OPTIMIZATION FOR MACHINE LEARNING

September 23, 2024

Today: Gradient descent (theory)

Next session: Lab on gradient descent (bring laptops!)

↑ Tomorrow September 24, 8.30 - 11.45
(Lab room TBA)

B048

GRADIENT DESCENT

Q: How do you solve an optimization problem in practice?

- solve: exactly? approximately?
- in practice: → on a computer → using an algorithm

Problem of interest

$$\min_{w \in \mathbb{R}^d} f(w)$$

$f : \mathbb{R}^d \to \mathbb{R}$ objective function

$w \in \mathbb{R}^d$ variables

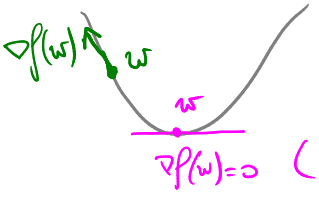
Assumptions:

- f is convex
- f is C^1

$\Downarrow$

$\forall w \in \mathbb{R}^d$, $\exists \nabla f(w) \in \mathbb{R}^d$ that indicates how the function changes around w

(f changes like a linear function of slope $\nabla f(w)$ around w)



(f locally flat around w^*)

Recall: optimality conditions for f C^1 convex

$$\left[\bar{w} \in \operatorname*{argmin}_{w \in \mathbb{R}^d} f(w) \right] \iff \left[\nabla f(\bar{w}) = 0_{\mathbb{R}^d} \right]$$

$$\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_d \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

argmin f: set of global minima of f

$f : \mathbb{R}^d \to \mathbb{R}$

$w \in \mathbb{R}^d \mapsto f(w) \in \mathbb{R}$

f C^1

$\nabla f : \mathbb{R}^d \to \mathbb{R}^d$

$w \mapsto \begin{bmatrix} \frac{\partial f}{\partial w_1}(w) \\ \vdots \\ \frac{\partial f}{\partial w_d}(w) \end{bmatrix} = \nabla f(w)$

↳ In general, cannot directly compute a vector $\bar{w} \in \mathbb{R}^d$ such that $\nabla f(\bar{w}) = 0_{\mathbb{R}^d}$

- Either there is no mathematical expression for the solution of the equation $\nabla f(\bar{w}) = 0_{\mathbb{R}^d}$ (e.g. deep neural networks)
- Or the solution is too expensive to compute in practice

Ex: Linear regression

$$\underset{w \in \mathbb{R}^d}{\text{minimize}} \quad \underbrace{\frac{1}{2n} \|Xw - y\|^2}_{f(w)} = \frac{1}{2n} \sum_{i=1}^{n} \left(x_i^T w - y_i\right)^2$$

Data: $\{(x_i, y_i)\}_{i=1..n}$ $\quad x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$

$$X = \begin{bmatrix} x_1^T \\ \vdots \\ x_n^T \end{bmatrix} \in \mathbb{R}^{n \times d}$$

n : (n)umber of data points

d : (d)imension of the parameter space

↳ On paper, we can give an explicit formula for the set of $\bar{w} \in \mathbb{R}^d$ such that $\nabla f(\bar{w}) = 0_{\mathbb{R}^d}$

↳ However, if $n \gg 1$, computing these expressions is not possible !

⇒ The large number of data points makes the calculations too expensive (inverse of a matrix in $\mathbb{R}^{n \times n}$, storage of such a matrix, ...)

↳ Instead of solving $\nabla f(\bar{w}) = 0_{\mathbb{R}^d}$ directly, we will build an algorithm that converges to a point for which $\nabla f(\bar{w}) = 0_{\mathbb{R}^d}$

⇒ Gradient descent

Idea: Given any $w \in \mathbb{R}^d$, compute $\nabla f(w) \in \mathbb{R}^d$

① If $\nabla f(w) = 0_{\mathbb{R}^d}$, then $w \in \operatorname{argmin}_{w \in \mathbb{R}^d} f(w)$ (f C^1 convex)

(problem solved!)

② If $\nabla f(w) \neq 0_{\mathbb{R}^d}$, use $\nabla f(w)$ to find a better point $w^+ \in \mathbb{R}^d$ such that $f(w^+) < f(w)$

$\Rightarrow$ Such a point necessarily exists!

$\Rightarrow$ Mathematically, one can show that there exists $\alpha > 0$ such that

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

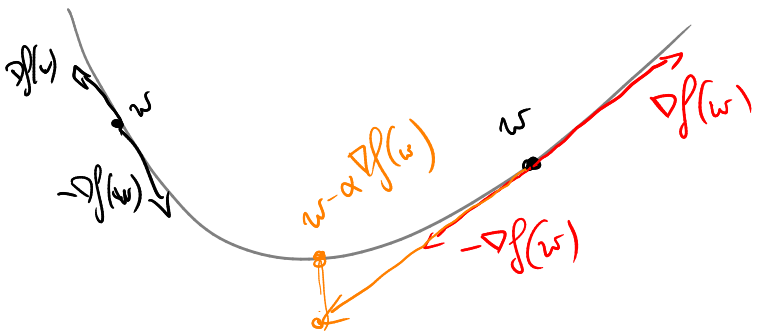
Not true for any $\alpha > 0$

opposite/negative gradient direction

When $\nabla f(w) \neq 0_{\mathbb{R}^d}$

$\nabla f(w)$: always defines a direction in which f increases

$-\nabla f(w)$ always defines a direction in which f decreases



Generic Gradient Descent algorithm

minimize $f(w)$ $w \in \mathbb{R}^d$ — f C^1 convex

Pseudo-code:

Initialization: Choose $w_0 \in \mathbb{R}^d$ (initial point), set $k = 0$ (iteration index)

Loop: For $k = 0, 1, 2, \ldots$ (infinite loop)

1. Compute $\nabla f(w_k) \in \mathbb{R}^d$
2. If $\nabla f(w_k) = 0_{\mathbb{R}^d}$, stop and return w_k. (stopping criterion)
3. Otherwise, compute $\alpha_k > 0$ (stepsize) and set $w_{k+1} = w_k - \alpha_k \nabla f(w_k)$ (Gradient Descent Update)

↳ The algorithm above is an infinite loop and a conceptual method

⇒ To implement it, we change the stopping criterion and we specify how to compute the stepsize

Stopping the algorithm (or making sure that it stops)

↳ Two ways to stop the algorithm

- Convergence criterion: Checks that the algorithm has computed a solution or a good enough approximation

Ex) $\nabla f(w_k) = 0_{\mathbb{R}^d}$? ($\Leftrightarrow \|\nabla f(w_k)\| = 0$?) (Exact CV criterion)

$\|v\| = \sqrt{\sum_{j=1}^{d} v_j^2}$

$\|\nabla f(w_k)\| \leq \varepsilon$ where $\varepsilon > 0$ is a small real number ($10^{-3}, 10^{-4}, \ldots$)

- Budget criterion: Guarantees that the algorithm will stop in a finite amount of time

Ex) . Do the for loop over $K \geq 1$ iterations.
=> Do at most K iterations / Compute at most K gradients

- Stop when $\|w_{k+1} - w_k\|$ is smaller than some tolerance (e.g. 10^{-8}, 10^{-12}, ...)
=> Algorithm is not changing the current point significantly

$\|w_{k+1} - w_k\| = \alpha_k \|\nabla f(w_k)\|$ can decrease faster than $\|\nabla f(w_k)\|$

(pick $\alpha_0 = 10^{-16}$, $\|\nabla f(w_0)\| = 1$, $\varepsilon = 10^{-6}$

$\|\nabla f(w_0)\| < 10^{-6}$ not satisfied

$\|w_1 - w_0\| = 10^{-16}$ should stop because this norm is of order of machine precision

- Stop after a CPU/GPU time limit is exceeded

=> In practice, the algorithm includes a convergence criterion and one or several budget criteria.

COMPUTING THE STEPSIZE α_k (aka the learning rate)

↳ Theory: If $\nabla f(w) \neq 0_{\mathbb{R}^d}$, $\exists \bar{\alpha} > 0$ such that

$$f(w - \alpha \nabla f(w)) < f(w) \quad \forall \alpha \in (0, \bar{\alpha}]$$

=> there exist infinitely many values that produce a decrease

↳ Empirically, there are 3 main strategies to choose the stepsize in the algorithm

a) Constant stepsize: $\alpha_k = \alpha > 0 \quad \forall k = 0, 1, \dots$

⊕ Easy to compute a priori

⊕ Actually works on some standard problems when α is well chosen

⊖ It is difficult to set a good value for α in general (α too large $\Rightarrow$ increase the function value)
(α too small $\Rightarrow$ numerical issues, little or no progress)

$\Rightarrow$ the method is very sensitive to the choice of α.

b) Decreasing stepsize

Idea: Define a priori a sequence $\{\alpha_k\}_k$ such that

$\alpha_k > 0$ and $\alpha_k \xrightarrow[k \to \infty]{} 0$ with $\alpha_0 \geq \alpha_1 \geq \alpha_2 \dots$

Classical choices: $\alpha_k = \frac{1}{k+1}$, $\alpha_k = \frac{1}{\sqrt{k+1}}$, $\alpha_k = \frac{\alpha_0}{k+1}$, $\alpha_0 > 0$

⊕ the theory guarantees that α_k is a good choice for k sufficiently large $\Rightarrow$ Eventually, the function values only decrease

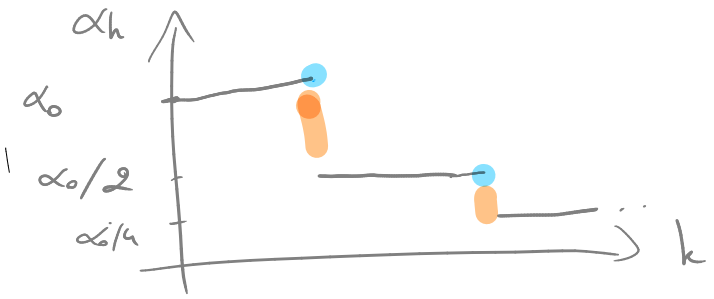
⊖ the stepsize can become unnecessarily small very quickly
$\Rightarrow$ No further progress

c) Hybrid stepsize approach

↳ Combination of constant/decreasing

↳ Use a constant value for α_k during a given number

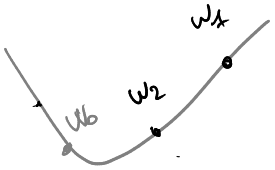
of iterations, then decrease the value and repeat



Implementation

- Define a rule on when to decrease α_k (often heuristic)
- Define how to decrease it (e.g. divide by 2 or 10)

A popular choice in ML: 0.1, 0.01, 0.001, ...



d) Adaptive stepsize

↳ Choose α_k according to $w_k, f(w_k), \nabla f(w_k)$

⇒ Goal: Find a good value adapted to the current point

↳ One example: Backtracking line search

$(w_k, \nabla f(w_k), f(w_k))$

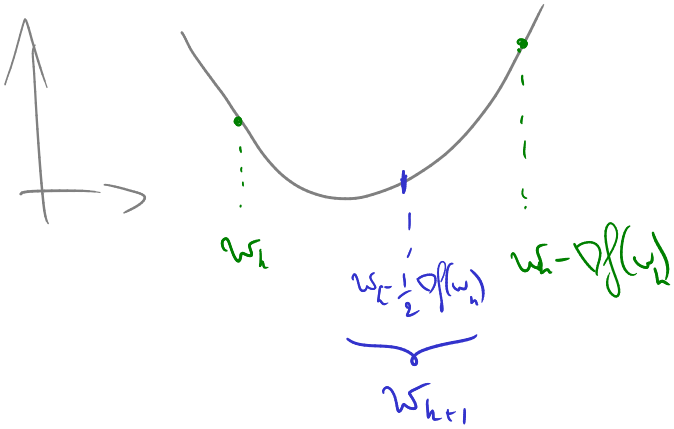
① Set $\alpha = 1$.

② If $f(w_k - \alpha \nabla f(w_k)) < f(w_k)$

Set $\alpha_k = \alpha$ and stop

Otherwise set $\alpha = \alpha/2$ and repeat ②.

NB: 1 and 1/2 are arbitrary values, they can be replaced by a positive value and a number in $(0,1)$



⊕ Theory guarantees that the backtracking line-search process will stop (with a good stepsize)

⊖ Extra cost in function values

(in the worst case: proportional to $-\log(\|\nabla f(w_k)\|)$)

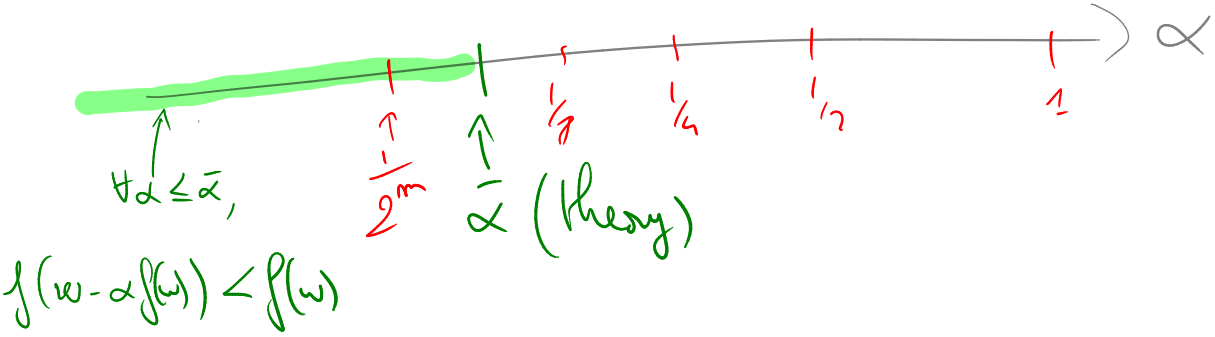
What is the best choice of α_k?

It depends!

- If function values are expensive (like in deep learning!), then predefined stepsizes (constant, decreasing, some hybrid) are preferable
- But if function values can be afforded, adaptive stepsizes give better results at the price of extra function value calculations

⚠ The gradient descent algorithm only computes gradient values and not function values in general

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



↳ For particular classes of problems (such as linear regression, logistic regression, linear SVM, ...), we can compute a constant value for α_k that guarantees decrease at every iteration and guarantees CV to an approximate solution in finite time

$C_L^{1,1}$ functions and gradient descent

Def: $f: \mathbb{R}^d \to \mathbb{R}$ is $C_L^{1,1}$ if it is C^1 and $\nabla f: \mathbb{R}^d \to \mathbb{R}^d$ is L-Lipschitz continuous for some $L > 0$, that is

$$\forall (v, w) \in (\mathbb{R}^d)^2, \quad \|\nabla f(v) - \nabla f(w)\| \leq L \|v - w\|$$

"The change in the gradients can be bounded by the change in the points."

Theorem Let $f: \mathbb{R}^d \to \mathbb{R}$ be $C_L^{1,1}$ and let $w \in \mathbb{R}^d$ such that $\nabla f(w) \neq 0_{\mathbb{R}^d}$. Then,

$$f\left(w - \frac{1}{L}\nabla f(w)\right) \leq f(w) - \frac{1}{2L}\|\nabla f(w)\|^2 < f(w)$$

Corollary: If f is $C_L^{1,1}$, we can implement gradient descent on f with $\alpha_k = \frac{1}{L}$ $\forall k$, and this will guarantee (without checking the function value!) that $f(w_{k+1}) < f(w_k)$ as long as $\nabla f(w_k) \neq 0_{\mathbb{R}^d}$

Corollary (Rate of convergence for gradient descent) — Gradient Descent

Let $f: \mathbb{R}^d \to \mathbb{R}$ be $C_L^{1,1}$ convex. Suppose that we run GD with $\alpha_k = \frac{1}{L}$ $\forall k$ starting with $w_0 \in \mathbb{R}^d$. Then, for any $K \geq 1$,

$$f(w_K) - \min_{w \in \mathbb{R}^d} f(w) \leq O\left(\frac{1}{K}\right)$$

w_K: iterate after K iterations

$\min_{w \in \mathbb{R}^d} f(w)$: best possible value for f

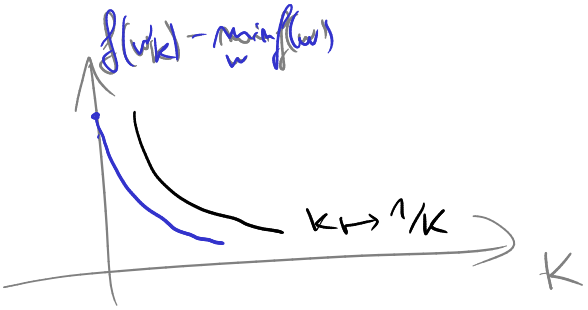
$O\left(\frac{1}{K}\right)$: "Big-O of $\left(\frac{1}{K}\right)$"

where $O\left(\frac{1}{k}\right) = \text{constant} \times \frac{1}{k}$, where the constant is positive and does not depend on k

(Here the constant depends on w_0, L, $\operatorname{argmin}_{w \in \mathbb{R}^d} f(w)$)

$\frac{L}{2} \max_{w^* \in \operatorname{argmin} f} \|w_0 - w^*\|$

$\Rightarrow$ We say that GD converges at the rate of $\frac{1}{k}$ for $C^{1,1}_L$ and convex functions



Equivalent result: Complexity bound

Under the same assumptions as the corollary, let $\varepsilon > 0$.

Then GD computes w_k such that $f(w_k) - \min_{w \in \mathbb{R}^d} f(w) \leq \varepsilon$ in at most $O\left(\frac{1}{\varepsilon}\right)$ iterations.