STAT 535

11/10/22

Lecture 13

- How to evaluate descent methods

Quiz 2 - today

HW 5 out

- difficult/easy objective J (or f) — ML / opt
- speed - how to evaluate
  - modern ML trade-offs
- SGD
- other acceleration methods for descent algorithms

Lecture IV: Training predictors, ~~Part I~~

Marina Meilă
mmp@stat.washington.edu

Department of Statistics
University of Washington

October, 2022

Optimization glossary

Change in notation!

Here, f is a function to be minimized, and x the variable in the domain of the function. In a learning task, f will be replaced by an objective J like $\hat{L}_{logl}$ and x by the parameters of the predictor, e.g. $w, \beta, \theta, \ldots$.

The methods in this lecture belong to the class of **unconstrained optimization** methods.
Problem Find $\min_x f(x)$ for $x \in \mathbb{R}^d$ or $x \in D$ the **domain** of f. We assume that f is a twice differentiable function with continuous second derivatives.
Notation The **gradient** of f is the column vector

$$\nabla f(x) = \left[\frac{\partial f}{\partial x_i}(x)\right]_{i=1}^{n} \tag{11}$$

and the **Hessian** of f is the square symmetric matrix of second partial derivatives of f

$$\nabla^2 f(x) = \left[\frac{\partial^2 f}{\partial x_i x_j}(x)\right]_{i,j=1}^{n} \tag{12}$$

Local and global minima

- A **local minimum** for f is point x^* for which

$$f(x^*) \leq f(x) \quad \text{whenever } ||x - x^*|| < \epsilon$$

- A **global minimum** for f is point x^* for which

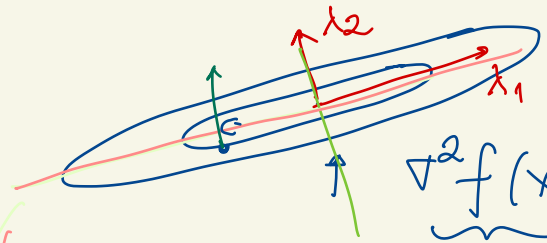
$$f(x^*) \leq f(x) \quad \text{for all } x \text{ in the domain of } f$$

  We say x^* is a **strict local/global minimum** when the above inequalities are strict for $x \neq x^*$.
- A minimum is **isolated** if it is the only local minimum in an ϵ-ball around itself.
- A **stationary point** for f is a point x^* for which $\nabla f(x^*) = 0$.

Theorem

If f has continuous second derivative everywhere in D, and $x^* \in D$ is a point for which $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*) \geq 0$ ($\nabla^2 f(x^*) > 0$) then x^* is a (nonsingular) local minimum for f.

In what follows, we will deal only with non-singular local minima. A non-singular local minimum is strict and isolated.



e-vectors of H : $\lambda_1 \ll \lambda_2$

$\nabla^2 f(x^*)$ ill conditioned $\Rightarrow$ $\frac{\lambda_{max}}{\lambda_{min}} = \kappa >> 1$

$\underbrace{\nabla^2 f(x^*)}_{H}$

condition number

1. **Hard** $\Leftrightarrow$ $\kappa(H) >> 1$

∇f grows slowly near x^*

$\Rightarrow \lambda_1$ small

$\nabla f(x)^T (x - x^*) \approx 0$

direction of descent — towards x^*

Not all f's are twice differentiable

By default we will assume f is twice differentiable and that its Hessian $\nabla^2 f$ is continuous and strictly positive definite around x^*. But other, weaker, conditions on f also indicate an f that is "easy" to minimize. Here are some of the most common ones

- f is B-**Lipschitz** if there is $B > 0$ so that

$$|f(y) - f(x)| \leq B||y - x|| \quad \text{for all } x, y$$

  A Lipschitz function, behaves "almost" like a differentiable function with bounded gradient.

- The problem $\min_x f$ is a **smooth** minimization problem if f it is upper bounded by a quadratic function around x^*, i.e. iff there exists $M > 0$ so that

$$f(x) - f(x^*) \leq \frac{1}{2} M||x - x^*||^2$$

  on a neighborhood of x^*. This property indicates that f, even though it may not be differentiable, behaves "almost like a quadratic", in the sense that local quantities (gradient, Hessian) are informative w.r.t the minimum. An example of a non-smooth minimization problem is $\min_x |x|$. The gradient ∇f is ± 1 everywhere but in 0, so it gives us information on which side of x the minimum lies, but its size does not tell us how far we are from x^*.

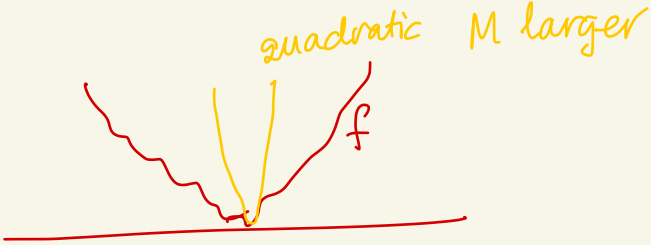
- A function f that is lower bounded by a quadratic function around x^* is called **strongly convex** (more about this later).

$$f(y) - f(x) - \nabla f(x)^T (y - x) \geq \frac{1}{2} m||x - y||^2$$

  If f is strongly convex, then the minimum x^* is well localized.

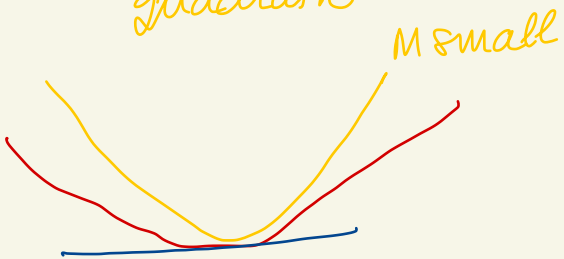
$\nabla f(x)$ discontinuous

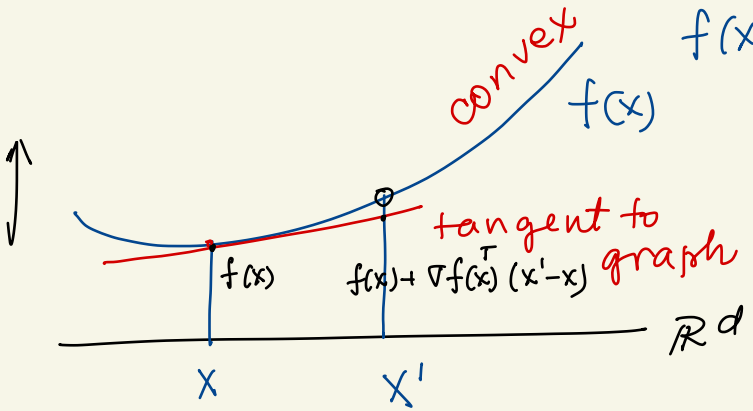
$\nexists \nabla f(x^*)$



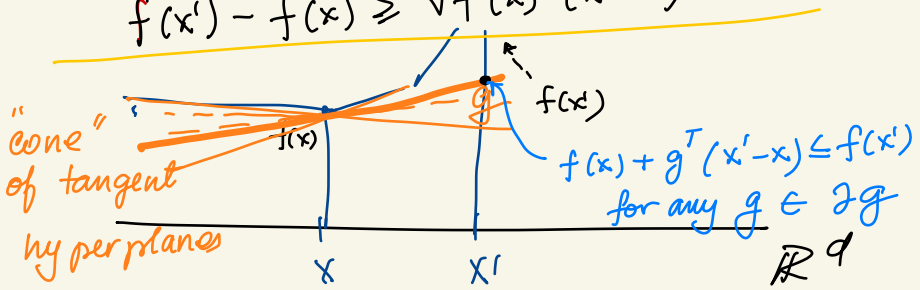
$f(x) \le \frac{1}{2} M \|x - x^*\|^2 \rightarrow \nabla^2 = M I_n$

quadratic





$f(x') - f(x) \ge \nabla f(x)^T (x' - x)$



$g \in \partial f$

set of tangent direction

convex :

$\frac{M}{m}$ = κ "condition number" for f

near x^*

Examples from Machine Learning

$\|\beta\|_1$ is 1-Lipschitz

- $f(x) = |x|$ is 1-Lipschitz
- If ∇f exists and $||\nabla f(x)|| \leq B$ for all x, then f is B-Lipschitz
- A C^2 function (continuous Hessian everywhere on the domain) with $\nabla^2 f(x^*) \succ 0$ is both strongly convex and smooth
- $\hat{L}_{01}$ the misclassification cost is piecewise constant. Hence, not differentiable everywhere, not strongly convex, but smooth. Very bad J to optimize, though.
- The LASSO cost function is not differentiable everywhere, it is not smooth but strongly convex. This is a very frequent in machine learning problems.

Taylor:

$$f(x) \approx f(x^*) + \frac{1}{2}(x - x^*)^T H (x - x^*) + \ldots$$

negligible near x^*

How to evaluate an optimization method? [Optional]

- Does it converge to a minimum?

As we shall see, all the methods described here converge to a minimum, but some of the require the function f to have additional "good" properties. A method which converges for larger classes of f's is more **robust**. tolerates non-differentiable larger B, M, smaller m

- How fast?

Answer is usually in terms of **rates of convergence**
Let $e^k = x^k - x^*$ or $e^k = f(x^k) - f(x^*)$ denote the "error" at step k. Then, an algorithm has a **rate of convergence of order** p if

$$||e^{k+1}|| \leq \beta(||e^k||)^p \quad \text{for some } 0 < \beta < 1 \tag{14}$$

$p=1 \Rightarrow ||e^{k+1}|| \leq \beta^k ||e^0||$

In the above, $p > 0$ but not necessarily an integer.
Most common cases are $p = 1$ (**linear**[1]) and $p = 2$ (quadratic). A rate of $p < 1$ is possible, and relevant for machine learning (see Part II). Superlinear scales are desirable – and often achievable.
Some modern machine learning algorithms have **sublinear** rates, e.g

$$||e^k|| \leq \frac{\beta}{k}||e^0|| \tag{15}$$

$\frac{1}{k}$ very slow

This is considered a slow convergence rate in classical optimization. Exercise Why may we like this rates in statistics/machine learning?

- Practical issues: Is it easy to implement or tune? Available software?

[1] The use of the term "linear" here is inconsistent with its use in e.g complexity theory. If an optimization algorithm is *linear*, that means that the error decreases **exponentially** with k, as $||e^k|| \leq \beta^k ||e^0||$.

Dichotomies ML ⟷ classical optimization

- $e = \|x - x^*\|$ small
  HARDER
- rate of convergence
  $p = 1$ almost too good

$e = f(x) - f(x^*)$ small

$p = 1$ barely good

$p = 2$

superlinear

$\frac{\|e^{k+1}\|}{\|e^k\|} \to 0$

- care more about transitory
  - cheap, fast, robust
    ↑ relie on few assumptions
- asymptotic regime
  - relie strongly on assumptions
    Newton, Conjugate Gradients

- objective

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} L(y^i, f(x^i)) + \lambda R(\theta)$$

not same f

prediction

↑ unit of cost for f or ∇f

- any f objective

computing f = unit of cost

(or ∇f)

Computational complexity – theory and practice

In optimization problems, there are various ways of expressing the computational complexity of an algorithm:

- number of flops (floating point operations) per iteration, usually as a function of n the dimension of the problem
- *number of function* or gradient evaluations per iteration
- *number of iterations*; this latter quantity is given implicitly, by the rate of convergence.
- *memory requirements* ~ high dim $\mathbb{R}^d$ d large
- With the increased complexity and variation of computer systems, the above mentioned number of operations is becoming obsolete. Algorithms are increasingly judged by other, system-related qualities, like: type of memory access (do they access memory in blocks or randomly), cache misses, etc. These criteria are beyond the scope of this course, but what you need to remember is that the texbook properties on an algorithm alone do not always predict its performance on the system you are going to run it. You may need to experiment with parameters and with algorithms to determine which algorithm is better suited for your data and system.

What's in a function evaluation?

In classical optimization, the computation of $f(x)$ or $\nabla f(x)$ is considered one unit of computation.
For machine learning, let us look inside the box. Take for example the LASSO objective function

$$J(\beta) = \sum_{i=1}^{n} (y^i - \beta^T x_i)^2 + \lambda ||\beta||_1.$$

We notice that

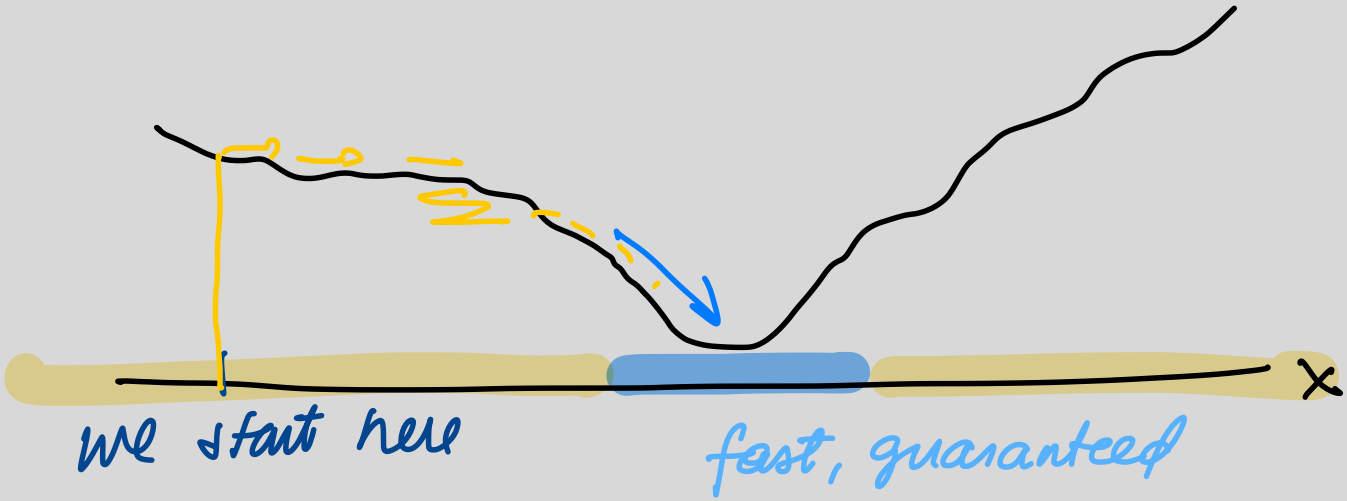
- J is a sum with $n+1$ terms
- each $x^i \in \mathbb{R}^d$, $\beta \in \mathbb{R}^d$ so each term takes $\mathcal{O}(n)$ operations (additions and multiplications) to compute
- Total number of operations to compute $J(\beta)$ once is $\mathcal{O}(nd)$
- Do this exercise Exercise The gradient ∇J is in $\mathbb{R}^d$ and is also a sum of $n+1$ terms. How many operations to compute it?

Transitory and asymptotic regimes

There are two regimes for each algorithm:

- the **transitory**, or approach regime, when x^k is far away from x^*
- the **asymptotic** regime, near x^* – most classic results are about this regime

The theoretical (and practical) behavior of optimization algorithms will strongly depend on the properties of the function f to be optimized around its minimum x^*.



Steepest (gradient) descent

- The steepest descent method follows the direction of the gradient.
- Rate of convergence: with line minimization or Armijo (see next slide), linear rate. For other line search methods, including constant step size, the rate of convergence is no larger.
- Ill-conditioning: The convergence coefficient β of equation (14) can get very close to 1 (very slow convergence) if the Hessian is ill conditioned.

Let M, m denote respectively the largest and the smallest eigenvalue of $\nabla^2 f(x^*)$. By continuity, we can assume that the Hessian around x^* is approximately the same. If $M >> m$ then the function will have a "long, narrow valley" with an almost flat "bottom" around x^*, oriented along the smallest eigenvector. The gradient will be almost perpendicular to the valley, and the algorithm, even with the optimal line minimization, will advance very slowly. Hence, many optimization methods (but not stochastic gradient) can be seen as "applying some coordinate transformation" that will turn the elongated ellipses into circles, so that steepest descent in this new coordinates can move rapidly towards the optimum. Equivalently, having such a transformation (which is represented by the Hessian matrix), one can apply the "inverse transformation" to the descent direction, which is precisely what the Newton-Raphson method does.