Optimization: Algorithms, Complexity & Approximations

Anastasios Kyrillidis [*]

[*]Instructor, Computer Science at Rice University

Contributors: Nick Sapoval, Carlos Quintero Pena, Delaram Pirhayatifard, McKell Stauffer, Mohammad Taha Toghani, Senthil Rajasekaran, Gaurav Gupta, Pranay Mittal, Yi Lin, Shawn Fan, Hannah Lei, Erin Liu, Bohan Wu

Chapter 1

This chapter introduces optimization through data science and machine learning applications. We discuss some of the places optimization appears, and gradually introduce the reasoning that will lead to the well-known definitions and assumptions usually made in optimization. We will fix the mathematical notation of most of the optimization problems we will discuss in this course, and introduce the notion of *Black Box* in optimization. This chapter will conclude with a primer on linear algebra and the basic numerical analysis operations needed for this course.

What are these notes about | some optimization examples | Black Box oracle | linear algebra primer

Notation. We obey the following notation: a p-dimensional vector x (*we will assume the real case for most of the course, unless otherwise stated*) is denoted by

$$x = (x_1, x_2, \dots, x_p)^\top \in \mathbb{R}^p.$$

A variant often found in the literature uses brackets $[\cdot]$ instead of parenthesis, and it is considered equivalent, based on the context. With a slight abuse of notation, we use plain lowercase letters for both scalars and vectors, but the distinction should be clear from the context, or stated explicitly. The notation $\cdot^\top$ denotes the transpose operation, that translates a column vector into a row vector, and vice versa. Throughout the course, we might use: *i)* x_i to denote the i-th entry of a vector, *ii)* x_i to denote the putative solution of an iterative method at the i-th iteration, or *iii)* x_i to denote the i-th column of a matrix. The distinction should be clear again from the context.

A general optimization criterion is described as follows [1–5]:

$$\begin{aligned} \min_{x \in \mathcal{C} \subseteq \mathbb{R}^p} \quad & f(x) \\ \text{subject to} \quad & g(x) \leq 0. \end{aligned}$$

Here, $f : \mathbb{R}^p \rightarrow \mathbb{R}$ is the objective criterion, $g : \mathbb{R}^p \rightarrow \mathbb{R}$ is a function that represents some of the constraints, and $\mathcal{C} \subseteq \mathbb{R}^p$ is a restriction on the values that the solution can take. Usually, $\mathcal{C}$ is defined such as:

$$\mathcal{C} = \{x \in \mathbb{R}^p \mid \text{further conditions on } x\}.$$

We generally use calligraphic uppercase letters to denote sets; e.g., $\mathcal{C}, \mathcal{S}, \dots$ One can argue that we can also include g in the description of $\mathcal{C}$, i.e., $\mathcal{C} = \{x \in \mathbb{R}^p \mid \text{further conditions on } x,\ g(x) \leq 0\}$, but we often include such additional constraints in the description, especially when we can handle them in a specific/clever way to approximate the solution for this objective. In any case, both descriptions are equivalent.

The set of x's that satisfy the intersection of $g(x) \leq 0$ and $\mathcal{C}$ constitutes the *feasible set*. *Finding the solution $x^\star$ that minimizes the objective, while belonging to the feasibility set, is the task of optimization.*

As it will be an important description for several problems in this course, we also describe a matrix version as an optimization criterion. A $p \times d$ matrix X is denoted as:

$$X = \begin{bmatrix} X_{11} & X_{12} & \dots & X_{1d} \\ X_{21} & X_{22} & \dots & X_{2d} \\ \vdots & & \ddots & \vdots \\ X_{p1} & X_{p2} & \dots & X_{pd} \end{bmatrix} \in \mathbb{R}^{p \times d}.$$

We will generally use uppercase plain letters for matrices, unless otherwise stated (*E.g., a common use of an uppercase letter as a non-matrix variable is as a universal constant, say C or the Lipschitz gradient continuity constant L*). Then, after changes in the domain of f, g, such that $f : \mathbb{R}^{p \times d} \rightarrow \mathbb{R}$ and $g : \mathbb{R}^{p \times d} \rightarrow \mathbb{R}$, we have a matrix-variable optimization problem as in:

$$\begin{aligned} \min_{X \in \mathcal{C} \subseteq \mathbb{R}^{p \times d}} \quad & f(X) \\ \text{subject to} \quad & g(X) \leq 0. \end{aligned}$$

Types of optimization.

- *Constrained optimization*: whenever any constraints on x are present, like in the description above.
- *Unconstrained optimization*: no constraints; this means that the problems above look like:

$$\min_{x \in \mathbb{R}^p} \quad f(x) \qquad \text{or} \qquad \min_{X \in \mathbb{R}^{p \times d}} \quad f(X)$$

In this course, we will consider only problems with non-empty feasibility set.

Types of solutions. A key distinction between non-trivial solutions to an optimization criterion is:

- *Global solution*: we usually denote the global optimal solution with $x^\star$. $x^\star$ has the property that $f(x^\star) \leq f(x)$, for any other x in the feasibility set. (*Note that there might be other x's that get even smaller objective value, but they do not satisfy the constraint set.*)
- *Local solution*: let us use here the notation $\bar{x}$ for a local solution. Then, $\bar{x}$ satisfies $f(\bar{x}) \leq f(x^\star)$. (*For the moment, the way we define the local solutions is arbitrary and contains all the non-global points in the feasibility set; and this is enough for now. Distinction between local minimum/maximum, saddle points, etc. will be provided later during the course*).

Some examples where optimization is used. Let us describe some classical and some less classical problems, that are using optimization for their solution.

Linear regression a.k.a. least squares. [6–8] Let $A \in \mathbb{R}^{n \times p}$ be a given matrix such that:

$$A = \begin{bmatrix} — & a_1^\top & — \\ — & a_2^\top & — \\ & \vdots & \\ — & a_n^\top & —. \end{bmatrix}$$

Here, n denotes the number of samples/data points we have, and p is the number of features for the problem at hand. We also have some observations $y \in \mathbb{R}^n$. Assume that we know that the predictor $f(x, a_i) \equiv f_i(x) = a_i^\top x$ is the right model to fit the data. Then, the goal is to find the vector x that minimizes the ℓ_2-norm discrepancy between the observations y_i and the prediction $f_i(x)$ as in:[1]

$$\min_{x \in \mathbb{R}^p} \quad \left\{ \tfrac{1}{n} \sum_{i=1}^n (y_i - f_i(x))^2 = \tfrac{1}{n} \sum_{i=1}^n \left(y_i - a_i^\top x\right)^2 \right\}$$

The above objective is known as the linear regression objective, but also as the least squares objective. *(Definitions of inner products are provided in the linear algebra primer that follows.)*

Quantum state tomography, a.k.a., least-squares over matrices with low-rank constraints. [9] Assume we have a quantum computer in our possession. A quantum computer is generally described by its *quantum state*. I.e., all approximations and errors put aside, a quantum computer "evolves" its quantum state from an initial state (initialization) to the final state (output of the algorithm). That quantum state can be represented as a density matrix:

$$X_t^\star \in \mathbb{C}^{d \times d} \quad \text{such that } X_t^\star \succeq 0,$$

where the subscript t represents the time index. (*In the quantum information notation, densities are represented as* ρ_t). Observe that we work in the complex domain. What a quantum algorithm does then is, through a series of operations (= a sequence of quantum gate applications), to produce $X_0^\star \to X_1^\star \to \cdots \to X_T^\star$, such that $X_T^\star$ somehow contains the answer to a problem.

Bringing back errors, this sequence of density matrices includes noise, often magnified from step to step. Thus, even if we perform the first step $X_0^\star \to X_1^\star$, we are not sure whether the state of the quantum computer is actually (or even approximately) $X_1^\star$. One of the ways to test the validity of this step is through *tomography*: we obtain tomographic measurements by applying some special structured matrices on (the assumed to be) $X_1^\star$, and from these observations, we recover the best matrix that fits the measurements. If that matrix, say $\widehat{X}_1$, is really close to $X_1^\star$, then we are confident that this step is performed as expected.

In math, this translates into forming a set of matrices $A_i \in \mathbb{C}^{d \times d}$ that will lead to the set of tomographic measurements. E.g., assume that we take measurements from a state $X^\star$; the measurements look like:

$$y_i = \text{Tr}(A_i X^\star) + \varepsilon_i.$$

Here, we observe $X^\star$ indirectly through $\{y_i, A_i\}_{i=1}^n$ measurements, that are contaminated with noise ε_i.

Now, given $\{y_i, A_i\}_{i=1}^n$, how do we recover $X^\star$? If we have enough measurements, maybe it is sufficient to solve just a matrix version of the least squares problem:

$$\min_{X \in \mathbb{R}^{d \times d}} \quad \left\{ \tfrac{1}{n} \sum_{i=1}^n (y_i - f_i(X))^2 = \tfrac{1}{n} \sum_{i=1}^n (y_i - \text{Tr}(A_i X))^2 \right\}$$

However, we know more about state $X^\star$. By its physical composition, $X^\star$ has unit trace; i.e., $\text{Tr}(X^\star) = 1$. We could include this information in the optimization problem:

$$\begin{aligned} \min_{X \in \mathbb{R}^{d \times d}} \quad & \tfrac{1}{n} \sum_{i=1}^n (y_i - \text{Tr}(A_i X))^2 \\ \text{subject to} \quad & \text{Tr}(X) = 1. \end{aligned}$$

But, even then, solving such a problem requires a fully complete set of observations; i.e., we require the number of measurements n to be of the order of $O(d^2)$ in order to solve such a problem. Obtaining such a set of measurements is often infeasible: d is connected exponentially to the number of qubits of the system, $d = 2^q$, where q is the number of qubits. For a fairly small number $q = 20$, the number of measurements become enormous to obtain, store and process. *Is there a different way to overcome such a difficulty?* As we will see in later chapters, there is, through low-rank matrix recovery procedures.

Fleet allocation for EMS services; a more data science engineered objective. [10] Theoretical work on the topic of strategic vehicle allocation for both fire and Emergency Medical Service departments enjoys a rich and diverse history. The idea is to perform optimal long-term vehicle allocation and location, and most models are formulated as constrained optimization problems. (*You can skip the following paragraphs till the problem formulation if you are not interested in the reasoning behind building an optimization criterion*).

These problems attempt to maximize one performance dimension of the vehicle response system, while subjecting vehicle locations to a set of constraints representative of real-life operating characteristics. Of these optimization models, here we will describe how to *maximize the number of incidents covered* by emergency vehicles.

Appropriately, these models are referred to as "covering models". Notably, these covering models primarily make use of integer constraints (i.e. the constraints can be either 0 or 1 in value) and linear or quadratic objective functions to increase the simplicity with which solutions to the problem formulations are obtained.

Briefly, all covering models describe the spatial location of incidents as well as the vehicles and their locations on a directed graph $\mathcal{G} = (\mathcal{V}, \ \mathcal{W}, \ \mathcal{E})$ where $\mathcal{V}$ (indexed by i) consists of all the demand points (summarizing all the incidents in a dataset), $\mathcal{W}$ consists of the locations of vehicle stations (indexed by j), and $\mathcal{E}$ refers to the set of edges (representing routes) between every demand point and every vehicle station. Often these problem formulations discretize the demand into distinct "demand" points which are taken to be summaries of the overall demand in order to tractably arrive at a solution. Associated with each of these demand points in $\mathcal{V}$ is a quantity, d_i which represents the *magnitude* of the demand associated with the demand point. x_j is an integer scalar representing the number of vehicles located at station j, while y_i is a binary variable equal to 1 if and only if demand point i is covered *at least once*.

These problems also require several other inputs in addition to the base variables. Each edge in the graph $e_{i,j}$ is associated

[1] A small comment on notation compromises: Different research areas follow a different notation convention. E.g., in optimization, the optimization variable is denoted as x; in statistics, it is often denoted as β; often in machine learning, we use w to denote the set of variables = weights of a neural network. In signal processing, for the set of features in the least squares objective, researchers often use A or Φ, while in statistics and optimization, we commonly use X to represent the set of features or the design matrix. Error terms, like additive noise in observations, can be represented as w, or n or ε. In all cases though, as long as the notation is consistent, the work should not be judged by the selection of letters used—this is a matter of personal taste at the end. Thus, it is recommended to readers to re-wire their knowledge around concepts (e.g., this is the set of features, irrespective of which letter used).

with a weight, $t_{i,j}$ which denotes the time it would take to travel from station j to a demand point i. In these formulations, r refers to the user-defined "response time threshold" used to determine whether a demand point is "covered" or not. Lastly, the input quantity p is a number which refers to the total number of available vehicles, while p_j refers to the maximum vehicle "capacity" of each station, and $\mathcal{W}_i$ describes a set which consists of the set of vehicle which cover a given demand point i.

The MEXCLP [10] problem formulation is one of the most versatile covering model problem formulation, combining a reasonable degree of simplicity and realistic constraint setting. It is an explicitly probabilistic model, incorporating a new parameter termed the "busy fraction" representing the probability that a given vehicle is not available to respond to a call, despite the call being "covered" by the vehicle in question; denote that parameter as q. Its formulation is as follows:

$$\begin{aligned}
\min_{x\in\{0,1\}^m, y\in\{0,1\}} \quad & \left\{ f(y) = \sum_{i\in\mathcal{V}} \sum_{k=1}^{p} d_i (1-q) q^{k-1} y_{ik} \right\} \\
\text{subject to} \quad & \sum_{j\in\mathcal{W}_i} x_j \geq \sum_{k=1}^{p} y_{ik}, i \in \mathcal{V}, \\
& \sum_{j\in\mathcal{W}} x_j = p, \\
& x_j \leq p_j
\end{aligned}$$

The above problem formulation shows the versatility of optimization criterions: Given a problem description, we can have continuous variables but also discrete or integer variables; we can have equality and inequality constraints; we can have one constraint or multiple constraints, etc. *In this class, topics like integer programming/discrete programming are out of scope.*

Training a neural network classifier. [11] Consider a problem where we are given a set of n input data $\{x_i\}_{i=1}^n$, with corresponding labels y_i. To make our discussion concrete, consider that each input data point x_i is a 20-dimensional flattened image of size 5×4, with corresponding label y_i belonging to one out of 10 classes. Thus, each y_i can be represented as a one-hot vector such that $y_i \in \{0,1\}^{10}$, with only one entry of y_i being 1 at the correct class.

Assume we are certain that the following neural network architecture is sufficient to train such a classifier.

- The input layer accepts vectors in $\mathbb{R}^{20}$.
- Each input data is transformed via a weight matrix $W_1 \in \mathbb{R}^{12\times 20}$ such that $\bar{h}_1 = W_1 x_i \in \mathbb{R}^{12}$.
- $\bar{h}_1$ goes through a non-linear activation function, say $\sigma : \mathbb{R}^{12} \to \mathbb{R}^{12}$, that operates in an entrywise fashion. This leads to $h_1 = \sigma(\bar{h}_1) \in \mathbb{R}^{12}$.
- Going into the second hidden layer, h_1 is further transformed by another weight matrix $W_2 \in \mathbb{R}^{10\times 12}$ such that $\bar{h}_2 = W_2 h_1 \in \mathbb{R}^{10}$.
- $\bar{h}_2$ goes through a non-linear activation function, usually the same as in the previous layer. Thus: $h_2 = \sigma(\bar{h}_2) \in \mathbb{R}^{10}$.
- Going into the third hidden layer, h_2 is further transformed by another weight matrix $W_3 \in \mathbb{R}^{10\times 10}$ such that $\bar{h}_3 = W_3 h_2 \in \mathbb{R}^{10}$.
- $\bar{h}_3$ goes through a non-linear activation function, usually the same as in the previous layer. Thus: $h_3 = \sigma(\bar{h}_3) \in \mathbb{R}^{10}$.
- Finally, h_3 is normalized according to the softmax layer in order to represent a probability distribution. That is, the output $\widehat{y}_i$, corresponding to the input x_i, is a 10-dimensional vector with entries: $(y_i)_j = \frac{e^{(h_3)_j}}{\sum_\ell e^{(h_3)_\ell}}$.

The above can be depicted in the following neural network illustration.

In math, the above can be described as:

$$\widehat{y}_i = \texttt{softmax}\left(\sigma\left(W_3 \cdot \sigma\left(W_2 \cdot \sigma\left(W_1 \cdot x_i\right)\right)\right)\right),$$

where W_i are the trainable variables we want to optimize.

A (not-that-natural) way to measure the discrepancy between the trained output $\widehat{y}_i$ and the actual one-hot vector y_i is via

$$\mathcal{L}\left(\widehat{y}_i, y_i\right) := \left(\widehat{y}_i - y_i\right)^2.$$

Using all the data we have, our goal is to learn W_i's via:[2]

$$\min_{W_i} \quad f(W_1, W_2, W_3) := \tfrac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(\widehat{y}_i, y_i\right).$$

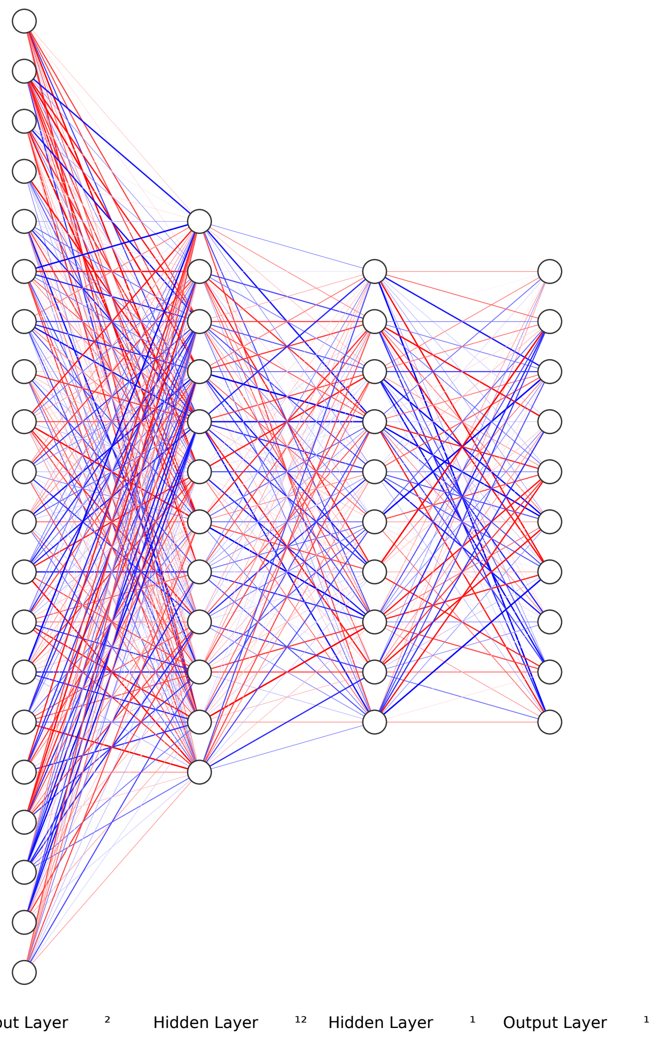

Fig. 1. An illustration of the artificial neural network.

[2] The interesting part of modern machine learning, in terms of optimization, is that we no longer care about finding the minimum of the objective criterion at hand, but rather find a solution that will behave nicely in a different objective function, which we do not have access to. While this seems an "unfair" requirement, at the same time opens new research directions, e.g., how to do *indirect* optimization.

A pessimistic view on optimization. Borrowing from Nesterov's book [2], one of the first bold statements made is the following:

"In general, optimization problems should be UNSOLVABLE"

Nevertheless, we use optimization in almost all aspects of technology. What is the caveat here? We need to define at what level we are comfortable to accept an approximate solution as the "optimal" solution. For linear regression, when is a solution $\widehat{x}$ considered optimal? E.g., would $\|\widehat{x} - x^\star\|_2 \leq \varepsilon$ suffice? And, what is an acceptable ε value? Should it be $\varepsilon = 10^{-3}$? $\varepsilon = 10^{-16}$? $\varepsilon = 10^{-100}$? Is the ℓ_2-norm the right metric to check? What about other norms such as ℓ_1- or ℓ_∞-norm? What if we move to the matrix variable scenarios? Would a Frobenius norm (Euclidean norm for matrices) be sufficient, as in $\|\widehat{X} - X^\star\|_F \leq \varepsilon$? What about induced norms or nuclear norm?

Even more importantly, recent applications of optimization in machine learning have shown that the classical way of thinking "solvability" (i.e., for example we ask for $\|\widehat{x} - x^\star\|_2 \leq \varepsilon$, for very small ε) are suboptimal compared to less accurate but better generalizable solutions (i.e., solutions that are not perfect over the training set, but work amazingly well on unseen data, compared to a solution that overfits training data).

So, which problems do we know how to solve *exactly*? Are there any or is optimization "doomed" to be compromised with an approximate solution? Consider the case of a quadratic equation in one-dimension:

$$f(x) = ax^2 + bx + c = 0.$$

One could use optimization to solve this problem, but fortunately, we know the solution to this problem, up to exact accuracy:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Another example is the inverse of a 2×2 matrix. E.g., under proper assumptions on the range of the entries of the matrix, a matrix

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

has inverse:

$$A^{-1} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \tfrac{1}{ad-bc} \cdot \begin{bmatrix} d & -b \\ -c & a \end{bmatrix},$$

which is provided in closed-form. Of course, someone could have used optimization methods (*and there are such methods in practice; actually this is how we compute the inverse of a matrix with arbitrary sizes*) to complete this task. But the above *closed-form* solution is *unbeatable*, because we can get infinite accuracy, while numerical methods are restricted by the numbers they are limited to represent.

While these problems might seem too simple for some readers, the well known result on solving least-squares problems $y = Ax$ as $x^\star = A^{-1}y$ (*under assumptions on A, y - also this assumes we can compute A^{-1} up to infinite accuracy*) adds to the problems that are amenable to closed-form solutions.

Finally, there are solutions that seem to be unbeatable to very narrow set of problems, but overall should not be considered as sophisticated algorithms for generic problems. E.g., consider an algorithm that always returns $x^\star = 0$; such an algorithm is the best we can hope for (both in terms of accuracy and computational efficiency) for problems that have zero as their optimal solution. But, we do not that a priori, and not many problems have as solutions such a trivial answer.

The common alternative to closed-form solutions: Iterative numerical methods. What is the alternative then? The most natural way of solving optimization problems is *iteratively*. Meaning, we start from an initial point, and we exploit the fact that we might know *something* about the objective at hand, in order to make a more educated guess on where to move next.

Let us define the notion of an *oracle*: we assume that we learn more about our objective at hand by asking questions to an oracle. Each query comes with a "price", which is translated into how much computational time the oracle needs to answer the question. But, what types of questions we might want to ask? Since we try to minimize an objective function $f(x)$, one possible question could be "What is the value of f at a point x?". Other questions include gradient information, or even second-order Hessian information (*both to be defined later in the text*). Then, the above lead to the following general description of an iterative method:

1. Start from an initial point x_0.
2. Given an oracle $\mathcal{O}$, make queries to $\mathcal{O}$.
3. Obtain oracle's answer and exploit such a knowledge to reach to a new point as a putative solution.
4. Repeat steps 2.-3. until we get to a point where we are satisfied, according to a stopping criterion.

There are several issues, or open questions, with the above description.

- Is there a particular way to select the initial point? How does such a selection affect the performance of the algorithm?
- What type of oracles would we wish to have? How reasonable (e.g., computationally) is to have such an oracle? E.g., an oracle that, given an input, tells you whether it is the optimal point is very valuable, but it almost never exists in practice.
- How can we exploit the answers from the oracle? In other words, what is the method to use that exploits such information and translates it into a sequence of approximates towards a good solution?
- How we stop the procedure? Is there an easy way to check whether we are close to an acceptable solution?
- What is the total complexity of the algorithm?

These are some of the questions we will briefly answer in this course. Starting with the last question, we identify two types of complexity:

- *Analytical complexity*: The number of accesses to the oracle, in order to meet the stopping criterion.
- *Arithmetic complexity*: The total number of arithmetic computations in order to meet the stopping criterion.

To distinguish between the two, consider the following scenario: you have one algorithm that requires a constant number of access to an oracle, and another algorithm that requires many more number of access, but using a less computationally expensive oracle (that provides the same information). That being said, the first algorithm has a better analytical complexity (i.e., hiding the amount of effort an oracle makes, it requires less number of iterations), while the second algorithm could potentially be more efficient, and have better arithmetic complexity, by introducing a tradeoff between number of iterations and effort per iteration.

An easy way to compare the two complexities, and assuming that the per iteration complexity is the same in Big-Oh notation, is that the arithmetic complexity is just the computational complexity per iteration, multiplied with the analytical complexity. More in the chapters that follow.

The Black-Box model. Actually, what we have been describing above is the *Black-Box* model, widely used in optimization. While the idea is simple, such clear descriptions of what is allowed and what is not was missing at the beginning of the optimization research. The Black-Box model assumes:

- The only information regarding the problem is through the oracle.
- The oracle is more often than not assumed *local*: in the sense that if we ask a question, it relates to the current putative solution and how the function behaves locally.

These simple assumptions are necessary to avoid (infeasible and impractical) oracles that answer questions like "*Where is the optimum?*". But, these assumptions also help us avoid oracles that provide information that looks "innocent": e.g., assume an oracle that provides as a side information what is the distance to the optimum from the current point x_t, say $\|x_t - x^\star\|_2$. While it is not a direct information about $x^\star$, it is something that can definitely help the optimization, and lead to unfair comparisons to algorithms that do not have access to that information.

Common types of oracles. Some common types of oracles are:

- *Zeroth-order oracle*: Given a query point x, the oracle only returns $f(x)$.
- *First-order oracle*: Given a query point x, the oracle returns $f(x)$, and its gradient at x, $\nabla f(x)$ (assuming differentiability).
- *Second-order oracle*: Given a query point x, the oracle returns $f(x)$, its gradient $\nabla f(x)$, and the Hessian at x, $\nabla^2 f(x)$ (assuming twice-differentiability).

The Black-Box model and the various types of oracles will be more clear in the next lectures. What we need to remember for now is that this (very abstract) computational model is the prevailing model for continuous optimization. While this does not necessarily exclude alternatives, to the best of author's knowledge, it is the most obvious one and very hard to beat.

What is this class about. Given the above introduction, this class evolves around the following optimization criterion:

$$\begin{aligned} \min_{x \in \mathbb{R}^p} \quad & f(x) \\ \text{subject to} \quad & x \in \mathcal{C}. \end{aligned}$$

In other words, we will only consider minimization problems of continuous functions, with specific constraints on the variable x. During the class we will consider:

- Diverse objectives that belong to general classes of functions with characteristic properties.
- Different strategies to optimize problems within such classes of functions.
- Approaches that handle the same problems more carefully, assuming pragmatic restrictions, such as limited computational resources.
- How constraints can change the above strategies.

Our goal here is to provide a course that combines theory and practice, without heavily relying on one of the two perspectives, but rather following a balanced approach: we will consider applications in AI/Machine Learning/Signal Processing, but we will also study how theory helps setting up the algorithms.

What is this class NOT about. While optimization as a research field has many different "branches" that deserve our attention, our limited time within a semester forces us to restrict our scope heavily. These means some of the subjects we will not cover are: *i*) parts of convex optimization and analysis *(e.g., we do note cover duality in this course)*; *ii*) (mixed) integer programming; *iii*) combinatorial optimization algorithms (e.g., graph algorithms); *iv*) randomized algorithms; *v*) online algorithms (e.g., bandits); *vi*) Bayesian optimization; and *v*) in-depth discussion of deep learning architectures.

Finally, any of the excluded optimization topics, depending on the audience's preferences, could be included as "guest lectures" in future versions of the course.

∞

(Some good sources for linear algebra are [12–14])

Vectors. A p-dimensional vector x is denoted by

$$x = (x_1, x_2, \dots, x_p)^\top \in \mathbb{R}^p.$$

Here, we abuse the notation and use plain lowercase letters for both scalars and vectors, but the distinction should be clear from the context, or stated explicitly. The notation $\cdot^\top$ denotes the transpose operation, that translates a column vector into a row vector, and vice versa. Some properties of vectors include:

- *Commutative*: $x + y = y + x, \quad x,\ y \in \mathbb{R}^p$.
- *Associative*: $(x + y) + z = x + (y + z), \quad x,\ y,\ z \in \mathbb{R}^p$.
- *Distributive*: $x^\top (y + z) = x^\top y + x^\top z, \quad x,\ y,\ z \in \mathbb{R}^p$.
- $0 + x = x$.

The space that span a set of vectors $x_1,\ x_2\ , \dots,\ x_k$ is denoted as:

$$\text{span}(x_1,\ x_2\ , \dots,\ x_k) = \{\alpha_1 x_1 + \alpha_2 x_2 + \dots + \alpha_k x_k \mid \alpha_i \in \mathbb{R}\}.$$

A set of vectors $x_1,\ x_2\ , \dots,\ x_k$ are said to be *linearly independent* if:

$$\alpha_1 x_1 + \alpha_2 x_2 + \dots + \alpha_k x_k = 0 \quad \Rightarrow \quad \alpha_i = 0,\ \forall i.$$

Question: How does k compare to p, the vector dimension?

The inner product of two vectors in p-dimensions is mathematically defined as:

$$x^\top y \equiv \langle x, y \rangle = \sum_{i=1}^{p} (x_i \cdot y_i).$$

Here, $\langle \cdot, \cdot \rangle$ is a different notation for the inner product; the subscripts x_i, y_i denote the i-th elements of the corresponding vectors. The inner product can also be interpreted visually as follows:

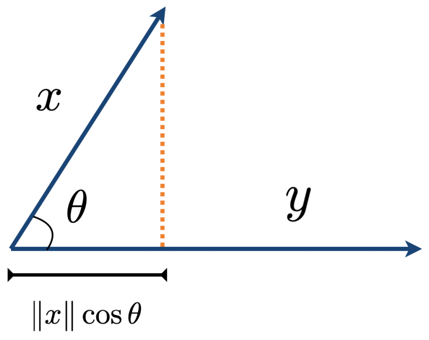

Fig. 2. Illustration of inner product.

This is based on the exact characterization of the inner product as:

$$\langle x, y \rangle = \|x\|_2 \cdot \|y\|_2 \cdot \cos\theta$$

We say that two *non-zero* vectors are *orthogonal* if $x^\top y = 0$; in other words, when $\theta = 90^o$.

Some norms associated with vectors are the following:

- *Euclidean or ℓ_2-norm*: $\|x\|_2 = \sqrt{\sum_i x_i^2}$.
- *ℓ_1-norm*: $\|x\|_1 = \sum_i |x_i|$.
- *ℓ_∞-norm*: $\|x\|_\infty = \max_i |x_i|$.

Key properties of norms include:

- $\|x\| \geq 0$.
- $\|x\| = 0 \Leftrightarrow x = 0$.
- $\|\alpha x\| = |\alpha| \cdot \|x\|, \forall \alpha \in \mathbb{R}$.
- *Triangle inequality*: $\|x + y\| \leq \|x\| + \|y\|$.
- *Cauchy-Schwarz inequality*: $|x^\top y| \leq \|x\| \cdot \|y\|$.

We will also consider functions over vectors, that could be considered as norms, but they do not satisfy some of the properties above (thus, often called pseudo-norms). One such key function is the ℓ_0-*"pseudo" norm*:

$$\|x\|_0 \equiv \text{card}(x) = \{\# \text{ of non-zeros in } x\}.$$

Question: Why is ℓ_0-pseudo norm not a regular norm?

Matrices. A $p \times d$ matrix X is denoted as:

$$X = \begin{bmatrix} X_{11} & X_{12} & \dots & X_{1d} \\ X_{21} & X_{22} & \dots & X_{2d} \\ \vdots & & \ddots & \vdots \\ X_{p1} & X_{p2} & \dots & X_{pd} \end{bmatrix} \in \mathbb{R}^{p \times d}.$$

We will generally use uppercase plain letters for matrices, unless otherwise stated.

Some key types of matrices include: *i*) square matrix where $p = d$; *ii*) tall matrix, when $p \gg d$; *iii*) fat matrix, when $p \ll d$; *iv*) zero matrix, when all the entries are zero; *v*) diagonal matrix, when all entries are zero outside the diagonal (usually used for square matrices); *vi*) identity matrix, a diagonal matrix with ones on the diagonal.

The notation $\cdot^\top$ denotes the transpose operation, that exchanges the dimensions of the matrix. In particular, $X^\top \in \mathbb{R}^{d \times p}$ where:

$$X^\top = \begin{bmatrix} X_{11} & X_{21} & \dots & X_{p1} \\ X_{12} & X_{22} & \dots & X_{p2} \\ \vdots & & \ddots & \vdots \\ X_{1d} & X_{2d} & \dots & X_{pd} \end{bmatrix}.$$

Similarly to the vector case, some properties of matrices include:

- *Commutative*: $A + B = B + A, \quad A,\ B \in \mathbb{R}^{p \times d}$.
- *Associative*: $(A+B)+C = A+(B+C), \quad A,\ B,\ C \in \mathbb{R}^{p \times d}$.
- *Distributive*: $A \cdot (B + C) = A \cdot B + A \cdot C, \quad A \in \mathbb{R}^{p \times d},\ B,\ C \in \mathbb{R}^{d \times m}$.
- $0 + A = A$.
- $(A + B)^\top = A^\top + B^\top$.

Above, we used matrix multiplication between matrices. For matrices $C \in \mathbb{R}^{p \times m}$, $A \in \mathbb{R}^{p \times d}$, and $B \in \mathbb{R}^{d \times m}$, this is defined as:

$$C = \begin{bmatrix} C_{11} & C_{12} & \dots & C_{1m} \\ C_{21} & C_{22} & \dots & C_{2m} \\ \vdots & & \ddots & \vdots \\ C_{p1} & C_{p2} & \dots & C_{pm} \end{bmatrix} = A \cdot B$$

where

$$C_{ij} = \sum_{\ell=1}^{d} A_{i,\ell} \cdot B_{\ell,j}.$$

Special cases of matrix multiplication are vector inner product (where a vector is seen as a single column matrix), matrix-vector multiplication, and vector outer-product.

More properties of matrix multiplication include (the corresponding dimensions are clear from the context - we also drop $(\cdot)$ for clarity):

- $A(BC) = (AB)C$.
- $\alpha(AB) = (\alpha A)B, \quad \alpha \in \mathbb{R}$.
- $(AB)^\top = B^\top A^\top$.
- $AB \neq BA$ in general.

The inner product between two matrices, with matching dimensions, is defined as:

$$\langle A, B \rangle \equiv \text{Tr}(A^\top B) \equiv \text{Tr}(B^\top A), \quad \forall A, B \in \mathbb{R}^{p \times d}.$$

Here, $\text{Tr}(\cdot)$ denotes the linear operator that sums the elements on the diagonal of its matrix input argument.

Question: how does Tr$(B^\top A)$ compare to vec$(B)^\top$vec(A)?

The *rank* of a matrix A is defined as the maximum number of independent columns or rows. The rank of a matrix is also directly connected with the *singular value decomposition* (SVD) of a matrix. In particular, every matrix A has a singular value decomposition of the form:

$$A = U \Sigma V^\top, \quad U \in \mathbb{R}^{p \times r},\ \Sigma \in \mathbb{R}^{r \times r},\ V \in \mathbb{R}^{d \times r}.$$

Here, r denotes the rank of the matrix, which has the following meanings within the SVD:

- A rank-r matrix A has only r non-zero singular values, which are the diagonal elements of Σ. By definition, Σ is a diagonal matrix in SVD.
- A rank-r matrix A has r orthonormal (i.e., orthogonal and of unit norm) *left* singular vectors, that span a rank-r subspace of the p-dimensional row space of A.
- A rank-r matrix A has r orthonormal (i.e., orthogonal and of unit norm) *right* singular vectors, that span a rank-r subspace of the d-dimensional column space of A.

Finally, an important class of matrices is that of *positive (semi)definite* matrices; we often use the abbreviation PD or PSD. A PSD (resp. PD) matrix $A \in \mathbb{R}^{p \times p}$ has the following properties:

- A is a square matrix.
- A is symmetric, i.e., $A = A^\top$.
- For every non-zero vector $x \in \mathbb{R}^p$, it holds $x^\top Ax \geq 0$ (resp. $x^\top Ax > 0$).

While the definition of PSD/PD matrices has a clear algebraic interpretation, we will also provide a geometrical interpretation. Decomposing the third property of PSD/PD matrices, define $y := Ax \in \mathbb{R}^p$. One can see y as the translation of the original vector x through A: i.e., $x \mapsto Ax$. Then, A being PSD/PD matrix has the property that the translated vector, y, has positive inner product with the original x: i.e., x and y point towards non-antithetical directions.

Some norms associated with matrices are the following:

- *Frobenius or ℓ_2-norm*: $\|X\|_F = \sqrt{\sum_{ij} X_{ij}^2}$.
- *Nuclear norm*: $\|X\|_* = \sum_i \sigma_i(X)$, where $\sigma_i(X)$ is the i-th singular value.
- *Spectral or ℓ_2-norm*: $\|X\|_2 = \max_i \sigma_i(X)$.

Properties of norms convey from the vector case to the matrix case, so we defer to the corresponding part of the vector case.

Finally, we define the notion of an *inverse* of a matrix. Inverses are squared matrices, denoted with the superscript $\cdot^{-1}$, such that:

$$AA^{-1} = A^{-1}A = I,$$

where I is the identity matrix with matching dimensions.

Appendix

1. J. Nocedal and S. Wright. Numerical optimization. Springer Science & Business Media, 2006.
2. Y. Nesterov. Introductory lectures on convex optimization: A basic course, volume 87. Springer Science & Business Media, 2013.
3. S. Boyd and L. Vandenberghe. Convex optimization. Cambridge university press, 2004.
4. D. Bertsekas. Convex optimization algorithms. Athena Scientific Belmont, 2015.
5. Sébastien Bubeck. Convex optimization: Algorithms and complexity. Foundations and Trends® in Machine Learning, 8(3-4):231–357, 2015.
6. S. Weisberg. Applied linear regression, volume 528. John Wiley & Sons, 2005.
7. T. Hastie, R. Tibshirani, and M. Wainwright. Statistical learning with sparsity: the lasso and generalizations. CRC press, 2015.
8. J. Friedman, T. Hastie, and R. Tibshirani. The elements of statistical learning, volume 1. Springer series in statistics New York, 2001.
9. M. Paris and J. Rehacek. Quantum state estimation, volume 649. Springer Science & Business Media, 2004.
10. M. Daskin. A maximum expected covering location model: formulation, properties and heuristic solution. Transportation science, 17(1):48–70, 1983.
11. I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. MIT press, 2016.
12. L. Trefethen and D. Bau III. Numerical linear algebra, volume 50. Siam, 1997.
13. G. Strang. Introduction to linear algebra, volume 3. Wellesley-Cambridge Press Wellesley, MA, 1993.
14. G. Golub. Cmatrix computations. The Johns Hopkins, 1996.
15. A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.
16. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.
17. S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems, pages 91–99, 2015.
18. T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems, pages 3111–3119, 2013.
19. Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473, 2014.
20. Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 1243–1252. JMLR. org, 2017.
21. Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In Fifteenth annual conference of the international speech communication association, 2014.
22. Tom Sercu, Christian Puhrsch, Brian Kingsbury, and Yann LeCun. Very deep multilingual convolutional neural networks for LVCSR. In 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 4955–4959. IEEE, 2016.
23. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. page arXiv:1706.03762, 2017.
24. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. page arXiv:1810.04805, 2018.
25. Luowei Zhou, Hamid Palangi, Lei Zhang, Houdong Hu, Jason J Corso, and Jianfeng Gao. Unified vision-language pre-training for image captioning and VQA. In AAAI, pages 13041–13049, 2020.
26. Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. arXiv preprint arXiv:2005.14165, 2020.
27. Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. arXiv preprint arXiv:1909.08053, 2019.
28. Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. arXiv preprint arXiv:1910.10683, 2019.
29. Gary Marcus, Ernest Davis, and Scott Aaronson. A very preliminary analysis of DALL-E 2. arXiv preprint arXiv:2204.13807, 2022.
30. John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with AlphaFold. Nature, 596(7873):583–589, 2021.
31. Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.
32. Or Sharir, Barak Peleg, and Yoav Shoham. The cost of training nlp models: A concise overview. arXiv preprint arXiv:2004.08900, 2020.
33. H. Karimi, J. Nutini, and M. Schmidt. Linear convergence of gradient and proximal-gradient methods under the Polyak-Łojasiewicz condition. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pages 795–811. Springer, 2016.
34. Philip Wolfe. Convergence conditions for ascent methods. SIAM review, 11(2):226–235, 1969.
35. Larry Armijo. Minimization of functions having Lipschitz continuous first partial derivatives. Pacific Journal of mathematics, 16(1):1–3, 1966.
36. Stephen Wright and Jorge Nocedal. Numerical optimization. Springer Science, 35(67-68):7, 1999.
37. B. Polyak. Introduction to optimization. Inc., Publications Division, New York, 1, 1987.
38. Stephen Boyd, Lin Xiao, and Almir Mutapcic. Subgradient methods. lecture notes of EE392o, Stanford University, Autumn Quarter, 2004:2004–2005, 2003.
39. Marguerite Frank, Philip Wolfe, et al. An algorithm for quadratic programming. Naval research logistics quarterly, 3(1-2):95–110, 1956.
40. M. Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In Proceedings of the 30th international conference on machine learning, number CONF, pages 427–435, 2013.
41. J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the ℓ_1-ball for learning in high dimensions. In Proceedings of the 25th international conference on Machine learning, pages 272–279, 2008.
42. Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. Computer, 42(8):30–37, 2009.
43. A. Mnih and R. Salakhutdinov. Probabilistic matrix factorization. In Advances in neural information processing systems, pages 1257–1264, 2008.
44. T. Booth and J. Gubernatis. Improved criticality convergence via a modified Monte Carlo power iteration method. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.
45. Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. Quarterly of applied mathematics, 2(2):164–168, 1944.
46. Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. Journal of the society for Industrial and Applied Mathematics, 11(2):431–441, 1963.
47. Andreas Griewank. The modification of Newton's method for unconstrained optimization by bounding cubic terms. Technical report, Technical report NA/12, 1981.
48. Yurii Nesterov and Boris T Polyak. Cubic regularization of Newton method and its global performance. Mathematical Programming, 108(1):177–205, 2006.
49. Yurii Nesterov and Arkadii Nemirovskii. Interior-point polynomial algorithms in convex programming. SIAM, 1994.
50. Ulysse Marteau-Ferey, Francis Bach, and Alessandro Rudi. Globally convergent Newton methods for ill-conditioned generalized self-concordant losses. Advances in Neural Information Processing Systems, 32, 2019.
51. Quoc Tran-Dinh, Anastasios Kyrillidis, and Volkan Cevher. Composite self-concordant minimization. J. Mach. Learn. Res., 16(1):371–416, 2015.
52. Konstantin Mishchenko. Regularized Newton method with global $o(1/k^2)$ convergence. arXiv preprint arXiv:2112.02089, 2021.
53. S. Zavriev and F. Kostyuk. Heavy-ball method in nonconvex optimization problems. Computational Mathematics and Modeling, 4(4):336–341, 1993.
54. E. Ghadimi, H. Feyzmahdavian, and M. Johansson. Global convergence of the heavy-ball method for convex optimization. In 2015 European control conference (ECC), pages 310–315. IEEE, 2015.
55. Y. Nesterov. A method for solving the convex programming problem with convergence rate $o(1/k^2)$. In Dokl. akad. nauk Sssr, volume 269, pages 543–547, 1983.
56. B. O'Donoghue and E. Candes. Adaptive restart for accelerated gradient schemes. Foundations of computational mathematics, 15(3):715–732, 2015.
57. O. Devolder, F. Glineur, and Y. Nesterov. First-order methods of smooth convex optimization with inexact oracle. Mathematical Programming, 146(1-2):37–75, 2014.
58. L. Bottou, F. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. Siam Review, 60(2):223–311, 2018.