Optimization for Machine Learning

September 25, 2023

Today:
- Exercises on LPs/least squares
- Gradient descent (Pt 1)

Thursday
- Gradient descent (Pt 2)
- Lab session on gradient descent (laptops welcome!)

GRADIENT DESCENT

(≈ chapter 3 of the lecture notes)

↳ last lecture: linear programs, least squares

⇒ Problems that can be solved very efficiently using state-of-the-art solvers

least squares: CVXPY / SciPy

linear programs:

CPLEX / Gurobi (commercial)

CVXPY / HiGHS (academic)

* in closed form

↳ For most problems, we cannot characterize the solution explicitly* ⇒ we need to design numerical algorithms that compute solutions in an iterative fashion

↳ Classical general setup:

$$\underset{w \in \mathbb{R}^d}{\text{minimize}} \; f(w)$$

where f is C^1, (continuously differentiable)

In this course "$f \in C^1$" = at every $w \in \mathbb{R}^d$ we can compute $f(w)$ and $\nabla f(w) \in \mathbb{R}^d$

For C^1 functions, we know that any local minimum of the function has a zero gradient

$$[\bar{w} \in \mathbb{R}^d \text{ local minimum of } f] \Rightarrow [\|\nabla f(\bar{w})\| = 0]$$

$A \Rightarrow B$

$\neg B \Rightarrow \neg A$

Implication (becomes $\Leftrightarrow$ when f is convex)

$\nabla f(\bar{w}) = 0$

Idea:

→ If $\nabla f(w) \neq 0$ for some $w \in \mathbb{R}^d$, then w is certainly not a local minimum

($[\|\nabla f(\bar{w})\| \neq 0] \Rightarrow [\bar{w}$ not a local minimum of $f]$)

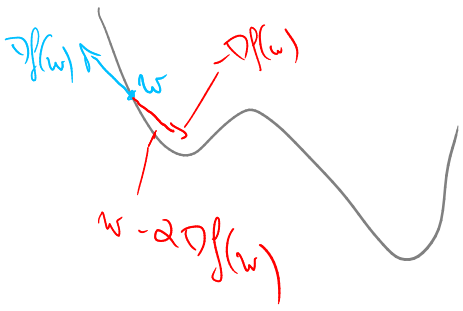
→ there must then exist a point $\hat{w} \in \mathbb{R}^d$ close to w such that $f(\hat{w}) < f(w)$

→ Key result:

If $\nabla f(w) \neq 0$, then $\exists \alpha > 0$ such that

$$f(w - \alpha \nabla f(w)) < f(w)$$

(can be shown thanks to the C^1 nature of f)



⇒ Basic idea behind gradient descent: Move along the negative gradient direction as long as the gradient is nonzero

Gradient descent algorithm (basic pseudo-code)

Initialization: Pick $w_0 \in \mathbb{R}^d$

For $k = 0, 1, 2, \ldots$

- Evaluate $\nabla f(w_k)$. If $\nabla f(w_k) = 0$, terminate.
- Compute a stepsize $\alpha_k > 0$.
- Set $w_{k+1} = w_k - \alpha_k \nabla f(w_k)$.

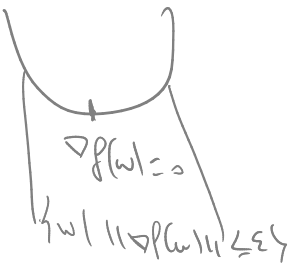
Gradient descent (GD) iteration

$w_{k+1} - w_k = -\alpha_k \nabla f(w_k)$: GD step

Implementation

↳ In practice, the for loop is replaced by a while loop with two conditions:

1) Convergence criterion $\|\nabla f(w_k)\| = 0$, $\|\nabla f(w_k)\| \leq \varepsilon$ for a small tolerance $\varepsilon > 0$



2) Budget criterion: • $k > K$ (fixed number of iterations),

- fixed number of gradient evaluations or function evaluations equal to the number of iterations (not necessarily)
- CPU time exceeded

Computing the stepsize α_k

three main categories

① Constant, predefined stepsize (quite popular in ML)

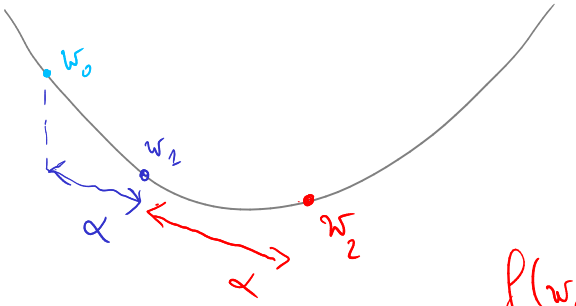
$\forall k \geq 0$, $\alpha_k = \alpha > 0$

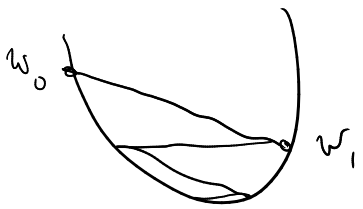
⊕ Simple, value is computed once before the algorithm starts

⊕ Can choose $\alpha > 0$ so that it is always large numerically (i.e. $\alpha = 10^{-3}$ than $\alpha = 10^{-50}$)

⊖ Often chosen independently of the function of GD

⊖ Always chosen independently of the iterates $(w_0, w_1, \dots)$





$f(w_2) > f(w_1)$

② Decreasing, predefined stepsizes

Define $\{\alpha_k\}_{k \geq 0}$ in advance and guarantee that $\alpha_k > 0$ and $\alpha_k \to 0$ $k \to \infty$

(ex) $\alpha_k = \frac{1}{k+1}$, $\alpha_k = \frac{1}{\sqrt{k+1}}$, $\alpha_k = \frac{1}{(k+1)^2}$, ...

⊕ This approach guarantees $f(w_{k+1}) < f(w_k)$ for sufficiently large k

⊕ Prevents divergence ($\|w_k\| \to \infty$) which can happen with a constant stepsize

⊖ The steps become increasingly small ($\|w_{k+1} - w_k\| \to 0$)

⊖ Typically chosen independently of f

(3) Adaptive stepsizes

(Really classical in optimization, not so much in ML)

↳ Idea: choose α_k according to w_k, $f(w_k)$ and $\nabla f(w_k)$

↳ Classical strategy: Line search

$$\alpha_k = \underset{\alpha > 0}{\operatorname{argmin}} \; f(w_k - \alpha \nabla f(w_k))$$

⇒ In practice, "Exact line search" usually exact line search is not doable

⇒ Exact minimization is replaced by an approximation, such as backtracking

Ex) Backtracking line search $[w_k, f(w_k), \nabla f(w_k)]$

- Start with $\alpha = 1$ → can use another value
- While $f(w_k - \alpha \nabla f(w_k)) \geq f(w_k) - c\alpha \|\nabla f(w_k)\|^2$ (forces the stepsize to improve the function value sufficiently)
  - $\alpha \leftarrow \alpha/2$ → can use any value between 0 and 1
- Return $\alpha_k = \alpha$.

$\alpha = 1$

$\alpha = 1/2$

$\alpha = 1/4$

⊕ Strategy adapted to every iterate

⊕ Can prove that it works

⊖ More expensive than the other strategies

⇒ Involves at least evaluations of f

⇒ Partly explains why this is not so popular in ML

Remark: Choosing a stepsize is a tradeoff between the cost of computing the stepsize and the quality of the corresponding step.

Remark: About GD and local minima

↳ GD stops when $\nabla f(w_k) = 0$ but w_k is not necessarily a minimum if f is not convex

↳ In practice, GD

usually converges to (local) minima

↳ In theory, very easy to construct examples on which it fails (converges to local maxima, saddle points, ...)

Theorem (Lee et al, 2015)

Choose $w_0 \in \mathbb{R}^d$ at random.

Then GD converges to a local minimum with probability 1. / almost surely.