Lecture 4a. Iterative Optimisation.

COMP90051 Statistical Machine Learning

Semester 2, 2020
Lecturer: Ben Rubinstein

This lecture

- **Iterative optimisation** for extremum estimators
  - First-order method: **Gradient descent**
  - Second-order: **Newton-Raphson method**
  - Later: Lagrangian duality
- Logistic regression: workhorse linear classifier
  - Possibly familiar derivation: frequentist
  - Decision-theoretic derivation
  - Training with Newton-Raphson looks like repeated, weighted linear regression

Gradient Descent

Brief review of most basic optimisation approach in ML

Optimisation formulations in ML

- Training = Fitting = Parameter estimation
- Typical formulation

$$\hat{\boldsymbol{\theta}} \in \operatorname*{argmin}_{\boldsymbol{\theta} \in \Theta} L(data, \boldsymbol{\theta})$$

  - argmin because we want a minimiser not the minimum
    - Note: argmin can return a set (minimiser not always unique!)
  - Θ denotes a model family (including constraints)
  - L denotes some objective function to be optimised
    - E.g. MLE: (conditional) likelihood
    - E.g. Decision theory: (regularised) empirical risk

One we've seen: Log trick

- Instead of optimising $L(\theta)$, try convenient $\log L(\theta)$
- Why are we allowed to do this?
- Strictly monotonic function: a $> b \Longrightarrow f(a) > f(b)$
  - Example: log function!
- **Lemma**: Consider any objective function $L(\theta)$ and any strictly monotonic f. θ^* is an optimiser of $L(\theta)$ if and only if it is an optimiser of $f(L(\theta))$.
  - Proof: Try it at home for fun!

Two solution approaches

- Analytic (*aka* closed form) solution
  - Known only in limited number of cases
  - Use 1st-order necessary condition for optimality*:

$$\frac{\partial L}{\partial \theta_1} = \cdots = \frac{\partial L}{\partial \theta_p} = 0$$

Assuming unconstrained, differentiable *L*

- Approximate iterative solution
  1. Initialisation: choose starting guess $\boldsymbol{\theta}^{(1)}$, set $i = 1$
  2. Update: $\boldsymbol{\theta}^{(i+1)} \leftarrow SomeRule\left[\boldsymbol{\theta}^{(i)}\right]$, set $i \leftarrow i + 1$
  3. Termination: decide whether to Stop
  4. Go to Step 2
  5. Stop: return $\widehat{\boldsymbol{\theta}} \approx \boldsymbol{\theta}^{(i)}$

* Note: to check for local minimum, need positive 2nd derivative (or Hessian positive definite); this assumes unconstrained – in general need to also check boundaries. See also Lagrangian techniques later in subject.

Reminder: The gradient

- Gradient at $\boldsymbol{\theta}$ defined as $\left[\frac{\partial L}{\partial \theta_1}, \dots, \frac{\partial L}{\partial \theta_p}\right]'$ evaluated at $\boldsymbol{\theta}$
- The gradient points to the direction of maximal change of $L(\boldsymbol{\theta})$ when departing from point $\boldsymbol{\theta}$
- Shorthand notation
  * $\nabla L \stackrel{\text{def}}{=} \left[\frac{\partial L}{\partial \theta_1}, \dots, \frac{\partial L}{\partial \theta_p}\right]'$ computed at point $\boldsymbol{\theta}$
  * Here ∇ is the "nabla" symbol
- Hessian matrix at $\boldsymbol{\theta}$: $\nabla L_{ij} = \frac{\partial^2 L}{\partial \theta_i \partial \theta_j}$

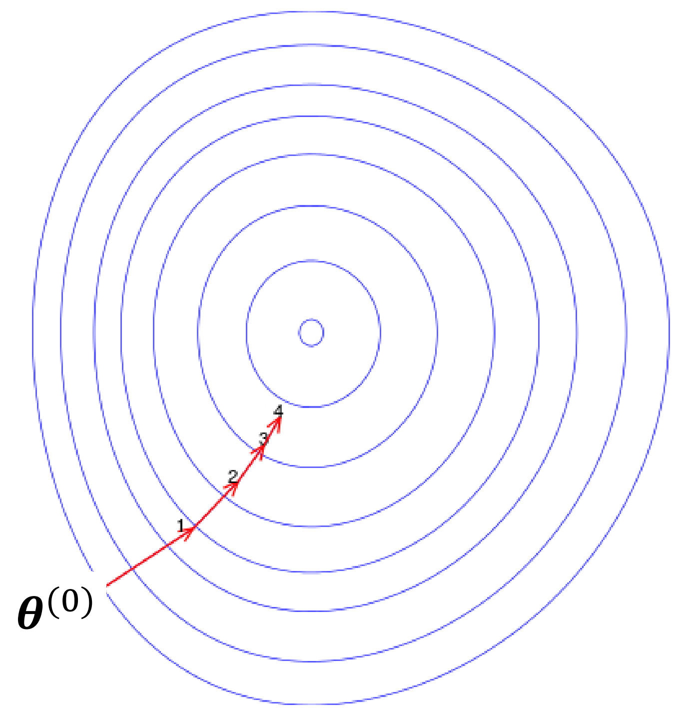
Harps, p. 984.

Gradient descent and SGD

Assuming L is differentiable

1. Choose $\boldsymbol{\theta}^{(1)}$ and some T
2. For i from 1 to T*
   1. $\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \eta \nabla L(\boldsymbol{\theta}^{(i)})$
3. Return $\widehat{\boldsymbol{\theta}} \approx \boldsymbol{\theta}^{(i)}$

- Note: η dynamically updated per step
- Variants: Momentum, AdaGrad, …
- Stochastic gradient descent: two loops
  - Outer for loop: each loop (called epoch) sweeps through all training data
  - Within each epoch, randomly shuffle training data; then for loop: do gradient steps only on batches of data. Batch size might be 1 or few

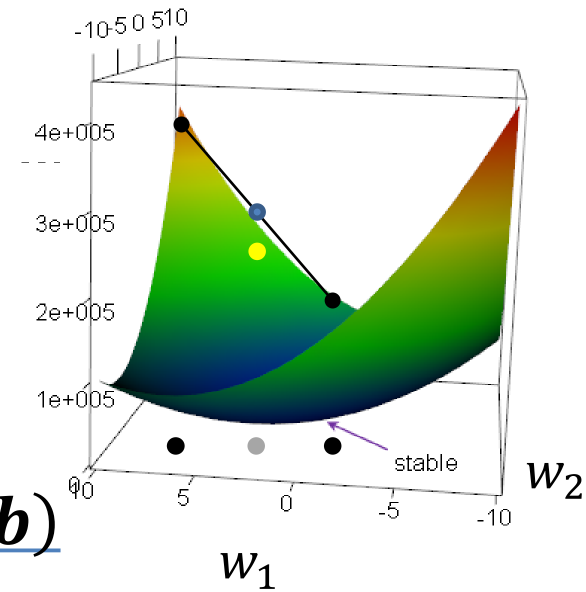


Wikimedia Commons. Authors: Olegalexandrov, Zerodamage

*Other stopping criteria can be used

Convex objective functions

- 'Bowl shaped' functions
- Informally: if line segment between any two points on graph of function lies above or on graph
- Formally* $f: D \to \mathbf{R}$ is convex
  if $\forall \boldsymbol{a}, \boldsymbol{b} \in D, t \in [0,1]$:
  $f(t\boldsymbol{a} + (1-t)\boldsymbol{b}) \leq tf(\boldsymbol{a}) + (1-t)f(\boldsymbol{b})$
  Strictly convex if inequality is strict (<)
- Gradient descent on (strictly) convex function guaranteed to find a (unique) global minimum!

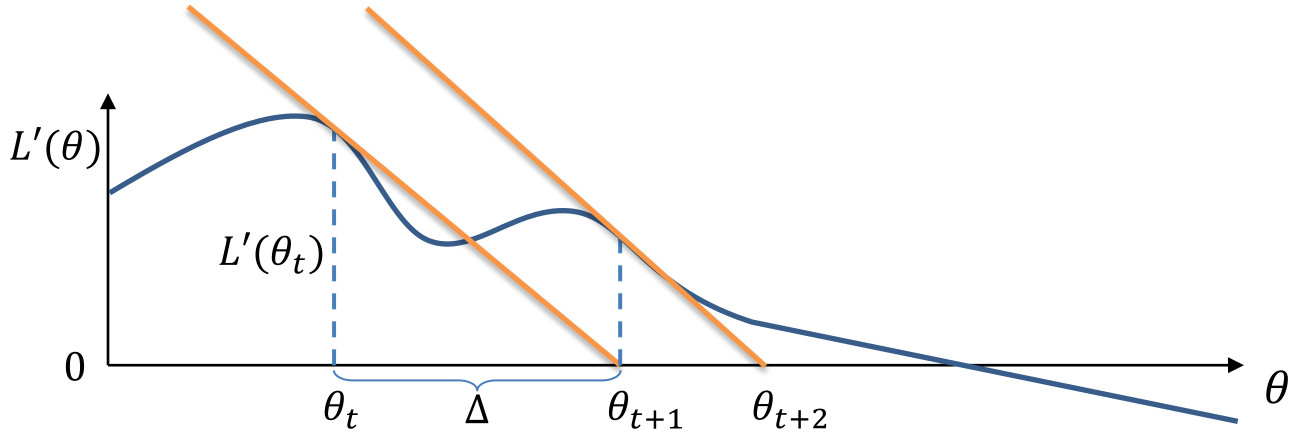


* Aside: Equivalently we can look to the second derivative. For *f* defined on scalars, it should be non-negative; for multivariate f, the Hessian matrix should be positive semi-definite (see linear algebra supplemental deck).

Newton-Raphson

A second-order method;
Successive root finding in the objective's derivative.

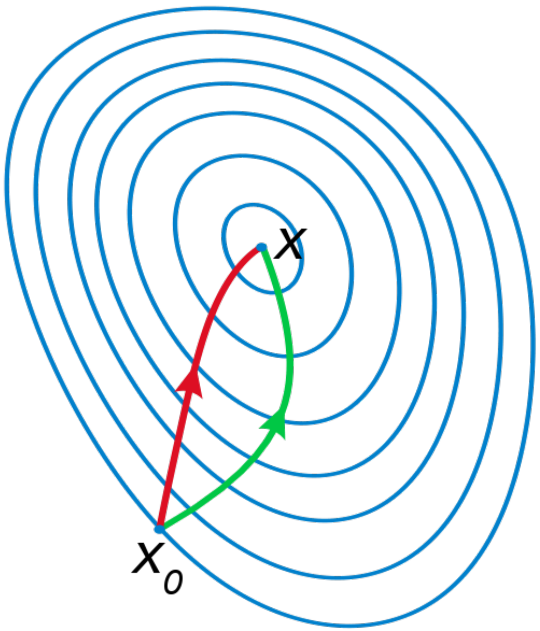
Newton-Raphson: Derivation (1D)



- Critical points of $L(\theta)$ = Zero-crossings of $L'(\theta)$
- Consider case of scalar θ. Starting at given/random θ_0, iteratively:
  1. Fit tangent line to $L'(\theta)$ at θ_t
  2. Need to find $\theta_{t+1} = \theta_t + \Delta$ using linear approximation's zero crossing
  3. Tangent line given by derivative: rise/run = $-L''(\theta_t) = L'(\theta_t)/\Delta$
  4. Therefore iterate is $\theta_{t+1} = \theta_t - L'(\theta_t)/L''(\theta_t)$

Newton-Raphson: General case

- Newton-Raphson summary
  - Finds $L'(\theta)$ zero-crossings
  - By successive linear approximations to $L'(\theta)$
  - Linear approximations involve derivative of $L'(\theta)$, ie. $L''(\theta)$
- Vector-valued θ:
  How to fix scalar $\theta_{t+1} = \theta_t - L'(\theta_t)/L''(\theta_t)$???
  - $L'(\theta)$ is $\nabla L(\theta)$
  - $L''(\theta)$ is $\nabla_2 L(\theta)$
  - Matrix division is matrix inversion

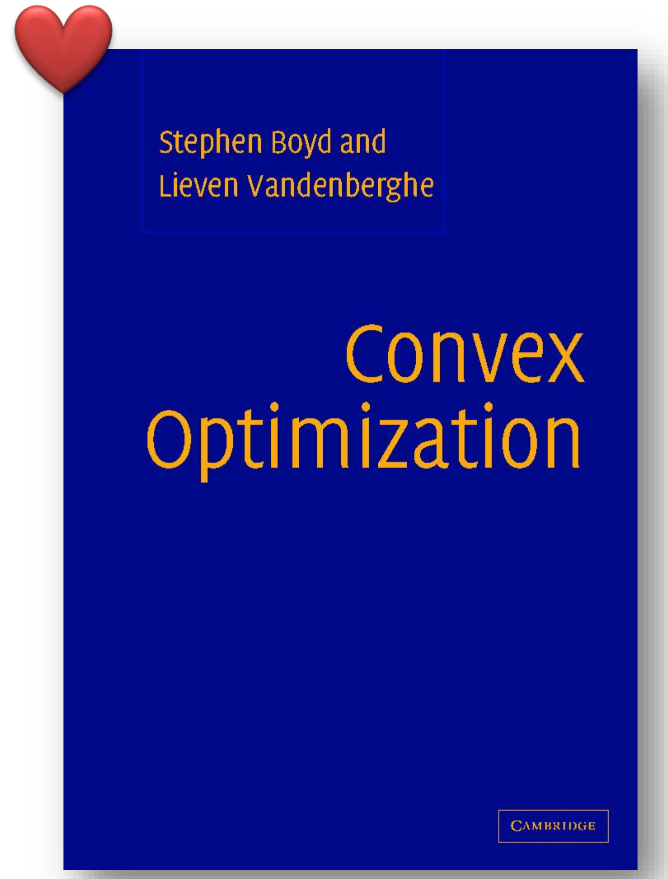


public domain wikipedia

- General case: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \left(\nabla_2 L(\boldsymbol{\theta}_t)\right)^{-1} \nabla L(\boldsymbol{\theta}_t)$
  - Pro: May converge faster; fitting a quadratic with curvature information
  - Con: Sometimes computationally expensive, unless approximating Hessian

...And much much more

- What if you have constraints?
  - See Lagrangian multipliers (let's you bring constraints into objective)
  - Or, projected gradient descent (you iterate between GD on objective, and GD on each constraints)
- What about speed of convergence?
- Do you really need differentiable objectives? (no, subgradients)
- Are there more tricks? (Hell yeah! But outside scope here)



Free at http://web.stanford.edu/~boyd/cvxbook/

Summary

- Iterative optimisation for ML
  - First-order: Gradient Descent and Stochastic GD
  - Convex objectives: Convergence to global optima
  - Second-order: Newton-Raphson can be faster, can be expensive to build/invert full Hessian

Next time: Logistic regression for binary classification