NLP 202: Optimization

Jeffrey Flanigan

Winter 2023

University of California Santa Cruz
jmflanig@ucsc.edu

Optimization

Structured Prediction Problem

1. Define output space $\mathcal{Y}(x)$
2. Pick a scoring function $score_\theta(x, y)$ with local parts and a decoding algorithm which exactly or approximately finds the argmax

$$\hat{y} = \operatorname*{argmax}_{y \in \mathcal{Y}(x)} score_\theta(x, y)$$

3. Pick a loss function for training

$$\hat{\theta} = \operatorname*{argmin}_{\theta} L_\theta(\mathcal{D})$$

4. **Pick an optimizer to minimize the loss**

Loss functions

$$\hat{\theta} = \operatorname*{argmin}_{\theta} \sum_{i=1}^{N} L(x_i, y_i, \theta)$$

$L(x_i, y_i, \theta) =$

Perceptron loss: $-\operatorname{score}_\theta(x_i, y_i) + \max_{y' \in \mathcal{Y}} \operatorname{score}_\theta(x_i, y')$

SVM loss: $-\operatorname{score}_\theta(x_i, y_i) + \max_{y' \in \mathcal{Y}} \operatorname{score}_\theta(x_i, y') + \operatorname{cost}(y_i, y')$

CRF loss: $-\operatorname{score}_\theta(x_i, y_i) + \log \sum_{y' \in \mathcal{Y}} e^{\operatorname{score}_\theta(x_i, y')}$

Softmax margin: $-\operatorname{score}_\theta(x_i, y_i) + \log \sum_{y' \in \mathcal{Y}} e^{\operatorname{score}_\theta(x_i, y') + \operatorname{cost}(y_i, y')}$

Kinds of Optimization Problems

- General (non-convex) optimization $\min_x f(x)$
- Constrained optimization:

$$\begin{aligned} \min_{x} \quad & f(x) \\ & s.t.\ x \in S \end{aligned}$$

  $s.t.$ means "subject to" and S is a set
- Convex optimization: $f(x)$ is convex, S is convex
- Linear programming $f(x) = \mathbf{c}^{\mathrm{T}}\mathbf{x}$:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ & s.t. \quad && \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & && \mathbf{x} \geq 0 \end{aligned}$$

Types of information

- Only function evaluation (black-box optimization)
  Used to find hyperparameters
- Gradient (or subgradient) at any point
  Used when learning parameters
- 2nd-order gradients (full Hessian) at any point
  Much, much faster, but too expensive to use.
  If n parameters, need to compute $O(n^2)$ values for Hessian.
  Usually need to invert Hessian $\Rightarrow O(n^3)$

Methods

- Only function evaluation (black-box optimization)
  - Pure random search, grid search, Nelder-Mead, Powell's method
  - Evolutionary strategies (CMA-ES)
  - Bayesian optimization
- Gradient (or subgradient) at any point
  - Gradient method, subgradient method
  - Conjugate gradient method
  - Quasi-Newton method, L-BFGS
- 2nd-order gradients (full Hessian) at any point
  - Newton's method

Stochastic Optimization

Also stochastic versions

- Stochastic zeroth-order: Get random variable $g_0(x)$, where $E[g_0(x)] = f(x)$
- Stochastic first-order: Also get random variable $g_1(x)$, where $E[g_1(x)] = \nabla f(x)$
- Stochastic second-order: Also get random variable $g_2(x)$, where $E[g_2(x)] = \nabla^2 f(x)$

Parameter Learning

For parameter learning, we usually use **stochastic first-order methods**:
stochastic gradient descent (SGD), Adagrad, RMSProp, Adam, Nadam, etc

Online Learning as Stochastic Optimization

$$\hat{\theta} = \operatorname*{argmin}_{\theta} L(\theta) = \operatorname*{argmin}_{\theta} \sum_{i=1}^{N} \frac{1}{N} L_i(\theta)$$

$L_i(\theta)$ is shorthand for $L(x_i, y_i, \theta)$, loss for datapoint (x_i, y_i)

True gradient is:

$$\nabla L(\theta) = \sum_{i=1}^{N} \frac{1}{N} \nabla L_i(\theta)$$

At each timestep, we get a R.V. $g(\theta) = \nabla L_j(\theta)$

for some j chosen randomly $j \in \{1 \dots N\}$

$$E[g(\theta)] = E[\nabla L_j(\theta)] = \sum_{i=1}^{N} \frac{1}{N} \nabla L_i(\theta) = \nabla L(\theta)$$

Today: Non-stochastic

Today: classic unconstrained optimization.

Next time: modern stochastic optimizers used in deep learning.

Insights learned today will apply to stochastic optimizers

Convex Sets

A set C is **convex** if the line segment between any two points of C lies in C, i.e., if for any $\mathbf{x}, \mathbf{y} \in C$ and any λ with $0 \leq \lambda \leq 1$, we have

$$\lambda \mathbf{x} + (1 - \lambda)\mathbf{y} \in C.$$

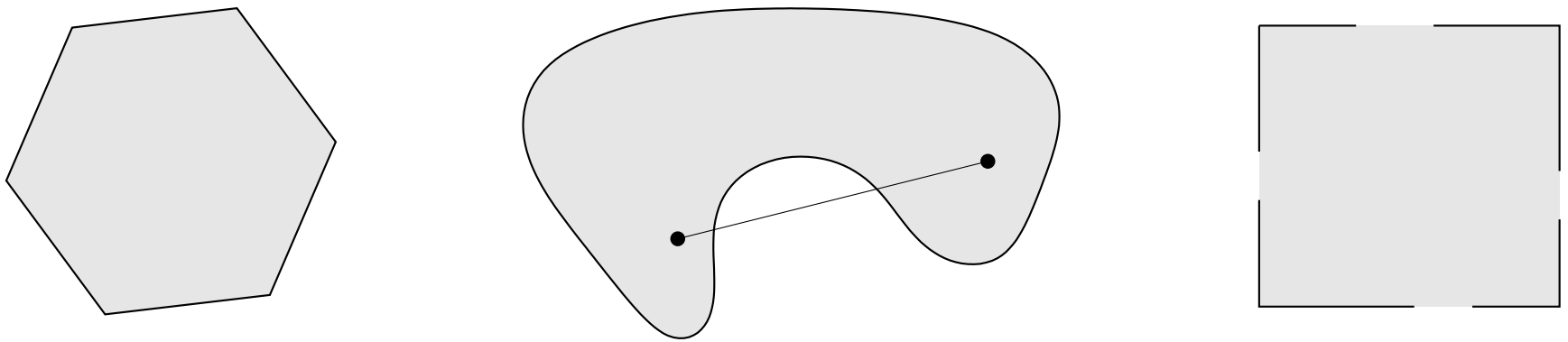


*Figure 2.2 from S. Boyd, L. Vandenberghe

Left Convex.

Middle Not convex, since line segment not in set.

Right Not convex, since some, but not all boundary points are contained in the set.

Convex Functions

Definition

A function $f : \mathbb{R}^d \to \mathbb{R}$ is **convex** if (i) $\mathbf{dom}(f)$ is a convex set and (ii) for all $\mathbf{x}, \mathbf{y} \in \mathbf{dom}(f)$, and λ with $0 \leq \lambda \leq 1$, we have

$$f(\lambda \mathbf{x} + (1 - \lambda)\mathbf{y}) \leq \lambda f(\mathbf{x}) + (1 - \lambda) f(\mathbf{y}).$$

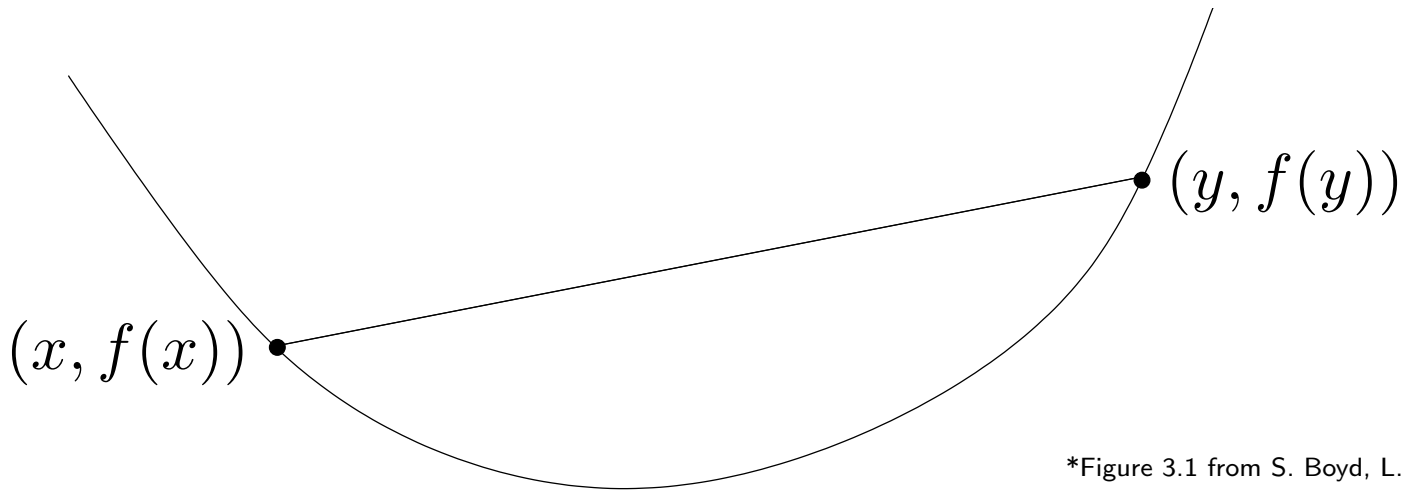


*Figure 3.1 from S. Boyd, L. Vandenberghe

Geometrically: The line segment between $(\mathbf{x}, f(\mathbf{x}))$ and $(\mathbf{y}, f(\mathbf{y}))$ lies above the graph of f.

Motivation: Convex Optimization

Convex Optimization Problems are of the form

$$\min \; f(\mathbf{x}) \qquad \text{s.t.} \qquad \mathbf{x} \in C$$

where both

- f is a convex function
- $X \subseteq \mathbf{dom}(f)$ is a convex set (note: $\mathbb{R}^d$ is convex)

Crucial Property of Convex Optimization Problems

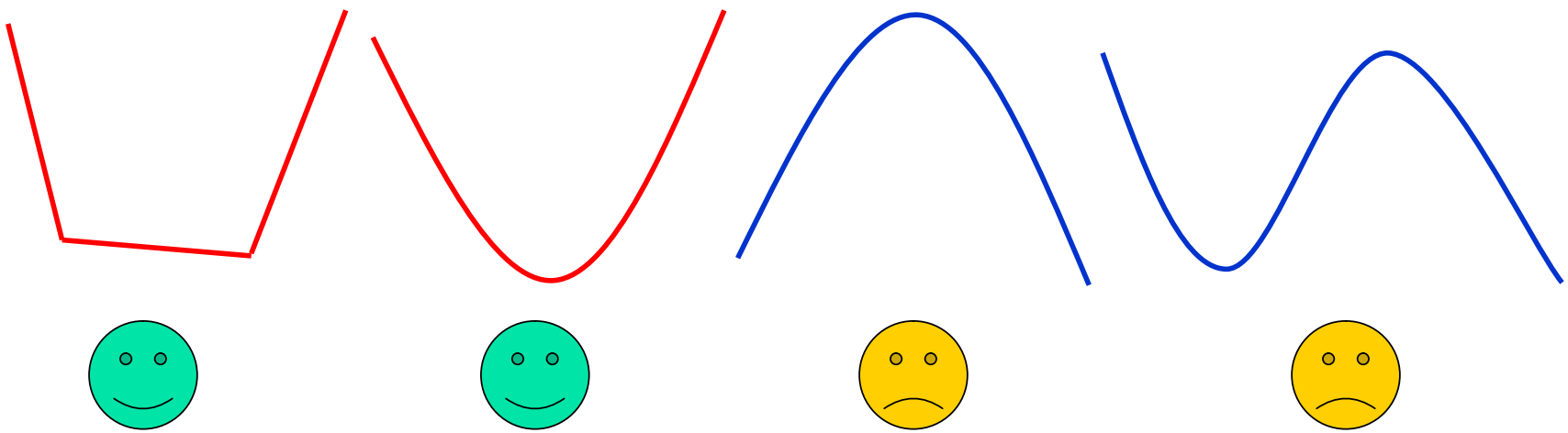
- Every local minimum is a **global minimum**, see later...

Preliminaries: convex analysis
Convex functions

- A function f is convex iff

$$\forall\ \mathbf{x}, \mathbf{y}, \lambda \in (0,1)$$

$$f(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}) \leq \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{y})$$

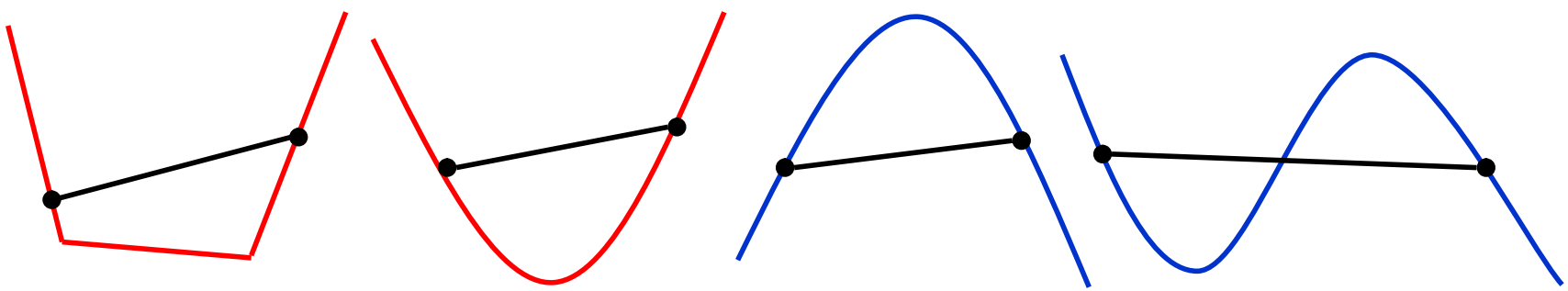


Preliminaries: convex analysis
Convex functions

- A function f is convex iff

$\forall\ \mathbf{x}, \mathbf{y}, \lambda \in (0,1)$

$$f(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}) \leq \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{y})$$



Examples of convex functions

- Univariate functions:
  - Exponential function: e^{ax} is convex for any a over $\mathbb{R}$
  - Power function: x^a is convex for $a \geq 1$ or $a \leq 0$ over $\mathbb{R}_+$ (nonnegative reals)
  - Power function: x^a is concave for $0 \leq a \leq 1$ over $\mathbb{R}_+$
  - Logarithmic function: $\log x$ is concave over $\mathbb{R}_{++}$
- Affine function: $a^T x + b$ is both convex and concave
- Quadratic function: $\frac{1}{2} x^T Q x + b^T x + c$ is convex provided that $Q \succeq 0$ (positive semidefinite)
- Least squares loss: $\|y - Ax\|_2^2$ is always convex (since $A^T A$ is always positive semidefinite)

Operations preserving convexity

- Nonnegative linear combination: $f_1, \ldots, f_m$ convex implies $a_1 f_1 + \cdots + a_m f_m$ convex for any $a_1, \ldots, a_m \geq 0$
- Pointwise maximization: if f_s is convex for any $s \in S$, then $f(x) = \max_{s \in S} f_s(x)$ is convex. Note that the set S here (number of functions f_s) can be infinite
- Partial minimization: if $g(x, y)$ is convex in x, y, and C is convex, then $f(x) = \min_{y \in C} g(x, y)$ is convex

More operations preserving convexity

- Affine composition: if f is convex, then $g(x) = f(Ax + b)$ is convex
- General composition: suppose $f = h \circ g$, where $g : \mathbb{R}^n \to \mathbb{R}$, $h : \mathbb{R} \to \mathbb{R}$, $f : \mathbb{R}^n \to \mathbb{R}$. Then:
  - f is convex if h is convex and nondecreasing, g is convex
  - f is convex if h is convex and nonincreasing, g is concave
  - f is concave if h is concave and nondecreasing, g concave
  - f is concave if h is concave and nonincreasing, g convex

  How to remember these? Think of the chain rule when $n = 1$:

$$f''(x) = h''(g(x))g'(x)^2 + h'(g(x))g''(x)$$

Board Work

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \sum_{i=1}^{N} L(x_i, y_i, \mathbf{w})$$

$L(x_i, y_i, \mathbf{w}) =$

Perceptron loss: $-\mathbf{w} \cdot f(x_i, y_i) + \max_{y' \in \mathcal{Y}} \mathbf{w} \cdot f(x_i, y')$

SVM loss: $-\mathbf{w} \cdot f(x_i, y_i) + \max_{y' \in \mathcal{Y}} \mathbf{w} \cdot f(x_i, y') + \text{cost}(y_i, y')$

Ramp loss = SVM loss - Perceptron loss

Example: log-sum-exp function

Log-sum-exp function: $g(x) = \log(\sum_{i=1}^k e^{a_i^T x + b_i})$, for fixed a_i, b_i, $i = 1, \ldots, k$. Often called "soft max", as it smoothly approximates $\max_{i=1,\ldots k} (a_i^T x + b_i)$

How to show convexity? First, note it suffices to prove convexity of $f(x) = \log(\sum_{i=1}^n e^{x_i})$ (affine composition rule)

Now use second-order characterization. Calculate

$$\nabla_i f(x) = \frac{e^{x_i}}{\sum_{\ell=1}^n e^{x_\ell}}$$

$$\nabla_{ij}^2 f(x) = \frac{e^{x_i}}{\sum_{\ell=1}^n e^{x_\ell}} 1\{i = j\} - \frac{e^{x_i} e^{x_j}}{(\sum_{\ell=1}^n e^{x_\ell})^2}$$

Write $\nabla^2 f(x) = \mathrm{diag}(z) - zz^T$, where $z_i = e^{x_i}/(\sum_{\ell=1}^n e^{x_\ell})$. This matrix is diagonally dominant, hence positive semidefinite

Preliminaries: convex analysis
Strong convexity

- A function f is called σ-strongly convex wrt a norm $\|\cdot\|$ iff

$$f(\mathbf{x}) - \frac{1}{2}\sigma \|\mathbf{x}\|^2 \quad \text{is convex}$$

$\forall\ \mathbf{x}, \mathbf{y}, \lambda \in (0,1)$

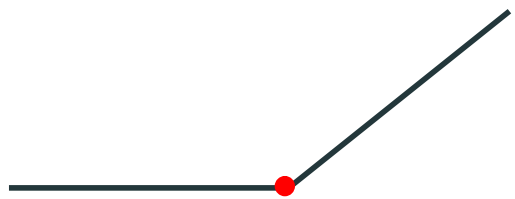
$$f(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}) \leq \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{y}) - \sigma \cdot \frac{\lambda(1-\lambda)}{2} \|\mathbf{x} - \mathbf{y}\|^2$$

Preliminaries: convex analysis
Lipschitz continuous gradient

- Lipschitz continuity
  - Stronger than continuity, weaker than differentiability
  - Upper bounds rate of change

$$\exists L > 0$$

$$|f(\mathbf{x}) - f(\mathbf{y})| \leq L \|\mathbf{x} - \mathbf{y}\| \qquad \forall\, \mathbf{x}, \mathbf{y}$$

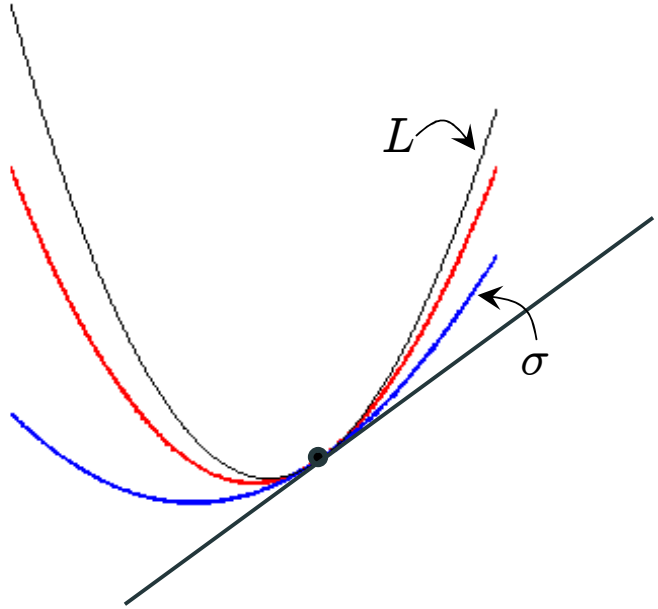


Preliminaries: convex analysis
Lipschitz continuous gradient

- Gradient is Lipschitz continuous (must be differentiable)

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq L \|\mathbf{x} - \mathbf{y}\| \qquad \forall\ \mathbf{x}, \mathbf{y}$$

$$\Leftrightarrow \quad f(\mathbf{y}) \leq f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \frac{L}{2} \|\mathbf{x} - \mathbf{y}\|^2 \qquad \forall\ \mathbf{x}, \mathbf{y}$$



L-l.c.g

Preliminaries: convex analysis
Lipschitz continuous gradient

- Gradient is Lipschitz continuous (must be differentiable)

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \leq L \|\mathbf{x} - \mathbf{y}\| \qquad \forall\ \mathbf{x}, \mathbf{y}$$

$$\Leftrightarrow \quad f(\mathbf{y}) \leq f(\mathbf{x}) + \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + \frac{L}{2} \|\mathbf{x} - \mathbf{y}\|^2 \qquad \forall\ \mathbf{x}, \mathbf{y}$$

$$\Leftrightarrow \quad \left\langle \nabla^2 f(\mathbf{x})\mathbf{y}, \mathbf{y} \right\rangle \leq L \|\mathbf{y}\|^2 \qquad \forall\ \mathbf{x}, \mathbf{y}$$

$$\nabla^2 f(x) \preceq L\mathbb{I} \qquad \text{if } L_2 \text{ norm}$$

Gradient descent

Consider unconstrained, smooth convex optimization

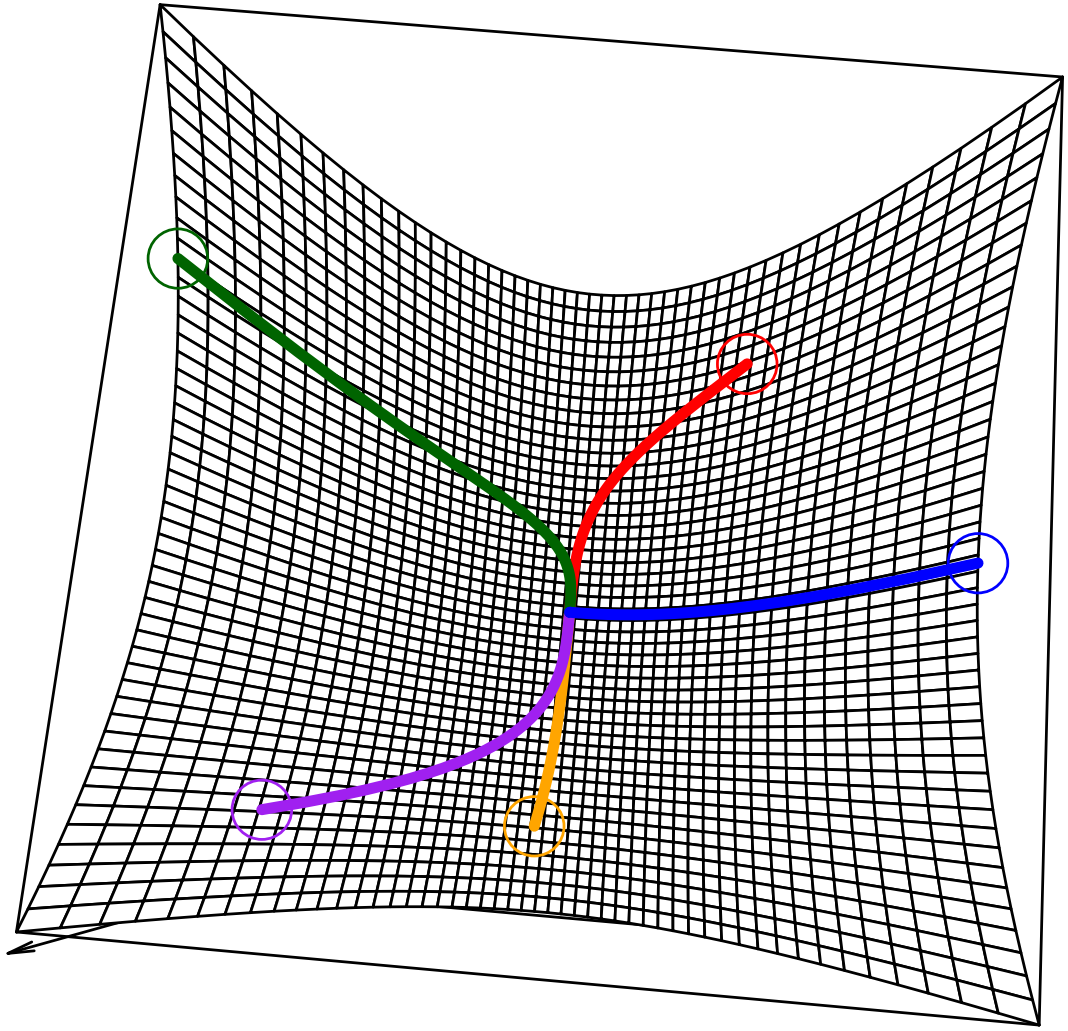
$$\min_x \ f(x)$$

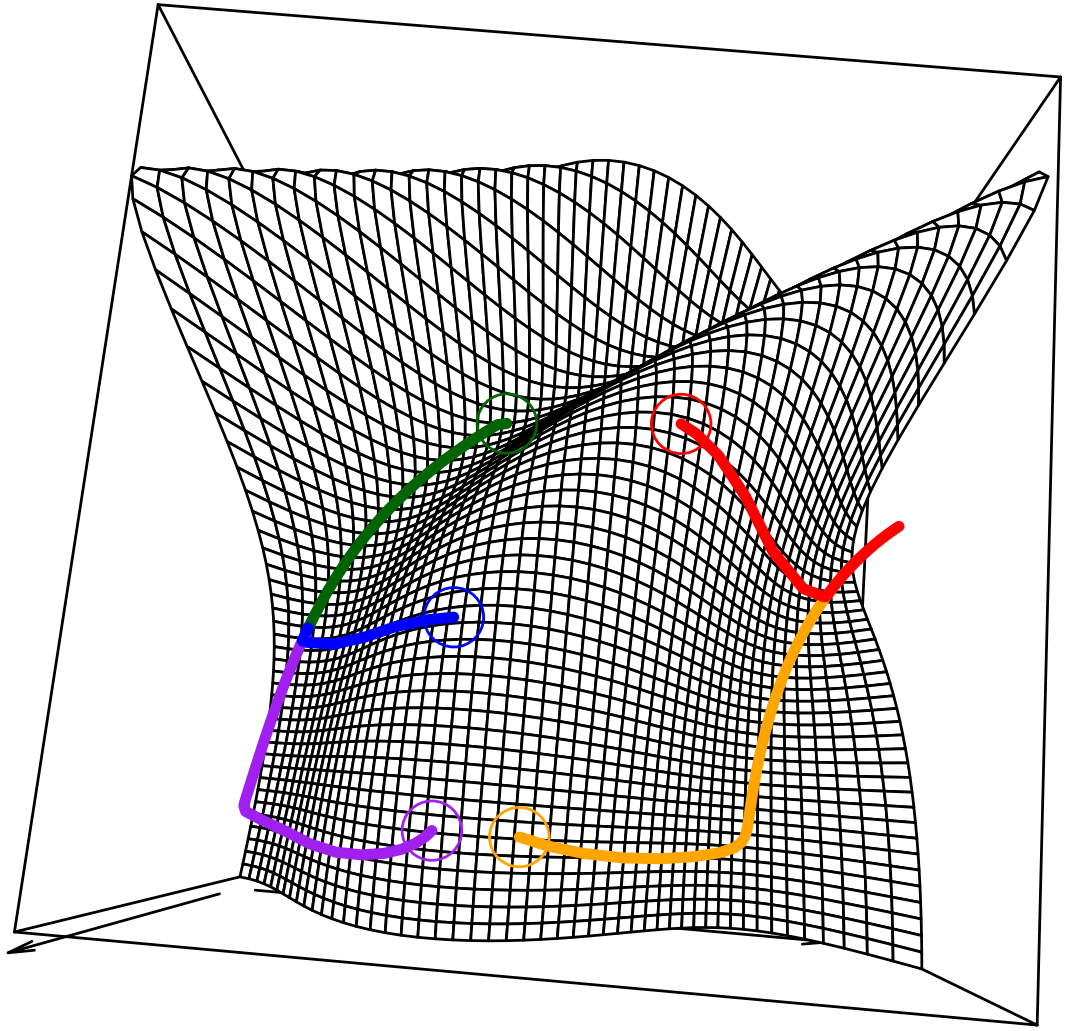
That is, f is convex and differentiable with $\mathrm{dom}(f) = \mathbb{R}^n$. Denote optimal criterion value by $f^\star = \min_x \ f(x)$, and a solution by $x^\star$

Gradient descent: choose initial point $x^{(0)} \in \mathbb{R}^n$, repeat:

$$x^{(k)} = x^{(k-1)} - t_k \cdot \nabla f(x^{(k-1)}), \quad k = 1, 2, 3, \ldots$$

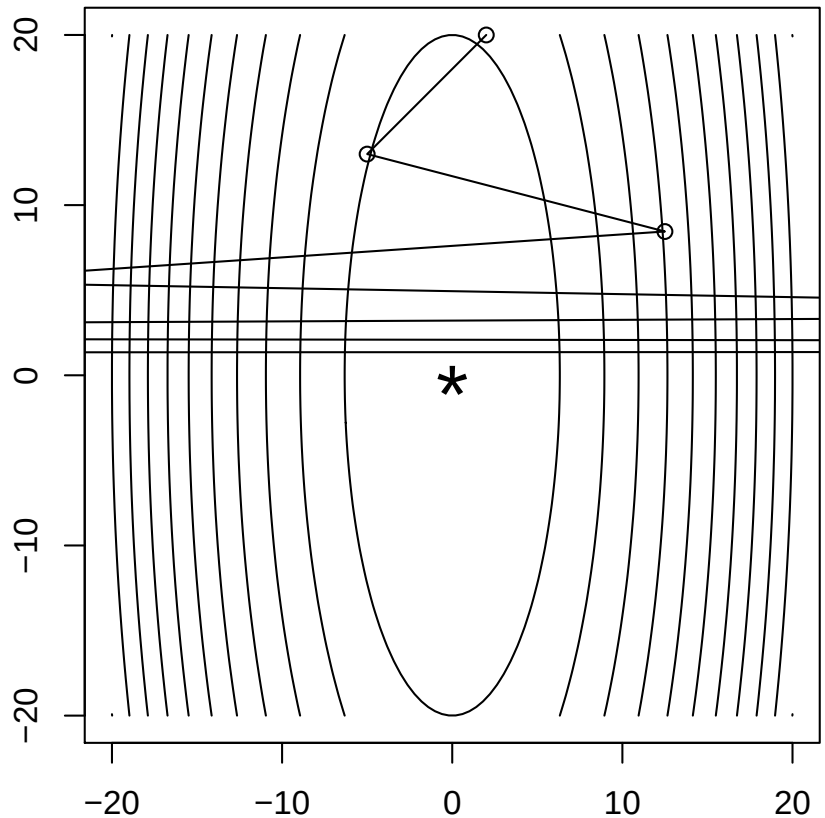
Stop at some point



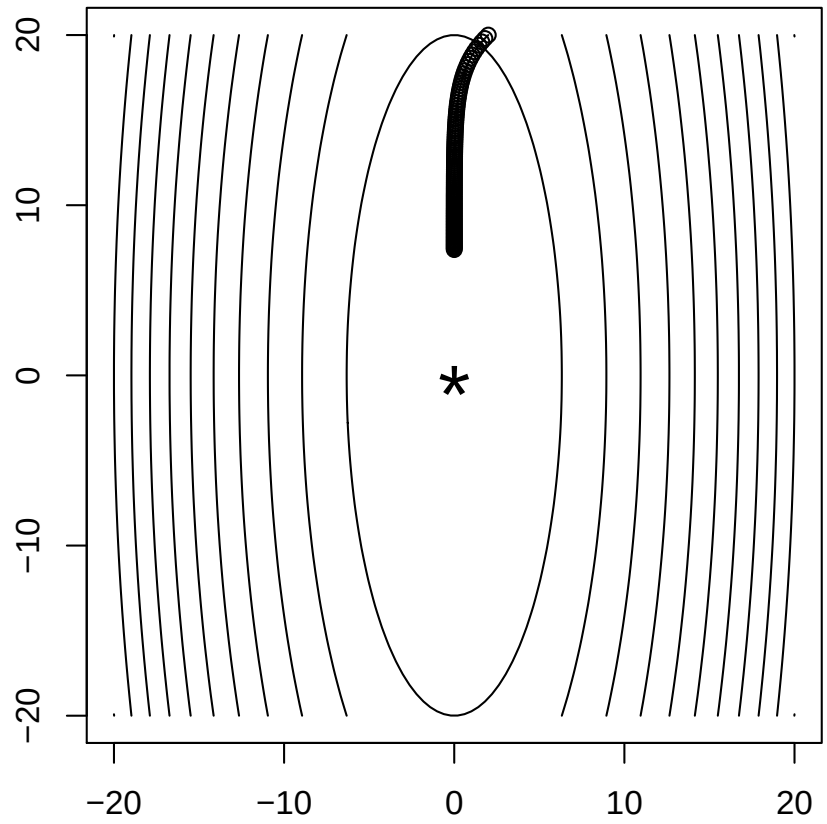


Fixed step size

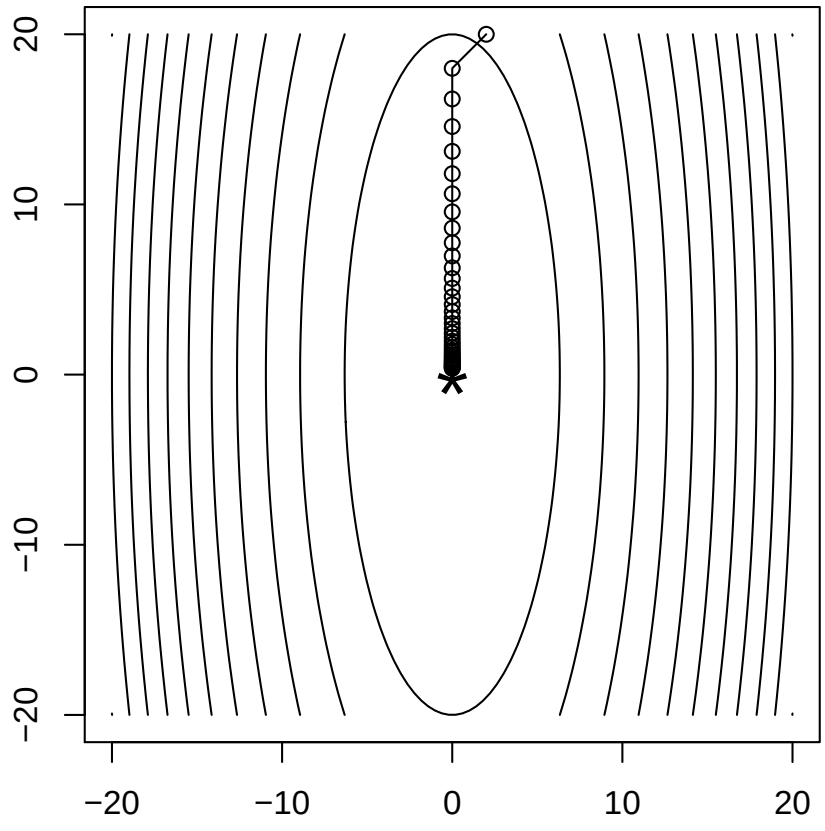
Simply take $t_k = t$ for all $k = 1, 2, 3, \ldots$, can diverge if t is too big.
Consider $f(x) = (10x_1^2 + x_2^2)/2$, gradient descent after 8 steps:



Can be slow if t is too small. Same example, gradient descent after 100 steps:

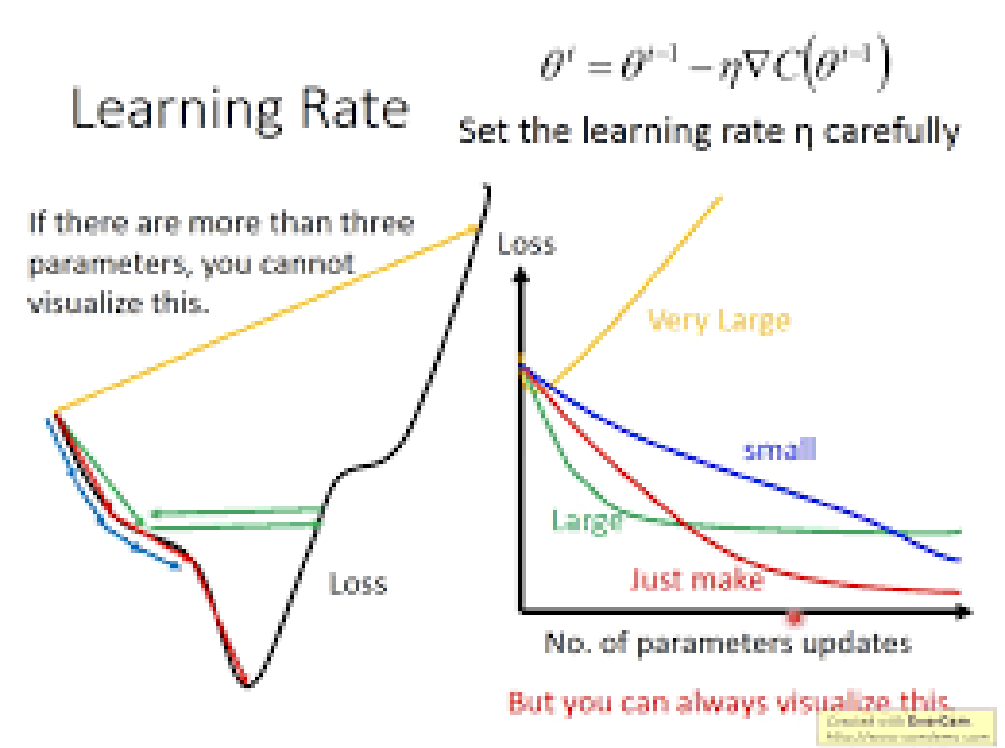


Converges nicely when t is "just right". Same example, 40 steps:



Convergence analysis later will give us a precise idea of "just right"

Effects of Step Size



Backtracking line search

One way to adaptively choose the step size is to use backtracking line search:

- First fix parameters $0 < \beta < 1$ and $0 < \alpha \leq 1/2$
- At each iteration, start with $t = t_{\text{init}}$, and while

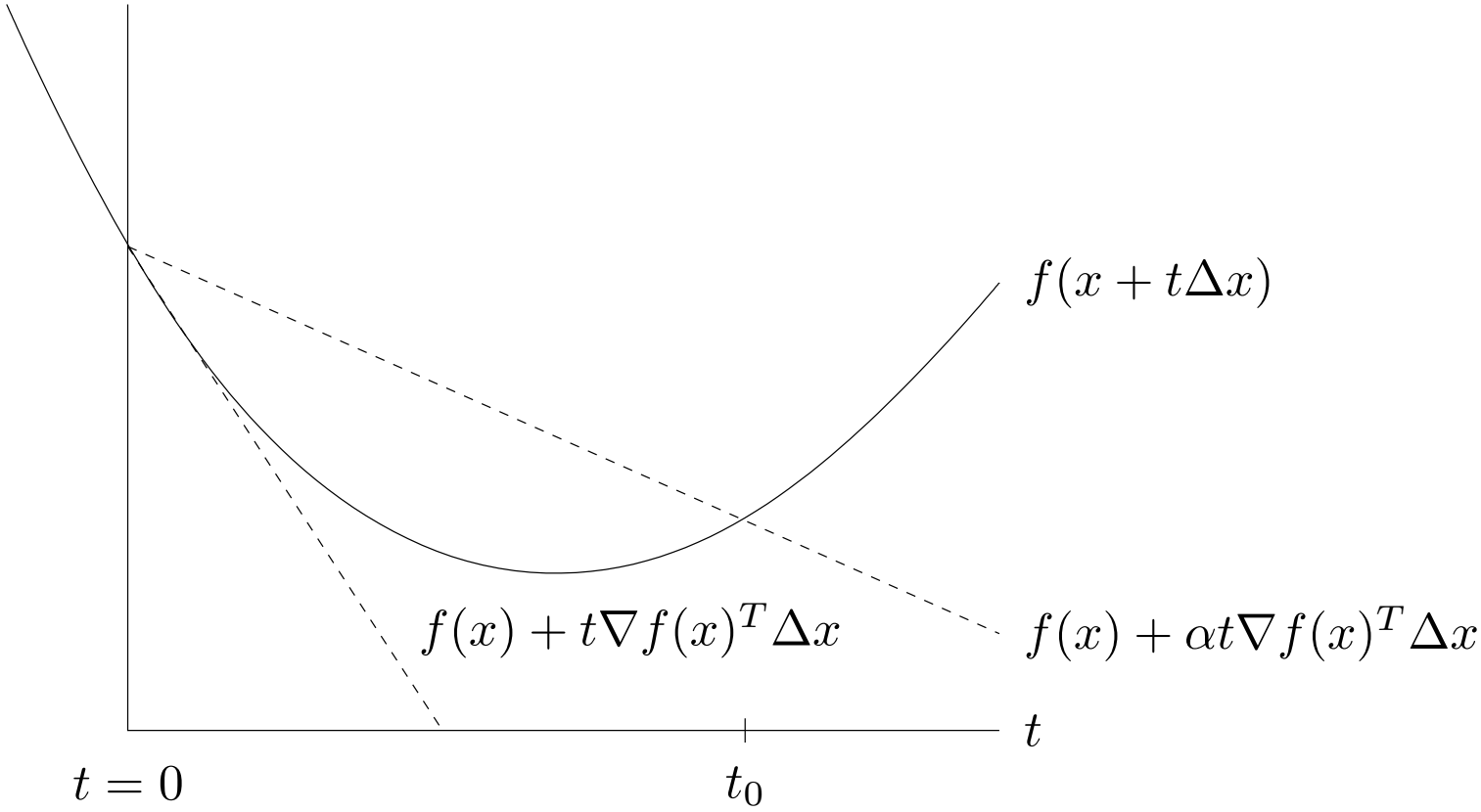
$$f(x - t\nabla f(x)) > f(x) - \alpha t \|\nabla f(x)\|_2^2$$

shrink $t = \beta t$. Else perform gradient descent update

$$x^+ = x - t\nabla f(x)$$

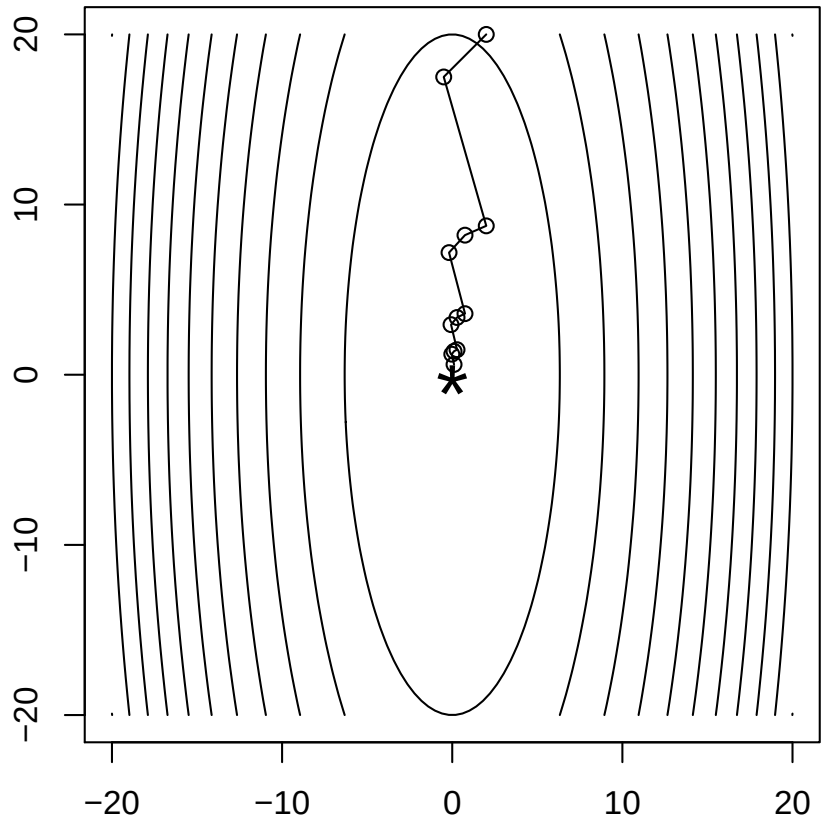
Simple and tends to work well in practice (further simplification: just take $\alpha = 1/2$)

Backtracking interpretation



For us $\Delta x = -\nabla f(x)$

Setting $\alpha = \beta = 0.5$, backtracking picks up roughly the right step size (12 outer steps, 40 steps total),



Exact line search

We could also choose step to do the best we can along direction of negative gradient, called exact line search:

$$t = \operatorname*{argmin}_{s \geq 0} \; f(x - s\nabla f(x))$$

Usually not possible to do this minimization exactly

Approximations to exact line search are typically not as efficient as backtracking, and it's typically not worth it

Convergence analysis

Assume that f convex and differentiable, with $\mathrm{dom}(f) = \mathbb{R}^n$, and additionally that ∇f is Lipschitz continuous with constant $L > 0$,

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq L\|x - y\|_2 \quad \text{for any } x, y$$

(Or when twice differentiable: $\nabla^2 f(x) \preceq LI$)

Theorem: Gradient descent with fixed step size $t \leq 1/L$ satisfies

$$f(x^{(k)}) - f^\star \leq \frac{\|x^{(0)} - x^\star\|_2^2}{2tk}$$

and same result holds for backtracking, with t replaced by β/L

We say gradient descent has convergence rate $O(1/k)$. That is, it finds ϵ-suboptimal point in $O(1/\epsilon)$ iterations

Analysis for strong convexity

Reminder: strong convexity of f means $f(x) - \frac{m}{2}\|x\|_2^2$ is convex for some $m > 0$ (when twice differentiable: $\nabla^2 f(x) \succeq mI$)

Assuming Lipschitz gradient as before, and also strong convexity:

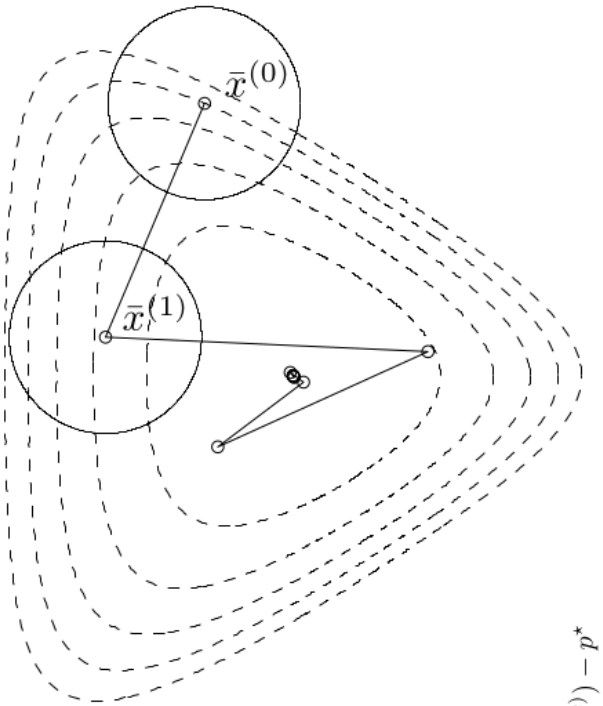
Theorem: Gradient descent with fixed step size $t \leq 2/(m+L)$ or with backtracking line search search satisfies

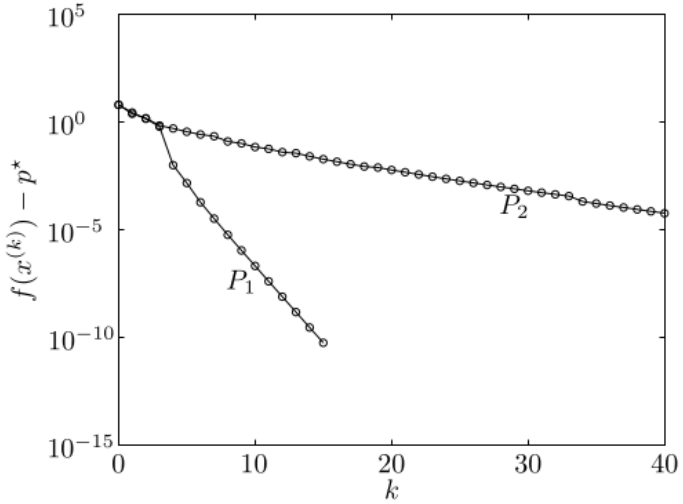
$$f(x^{(k)}) - f^\star \leq \gamma^k \frac{L}{2}\|x^{(0)} - x^\star\|_2^2$$

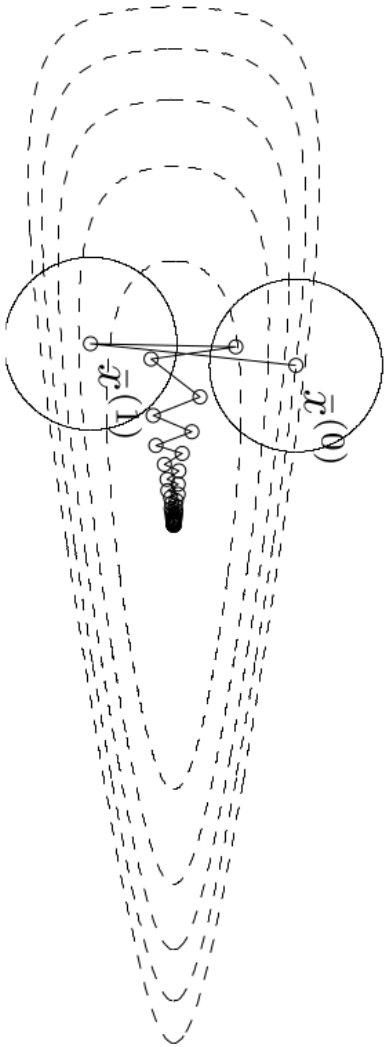
where $0 < \gamma < 1$

Rate under strong convexity is $O(\gamma^k)$, exponentially fast! That is, it finds ϵ-suboptimal point in $O(\log(1/\epsilon))$ iterations

Conditioning







Can we do better?

Gradient descent has $O(1/\epsilon)$ convergence rate over problem class of convex, differentiable functions with Lipschitz gradients

First-order method: iterative method, which updates $x^{(k)}$ in

$$x^{(0)} + \text{span}\{\nabla f(x^{(0)}), \nabla f(x^{(1)}), \ldots \nabla f(x^{(k-1)})\}$$

Theorem (Nesterov): For any $k \leq (n-1)/2$ and any starting point $x^{(0)}$, there is a function f in the problem class such that any first-order method satisfies

$$f(x^{(k)}) - f^\star \geq \frac{3L\|x^{(0)} - x^\star\|_2^2}{32(k+1)^2}$$

Can attain rate $O(1/k^2)$, or $O(1/\sqrt{\epsilon})$? Answer: yes (we'll see)!

Analysis for nonconvex case

Assume f is differentiable with Lipschitz gradient, now nonconvex. Asking for optimality is too much. Let's settle for a ϵ-substationary point x, which means $\|\nabla f(x)\|_2 \leq \epsilon$

Theorem: Gradient descent with fixed step size $t \leq 1/L$ satisfies

$$\min_{i=0,\ldots,k} \|\nabla f(x^{(i)})\|_2 \leq \sqrt{\frac{2(f(x^{(0)}) - f^\star)}{t(k+1)}}$$

Thus gradient descent has rate $O(1/\sqrt{k})$, or $O(1/\epsilon^2)$, even in the nonconvex case for finding stationary points

This rate cannot be improved (over class of differentiable functions with Lipschitz gradients) by any deterministic algorithm[1]

[1] Carmon et al. (2017), "Lower bounds for finding stationary points I"

Stochastic gradient descent

Consider minimizing an average of functions

$$\min_x \frac{1}{m} \sum_{i=1}^m f_i(x)$$

As $\nabla \sum_{i=1}^m f_i(x) = \sum_{i=1}^m \nabla f_i(x)$, gradient descent would repeat:

$$x^{(k)} = x^{(k-1)} - t_k \cdot \frac{1}{m} \sum_{i=1}^m \nabla f_i(x^{(k-1)}), \quad k = 1, 2, 3, \ldots$$

In comparison, stochastic gradient descent or SGD (or incremental gradient descent) repeats:

$$x^{(k)} = x^{(k-1)} - t_k \cdot \nabla f_{i_k}(x^{(k-1)}), \quad k = 1, 2, 3, \ldots$$

where $i_k \in \{1, \ldots, m\}$ is some chosen index at iteration k

Two rules for choosing index i_k at iteration k:

- Randomized rule: choose $i_k \in \{1, \ldots, m\}$ uniformly at random
- Cyclic rule: choose $i_k = 1, 2, \ldots, m, 1, 2, \ldots, m, \ldots$

Randomized rule is more common in practice. For randomized rule, note that

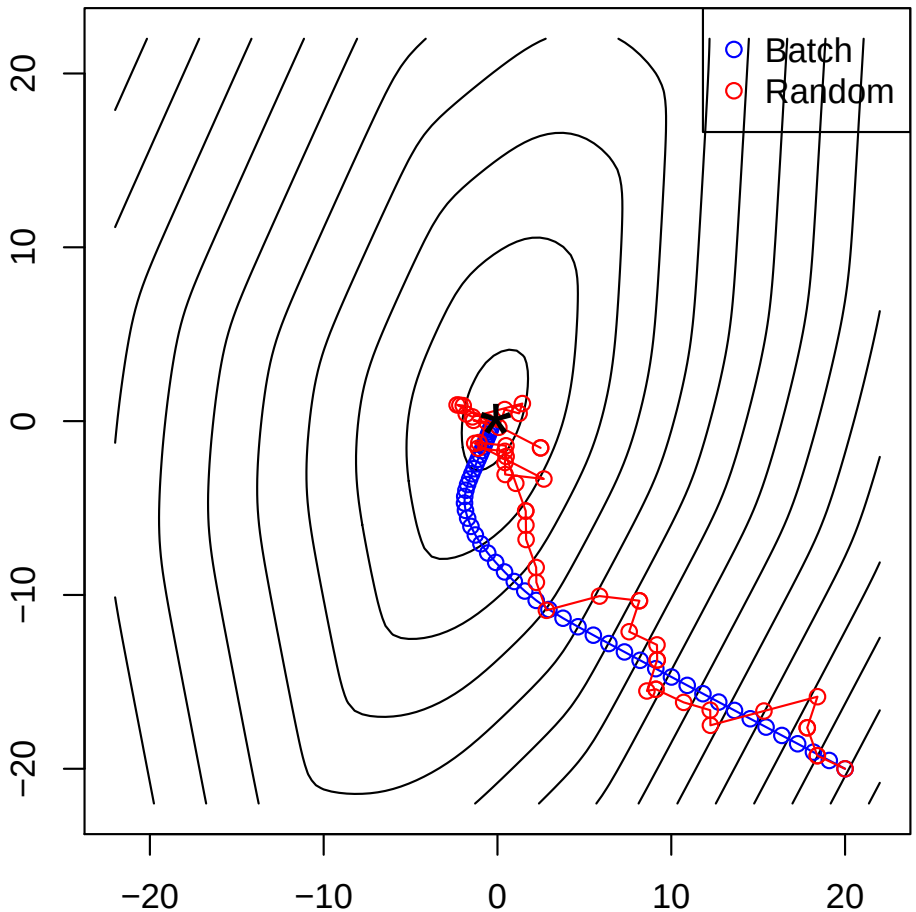
$$\mathbb{E}[\nabla f_{i_k}(x)] = \nabla f(x)$$

so we can view SGD as using an unbiased estimate of the gradient at each step

Main appeal of SGD:

- Iteration cost is independent of m (number of functions)
- Can also be a big savings in terms of memory useage

Small example with $n = 10$, $p = 2$ to show the "classic picture" for batch versus stochastic methods:



Blue: batch steps, $O(np)$
Red: stochastic steps, $O(p)$

Rule of thumb for stochastic methods:

- generally thrive far from optimum
- generally struggle close to optimum

Step sizes

Standard in SGD is to use diminishing step sizes, e.g., $t_k = 1/k$

Why not fixed step sizes? Here's some intuition. Suppose we take cyclic rule for simplicity. Set $t_k = t$ for m updates in a row, we get:

$$x^{(k+m)} = x^{(k)} - t \sum_{i=1}^{m} \nabla f_i(x^{(k+i-1)})$$

Meanwhile, full gradient with step size mt would give:

$$x^{(k+1)} = x^{(k)} - t \sum_{i=1}^{m} \nabla f_i(x^{(k)})$$

The difference here: $t \sum_{i=1}^{m} [\nabla f_i(x^{(k+i-1)}) - \nabla f_i(x^{(k)})]$, and if we hold t constant, this difference will not generally be going to zero

Convergence rates

Recall: for convex f, gradient descent with diminishing step sizes satisfies

$$f(x^{(k)}) - f^\star = O(1/\sqrt{k})$$

When f is differentiable with Lipschitz gradient, we get for suitable fixed step sizes

$$f(x^{(k)}) - f^\star = O(1/k)$$

What about SGD? For convex f, SGD with diminishing step sizes satisfies[1]

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

Unfortunately this does not improve when we further assume f has Lipschitz gradient

[1] For example, Nemirosvki et al. (2009), "Robust stochastic optimization approach to stochastic programming"

Mini-batches

Also common is mini-batch stochastic gradient descent, where we choose a random subset $I_k \subseteq \{1, \ldots, m\}$, $|I_k| = b \ll m$, repeat:

$$x^{(k)} = x^{(k-1)} - t_k \cdot \frac{1}{b} \sum_{i \in I_k} \nabla f_i(x^{(k-1)}), \quad k = 1, 2, 3, \ldots$$

Again, we are approximating full gradient by an unbiased estimate:

$$\mathbb{E}\left[\frac{1}{b} \sum_{i \in I_k} \nabla f_i(x)\right] = \nabla f(x)$$

Using mini-batches reduces variance by a factor $1/b$, but is also b times more expensive. Theory is not convincing: under Lipschitz gradient, rate goes from $O(1/\sqrt{k})$ to $O(1/\sqrt{bk} + 1/k)$[3]

[3]For example, Dekel et al. (2012), "Optimal distributed online prediction using mini-batches"

Back to logistic regression, let's now consider a regularized version:

$$\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} \Big(- y_i x_i^T \beta + \log(1 + e^{x_i^T \beta}) \Big) + \frac{\lambda}{2} \|\beta\|_2^2$$

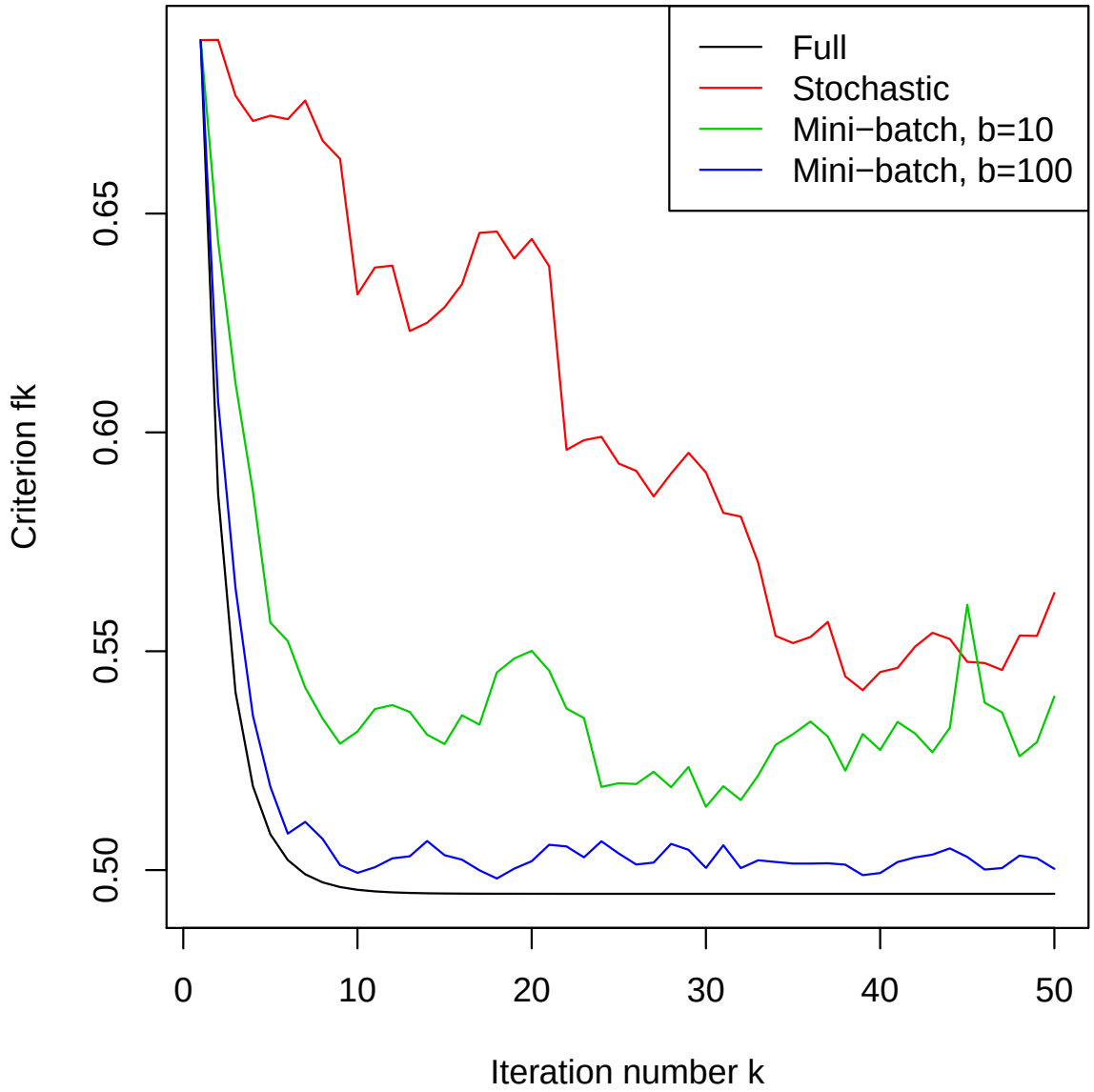
Write the criterion as

$$f(\beta) = \frac{1}{n} \sum_{i=1}^{n} f_i(\beta), \quad f_i(\beta) = -y_i x_i^T \beta + \log(1 + e^{x_i^T \beta}) + \frac{\lambda}{2} \|\beta\|_2^2$$

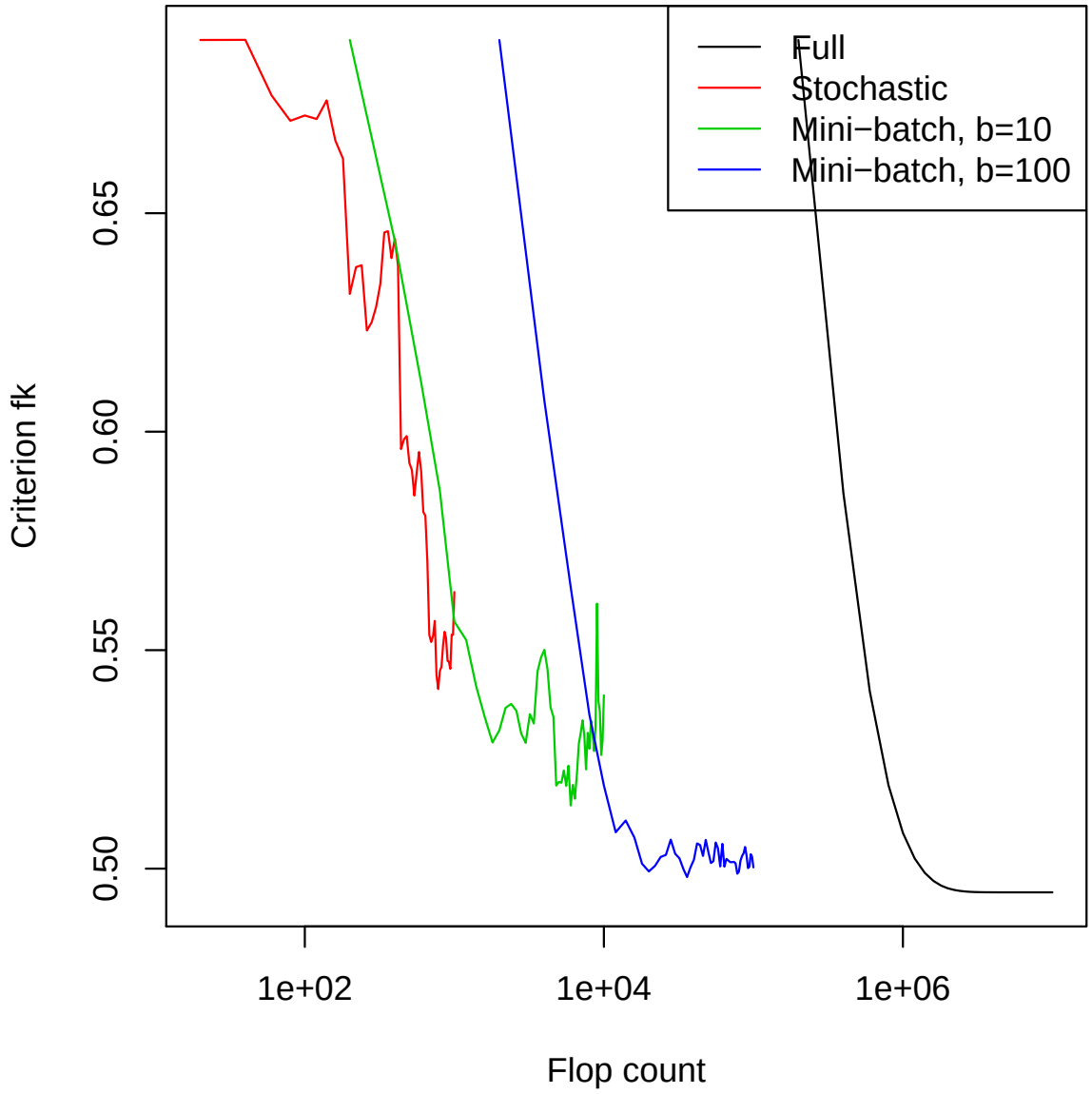
Full gradient computation is $\nabla f(\beta) = \frac{1}{n} \sum_{i=1}^{n} \big(y_i - p_i(\beta)\big) x_i + \lambda \beta$.
Comparison between methods:

- One batch update costs $O(np)$
- One mini-batch update costs $O(bp)$
- One stochastic update costs $O(p)$

Example with $n = 10,000$, $p = 20$, all methods use fixed step sizes:



What's happening? Now let's parametrize by flops:



End of the story?

Short story:

- SGD can be super effective in terms of iteration cost, memory
- But SGD is slow to converge, can't adapt to strong convexity
- And mini-batches seem to be a wash in terms of flops (though they can still be useful in practice)

Is this the end of the story for SGD?

For a while, the answer was believed to be yes. Slow convergence for strongly convex functions was believed inevitable, as Nemirovski and others established matching lower bounds ... but this was for a more general stochastic problem, where $f(x) = \int F(x, \xi)\, dP(\xi)$

New wave of "variance reduction" work shows we can modify SGD to converge much faster for finite sums (more later?)

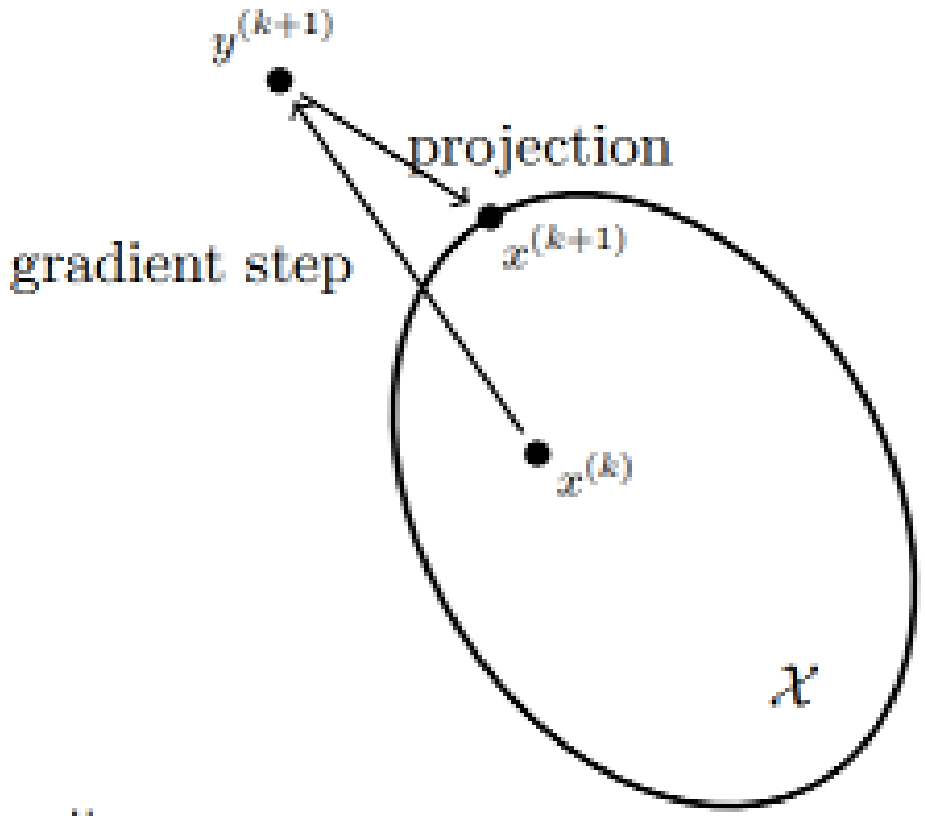
SGD in large-scale ML

SGD has really taken off in large-scale machine learning

- In many ML problems we don't care about optimizing to high accuracy, it doesn't pay off in terms of statistical performance
- Thus (in contrast to what classic theory says) fixed step sizes are commonly used in ML applications
- One trick is to experiment with step sizes using small fraction of training before running SGD on full data set[4]
- Momentum/acceleration, averaging, adaptive step sizes are all popular variants in practice
- SGD is especially popular in large-scale, continous, nonconvex optimization, but it is still not particular well-understood there (a big open issue is that of implicit regularization)

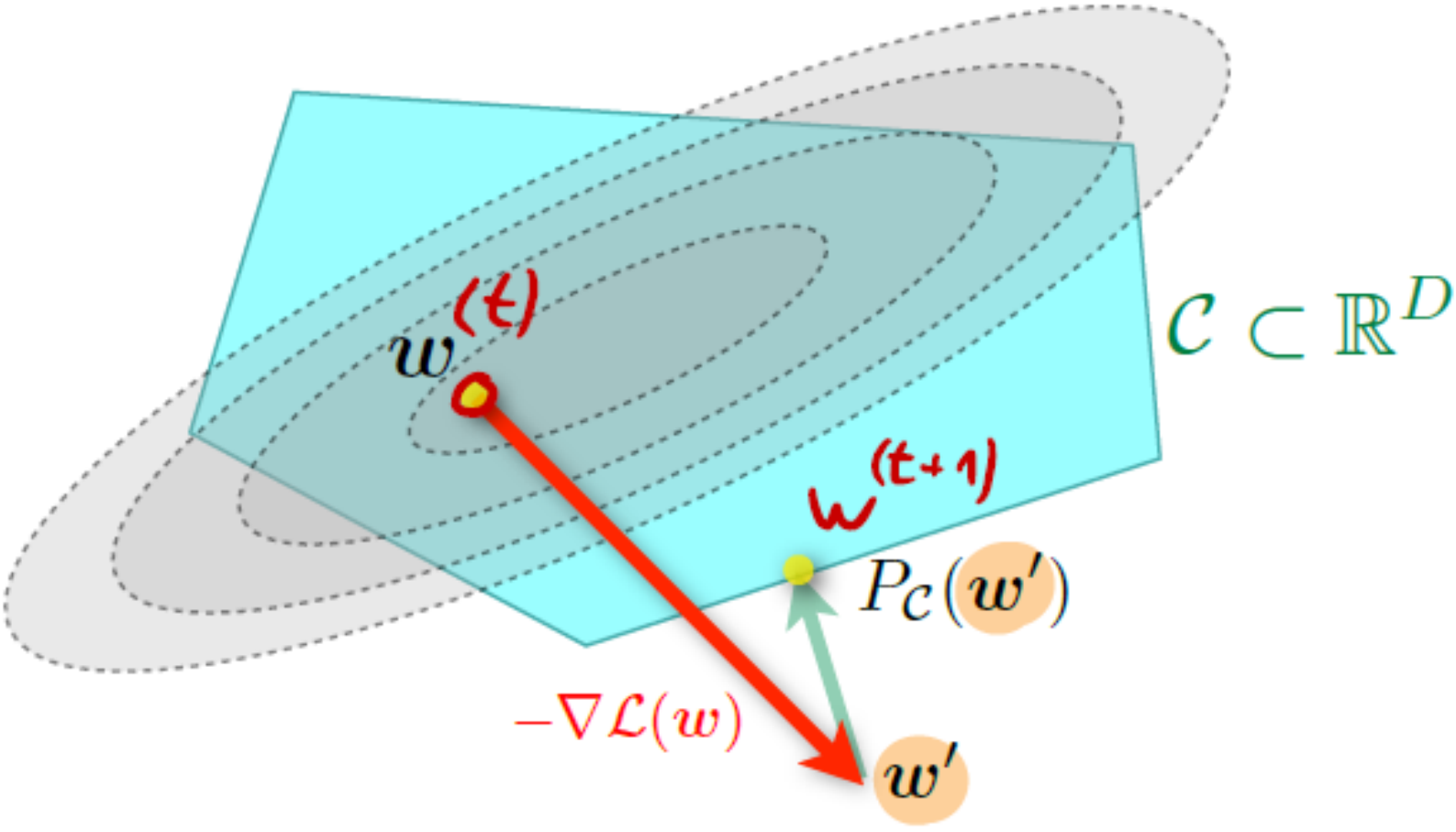
[4] For example, Bottou (2012), "Stochastic gradient descent tricks"

Handling Constraints: Projected Gradient Descent



..

Handling Constraints: Projected Gradient Descent



Constraint set $\mathcal{C}$ must be a convex set.

$P_{\mathcal{C}}(\mathbf{w})$ is the projection operator $P_{\mathcal{C}}(\mathbf{w}) = \text{argmin}_{\mathbf{v} \in \mathcal{C}} \|\mathbf{v} - \mathbf{w}\|_2$

Handling Constraints: Projected Gradient Descent

Algorithm 1 Projected Gradient Descent (PGD)

Input: Convex objective f, convex constraint set $\mathcal{C}$, step lengths η_t

Output: A point $\hat{\mathbf{x}} \in \mathcal{C}$ with near-optimal objective value

1: $\mathbf{x}^1 \leftarrow \mathbf{0}$
2: **for** $t = 1, 2, \ldots, T$ **do**
3: $\quad \mathbf{z}^{t+1} \leftarrow \mathbf{x}^t - \eta_t \cdot \nabla f(\mathbf{x}^t)$
4: $\quad \mathbf{x}^{t+1} \leftarrow \Pi_{\mathcal{C}}(\mathbf{z}^{t+1})$
5: **end for**

Return $\mathbf{x}^T$

Projection operator:

$$\Pi_{\mathcal{C}}(\mathbf{z}) = \operatorname*{argmin}_{\mathbf{x} \in \mathcal{C}} \|\mathbf{x} - \mathbf{z}\|_2$$

Newton's method

Given unconstrained, smooth convex optimization

$$\min_x \; f(x)$$

where f is convex, twice differentable, and $\mathrm{dom}(f) = \mathbb{R}^n$. Recall that gradient descent chooses initial $x^{(0)} \in \mathbb{R}^n$, and repeats

$$x^{(k)} = x^{(k-1)} - t_k \cdot \nabla f(x^{(k-1)}), \quad k = 1, 2, 3, \ldots$$

In comparison, Newton's method repeats

$$x^{(k)} = x^{(k-1)} - \left(\nabla^2 f(x^{(k-1)})\right)^{-1} \nabla f(x^{(k-1)}), \quad k = 1, 2, 3, \ldots$$

Here $\nabla^2 f(x^{(k-1)})$ is the Hessian matrix of f at $x^{(k-1)}$

Newton's method interpretation

Recall the motivation for gradient descent step at x: we minimize the quadratic approximation

$$f(y) \approx f(x) + \nabla f(x)^T (y - x) + \frac{1}{2t}\|y - x\|_2^2$$

over y, and this yields the update $x^+ = x - t\nabla f(x)$

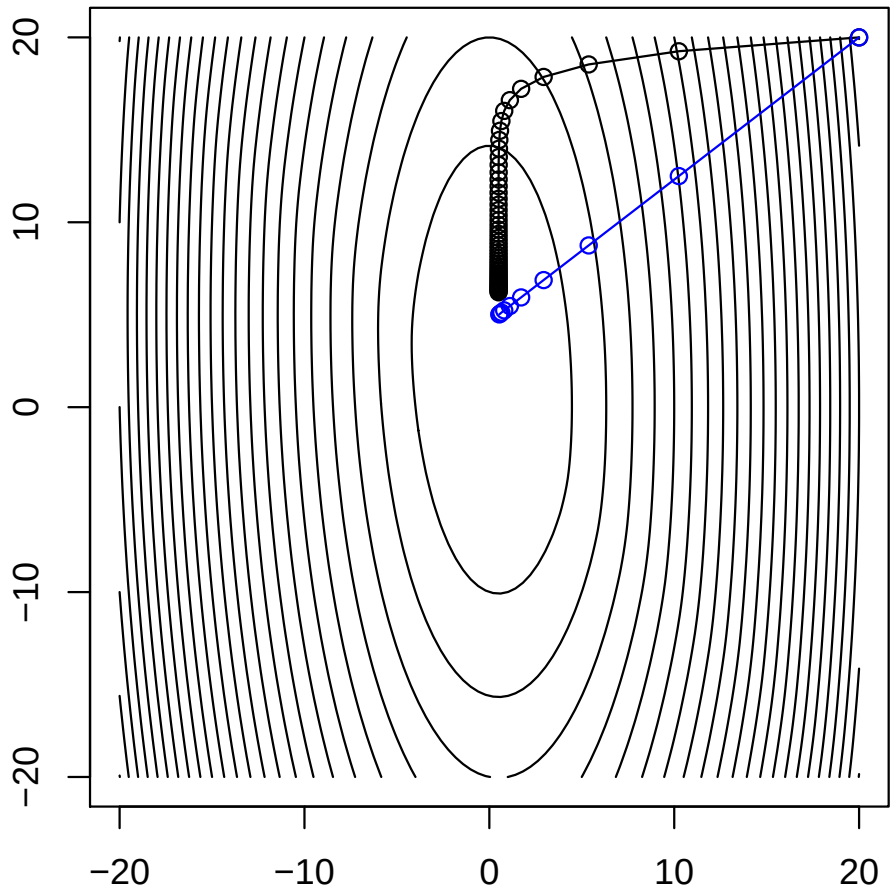
Newton's method uses in a sense a better quadratic approximation

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

and minimizes over y to yield $x^+ = x - (\nabla^2 f(x))^{-1} \nabla f(x)$

Consider minimizing $f(x) = (10x_1^2 + x_2^2)/2 + 5\log(1 + e^{-x_1 - x_2})$
(this must be a nonquadratic ... why?)

We compare gradient descent (black) to Newton's method (blue), where both take steps of roughly same length



Affine invariance of Newton's method

Important property Newton's method: affine invariance. Given f, nonsingular $A \in \mathbb{R}^{n \times n}$. Let $x = Ay$, and $g(y) = f(Ay)$. Newton steps on g are

$$\begin{aligned} y^+ &= y - \big(\nabla^2 g(y)\big)^{-1} \nabla g(y) \\ &= y - \big(A^T \nabla^2 f(Ay) A\big)^{-1} A^T \nabla f(Ay) \\ &= y - A^{-1} \big(\nabla^2 f(Ay)\big)^{-1} \nabla f(Ay) \end{aligned}$$

Hence

$$Ay^+ = Ay - \big(\nabla^2 f(Ay)\big)^{-1} \nabla f(Ay)$$

i.e.,

$$x^+ = x - \big(\nabla^2 f(x)\big)^{-1} \nabla f(x)$$

So progress is independent of problem scaling. This is not true of gradient descent!

Comparison to first-order methods

At a high-level:

- Memory: each iteration of Newton's method requires $O(n^2)$ storage ($n \times n$ Hessian); each gradient iteration requires $O(n)$ storage (n-dimensional gradient)
- Computation: each Newton iteration requires $O(n^3)$ flops (solving a dense $n \times n$ linear system); each gradient iteration requires $O(n)$ flops (scaling/adding n-dimensional vectors)
- Backtracking: backtracking line search has roughly the same cost, both use $O(n)$ flops per inner backtracking step
- Conditioning: Newton's method is not affected by a problem's conditioning, but gradient descent can seriously degrade

Quasi-Newton methods

If the Hessian is too expensive (or singular), then a quasi-Newton method can be used to approximate $\nabla^2 f(x)$ with $H \succ 0$, and we update according to

$$x^+ = x - tH^{-1}\nabla f(x)$$

- Approximate Hessian H is recomputed at each step. Goal is to make H^{-1} cheap to apply (possibly, cheap storage too)
- Convergence is fast: superlinear, but not the same as Newton. Roughly n steps of quasi-Newton make same progress as one Newton step
- Very wide variety of quasi-Newton methods; common theme is to "propogate" computation of H across iterations