Elements of Calculus

Prof. Sören Petrat, Constructor University

Lecture notes from Spring 2025

Session 24
(Week 12B)

6. Multivariable Calculus

6.2 Optimization in Many Variables

Topic for Week 12B: Least Squares and Gradient Descent

Let us make another connection to the Elements of Linear Algebra class from last semester.

There, in Week 12B, we considered the Least Square problem:

Given an $m \times n$ matrix A with $m > n$ and a vector $b \in \mathbb{R}^m$, find $x \in \mathbb{R}^n$ that minimizes $\|Ax-b\|$.

In Linear Algebra, we used the QR decomposition to solve this.

Now, let us use Calculus to solve this problem.

Let us define $f(x) = \|Ax-b\|^2$ which has the same minimum as $\|Ax-b\|$.

We have $f(x) = (Ax-b)^T(Ax-b) = (x^TA^T - b^T)(Ax-b)$

(recall: $\|x\|^2 = x^Tx$; A^T = transpose of A; recall $(AB)^T = B^TA^T$)

$$= x^TA^TAx - x^TA^Tb - b^TAx + b^Tb$$

$b^TAx = x^T(b^TA)^T = x^TA^Tb$

$$= x^TA^TAx - 2x^TA^Tb + b^Tb$$

$$= \sum_{j,\ell} x_j (A^TA)_{j\ell} x_\ell - 2\sum_j x_j (A^Tb)_j + b^Tb$$

Minimization: $0 = (\nabla f)(x)$, i.e., $\frac{\partial f}{\partial x_i} = 0 \quad \forall i = 1, \dots, n$.

$$\Rightarrow \quad \frac{\partial f}{\partial x_i} = \sum_e (A^TA)_{ie} x_e + \sum_j \underbrace{x_j (A^TA)_{ji}}_{= (A^TA)^T_{ij} x_j = (A^TA)_{ij} x_j} - 2(A^Tb)_i$$

$$\Rightarrow 0 = \nabla f = 2A^TAx - 2A^Tb \quad \Rightarrow \quad A^TAx = A^Tb$$

$\Rightarrow$ The minimum occurs when $A^TAx = A^Tb$ (which is an $n \times n$ system of linear equations that still needs to be solved)

Note: Here our function was very simple, hence we could do the minimization exactly.

But in many applications, especially in Machine learning, we cannot find minima exactly. Hence we need numerical algorithms.

Let us briefly discuss the important method of **gradient descent**. The goal is to numerically find minima of $f: \mathbb{R}^n \to \mathbb{R}$.

We start from our observation that ∇f points in the direction of the strongest increase. Hence we could use the following

Algorithm:

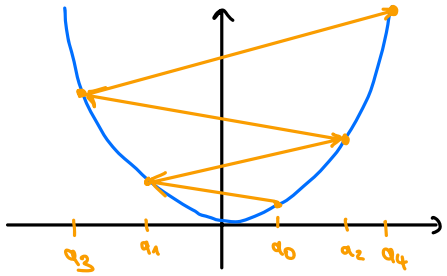
- Choose an initial point $a_0 \in \mathbb{R}^n$.
- Choose a step size $\eta > 0$ and go a step η in the direction of $-\nabla f$ (steepest descent):
  $a_1 = a_0 - \eta (\nabla f)(a_0)$
- Repeat: $a_{n+1} = a_n - \eta (\nabla f)(a_n)$
- Stop when $\|(\nabla f)(a_n)\| < \varepsilon$ for some tolerance ε.

Note: In the context of Machine learning, the step size η is called "learning rate".

Applications and implementations of this method are discussed in other classes. Here, let us discuss some potential problems and generalizations.

Potential problems:

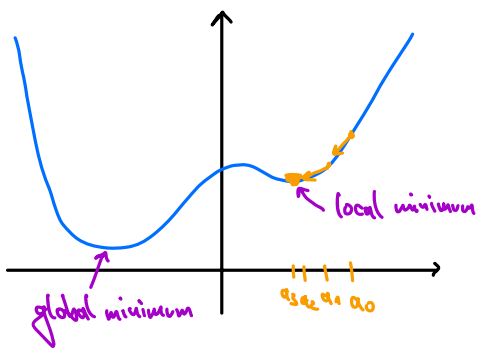
- Learning rate: • If η is chosen too large, we might not find the minimum, or the algorithm might even diverge.



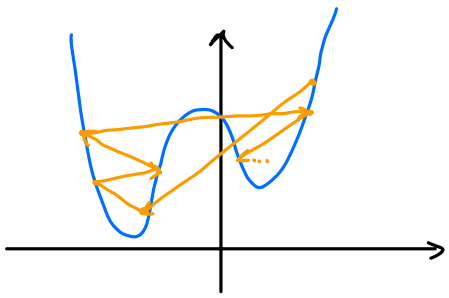
"overshooting"

• If η is chosen too small, the convergence might be very slow.

- Local vs. global minima: The algorithm might get stuck in a local minimum instead of a global minimum, especially if the learning rate is chosen too small.



- Related unlucky situations:



Generalizations:

- Making the learning rate n-dependent: For example, decreasing the learning rate with increasing number of iterations mitigates the risk of overshooting the minimum. E.g., $\eta_n = \frac{1}{1+n}$

- Stochastic gradient descent: Often, $f(x) = \frac{1}{N}\sum_{i=1}^{N} f_i(x)$ with N large, so evaluating ∇f is computationally expensive. Here, instead of using the iteration $a_{n+1} = a_n - \eta \frac{1}{N}\sum_{i=1}^{N} (\nabla f_i)(a_n)$, we choose one i randomly in each step and perform the iteration $a_{n+1} = a_n - \eta (\nabla f_i)(a_n)$.

  Or we could take an average over a small subset $M \subset \{1,\dots,N\}$: $\frac{1}{|M|}\sum_{j\in M}(\nabla f_j)(a_n)$

  Advantages: · computationally cheap
  · might be able to escape local minima

  Disadvantages: · might need more steps to reach minimum
  · might "jump around" near an actual minimum