Optimization

Jong-Han Kim

EE787 Machine learning
Kyung Hee University

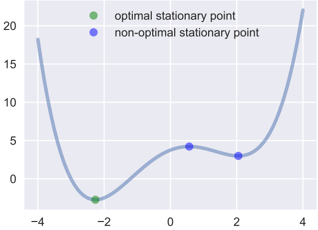
Optimization problems and algorithms

Optimization problem

$$\text{minimize} \quad f(\theta)$$

- $\theta \in \mathbf{R}^d$ is the *variable* or *decision variable*
- $f : \mathbf{R}^d \to \mathbf{R}$ is the *objective function*
- goal is to choose θ to minimize f
- $\theta^\star$ is *optimal* means that for all θ, $f(\theta) \geq f(\theta^\star)$
- $f^\star = f(\theta^\star)$ is the *optimal value* of the problem
- optimization problems arise in many fields and applications, including machine learning

Optimality condition



- let's assume that f is *differentiable*, *i.e.*, partial derivatives $\frac{\partial f(\theta)}{\partial \theta_i}$ exist
- if $\theta^\star$ is optimal, then $\nabla f(\theta^\star) = 0$
- $\nabla f(\theta) = 0$ is called the *optimality condition* for the problem
- there can be points that satisfy $\nabla f(\theta) = 0$ but are not optimal
- we call points that satisfy $\nabla f(\theta) = 0$ *stationary points*
- not all stationary points are optimal

Solving optimization problems

- in some cases, we can solve the problem analytically
- *e.g.*, least squares: minimize $f(\theta) = ||X\theta - y||^2$
  - optimality condition is $\nabla f(\theta) = 2X^T(X\theta - y) = 0$
  - this has (unique) solution $\theta^\star = (X^TX)^{-1}X^Ty = X^\dagger y$
    (when columns of X are linearly independent)
- in other cases, we resort to an *iterative algorithm* that computes a sequence $\theta^1, \theta^2, \ldots$ with, hopefully, $f(\theta^k) \to f^\star$ as $k \to \infty$

Iterative algorithms

- *iterative algorithm* computes a sequence $\theta^1, \theta^2, \ldots$
- θ^k is called the kth *iterate*
- θ^1 is called the *starting point*
- many iterative algorithms are *descent methods*, which means

$$f(\theta^{k+1}) < f(\theta^k), \quad k = 1, 2, \ldots$$

  i.e., each iterate is better than the previous one
- this means that $f(\theta^k)$ converges, but not necessarily to $f^\star$

Stopping criterion

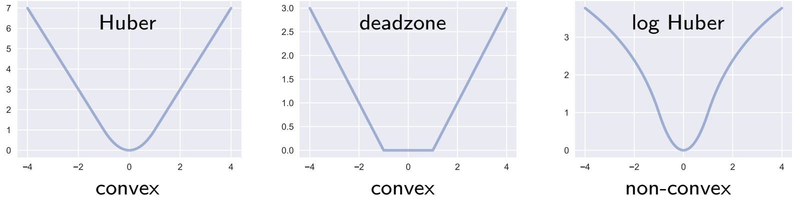
- in practice, we stop after a finite number K of steps
- typical stopping criterion: stop if $||\nabla f(\theta^k)|| \leq \epsilon$ or $k = k^{\mathrm{max}}$
- ϵ is a small positive number, the *stopping tolerance*
- k^{max} is the maximum number of iterations
- in words: we stop when θ^k is almost a stationary point
- we hope that $f(\theta^K)$ is not too much bigger than $f^\star$
- or more realistically, that θ^K is at least useful for our application

Non-heuristic and heuristic algorithms

- in some cases we *know* that $f(\theta^k) \to f^\star$, for any θ^1
- in words: *we'll get to a solution if we keep iterating*
- called *non-heuristic*

- other algorithms do not guarantee that $f(\theta^k) \to f^\star$
- we can hope that even if $f(\theta^k) \not\to f^\star$, θ^k is still useful for our application
- called *heuristic*

Convex functions



- a function $f : \mathbf{R}^d \to \mathbf{R}$ is *convex* if for any θ, $\tilde{\theta}$, and α with $0 \leq \alpha \leq 1$,

$$f(\alpha\theta + (1-\alpha)\tilde{\theta}) \leq \alpha f(\theta) + (1-\alpha) f(\tilde{\theta})$$

- roughly speaking, f has 'upward curvature'
- for $d = 1$, same as $f''(\theta) \geq 0$ for all θ

Convex optimization

- optimization problem

$$\text{minimize} \quad f(\theta)$$

  is called *convex* if the objective function f is convex

- for convex optimization problem, $\nabla f(\theta) = 0$ only for θ optimal, *i.e.*, *all stationary points are optimal*

- algorithms for convex optimization are non-heuristic

- *i.e.*, *we can solve convex optimization problems* (exactly, in principle)

Convex ERM problems

- regularized empirical risk function $f(\theta) = \mathcal{L}(\theta) + \lambda r(\theta)$, with $\lambda \geq 0$,

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} p(\theta^\mathsf{T} x^i - y^i), \qquad r(\theta) = q(\theta_1) + \cdots + q(\theta_d)$$

- f is convex if loss penalty p and parameter penalty q functions are convex

- convex penalties: square, absolute, tilted absolute, Huber
- non-convex penalties: log Huber, squareroot

Gradient method

Gradient method

- assume f is differentiable
- at iteration θ^k, create affine (Taylor) approximation of f valid near θ^k

$$\hat{f}(\theta;\theta^k) = f(\theta^k) + \nabla f(\theta^k)^T(\theta - \theta^k)$$

- $\hat{f}(\theta;\theta^k) \approx f(\theta)$ for θ near θ^k
- choose θ^{k+1} to make $\hat{f}(\theta^{k+1};\theta^k)$ small, but with $||\theta^{k+1} - \theta^k||$ not too large
- choose θ^{k+1} to minimize $\hat{f}(\theta;\theta^k) + \frac{1}{2h^k}||\theta - \theta^k||^2$
- $h^k > 0$ is a *trust parameter* or *step length* or *learning rate*
- solution is $\theta^{k+1} = \theta^k - h^k \nabla f(\theta^k)$
- roughly: take step in direction of negative gradient

Gradient method update

- choose θ^{k+1} to as minimizer of

$$f(\theta^k) + \nabla f(\theta^k)^T(\theta - \theta^k) + \frac{1}{2h^k}||\theta - \theta^k||^2$$

- rewrite as

$$f(\theta^k) + \frac{1}{2h^k}||(\theta - \theta^k) + h^k\nabla f(\theta^k)||^2 - \frac{h^k}{2}||\nabla f(\theta^k)||^2$$

- first and third terms don't depend on θ
- middle term is minimized (made zero!) by choice

$$\theta = \theta^k - h^k\nabla f(\theta^k)$$

How to choose step length

- if h^k is too large, we can have $f(\theta^{k+1}) > f(\theta^k)$
- if h^k is too small, we have $f(\theta^{k+1}) < f(\theta^k)$ but progress is slow

- a simple scheme:
  - if $f(\theta^{k+1}) > f(\theta^k)$, set $h^{k+1} = h^k/2$, $\theta^{k+1} = \theta^k$ (a *rejected step*)
  - if $f(\theta^{k+1}) \leq f(\theta^k)$, set $h^{k+1} = 1.2h^k$ (an *accepted step*)
- reduce step length by half if it's too long; increase it 20% otherwise

Gradient method summary

choose an initial $\theta^1 \in \mathbf{R}^d$ and $h^1 > 0$ (*e.g.*, $\theta^1 = 0$, $h^1 = 1$)

for $k = 1, 2, \ldots, k^{\text{max}}$

1. compute $\nabla f(\theta^k)$; quit if $\|\nabla f(\theta^k)\|$ is small enough
2. form tentative update $\theta^{\text{tent}} = \theta^k - h^k \nabla f(\theta^k)$
3. if $f(\theta^{\text{tent}}) \leq f(\theta^k)$, set $\theta^{k+1} = \theta^{\text{tent}}$, $h^{k+1} = 1.2h^k$
4. else set $h^k := 0.5h^k$ and go to step 2

Gradient method convergence

- (assuming some technical conditions hold) we have

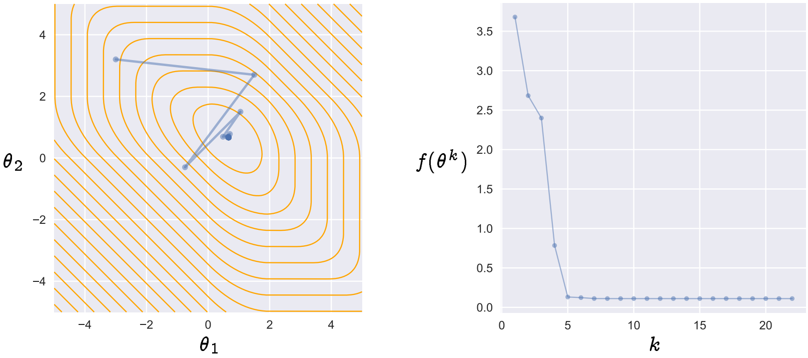
$$\|\nabla f(\theta^k)\| \to 0 \text{ as } k \to \infty$$

- *i.e.*, the gradient method always finds a stationary point

- for *convex problems*
  - gradient method is *non-heuristic*
  - for any starting point θ^1, $f(\theta^k) \to f^\star$ as $k \to \infty$

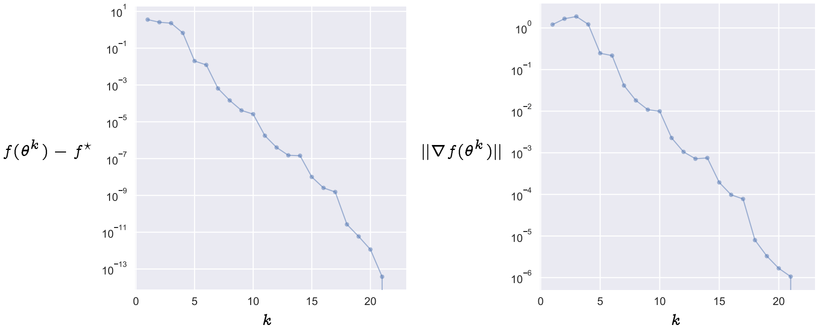
- for *non-convex problems*
  - gradient method is *heuristic*
  - we can (and often do) have $f(\theta^k) \not\to f^\star$

Example: Convex objective



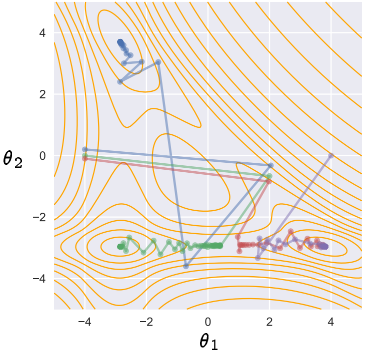
- $f(\theta) = \frac{1}{3}\left(p^{\text{hub}}(\theta_1 - 1) + p^{\text{hub}}(\theta_2 - 1) + p^{\text{hub}}(\theta_1 + \theta_2 - 1)\right)$
- f is convex
- optimal point is $\theta^\star = (2/3, 2/3)$, with $f^\star = 1/9$

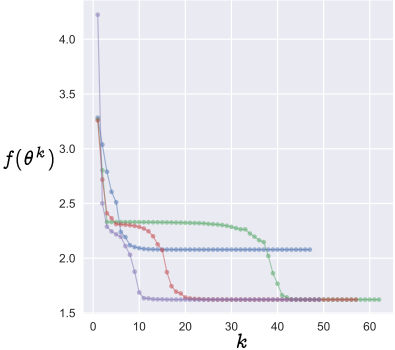
Example: Convex objective



- $f(\theta^k)$ is a decreasing function of k, (roughly) exponentially
- $||\nabla f(\theta^k)|| \to 0$ as $k \to \infty$

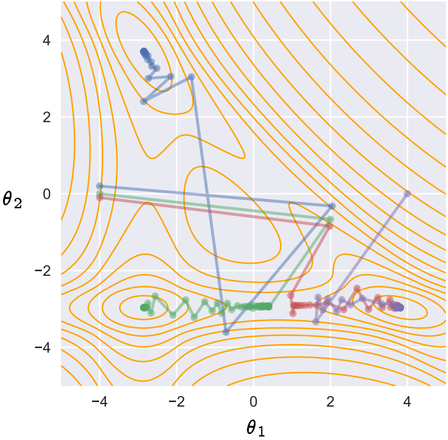
Example: Non-convex objective

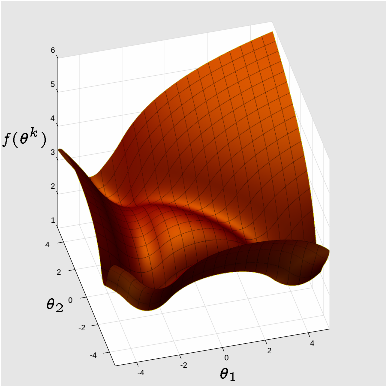




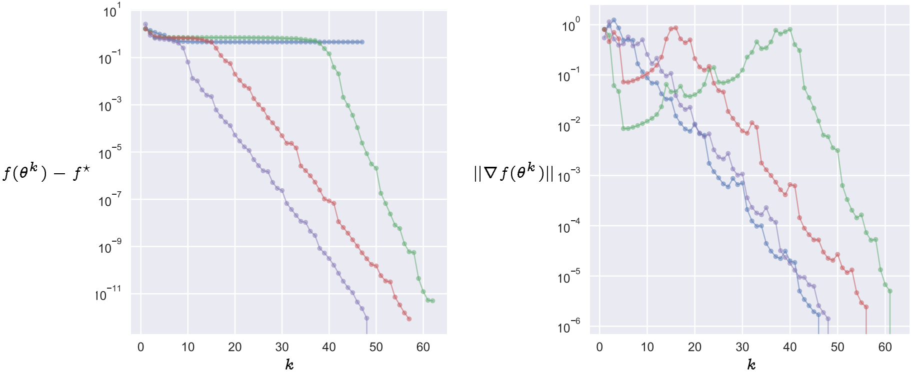
- $f(\theta) = \frac{1}{3}\left(p^{\mathrm{lh}}(\theta_1 + 3) + p^{\mathrm{lh}}(2\theta_2 + 6) + p^{\mathrm{lh}}(\theta_1 + \theta_2 - 1)\right)$
- f is sum of log-Huber functions, so not convex
- gradient algorithm converges, but limit depends on initial guess

Example: Non-convex objective





Example: Non-convex objective



Gradient method for ERM

Gradient of empirical risk function

- empirical risk is sum of terms for each data point

$$\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(\hat{y}^i, y^i) = \frac{1}{n}\sum_{i=1}^{n} \ell(\theta^T x^i, y^i)$$

- convex if loss function ℓ is convex in first argument
- gradient is sum of terms for each data point

$$\nabla\mathcal{L}(\theta) = \nabla\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell'(\theta^T x^i, y^i) x^i$$

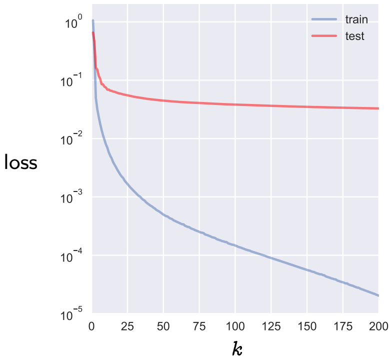
where $\ell'(\hat{y}, y)$ is derivative of ℓ with respect to its first argument $\hat{y}$

Evaluating gradient of empirical risk function

- compute n-vector $\hat{y}^k = X\theta^k$
- compute n-vector z^k, with entries $z_i^k = \ell'(\hat{y}_i^k, y^i)$
- compute d-vector $\nabla\mathcal{L}(\theta^k) = (1/n)X^T z^k$

- first and third steps are matrix-vector multiplication, each costing $2nd$ flops
- second step costs order n flops (dominated by other two)
- total is $4nd$ flops

Validation



- can evaluate empirical risk on train and test while gradient is running
- optimization is only a surrogate for what we want
  (*i.e.*, a predictor that predicts well on unseen data)
- predictor is often good enough well before gradient descent has converged