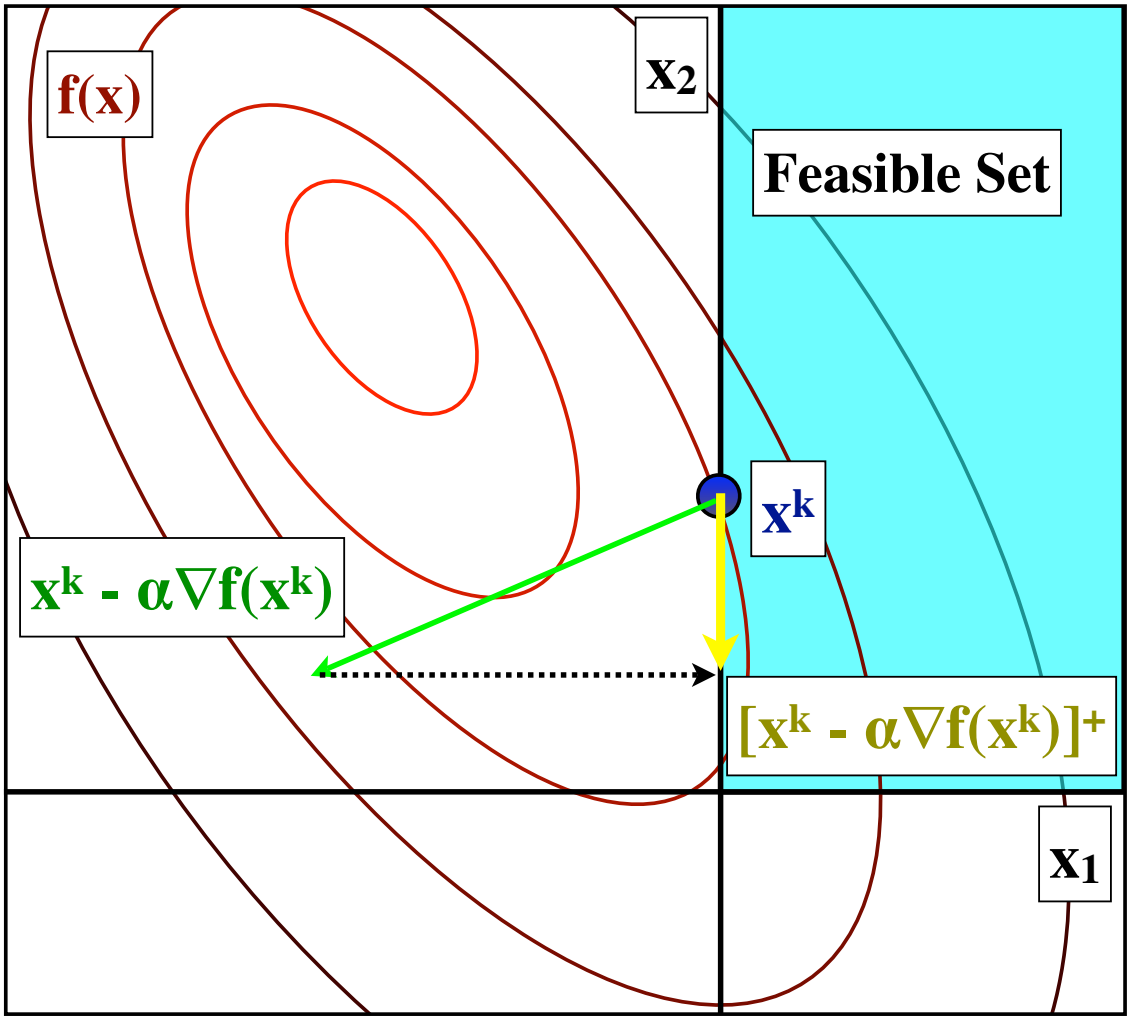
Gradient Projection



Gradient Projection



Gradient Projection



Simple Constraints

- Gradient projection has the same convergence rate as the unconstrained gradient method,

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \leq \left(1 - \frac{1}{\kappa_f}\right)[f(\mathbf{x}_0) - f(\mathbf{x}^*)].$$

- You can do line-search to select the step-size.

Simple Constraints

- Gradient projection has the same convergence rate as the unconstrained gradient method,

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \leq \left(1 - \frac{1}{\kappa_f}\right)[f(\mathbf{x}_0) - f(\mathbf{x}^*)].$$

- You can do line-search to select the step-size.
- Accelerated gradient projection,

$$\mathbf{x}_{k+1} = \text{project}[\mathbf{y}_k - \alpha_k \nabla f(\mathbf{y}_k)],$$
$$\mathbf{y}_{k+1} = \mathbf{x}_k + \beta_k(\mathbf{x}_{k+1} - \mathbf{x}_k),$$

gives a better dependence on the condition number,

$$f(\mathbf{x}_k) - f(\mathbf{x}^*) \leq \left(1 - \frac{1}{\sqrt{\kappa_f}}\right)[f(\mathbf{x}_0) - f(\mathbf{x}^*)].$$

Proximal-Gradient Method

- The proximal-gradient method addresses problem of the form

$$\min_{\mathbf{x}} f(\mathbf{x}) + r(\mathbf{x}),$$

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

Proximal-Gradient Method

- The proximal-gradient method addresses problem of the form

$$\min_{\mathbf{x}} f(\mathbf{x}) + r(\mathbf{x}),$$

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

- Alternates between gradient descent on f and proximity operator of r:

$$\mathbf{x}_{k+\frac{1}{2}} = \mathbf{x}_k - \alpha^k \nabla f(\mathbf{x}^k),$$

$$\mathbf{x}_{k+1} = \arg\min_{\mathbf{y}} \left\{ \frac{1}{2}\|\mathbf{y} - \mathbf{x}_{k+\frac{1}{2}}\|^2 + \alpha_k r(\mathbf{y}) \right\},$$

Proximal-Gradient Method

- The proximal-gradient method addresses problem of the form

$$\min_{\mathbf{x}} f(\mathbf{x}) + r(\mathbf{x}),$$

  where f is smooth but r is a general convex function.
- Alternates between gradient descent on f and proximity operator of r:

$$\mathbf{x}_{k+\frac{1}{2}} = \mathbf{x}_k - \alpha^k \nabla f(\mathbf{x}^k),$$

$$\mathbf{x}_{k+1} = \arg\min_{\mathbf{y}} \left\{ \frac{1}{2}\|\mathbf{y} - \mathbf{x}_{k+\frac{1}{2}}\|^2 + \alpha_k r(\mathbf{y}) \right\},$$

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

Special case of Projected-Gradient Methods

- Projected-gradient methods are a special case:

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

Special case of Projected-Gradient Methods

- Projected-gradient methods are a special case:

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

gives

$$\mathbf{x}_{k+1} = \text{project}_{\mathcal{C}}[\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)],$$

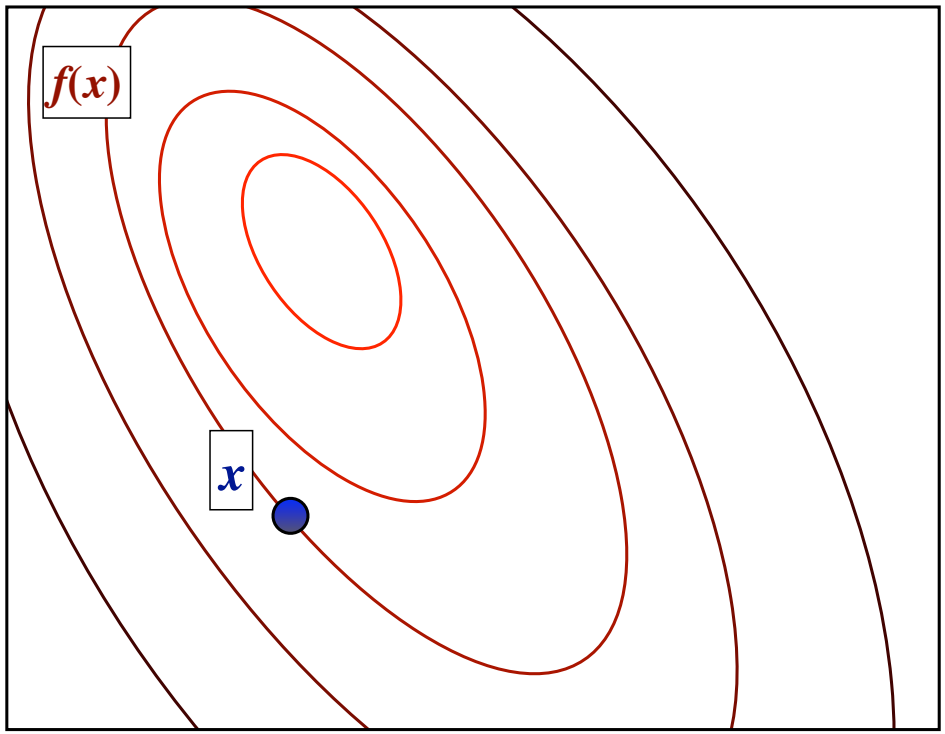
Special case of Projected-Gradient Methods

- Projected-gradient methods are a special case:

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

gives

$$\mathbf{x}_{k+1} = \text{project}_{\mathcal{C}}[\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)],$$



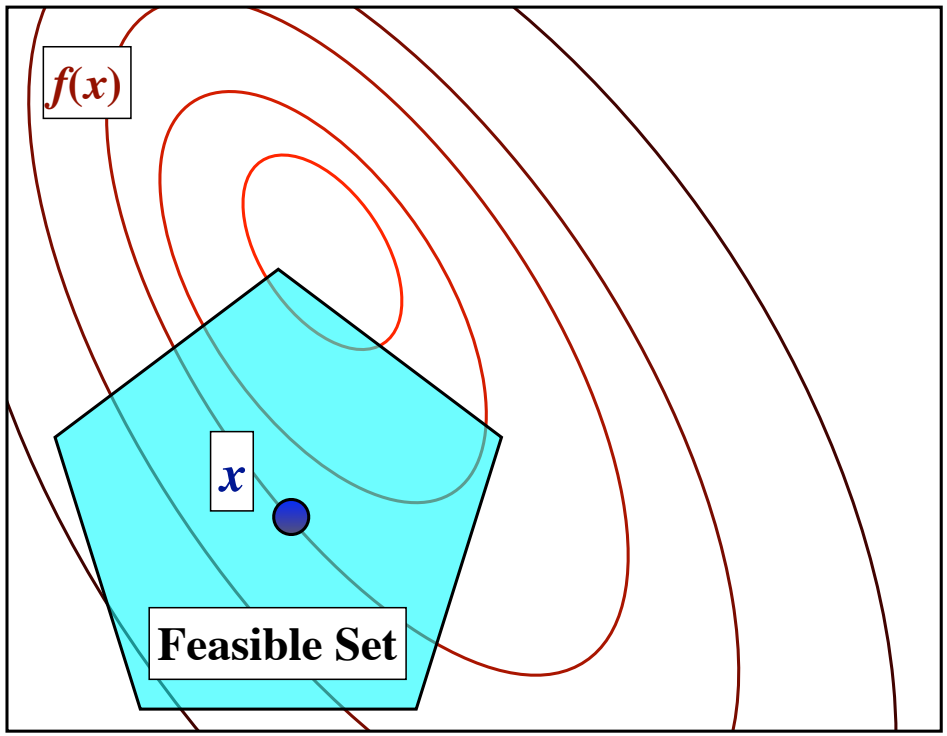
Special case of Projected-Gradient Methods

- Projected-gradient methods are a special case:

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

gives

$$\mathbf{x}_{k+1} = \text{project}_{\mathcal{C}}[\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)],$$



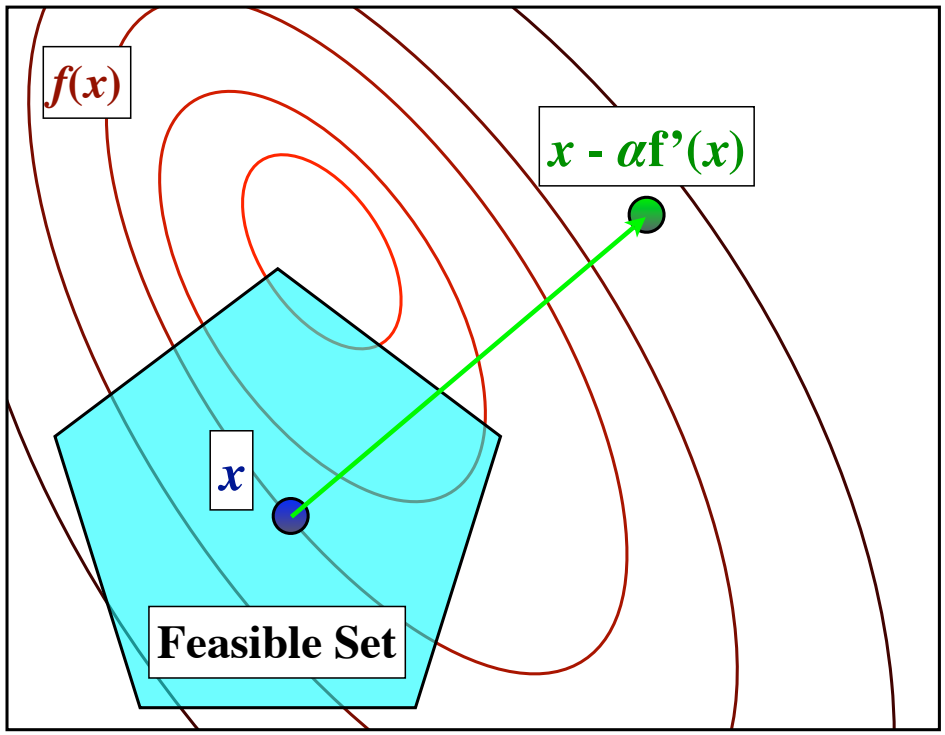
Special case of Projected-Gradient Methods

- Projected-gradient methods are a special case:

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

gives

$$\mathbf{x}_{k+1} = \text{project}_{\mathcal{C}}[\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)],$$



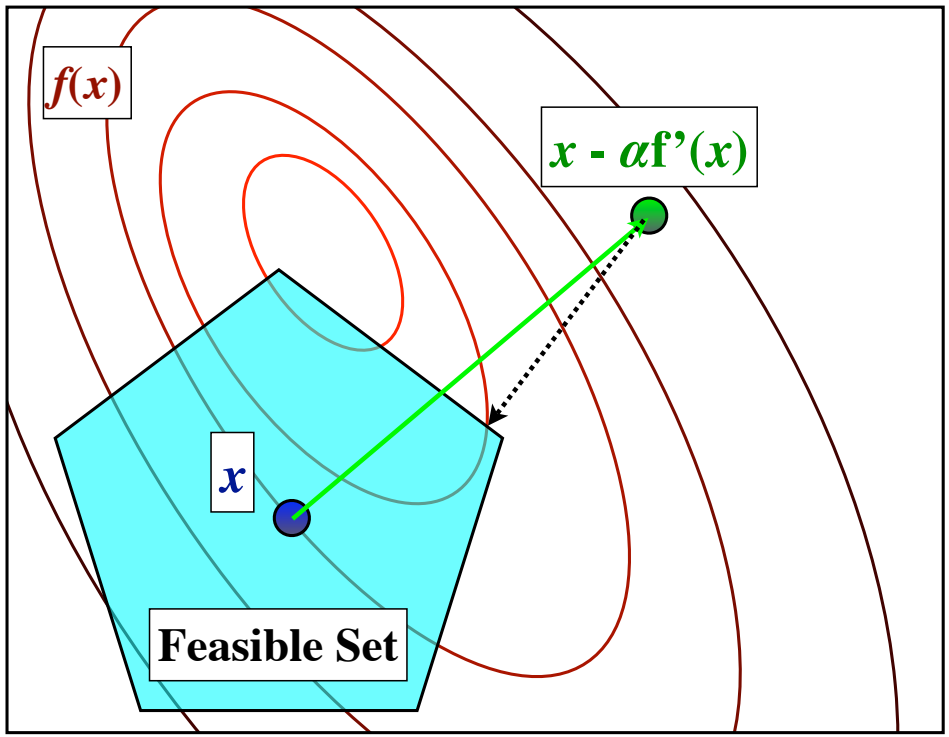
Special case of Projected-Gradient Methods

- Projected-gradient methods are a special case:

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

gives

$$\mathbf{x}_{k+1} = \text{project}_{\mathcal{C}}[\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)],$$



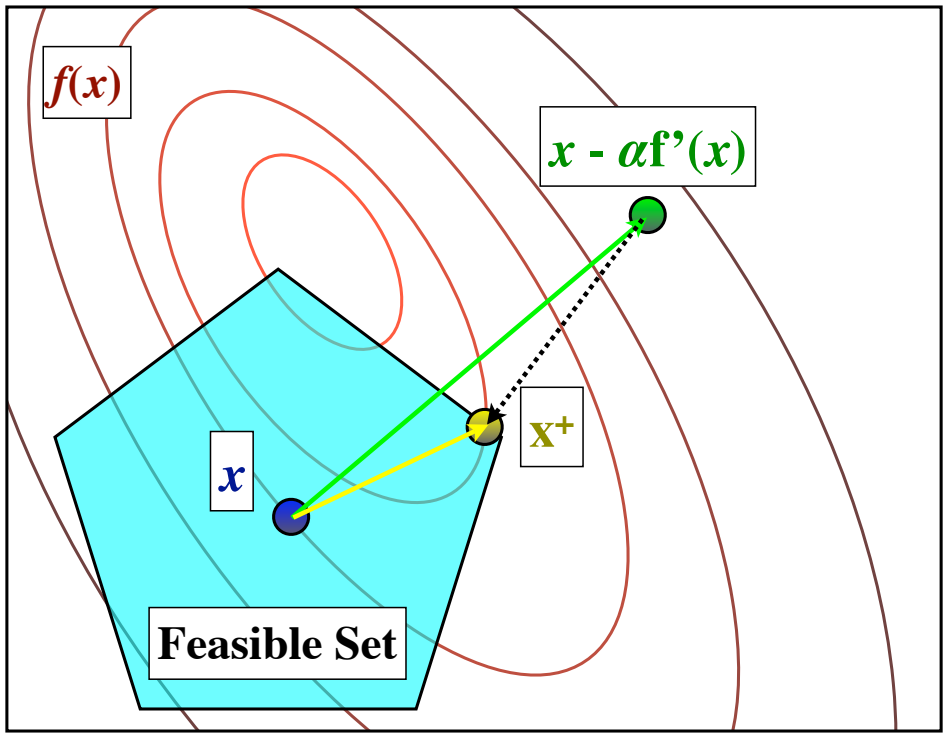
Special case of Projected-Gradient Methods

- Projected-gradient methods are a special case:

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

gives

$$\mathbf{x}_{k+1} = \text{project}_{\mathcal{C}}[\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)],$$



Proximal Operator, Iterative Soft Thresholding

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

$$\mathbf{x}_{k+1} = \text{softThresh}_{\alpha_k \lambda}[\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)].$$

Proximal Operator, Iterative Soft Thresholding

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

$$\mathbf{x}_{k+1} = \text{softThresh}_{\alpha_k \lambda}[\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)].$$

- 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

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

$$\mathbf{x}_{k+1} = \text{softThresh}_{\alpha_k \lambda}[\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)].$$

- 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

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

$$\mathbf{x}_{k+1} = \text{softThresh}_{\alpha_k \lambda}[\mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)].$$

- Example with $\lambda = 1$:

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

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:

Exact Proximal-Gradient Methods

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

Exact Proximal-Gradient Methods

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

Exact Proximal-Gradient Methods

- For what problems can we apply these methods?
- We can efficiently compute the proximity operator for:
  1. Lower and upper bounds.
  2. Small number of linear constraint.
  3. Probability constraints.
  4. L1-Regularization.
  5. Group ℓ_1-Regularization.

Exact Proximal-Gradient Methods

- For what problems can we apply these methods?
- We can efficiently compute the proximity operator for:
  1. Lower and upper bounds.
  2. Small number of linear constraint.
  3. Probability constraints.
  4. L1-Regularization.
  5. Group ℓ_1-Regularization.
  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. Lower and upper bounds.
  2. Small number of linear constraint.
  3. Probability constraints.
  4. L1-Regularization.
  5. Group ℓ_1-Regularization.
  6. A few other simple regularizers/constraints.
- Can solve huge instances of these constrained/non-smooth problem.

Outline

Big-N Problems

- Consider the problem of minimizing a finite sum,

$$\min_{\mathbf{x}} \frac{1}{m} \sum_{i=1}^{m} f_i(\mathbf{x}),$$

  where m is very large.
- This could be a big least squares problem, or another ML model.

Big-N Problems

- Consider the problem of minimizing a finite sum,

$$\min_{\mathbf{x}} \frac{1}{m} \sum_{i=1}^{m} f_i(\mathbf{x}),$$

  where m is very large.
- This could be a big least squares problem, or another ML model.
- Examples:
  - Each i is a Facebook user.
  - Each i is a product on Amazon.
  - Each i is a webpage on the internet.

Big-N Problems

- Consider the problem of minimizing a finite sum,

$$\min_{\mathbf{x}} \frac{1}{m} \sum_{i=1}^{m} f_i(\mathbf{x}),$$

  where m is very large.
- This could be a big least squares problem, or another ML model.
- Examples:
  - Each i is a Facebook user.
  - Each i is a product on Amazon.
  - Each i is a webpage on the internet.
- We can't afford to go through all m examples many times.
- One way to deal with this restriction is stochastic gradient methods.

Stochastic Gradient Methods

- Stochastic gradient methods consider minimizing expectations,

$$\min_{\mathbf{x}} \mathbb{E}[f(\mathbf{x})].$$

- They assume we can generate a random vector $\mathbf{p}_k$ whose expectation is the gradient

$$\mathbb{E}[\mathbf{p}_k] = \nabla f(\mathbf{x}^k),$$

and take a gradient step using this direction,

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{p}_k.$$

Stochastic Gradient Methods

- Stochastic gradient methods consider minimizing expectations,

$$\min_{\mathbf{x}} \mathbb{E}[f(\mathbf{x})].$$

- They assume we can generate a random vector $\mathbf{p}_k$ whose expectation is the gradient

$$\mathbb{E}[\mathbf{p}_k] = \nabla f(\mathbf{x}^k),$$

and take a gradient step using this direction,

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{p}_k.$$

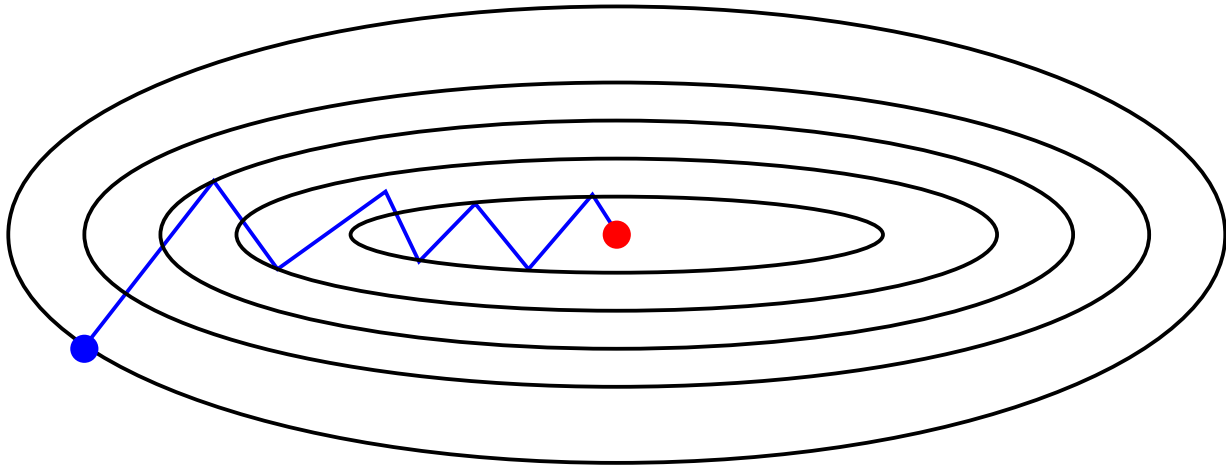
- For convergence, usually require the step-sizes α_k to converge to 0.
  - E.g., Robbins-Munro conditions,

$$\sum_{k=1}^{\infty} \alpha_k = \infty, \quad \sum_{k=1}^{\infty} \alpha_k^2 < \infty,$$

suggests using $\alpha_k = \gamma/k$ for some constant γ.

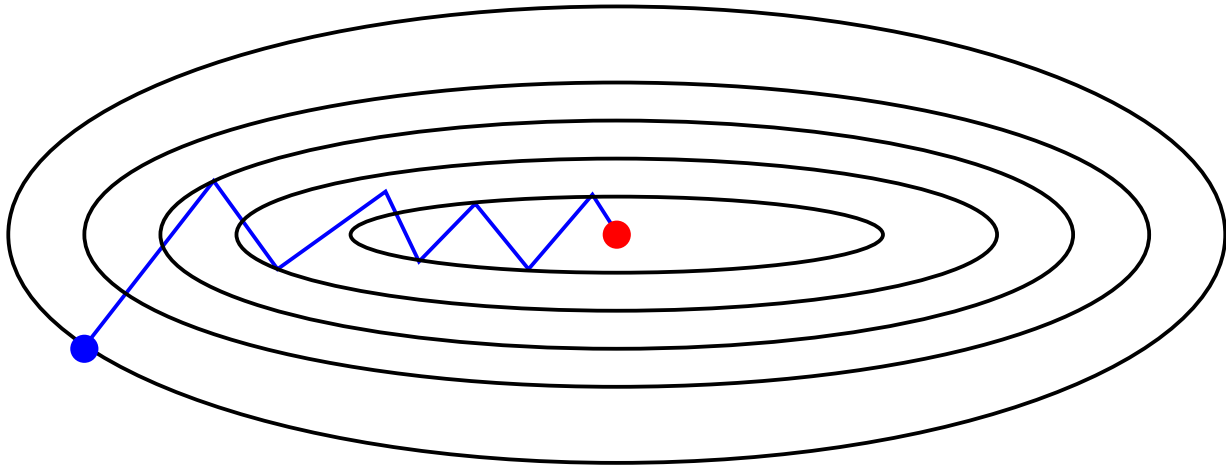
Gradient Method vs. Stochastic Gradient Method

Gradient method:

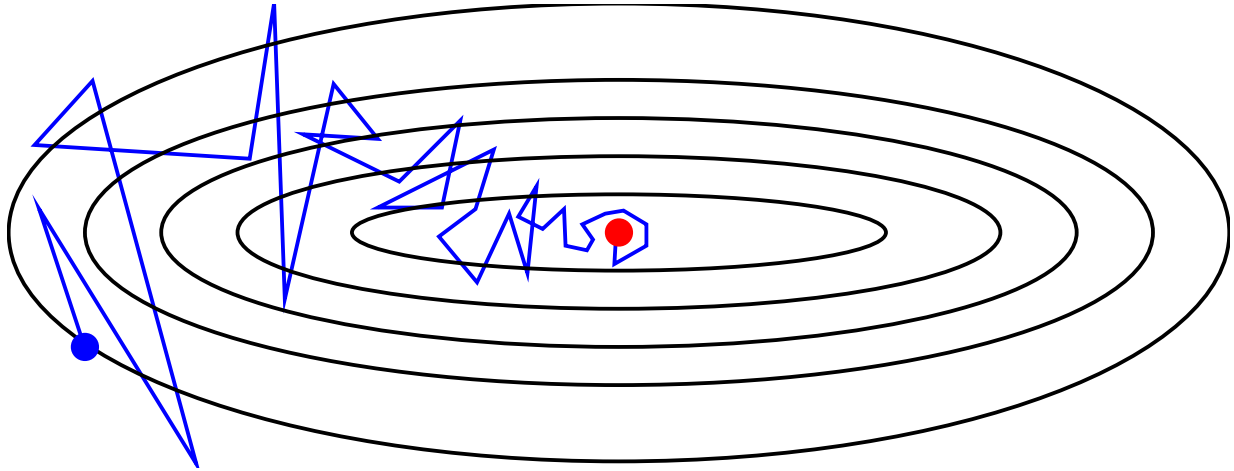


Gradient Method vs. Stochastic Gradient Method

Gradient method:



Stochastic gradient method:



Application to Finite Sums

- Returning to the problem of minimizing a finite sum,

$$\min_{\mathbf{x}} \frac{1}{m} \sum_{i=1}^{m} f_i(\mathbf{x}).$$

- Set $\mathbf{p}_k$ to the gradient of a random function f_{i_k},

Application to Finite Sums

- Returning to the problem of minimizing a finite sum,

$$\min_{\mathbf{x}} \frac{1}{m} \sum_{i=1}^{m} f_i(\mathbf{x}).$$

- Set $\mathbf{p}_k$ to the gradient of a random function f_{i_k},

$$\begin{aligned}\mathbb{E}[\mathbf{p}_k] &= \mathbb{E}_i[\nabla f_i(\mathbf{x}_k)] \\ &= \sum_{i=1}^{m} p(i) \nabla f_i(\mathbf{x}_k) \\ &= \frac{1}{m} \sum_{i=1}^{m} \nabla f_i(\mathbf{x}_k).\end{aligned}$$

Application to Finite Sums

- Returning to the problem of minimizing a finite sum,

$$\min_{\mathbf{x}} \frac{1}{m}\sum_{i=1}^{m} f_i(\mathbf{x}).$$

- Set $\mathbf{p}_k$ to the gradient of a random function f_{i_k},

$$\begin{aligned}\mathbb{E}[\mathbf{p}_k] &= \mathbb{E}_i[\nabla f_i(\mathbf{x}_k)] \\ &= \sum_{i=1}^{m} p(i)\nabla f_i(\mathbf{x}_k) \\ &= \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(\mathbf{x}_k).\end{aligned}$$

- This gives us the stochastic gradient algorithm

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f_{i_k}(\mathbf{x}_k).$$

- The iteration cost is independent of m.

Convergence Rate of Stochastic Gradient

- Stochastic gradient has much faster iterations.
- But how many iterations are required?

Convergence Rate of Stochastic Gradient

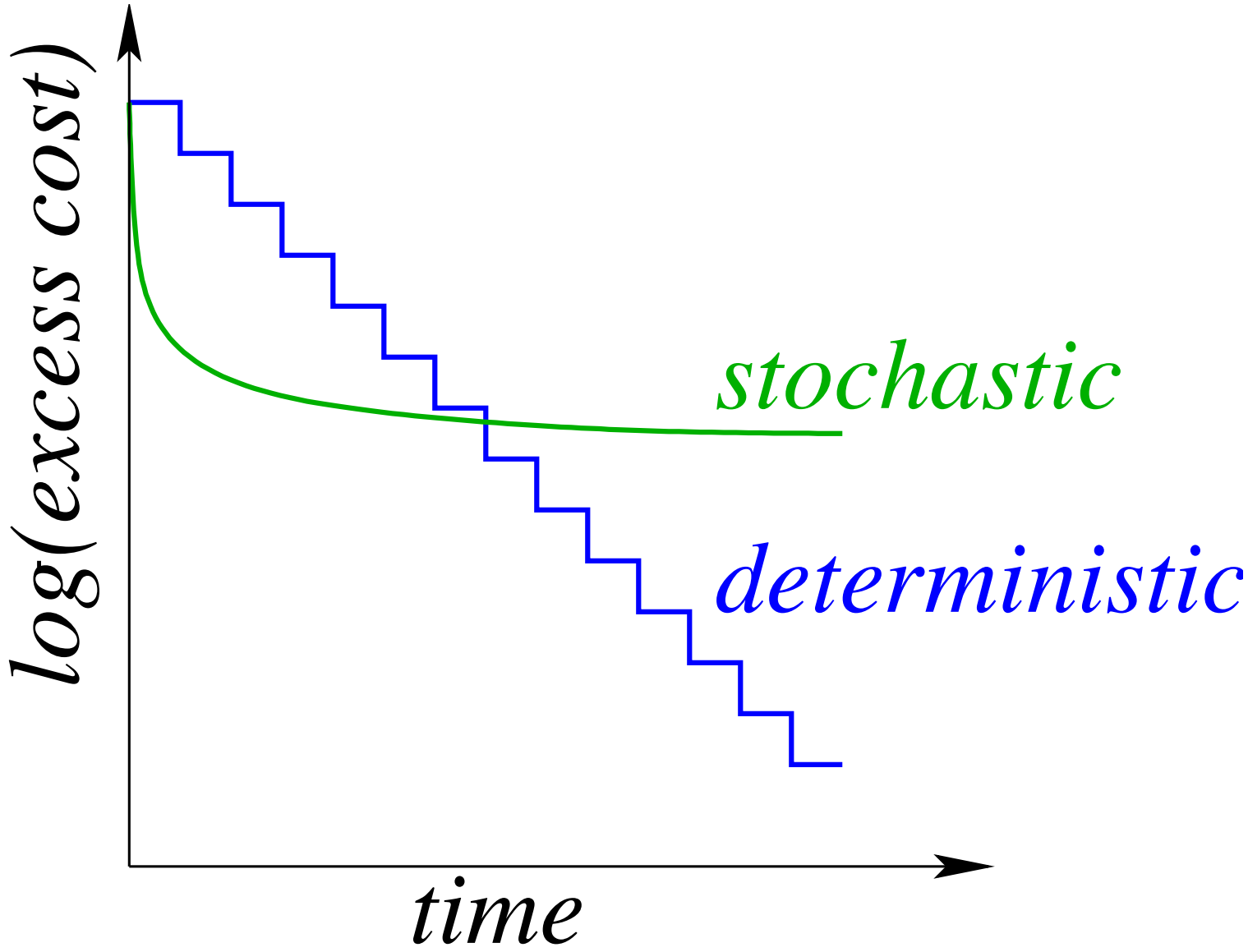
- Stochastic gradient has much faster iterations.
- But how many iterations are required?
- If we set $\alpha_k = 1/k\lambda_n$, we have that

$$\mathbb{E}[f(x^k) - f(x^*)] = O(1/k).$$

- This is a sublinear rate.
- Often works badly in practice:
  - Initial α_k might be huge.
  - Later α_k might be tiny.
- Nesterov/Newton-like variations can only improve the constant.

(even in low dimensions)

Comparison of Gradient and Stochastic Gradient



Improving Stochastic Gradient

- How can we improve this algorithm?

Improving Stochastic Gradient

- How can we improve this algorithm?
  1. Averaging: If you use a bigger step-size, $\alpha_k = \gamma/\sqrt{k}$, then the average of the iterations $(\frac{1}{k}\sum_{i=1}^{k}\mathbf{x}_i)$ has nearly-optimal constants.

Improving Stochastic Gradient

- How can we improve this algorithm?
  1. Averaging: If you use a bigger step-size, $\alpha_k = \gamma/\sqrt{k}$, then the average of the iterations ($\frac{1}{k}\sum_{i=1}^{k} \mathbf{x}_i$) has nearly-optimal constants.
  2. Constant step-sizes: If you use a constant step-size $\alpha_k = \alpha$, you can show

$$\mathbb{E}[f(x^k) - f(x^*)] = (1 - 2\alpha\lambda_n)^k[f(x^0) - f(x^*)] + O(\alpha),$$

  which shows rapid progress but non-convergence.

Improving Stochastic Gradient

- How can we improve this algorithm?
  1. Averaging: If you use a bigger step-size, $\alpha_k = \gamma/\sqrt{k}$, then the average of the iterations $(\frac{1}{k}\sum_{i=1}^{k}\mathbf{x}_i)$ has nearly-optimal constants.
  2. Constant step-sizes: If you use a constant step-size $\alpha_k = \alpha$, you can show

$$\mathbb{E}[f(x^k) - f(x^*)] = (1 - 2\alpha\lambda_n)^k[f(x^0) - f(x^*)] + O(\alpha),$$

     which shows rapid progress but non-convergence.
  3. Use special problem structures: For certain problems, you can show faster rates.

Improving Stochastic Gradient

- How can we improve this algorithm?
  1. Averaging: If you use a bigger step-size, $\alpha_k = \gamma/\sqrt{k}$, then the average of the iterations $(\frac{1}{k}\sum_{i=1}^{k}\mathbf{x}_i)$ has nearly-optimal constants.
  2. Constant step-sizes: If you use a constant step-size $\alpha_k = \alpha$, you can show

$$\mathbb{E}[f(x^k) - f(x^*)] = (1 - 2\alpha\lambda_n)^k[f(x^0) - f(x^*)] + O(\alpha),$$

     which shows rapid progress but non-convergence.
  3. Use special problem structures: For certain problems, you can show faster rates.
- Since 2012, large focus on better algorithms for finite sum structure.

Stochastic Average Gradient

- The stochastic average gradient (SAG) algorithm uses,

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \frac{\alpha_k}{m} \sum_{i=1}^{m} \mathbf{y}_k^i,$$

and evaluates a random $\nabla f_i(x^k)$, with $\mathbf{y}_k^i$ the last evaluation of ∇f_i.

Stochastic Average Gradient

- The stochastic average gradient (SAG) algorithm uses,

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \frac{\alpha_k}{m} \sum_{i=1}^{m} \mathbf{y}_k^i,$$

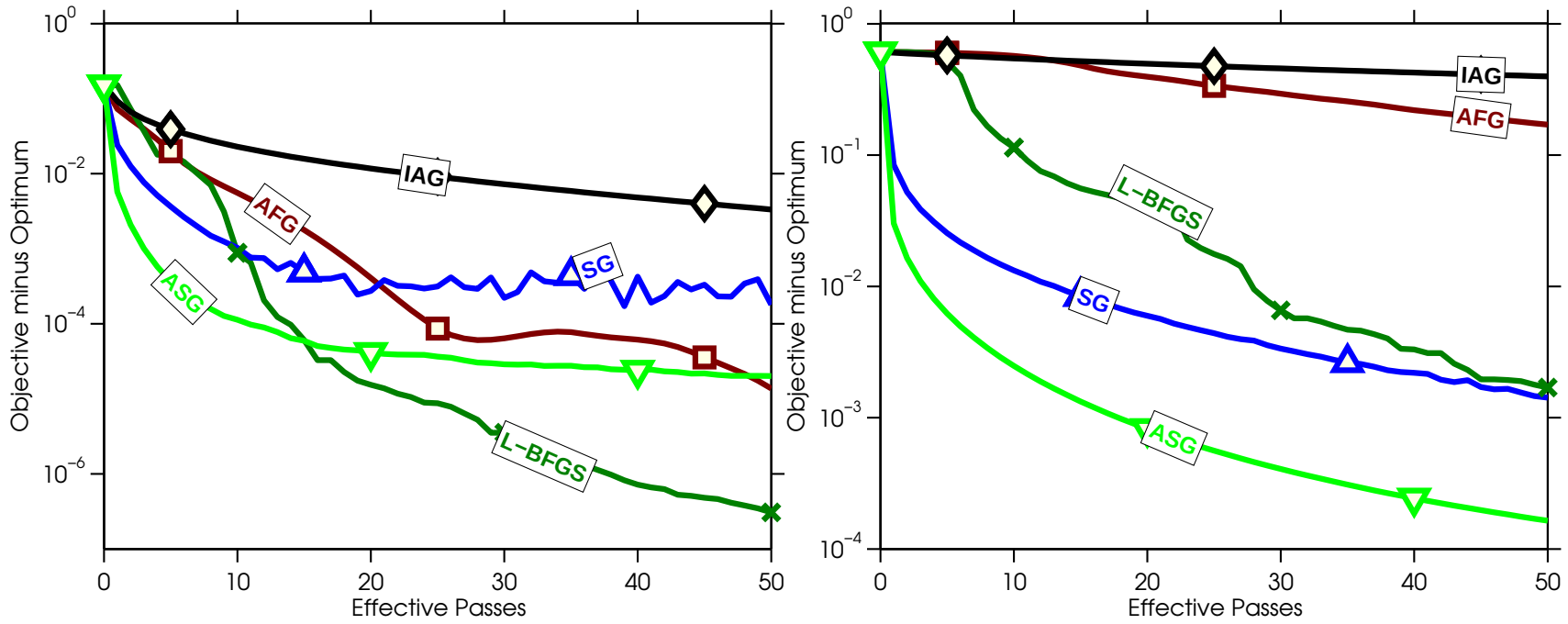
  and evaluates a random $\nabla f_i(x^k)$, with $\mathbf{y}_k^i$ the last evaluation of ∇f_i.
- With $\alpha_k = 1/16\Lambda_1$, the SAG algorithm has linear rate,

$$\mathbb{E}[f(\mathbf{x}_k)] - f(\mathbf{x}^*) \leq \left(1 - \min\left\{\frac{1}{8m}, \frac{\lambda_n}{16\Lambda_1}\right\}\right)^k [f(\mathbf{x}_0) - f(\mathbf{x}^*)],$$

  where Λ_1 bounds eigenvalues of *each* $\nabla^2 f_i(\mathbf{x})$.
- Iteration cost independent of m, rate similar to gradient method.

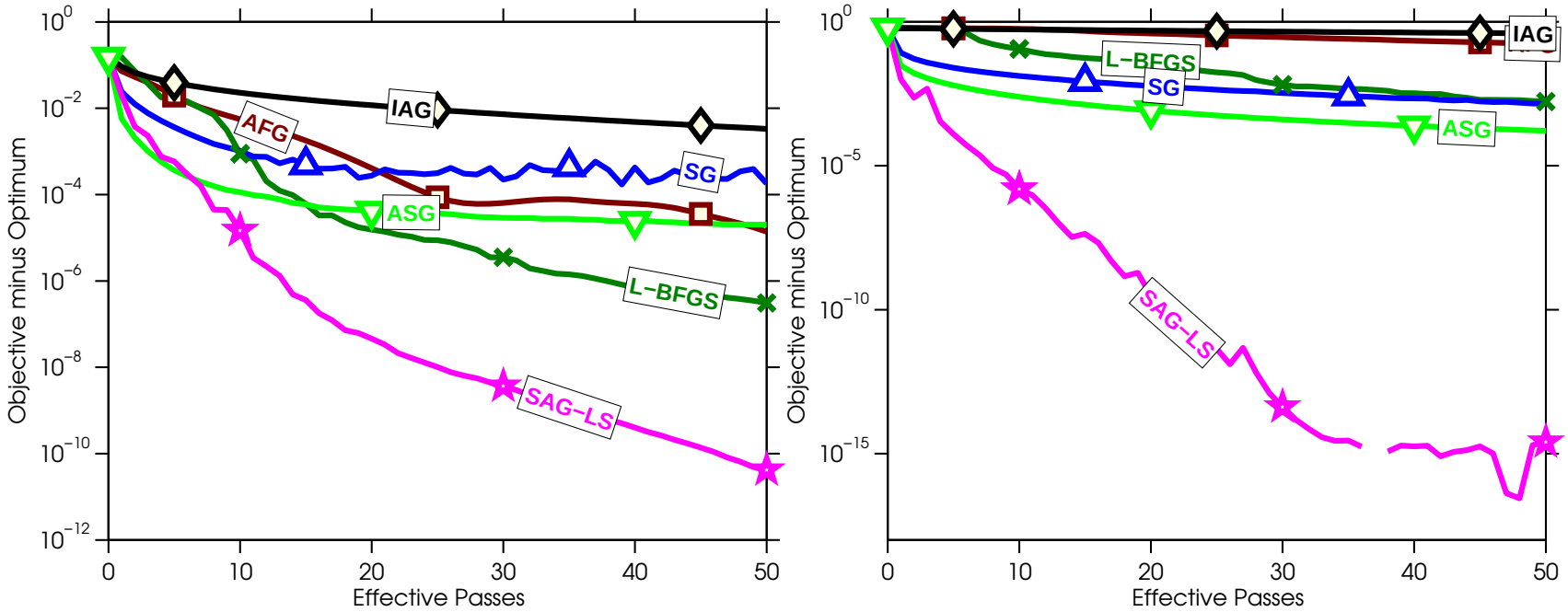
Comparing FG and SG Methods

- 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 ($n = 697641$, $p = 47236$)



Summary

- Part 1: Numerical optimization is at the core of many modern machine learning applications.
- Part 2: Gradient-based methods allow elegant scaling with dimensionality for smooth problems.
- Part 3: Proximal-gradient methods allow the same scaling for many non-smooth problems.
- Part 4: Stochastic gradient methods allow scaling to a huge number of data samples.