Lecture Notes:
Introduction to Convex Optimization

Changho Suh[1]

June 4, 2019

[1]Changho Suh is an Associate Professor in the School of Electrical Engineering at Korea Advanced Institute of Science and Technology, South Korea (Email: chsuh@kaist.ac.kr).

Lecture 1: Logistics and Overview

About instructor

Welcome to EE523: Convex Optimization! My name is Changho Suh, an instructor of the course. A brief introduction of myself. A long time ago, I was one of the students in KAIST like you. I spent six years at KAIST to obtain the Bachelor and Master degrees all from Electrical Engineering in 2000 and 2002, respectively. And then I left academia, joining Samsung electronics. At Samsung, I worked on the design of wireless communication systems like 4G-LTE systems. Spending four and a half years, I then left industry, joining UC-Berkeley where I obtained the PhD degree in 2011. I then joined MIT as a postdoc, spending around one year. And then I came back to KAIST. My research interests include information theory and machine learning which have something to do with the optimization theory, which I am going to cover in part from this course.

Today's lecture

In today's lecture, we will cover two very basic stuffs. The first is logistics of this course: details as to how the course is organized and will proceed. The second thing to cover is a brief overview to this course. In the second part, I am going to tell you a story of how convex optimization was developed, as well as what we will cover from this course.

My contact information, office hours and TAs' information

See syllabus uploaded on the course website. One special note: if you cannot make it neither for my office hours nor for TAs' ones, please send me an email to make an appointment in different time slots.

Prerequisite

The key prerequisite for this course is to have a good background in *linear algebra*. In terms of a course, this means that you should have taken the following course: MAS109, which is an introductory-level course on linear algebra. This is the one that is offered in the Department of Mathematical Sciences. Some of you might take a different yet equivalent course from other departments. This is also okay. Taking a somewhat advanced-level course (e.g., MAS212: Linear Algebra) is optional although it is recommended. If you feel a bit uncomfortable although you took the relevant course(s), you may want to review a material (part of a linear algebra course note at Stanford) that I uploaded on the course website.

Another prerequisite for the course is a familiarity with the concept on *probability*. In terms of a course, this means that you are expected to be somewhat comfortable with the contents dealt in EE210, which is an undergraduate-level course on probability. Taking an advanced course like EE528 (Random Processes) is optional. This is not a strong prerequisite. In fact, the convex optimization concept itself has nothing to do with probability. But some problems (especially the ones that arise in recent trending fields like machine learning and artificial intelligence, and I will also touch upon a bit in this course) deal with some quantities which are random and therefore are described with probability distributions. This is the only reason that understanding the concept on probability is needed for this course. Hence, reviewing another material (an easy-

level probability course note at MIT) that I uploaded on the course website may suffice.

There must be a reason as to why linear algebra is crucial for this course. The reason is related to the definition of optimization, which subsumes convex optimization (which we will study throughout this course) as a special case. A somewhat casual definition of optimization is to make the best choice among many candidates (or alternatives). A more math-oriented definition of optimization (which we should rely on as scientists and/or engineers) is to choose an *optimization variable* (or *decision variable*) so as to *minimize* or *maximize* a *certain quantity* of interest possibly given some constraint(s). Here the optimization variable and the certain quantity are the ones that relate the optimization to linear algebra. In many situations, the optimization variable are multiple real values which can be succinctly represented as a *vector*. Also the certain quantity (which is a function of the optimization variable) can be represented as a function that involves matrix-vector multiplication and/or matrix-matrix multiplication, which are basic operations that arise in linear algebra. In fact, many operations and techniques in linear algebra help formulating an optimization problem in a very succinct manner, and therefore help theorizing the optimization field. This is the very reason that this course requires a good understanding and manipulation techniques on linear algebra. Some of you may not be well trained with expressing an interested quantity with vector/matrix forms. Please don't be offended. You will have lots of chances to be trained via some examples that will be covered in lectures and homeworks as the course progresses. Whenever some advanced techniques are needed, then I will provide detailed explanations and/or materials which serve you to understand the required techniques.

If you think that you lack these prerequisites, then come and consult with me so that I can help you as much as possible.

Course website

We have a course website on the KLMS system. You can simply login with your portal ID. If you want to only sit in this course (or cannot register the course for some reason), please let me or one of TAs know your email address. Upon request, we are willing to distribute course materials to the email address that you sent us.

Text

The organization of the course will follow mostly but in part by: Prof. Laurent El Ghaoui's livebook (LB for short), titled "Optimization Models and Applications". To access, you need to register at http://livebooklabs.com/keeppies/c5a5868ce26b8125. A good news is that the registration comes for free.

I am going to provide you with lecture slides (LS for short) which I will use during lectures, as well as course notes (CN for short) like the one that you are now reading. Most times, these materials will be posted at night on a day before class, but sometimes course notes may be uploaded after lectures. These materials are almost self-contained, and cover the entire contents that will show up in homeworks and exams. So if you want to make a minimal effort to this course (for some personal reason), then these materials may suffice to pass the course with a reasonable grade.

For those who have enough energy, passion and time, I recommend you to consult with other references: (1) Calafiore & El Ghaoui, "Optimization Models", Cambridge University Press, Oct. 2014; (2) Boyd & Vandenberghe, "Convex Optimization", Cambridge University Press, 2004 (available online at http://stanford.edu/∼boyd/cvxbook/bv_cvxbook.pdf). Sometimes, I will make some homework problems from these references. If so, I will let you know and upload a

soft copy of the relevant part of the books.

Problem sets

There will be weekly or bi-weekly problem sets. So there would be seven to eight problem sets in total. Solutions will be usually available at the end of the due date. This means that in principle, we do not accept any late submission. We encourage you to cooperate with each other in solving the problem sets. However, you should write down your own solutions by yourselves. You are welcome to flag confusing topics in the problem sets; this will not lower your grade. Some problems may require programming like CVX that runs in MATLAB. We will also provide a tutorial for programming when a first related-homework is issued.

Exams

As usual, there will be two exams: midterm and final. Please see syllabus for the schedule that our institution assigns by default - that is Thursday 9am-noon on an exam week for this course. Please let us know if someone cannot make it for the schedule. If your reason is reasonable, then we can change the schedule or can give a chance for you to take an exam in a different time slot that we will organize individually.

Two things to notice. First, for both exams, you are allowed to use one cheating sheet, A4-sized and double-sided. So it is a kind of semi-closed-book exam. Second, for your convenience, we will provide an instruction note for each exam, which contains detailed guidelines as to how to prepare for the exam. Such information includes: (1) how many problems are in the exam; (2) what types of problems are dealt with in what contexts; (3) the best way to prepare for such problems.

Course grade

Here is a rule for the course grade that you are mostly interested in perhaps. The grade will be decided based on four factors: problem sets (22%); midterm (32%); final (40%); and interaction (6%). Here the interaction means any type of interaction. So it includes attendance, in-class participation, questions, discussion, and any type of interaction with me.

Overview

Now let's move onto the second part. Here is information for reading materials: Course Note (CN for short) 1 and Chapter 1 in LB. In this part, I will tell you how the theory of convex optimization was developed in which contexts. I will then provide you with specific topics that we will learn about through the course.

Optimization

To talk about a story of how the theory of convex optimization was developed, we need to first know about the history of optimization which includes convex optimization as a special case. What is optimization? As mentioned earlier, a casual informal definition of optimization is to make the best choice out of possible candidates. It comes up often in our daily life. For example, we may want to figure out a scheduling strategy for airplanes so that the total waiting time is minimized under some constraints, e.g., a certain airplane with emergency should take off no later than a specific time. Or family members may want to choose a restaurant to visit for dinner, so as to maximize the happiness of the members (if it can be quantified) given a distance constraint, e.g., a chosen restaurant should be within a few kilometers.

A more mathematical definition of optimization that we are interested in as scientists is to choose an optimization variable that minimizes (or maximizes) a certain quantity of interest possibly given some constraints. Of course this course aims at learning a theory concerning such a formal definition. Specifically we are interested in learning a *mathematical theory of optimization* which has been extensively developed and explored for a few past centuries. In fact, the birth of the theory traced back to an astronomy problem in the 1800s. So let us talk about the problem to see how the theory was developed.

An astronomy problem in the early 1800s

In the early 1800s, astronomers discovered a new planetoid (or called dwarf planet), which was later named *Ceres*. Giuseppe Piazzi is the first astronomer who discovered the planetoid. He wished to figure out an orbit of Ceres, so as an effort, he made 19 observations (of its locations) over 42 days. However, something happened in locating the trajectory. The object was lost due to the glare of the Sun. So many astronomers wanted to *predict* the hidden trajectory using the partial 19 observations.

One interesting trial was made by a German young mathematician, named Carl Friedrich Gauss ($1777 \sim 1855$)[1]. He made a specific yet smart approach to figure out the trajectory successfully. In the process, he could develop a mathematical problem that later formed the basis of the optimization field.

Gauss's approach

Here is what Gauss did - see also Fig. 1. First of all, he gathered all the observations (each pointing out a location of Ceres measured at a certain time) scattered around the Sun. Let b_i indicate a coordinate of the location of the ith observation where $i \in \{1, 2, \dots, m\}$. Here m denotes the total number of observations that could be up to 19 in the astronomy problem context - remember that Piazzi made 19 observations.

Next he fixed two arbitrary observation points. He then drew an orbit that crosses the two fixed points. Actually it was well studied in the astronomy field that an orbit can be fully represented with six parameters. So the orbits that cross the two fixed points can be represented with four parameters. Depending on the choice of such free parameters, there are many ways to draw such orbits. Let's denote by x a vector that stacks up the four parameters. Here what Gauss did is that he could represent a point on the orbit which is the closest to each b_i, in terms of the vector x. It turned he could approximate the point as $A_i x$ where A_i indicates a certain matrix which relates b_i to the nearest point on the orbit. He then believed that for the ground-truth orbit (which we wish to figure out), the distance to b_i, reflected in $\|A_i x - b_i\|$, should be within a location-measurement error[2]. This motivated him to solve the following optimization problem:

$$\min_x \sum_{i=1}^{m} \|A_i x - b_i\|^2. \tag{1}$$

[1]Yes, he is the guy who invented the *Gaussian distribution* (which is the one of the very famous and useful distributions in probability) and *Gaussian elimination* (an efficient method which allows us to do a matrix inversion or to solve linear equations).

[2]Here $\|x\|$ denotes the Euclidean norm (or called the ℓ_2 norm), defined as $\|x\| := \sqrt{x_1^2 + \cdots + x_d^2}$ where d is the dimension of x.

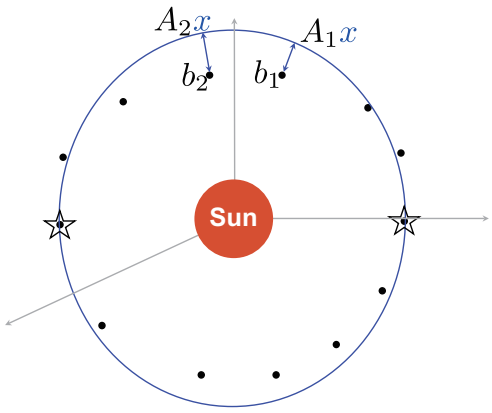


Figure 1: Gauss's approach to figure out the orbit of Ceres

Gauss then observed that $\sum_{i=1}^{m} \|A_i x - b_i\|^2$ can be simplified as:

$$\begin{aligned} \sum_{i=1}^{m} \|A_i x - b_i\|^2 &= \left\| \begin{bmatrix} A_1 x - b_1 \\ \vdots \\ A_m x - b_m \end{bmatrix} \right\|^2 \\ &= \left\| \begin{bmatrix} A_1 \\ \vdots \\ A_m \end{bmatrix} x - \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} \right\|^2. \end{aligned} \tag{2}$$

Letting $A := [A_1; \cdots; A_m]$ and $b := [b_1; \cdots; b_m]$[3], he then re-wrote the optimization problem as:

$$\min_x \|Ax - b\|^2. \tag{3}$$

Least-squares problem

The problem (3) is actually the famous problem that is now known as the *least-squares* problem. Notice that the word "*least*" comes from "min" and "*squares*" is due to the square exponent placed above the Euclidean norm. You may wonder why Gauss used the square as an exponent in the objective function. Why not other exponents like 1, 3 or 4? Actually this was due to the mathematical beauty that Gauss was obsessed with. Notice that using other exponents like 1 or 3 or 4, one cannot do the beautiful simplification like (2). If you cannot see why, then check in homework.

It turns out that the least-squares problem could open up the optimization field (since the time, people have tried to theorize the field with passion) and also has played a significant role in the field. There are two reasons as to why the problem played such a big role. The first is that (3) has the beautiful closed-form solution:

$$x^* = (A^T A)^{-1} A^T b \tag{4}$$

where $(\cdot)^T$ indicates a transpose of a matrix and $(\cdot)^{-1}$ denotes a matrix inversion. We will later show why the solution is of the form (4) - please be patient until we get to the point. The second reason is that there are efficient algorithms and software that enable us to compute the solution

[3]Here the notation $[\cdot;\cdot]$ means the column-wise concatenation.

involving a matrix inverse. Even in the 1800s, there was an efficient matrix-inversion algorithm, based on the *Gaussian elimination* due to again Gauss.

Since the development of the least-squares problem, people tried to translate any problem of their interest to a least-squares problem. So a variety of translation techniques (that we will also study in this course) have been developed. However, people encountered many situations in which such translation does not work. This challenge was sort of expected. As you can easily image, the least-squares problem is just a single tiny class of the entire optimization problems that can be formulated in the world.

A breakthrough by Kantorovich

Unfortunately there was no significant progress on the optimization theory for more than a century. But another history was made in 1939 by a Soviet economist, named Leonid Kantorovich. He made a breakthrough in the course of solving a military-related problem during World War II (sort of forced to do so by the Soviet Union government). The problem that he was trying to solve was to plan expenditures and returns of soldiers to minimize the entire cost of the Soviet Union Army as well as to maximize the losses imposed on the enemy.

In the process, he could formulate an optimization problem now known as the very famous *linear program* (LP)[4]. Simply put, the LP is an optimization problem in which the objective function and the functions that appear in constraints are all linear. We will study its formal definition later on. Unlike the least-squares problem, the LP has no closed form solution. But the good news is that Kantorovich could develop a very efficient algorithm that achieves the optimal solution x^*. Actually having a closed form solution is not that important as long as we know how to get to the optimal solution. This achievement won him the *Nobel Prize in Economics* in 1975.

The development of LP made people become excited again, trying to translate an interested problem into a least-squares problem or LP. While many LP-translation techniques (that we will also study in this course) have been developed, people encountered *still many situations* in which the translation is not doable.

A class of tractable optimization problems

Inspired by the development of LP, people tried to come up with a class of *tractable* optimization problems which can be solved reliably & efficiently - LP is one such example. In a decade, another tractable problem, called quadratic program (QP), was developed. It is a sort of a generalized version, as it includes as special cases the least-squares problem and LP - see Fig. 2. In the 1990s, another problem, called second-order cone program (SOCP), was developed which subsume as special cases all of the prior problems. Around the same time, a larger class of problem, called semi-definite program, was developed. More and more tractable problems have been developed so far. It turns out that all of the tractable problems share the common property (concerning the word "convex"), and this property established the class of tractable problems, named *convex optimization*.

Course outline

This course consists of three parts. In Part I, we will study the basic concepts and several mathematical definitions required to understand what convex optimization is as well as how to translate an interested problem into a convex problem. We will then explore five instances of

[4]The "program" (or "programming") is sort of a jargon frequently used in the field, which refers to an optimization problem. So the formal name of LP is a linear optimization problem.

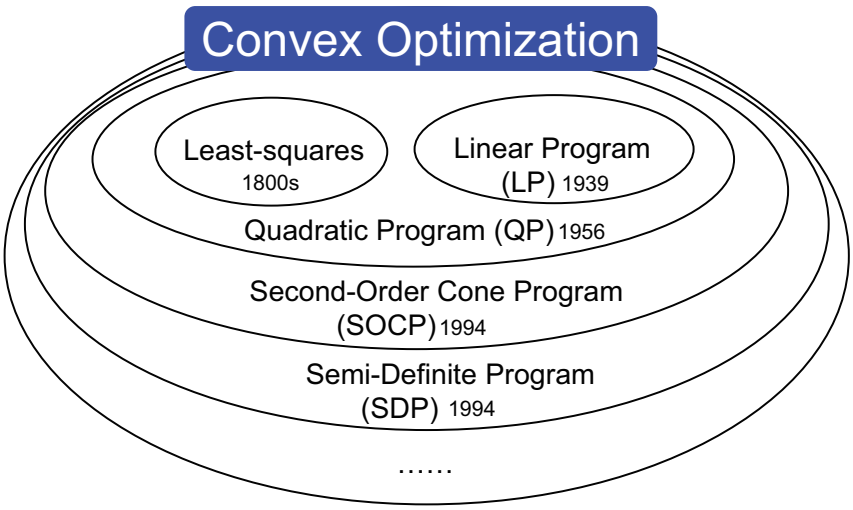


Figure 2: A class of tractable optimization problems: Convex optimization.

convex optimization problems: LP, least-squares, QP, SOCP and SDP. Specifically we will focus on techniques which serve recognizing (and translating to) such problems. We will also study some prominent algorithms for solving such problems. In Part II, we will study one of the key theories in the optimization field, called *duality*. There are two types of dualities: (1) strong duality; (2) weak duality. It turns out that the strong duality is quite useful for gaining some algorithmic insights for convex problems. The weal duality helps dealing with difficult non-convex problems, by providing an approximated solution. In the last third part, we will explore applications that arise in machine learning and finance: (i) supervised learning, one of the most popular machine learning techniques using labelled data; (ii) Generative Adversarial Networks (GANs), one of the breakthrough models for unsupervised learning; (iii) portfolio optimization.

EE523 Convex Optimization
KAIST, Spring 2019

March 5, 2019

Lecture 2: Definition of Convex Optimization

Recap

Last time, I told you a story of how the optimization theory was developed. There were two breakthroughs in the history of optimization. The first was made by the famous Gauss. In the process of solving an astronomy problem of figuring out the orbit of Ceres (which many astronomers were trying to address in the 1800s), he could develop an optimization problem, which is now known as the least-squares problem. The beauty of the least-squares problem is two-folded: (i) it has a closed form solution; (ii) there is an algorithm which enables computing the matrix inverse efficiently which is required to compute the solution. It turned out the beauty of the problem opened up the optimization field and has played a significant role in the field.

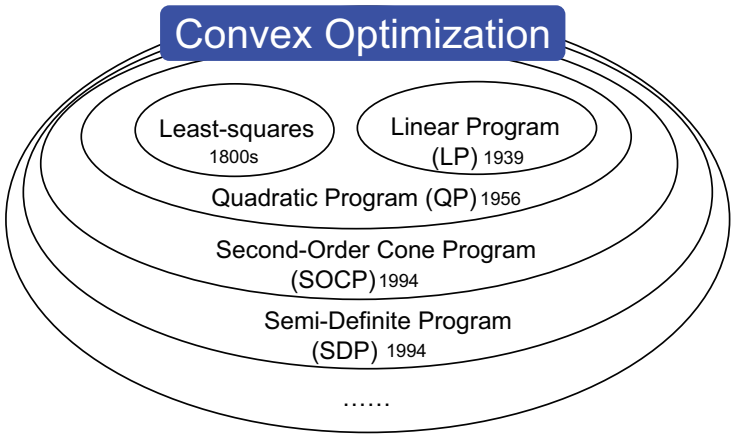


Figure 1: A class of tractable optimization problems: Convex optimization.

The second breakthrough was made by Leonid Kantorovich. In the process of solving a military-related resource-allocation problem, he could formulate a problem which is now known as linear program (LP). The good thing of LP is that there developed an efficient algorithm which allows us to compute the optimal solution reliably and efficiently although the closed form solution is unknown. In other words, Kantorovich came up with the concept of *tractable* optimization problems which can be solved via an algorithm without the knowledge of the optimal solution form. This motivated many followers to mimick his approach, thereby coming up with a class of tractable optimization problems: convex optimization; see Fig. 1.

Today's lecture

The goal of today's lecture is to understand what the convex optimization is. To this end, we will cover four stuffs sequentially. First we will study a standard mathematical formulation of optimization problems. It turns out the definition of *convex optimization problems* of our main interest requires the knowledge of *convex functions*. But the definition of convex functions relies on the concept of *convex sets*. So in the second part, we will study what the convex set is and also investigate some important examples in an effort to be familiar with the concept. Next we will study the definition of convex functions together with a couple of examples and crucial

properties. Using all theses, we will finally investigate a standard mathematical formulation of convex optimization problems.

Optimization problem in standard form

Let us start by recalling the definition of optimization: Choosing an optimization variable that minimizes (or maximizes) a certain quantity of interest possibly given constraints. Here we denote the optimization variable by $x := [x_1, x_2, \ldots, x_d]^T \in \mathbf{R}^d$ where d indicates the dimension of the variable. Denote the objective function (the certain quantity) by $f(x) \in \mathbf{R}$. There are two types of constraints: (i) inequality constraints; (ii) equality constraints. The inequality constraints are represented by the form like $f_i(x) \leq c_i$ where $i = 1, \ldots, m$. Here m indicates the number of the constraints. Without loss of generality (WLOG), the constant c_i can be merged with $f_i(x)$ and hence the form can be simplified as: $f_i(x) \leq 0$. Similarly the equality constraints can be represented by: $h_i(x) = 0$ where $i = 1, \ldots, p$ and p is the number of the equality constraints.

Using these notations, one can write the standard form of optimization problems as:

$$\begin{aligned} &\min_x f(x) \\ &\text{subject to } f_i(x) \leq 0, i = 1, 2, \ldots, m, \\ &\qquad\qquad h_i(x) = 0, i = 1, 2, \ldots, p. \end{aligned} \tag{1}$$

Without loss of generality, it suffices to consider the minimization problem, since the maximization problem can readily come by flipping the sign of $f(x)$: $\min f(x)$ is equivalent to $\max -f(x)$. Here we have two conventions that allow us to simplify the above form (1). One is that we use the colon ":" to indicate the "subject to". The second is that the x placed below min is omitted since the role of x is clear enough. Hence, the simpler form reads:

$$\begin{aligned} \min f(x) : \ & f_i(x) \leq 0, i = 1, 2, \ldots, m, \\ & h_i(x) = 0, i = 1, 2, \ldots, p. \end{aligned} \tag{2}$$

Two more things to note. One is the *optimal value*, denoted by $p^* := \min f(x)$. The other is the *optimal solution*, denoted by $x^* := \arg\min f(x)$. Here "arg min" stands for *"argues the one that minimizes"*.

Convex set

Now what is *convex optimization* that we wish to figure out in this lecture? As mentioned in the beginning of this lecture, to define this, we need to know about the concept of convex functions. But to define convex functions, we need to know about convex sets. So we will first study what the convex set is.

A set $\mathcal{S}$ is said to be *convex* if

$$x, y \in \mathcal{S} \Longrightarrow \lambda x + (1 - \lambda) y \in \mathcal{S}, \ \ \forall \lambda \in [0, 1]. \tag{3}$$

Examples: Point, line, plane, line segment, ...

To get a concrete feel about what the convex set means, let us investigate several examples. The first simplest example is the set containing a *single point*. This is obviously a convex set, as any linear combination that lies in between x and y, reflected in $\lambda x + (1 - \lambda) y$, is just the single point.

The second simplest example is perhaps the set that contains a *line* that lives in a 2-dimensional ambient space. This is also convex because any linear combination of two points lying on a line should also lie on the line. Here let us investigate how to represent such convex set. Actually this representation will help us to understand the concept of convex optimization later on. Notice that the line in a 2-dimensional space can be represented as: $x_2 = a_1 x_1 + b_1$ where a_1 and b_1 indicates the slope and y-intersect. Hence, one can represent the set as:

$$\mathcal{S} = \{x : x_2 = a_1 x_1 + b_1\}. \tag{4}$$

Using vector notations, one can define $a := [-a_1, 1]^T$ and $b_1 := b$, which in turn simplifies the representation (4) as:

$$\mathcal{S} = \{x : a^T x - b = 0\}. \tag{5}$$

The third example is the naive extension of the second example: *a plane living in a 3-dimensional space*. This is also obviously a convex set, as any combination of two points lying on a plane also lies on the plane. The representation of the convex set is exactly the same as (5), except that now the dimension of x and a are 3.

The fourth example is the one in which the dimension of an object of interest differs from d by 2. One such example is the set that contains a *line* living in a 3-dimensional space. This is also a convex set, as the object of interest is a line. But the representation of such convex set is different from (5). A line is actually the *intersection* of two planes in a 3-dimensional space. So the representation of the set should read:

$$\mathcal{S} = \{x : a_1^T x - b_1 = 0, a_2^T x - b_2 = 0\}. \tag{6}$$

Defining $A := [a_1, a_2]^T$ and $b := [b_1, b_2]^T$, this can be simplified as:

$$\mathcal{S} = \{x : Ax - b = 0\}. \tag{7}$$

Looking carefully at these examples, one can see that the representation of a line, a plane or a hyperplane (a subspace whose dimension is one less than that of its ambient space) lying in a larger-dimensional ambient space reads the form like (7). Depending on the dimension of the matrix $A \in \mathbf{R}^{p \times d}$, $\mathcal{S}$ may refer to the set containing a line, a plane or a higher-dimensional plane. For instance, when $d - p = 1$, $\mathcal{S}$ refers to a line. When $d - p = 2$, $\mathcal{S}$ indicates a plane. Actually the set represented bv the form (7) is called an *affine* set. An affine function is a linear function that allows for having a bias constant term; the formal definition will be given later on. Since the set in (7) includes the affine function $Ax - b$, it is called an affine set.

Another example that I would like to mention is the set that contains *a line segment*; see the left top in Fig. 2. Again this is obviously a convex set. On the other hand, a broken line, a line that is broken in a middle, is not convex, since some linear combination of two points in the broken line may fall into to somewhere in the broken place; see the right top in Fig. 2.

More examples: Polygon, polyhedron, polytope, ...

You may wonder if there are any other examples beyond point/line/plane. Of course, there are many. One object that you may be interested in is: a *polygon* living in a 2-dimensional space. In particular, the closed polygon in which points inside (and boundary at) the polygon are included in the set (see the left bottom in Fig. 2 for illustration) is a convex set. On the other hand, the boundary-only polygon (see the right bottom in Fig. 2) is not a convex set.

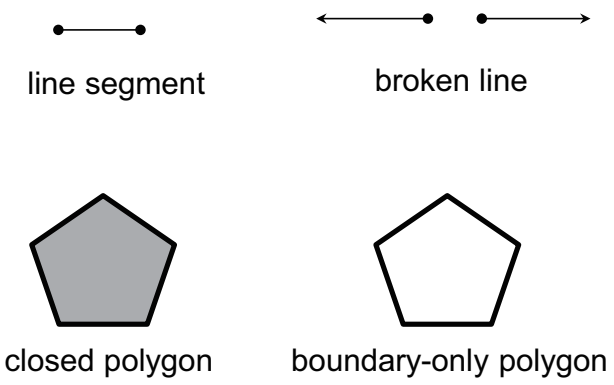


Figure 2: Examples of convex sets (left two) and non-convex sets (right two).

As you may imagine, representation of the closed-polygon convex set is different from the form (7) of affine sets. The closed-polygon can actually be represented as the *intersection* of half-planes, each being represented by $a_i^T x - b_i \leq 0$. Hence, the representation of such set reads:

$$S = \{x : Ax - b \leq 0\}. \tag{8}$$

where the inequality indicates a component-wise inequality.

Similarly, a *polyhedron* living in a 3-dimensional ambient space (or a *polytope* living in a d-dimension space and hence difficult to visualize) is a convex set and can also be represented as the form like (8).

Convex function

We are now ready to define the convex function. A function $f(x)$ is said to be *convex* if the following two conditions are satisfied:

(i) the domain of the function f, denoted by $\mathbf{dom}f$ (the set in which the input x of the function lies in), is a convex set; and

(ii) for $x, y \in \mathbf{dom}f$ and $\lambda \in [0, 1]$:

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y). \tag{9}$$

Here you can see why we needed to know about the concept of the *convex set*. The concept appears while mentioning the first condition that $\mathbf{dom}f$ should satisfy. Actually this "convex set" condition is required; otherwise, we have a problem when it comes to stating the second key condition in (9), because the function in the left hand side cannot be defined. Notice that the input argument in the function is $\lambda x + (1 - \lambda)y$ and this should be in $\mathbf{dom}f$ (meaning that $\mathbf{dom}f$ should be convex) - otherwise, $f(\lambda x + (1 - \lambda)y)$ cannot be defined.

If you think about some picture that reflects the second key condition (9), then you can readily get a feel about why the convex function should be defined in such a manner. Actually the meaning of "convex" is "bowl-shaped". So we can think about a bowl-shaped curve like the one illustrated in Fig. 3. Now let's consider two points, say x and y, and a λ-weighted linear combination, $\lambda x + (1-\lambda)y$. The function evaluated at $\lambda x + (1-\lambda)y$ is on the bowl-shaped curve while the same-weighted linear combination $\lambda f(x) + (1-\lambda)f(y)$ of the two functions evaluated at x and y is *above* $f(\lambda x + (1-\lambda)y)$. Hence, the key condition (9) comes naturally as a consequence of the bowl-shaped feature of the curve.

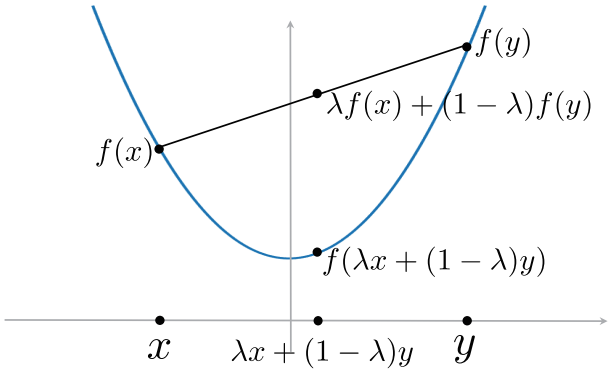


Figure 3: Geometric intuition behind convex functions.

There are tons of examples of convex functions and also many of these are the ones that you should be familiar with if you wish to be an expert on this field. Or at least you may want to know some of them in which problems of your interest can be linked to. But exploring many of such examples may be too much now - it is just the beginning of the course, so that way you will be exhausted shortly. Thus we will here investigate only a couple of examples. One example of a convex function is:

$$f(x) = \frac{1}{x} \qquad x > 0. \tag{10}$$

Here the function is like bowl-shaped, so it respects the second condition (9). Also $\mathbf{dom} f = \{x : x > 0\}$ is a convex set, satisfying the first condition. Hence, the function is convex.

Now what about a slightly different function:

$$f(x) = \frac{1}{x}? \tag{11}$$

Here the distinction is that $\mathsf{dom} f$ is not explicitly defined. In this case, we should think about an *implicit* constraint that x should satisfy. The implicit constraint is: $x \neq 0$, thus yielding:

$$\mathbf{dom} f = (-\infty, 0) \cup (0, \infty).$$

Since $\mathbf{dom} f$ is not convex, the function is not convex either.

Actually there is a way to handle this issue to make the bowl-shaped function convex. The way is to make $\mathbf{dom} f$ span the entire region (making it convex) while setting the function to some arbitrary quantities for newly added regions. For instance, we can define the function as:

$$f(x) = \begin{cases} \frac{1}{x}, & x > 0; \\ +\infty, & x \leq 0. \end{cases} \tag{12}$$

Notice that now $\mathbf{dom} = (-\infty, \infty)$ is a convex set while $f(x)$ is still being bowl-shaped. Hence, the function is convex.

There is another function which is defined very similarly to the convex function. That is, a *concave function*. We say that $f(x)$ is *concave* if $-f(x)$ is convex. The geometric intuition says that the function of a bell shape is concave. Also a function is said to be *affine* (linear plus bias) if it is convex and concave.

Convex function and convex set

Previously we investigated examples of convex sets where only affine functions are introduced. Actually there are many convex sets which concern convex functions. Here we list a couple of such examples.

One such example is:

$$S = \{x : f(x) \leq 0\} \tag{13}$$

where $f(x)$ is a convex function. Here is the proof that S is a convex set. Suppose $x, y \in S$. Then, $f(x) \leq 0$ and $f(y) \leq 0$. This together with the convexity of f, reflected in the condition (9), gives:

$$f(\lambda x + (1-\lambda)y) \leq 0,$$

which in turn implies that $\lambda x + (1-\lambda)y \in S$. This completes the proof.

Another example is the intersection of such convex sets:

$$\begin{aligned} &S = S_1 \cap S_2 \\ &S_1 = \{x : f_1(x) \leq 0\}, \;\; S_2 = \{x : f_2(x) \leq 0\}. \end{aligned} \tag{14}$$

Try the proof in Problem Set (PS) 1. Actually the intersection of arbitrary convex sets is also convex - check in PS1 as well.

Convex optimization problem in standard form

We are now ready to define the convex optimization problem. It is an optimization problem which satisfies the following three: (i) The objective function is convex; (ii) The set induced by inequality constraints is convex; and (iii) The set induced by equality constraints is convex. So the standard form of the convex optimization problem is (2) in which (i) $f(x)$ is convex; (ii) $f_i(x)$ is convex; and (iii) $h_i(x)$ is affine. Notice that the set induced by affine equality constraints $\mathcal{S} = \{x : Ax - b = 0\}$ is a convex set as we studied earlier.

Look ahead

Of course there must be a reason why the convex optimization problem is defined in such a manner. This is because the way of definition makes the problem *tractable*. Next time, we will provide an intuition as to why convex optimization is tractable. We will then start investigating one instance of convex optimization: Linear Program (LP).

Lecture 3: Tractability of Convex Optimization and Introduction to Linear Program

Recap

Last time, we studied the concept of convex sets and convex functions to come up with a standard form of convex optimization problems. In words, convex optimization is a problem in which the objective function is convex and the set induced by inequality and equality constraints (that we often call the *feasible set*) is convex. In terms of mathematical notations, it is formulated as:

$$\begin{aligned} \min f(x): \ & f_i(x) \leq 0, i = 1, \ldots, m, \\ & h_i(x) = 0, i = 1, \ldots, p \end{aligned} \tag{1}$$

where $f(x)$ and $f_i(x)$'s are convex and $h_i(x)$'s are affine functions. And then I told you that there is a reason that many people have been interested in such convex optimization defined as above. The reason is that the way of defining the problem makes the problem *tractable*. Here what it means by *tractable* is that the optimal solution can be achieved via an *algorithm* (with the help of a computer) even if the closed form solution is unknown.

Today's lecture

The main goal of today's lecture is to provide a rationale behind the claim regarding tractability. To this end, I will provide you with an explanation as to why convex optimization is tractable.

Specifically we will deal with two cases: (i) *unconstrained* minimization; (ii) *constrained* minimization. For the unconstrained case, we will first derive a simple optimality condition and then demonstrate that the condition naturally leads to efficient algorithms. For the constrained case, we will also characterize an optimality condition which turns out to shed lights into efficient algorithms. While investigating the two cases, we will assume that: (i) the objective function $f(x)$ is *differentiable* at every point x in $\mathbf{dom} f$; (ii) the domain is open.

Another goal of this lecture is to give an overview to the contents regarding one instance of convex optimization problems: Linear Program (LP). This is what we will cover through a couple of upcoming lectures.

Unconstrained minimization

Let us start by investigating the unconstrained convex optimization problem:

$$\min f(x).$$

Recall the meaning of *convex*. It means "bowl-shaped". So one can think of a graph illustrated in Fig. 1. Here you can easily see that at the optimal point x^*, the slope of the objective function is 0, and also vice versa (meaning that the point in which the slope of the function is 0 is the optimal solution). This naturally leads us to conjecture that $\nabla f(x^*) = 0$ is a sufficient and necessary condition for x^* to be optimal:

$$\nabla f(x^*) = 0 \longleftrightarrow f(x) \geq f(x^*) \quad \forall x \in \mathbf{dom} f. \tag{2}$$

It turns out this conjecture is indeed the case. Here is the proof.

Proof of the direct part $\longrightarrow$ *in* (2)*:* To gain some insights, let us see a convex function $f(x)$ in Fig. 1. Pick up a point $(x^*, f(x^*))$. Now consider a line that crosses the point $(x^*, f(x^*))$ with a slope $\nabla f(x^*)$ so that it is tangent to $f(x)$. Then, the line should read: $\nabla f(x^*)^T(x-x^*)+f(x^*)$. Here the picture suggests that the convex function $f(x)$ is above (or touching) the line:

$$f(x) \geq \nabla f(x^*)^T(x - x^*) + f(x^*) \quad \forall x \in \mathbf{dom} f. \tag{3}$$

It turns out this is indeed the case, meaning that the condition (3) (together with $\mathbf{dom} f$ being convex) holds if and only if $f(x)$ is convex. Actually this is one of the crucial properties of convex functions, called the "1st order condition of convex functions". The proof of this is omitted here, but you will have a chance to prove this in PS1. Now this together with the hypothesis gives: $f(x) \geq f(x^*)$, $\forall x \in \mathbf{dom} f$.

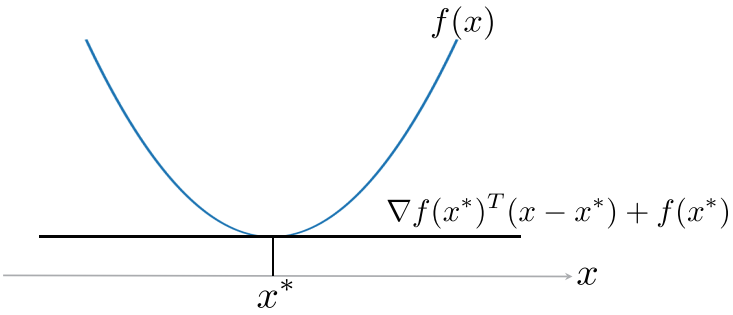


Figure 1: 1st order condition of convex functions: $f(x) \geq \nabla f(x^*)^T(x-x^*)+f(x^*)$, $\forall x \in \mathbf{dom} f$.

Proof of the converse part $\longleftarrow$ *in* (2)*:* The converse proof relie on the following fact (which we will prove once we are done with this converse proof):

$$f(x) \geq f(x^*) \;\; \forall x \in \mathbf{dom} f \longrightarrow \nabla f(x^*)^T(x - x^*) \geq 0 \;\; \forall x \in \mathbf{dom} f. \tag{4}$$

Now suppose $\nabla f(x^*) \neq 0$. Here the key thing to note is that there is no constraint on x, except that $x \in \mathbf{dom} f$. So one can choose x such that $x - x^*$ points to an *arbitrary* direction. This implies that we can easily choose x such that

$$\nabla f(x^*)^T(x - x^*) < 0. \tag{5}$$

This contradicts with $\nabla f(x^*) = 0$, thus completing the proof.

Let us now prove (4) which I deferred proving earlier.

Proof of (4)*:* The proof idea is by contradiction. Suppose that there exists x (i.e., $\exists x$) $\in \mathbf{dom} f$ such that:

$$\nabla f(x^*)^T(x - x^*) < 0. \tag{6}$$

Now consider a point: $z(\lambda) := \lambda x + (1 - \lambda)x^*$ where $\lambda \in [0, 1]$. Notice that $z(\lambda) \in \mathbf{dom} f$, as the function f is convex and therefore its domain is a convex set. Here what we want to show is that for a very small $\lambda \approx 0$, $f(z(\lambda)) < f(x^*)$. This is because $f(z(\lambda)) < f(x^*)$ contracts with the fact that x^* is an optimal solution, thus leading to contradiction. To show this, we consider the following quantity:

$$\begin{aligned} \frac{d}{d\lambda} f(z(\lambda)) &\overset{(a)}{=} \nabla f(z(\lambda))^T \frac{d}{d\lambda} z(\lambda) \\ &\overset{(b)}{=} \nabla f(z(\lambda))^T (x - x^*) \end{aligned}$$

where (a) follows from a chain rule (that you learned from vector calculus); and (b) is due to the definition of $z(\lambda) := \lambda x + (1-\lambda)x^*$. Now evaluating both sides at $\lambda = 0$, we get:

$$\frac{d}{d\lambda} f(z(\lambda))\Big|_{\lambda=0} = \nabla f(x^*)^T (x - x^*) < 0 \tag{7}$$

where the last inequality comes from our assumption (6). Here the derivative of $f(z(\lambda))$ being negative at $\lambda = 0$ implies that $f(z(\lambda))$ *decreases with* λ and therefore:

$$f(z(\lambda)) < f(x^*). \tag{8}$$

This contradicts with the hypothesis $f(x) \geq f(x^*)\ \forall x \in \mathbf{dom} f$. This completes the proof.

Gradient decent algorithm

So what we can conclude with respect to (w.r.t.) unconstrained minimization is that:

$$\nabla f(x^*) = 0 \text{ is a sufficient and necessary condition for } x^* \text{ to be optimal.}$$

This suggests that it suffices to find a point such that (s.t.) its gradient is 0. But there are some issues in deriving such point. Two issues. One is that computing $\nabla f(x)$ may not be that simple. The second is that analytically finding such point may not be doable even if one can explicitly derive the gradient. However, there is a good news. The good news is that there developed several algorithms which allow us to find such point efficiently without the knowledge of the closed form solution. One prominent algorithm that has been widely used in a variety of fields is: the *gradient decent algorithm*.

Here is how the algorithm works. The gradient decent algorithm is an *iterative* algorithm. Suppose that at the t-th iteration, we have an estimate of x^*, say $x^{(t)}$. We then compute the gradient of the function evaluated at the estimate: $\nabla f(x^{(t)})$. Next we update the estimate along a direction being *opposite* to the direction of the gradient:

$$x^{(t+1)} \longleftarrow x^{(t)} - \alpha^{(t)} \nabla f(x^{(t)}) \tag{9}$$

where $\alpha^{(t)} > 0$ indicates a step size which usually decays with t, e.g., $\alpha^{(t)} = \frac{1}{2^t}$. If you think about it, this update rule makes sense. Suppose $x^{(t)}$ is placed right relative to the optimal point x^*, as illustrated in Fig. 2.

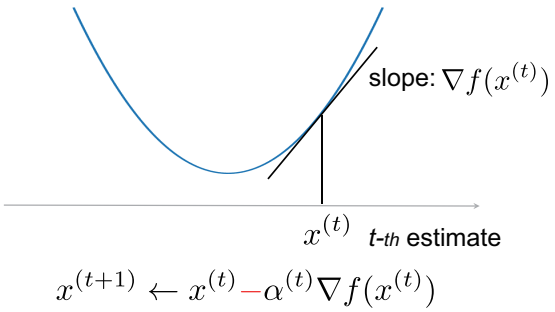


Figure 2: Gradient decent algorithm.

Then, we should move $x^{(t)}$ to the *left* so that it is closer to x^*. The update rule actually does this, as we *subtract* by $\alpha^{(t)} \nabla f(x^{(t)})$. Notice that $\nabla f(x^{(t)})$ points to the *right* direction given

that $x^{(t)}$ is placed right relative to x^*. We repeat this procedure until it converges. It turns out: as $t \to \infty$, it actually converges:

$$x^{(t)} \longrightarrow x^*, \tag{10}$$

as long as the step size is chosen properly, like the one delaying exponentially. We will not touch upon the proof of this convergence. We have a good excuse - it is out of the scope of this course. Actually the proof is not that simple - even there is a big field in statistics which intends to prove the convergence of a variety of algorithms (if it is the case).

Constrained minimization: Case I

Now let us consider the constrained minimization:

$$\begin{aligned} \min f(x): \quad & f_i(x) \leq 0, \quad i = 1, \ldots, m, \\ & h_i(x) = 0, \quad i = 1, \ldots, p. \end{aligned} \tag{11}$$

Before getting into detail, let us first introduce one terminology. We say that the set in which x satisfies all the constraints is *feasible*. For convex optimization of our interest, the feasible set, say $\mathcal{S}$, is *convex* because we put conditions to the functions such that the set induced by inequality and equality constraints is convex.

It turns out that for constrained minimization where x^* is the optimal solution, there is a simple sufficient/necessary condition for x^* to be optimal; and the condition gives insights into developing efficient algorithms that achieve the solution. In other words, the optimality condition plays a significant role in making convex optimization *tractable*.

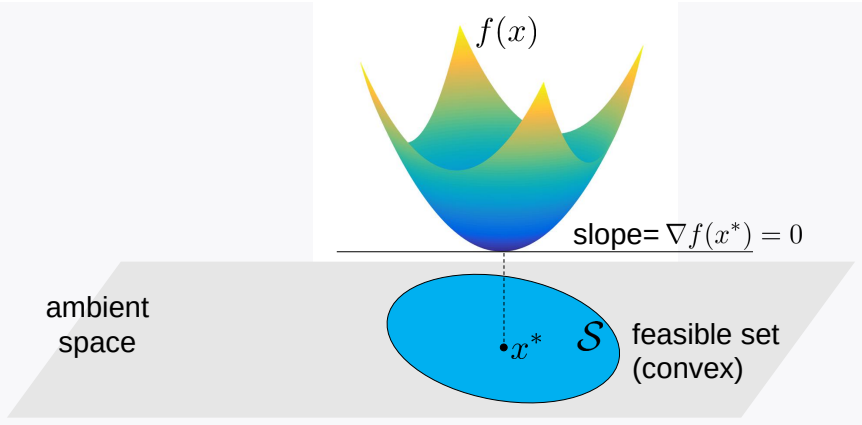


Figure 3: Case I: There exists $x^* \in \mathcal{S}$ (feasible set) such that $\nabla f(x^*) = 0$.

Now what is the optimality condition? We will consider two cases to investigate the condition. The first is a case in which $\exists x^* \in \mathcal{S}$ such that $f(x^*) = 0$. See Fig. 3. Recall in the *unconstrained case* that (see (2)):

$$\nabla f(x^*) = 0 \longleftrightarrow f(x) \geq f(x^*) \quad \forall x \in \mathbf{dom} f. \tag{12}$$

Since the above holds for $x \in \mathbf{dom} f$, it should also hold for a more restricted set $\mathcal{S}$:

$$\nabla f(x^*) = 0 \longleftrightarrow f(x) \geq f(x^*) \quad \forall x \in \mathcal{S}. \tag{13}$$

So in this case, the optimality condition is identical to that in the unconstrained case. Hence, one can use the already-mentioned efficient algorithms (like the gradient decent algorithm) for achieving the optimal solution.

Constrained minimization: Case II

The second is a case in which $\nexists x^* \in \mathcal{S}$ (feasible set) such that $\nabla f(x^*) = 0$. In this case, obviously:

$$\nabla f(x) \neq 0 \;\; \forall x \in \mathcal{S}. \tag{14}$$

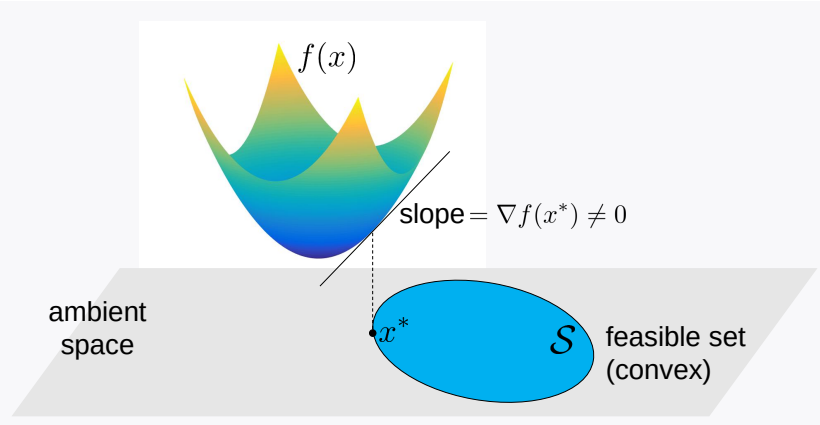


Figure 4: Case II: There does not exist $x^* \in \mathcal{S}$ (feasible set) such that $\nabla f(x^*) = 0$.

Recall the 1st-order condition of convex functions (3):

$$f(x) \geq \nabla f(x^*)^T (x - x^*) + f(x^*) \quad \forall x \in \mathbf{dom} f.$$

From this, one can see that:

$$\nabla f(x^*)^T (x - x^*) \;\; \forall x \in \mathbf{dom} f \geq 0 \longrightarrow f(x) \geq f(x^*) \;\; \forall x \in \mathbf{dom} f, \tag{15}$$

meaning that $\nabla f(x^*)^T (x - x^*)$ is a *sufficient* condition for x^* to be optimal. It turns out that the converse of (15) also holds, i.e., $\nabla f(x^*)^T (x - x^*)$ is also *necessary*. To see this, remember what we proved earlier (see (4)):

$$f(x) \geq f(x^*) \;\; \forall x \in \mathbf{dom} f \longrightarrow \nabla f(x^*)^T (x - x^*) \geq 0 \;\; \forall x \in \mathbf{dom} f. \tag{16}$$

Here one key point to note is that $\mathbf{dom} f$ is a convex set (why?). Since the feasible set $\mathcal{S}$ is convex, the proof of (16) is exactly the same as in the constrained case concerning $\mathcal{S}$. Remember in the proof of (4) that we only used the fact that $\mathbf{dom} f$ is convex. Hence, the converse of (15) holds, thus yielding:

$$\nabla f(x^*)^T (x - x^*) \geq 0 \text{ is a sufficient/necessary condition for } x^* \text{ to be optimal.} \tag{17}$$

Efficient algorithms

Actually the optimality condition stated in (17) forms the basis of *strong duality* that we will learn in Part II in a few weeks. As I mentioned in Lecture 1, the strong duality gives insights

into developing efficient algorithms that achieve the optimal solution. This will be discussed in depth later in Part II. Hence, we have efficient algorithms both for unconstrained and constrained convex optimization, suggesting that convex optimization is tractable.

Overview of Linear Program (LP)

From now on, we will start investigating several instances of convex optimization problems. One instance that we will take first is: Linear Program (LP).

$$\begin{aligned} \min f(x): \quad & f_i(x) \leq 0, \quad i = 1, \ldots, m, \\ & h_i(x) = 0, \quad i = 1, \ldots, p. \end{aligned} \tag{18}$$

Here we say that (18) is an LP if all functions $f(x), f_i(x)$'s, $h_i(x)$'s are *affine*.

Since Kantorovich's breakthrough, people realized that many interesting/important problems can be formulated as LPs such as: (i) resource allocation problems (like the military-related problem that Kantorovich considered); (ii) transportation problems (important problems in economics); (iii) the linear classification problem (one of the most classical and popular problems in machine learning); (iv) the network flow problem (a fundamental problem in computer networks); and so on and so forth.

Moreover, some very difficult problems in which the optimization variable is boolean (binary values) can be approximated as an LP via a relaxation technique. Very interestingly, in some cases, the LP relaxation provides the *exact* solution, i.e., coming without loss of optimality.

Look ahead

So through a couple of upcoming lectures, we will deal with the above examples together with algorithms and software implementation. Specifically we are going to cover four stuffs: (i) Will study a few examples that can be formulated as LPs; (ii) Will study the LP relaxation technique useful for some very difficult problems; (iii) Will investigate efficient algorithms; (iv) Will study how to implement such algorithms using software like CVX running in MATLAB.

Lecture 4: Examples of LP

Recap

Last time, we tried to understand why convex optimization problems, stated below, are *tractable*, i.e., can be solved efficiently on a computer even if the closed form solution is unknown:

$$\begin{aligned} \min f(x): \ & f_i(x) \leq 0, i = 1, \ldots, m, \\ & h_i(x) = 0, i = 1, \ldots, p \end{aligned} \tag{1}$$

where $f(x)$ and $f_i(x)$'s are convex and $h_i(x)$'s are affine functions. To this end, specifically we considered two cases: (i) unconstrained case; (ii) constrained case. For unconstrained minimization, we first derived the optimality condition for x^* to be optimal: $\nabla f(x^*) = 0$ (assuming that f is differentiable at every point $\in \mathbf{dom} f$, which is open) and then investigated one efficient algorithm, the *gradient decent algorithm*, which allows us to achieve such x^* in an efficient and iterative manner. For constrained minimization, we also derived the optimality condition which turns out to be: for optimal x^*,

$$\nabla f(x^*)(x - x^*) \geq 0 \quad \forall x \in \mathcal{S}, \tag{2}$$

where $\mathcal{S}$ denotes the feasible set. And then I told you that this condition forms the basis of *strong duality*, which turns out to give insights into developing efficient algorithms. This is how we understood as to why convex optimization problems are tractable.

We then moved onto one simple instance of convex optimization problems: Linear Program (LP), which has a standard form as:

$$\begin{aligned} \min w^T x: \ & Ax - b \leq 0, \\ & Cx - e = 0 \end{aligned} \tag{3}$$

where w, A, b, C and e are of compatible size and the inequality is component-wise one. At the end of the last lecture, we then claimed that many interesting and important problems can be translated into LPs.

Today's lecture

The goal of today's lecture is to try to prove the claim. To this end, we will study three prominent examples which can be translated into LPs. The first is a resource allocation problem which is also called an *optimal planning problem*. Actually this has been the most important problem in economics and operation research in the 20th century. Specifically we will investigate a historical problem (explored by the inventor of LP, Leonid Kantorovich), which later gave inspirations to the development of LP. The second is a transportation problem which has been playing a crucial role in a variety of fields for centuries. Specifically we will study a problem explored by the *Farther of Transportation Theory*, Gaspard Monge, a French mathematician in the 18th century. If time permits, we will lastly study the most classical problem in machine learning: the *linear classification problem*. While investigating these examples, we will learn a couple of translation techniques for LP: (i) how to express conditions (given in a problem) in terms of vector and matrix notations; (ii) how to set up a proper optimization variable that yields a LP

formulation; (ii) how to make a convex function into an affine function (of interest) by inducing some additional constraints.

Kantorovich's plywood cutting problem

One of the problems that Kantorovich considered is the *plywood cutting problem*. Actually Kantorovich encountered the problem in 1937 while interacting with plywood engineers. Just for simplicity, here we consider a much simpler version of the original problem.

The problem is about allocating the time for the use of different machines for peeling different kinds of woods. Suppose there are two kinds of woods to peel, say wood 1 and wood 2. Also there are two different peeling machines: machine 1 and machine 2. Each machine has a different capability for peeling. Machine 1 can peel 10 units/time for either type of wood. On the other hand, machine 2 can peel 20 units/time for wood 1 while peeling 40 units/time for wood 2. See Fig. 1.

	wood 1	wood 2
machine 1	10 units/time	10
machine 2	20	40

Figure 1: Machine capabilities for peeling woods.

Here the goal is to maximize the total wood production. But there is a constraint. The constraint is that the production is desired to meet the equal proportion i.e., the amount of wood 1 peeled is desired to be the same as that of wood 2. If there is a remnant part which exceeds the equal proportion, then it is simply discarded. So the objective is to maximize the *minimum* of wood 1 and 2 products:

$$\max \min \{\text{wood 1 product}, \text{wood 2 product}\}. \tag{4}$$

Now what is an optimization variable? In other words, what is a quantity that we can control over to affect the objective function? That is, the time that we use for peeling each wood with a certain machine. Specifically let x_1 be machine 1's time for peeling wood 1. Normalizing the time, we can assume that $0 \le x_1 \le 1$. Assuming that machines are always operating, machine 1's time for wood 2 would be $1 - x_1$. Similarly define $0 \le x_2 \le 1$ as machine 2's time for peeling wood 1. Using these notations together with machine capabilities illustrated in Fig. 1, we get: wood 1 product $= 10x_1 + 20x_2$ (units/time); wood 2 product $= 10(1 - x_1) + 40(1 - x_2)$ (units/time). Now applying this to (4) and flipping the sign of the objective function, we obtain a minimization problem (so as to respect the standard form) as follows:

$$\begin{aligned} \min \max \{-10x_1 - 20x_2, 10(x_1 - 1) + 40(x_2 - 1)\} : \\ 0 \le x_1 \le 1,\ 0 \le x_2 \le 1. \end{aligned} \tag{5}$$

Translation to an LP

Notice in (5) that the objective function $\max\{\cdot, \cdot\}$ (marked in red) is convex (why? check in PS1), but it is *not affine*, which we wish to obtain for an LP formulation. This is exactly where one important translation technique kicks in. One technique that allows us to convert such a convex function into an affine function is the following: introducing another variable, say x_3, and letting:

$$x_3 := \max \{-10x_1 - 20x_2, 10(x_1 - 1) + 40(x_2 - 1)\}. \tag{6}$$

This new variable then induces additional constraints:

$$x_3 \geq -10x_1 - 20x_2, \;\; x_3 \geq 10(x_1 - 1) + 40(x_2 - 1). \tag{7}$$

Using these additional constraints together with the new variable x_3, we can then re-write (5) as:

$$\begin{aligned} &\min x_3 : \\ &0 \leq x_1 \leq 1, \;\; 0 \leq x_2 \leq 1, \\ &-10x_1 - 20x_2 - x_3 \leq 0, \\ &10x_1 + 40x_2 - x_3 - 50 \leq 0. \end{aligned} \tag{8}$$

Note that the objective function and all of the functions that appear in inequality constraints are affine. Hence, the problem is an LP. Using vector/matrix notations, we can also represent this in the following standard form:

$$\min w^T x : \;\; Ax - b \leq 0 \tag{9}$$

where:

$$w = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, A = \begin{bmatrix} -1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \\ -10 & -20 & -1 \\ 10 & 40 & -1 \end{bmatrix}, b = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 50 \end{bmatrix}. \tag{10}$$

Observe that the first rows of A and b come from $-x_1 \leq 0$, and the second rows are due to $x_1 - 1 \leq 0$, and so on and so forth.

Monge's transportation problem

The second problem that we will study is another historical problem: *Monge's problem*, explored by Gaspard Monge, a French mathematician who lived in the 18th century. In 1781, he published a memoir, titled: *Mémoire sur la théorie des déblais et des remblais.* In the memoir, he introduced a transportation problem which later laid the foundation of *transportation theory.* In particular, the field of transportation theory was revolutionized in the 20th century by the recognition that the Monge's transportation problem can be translated into an LP. Actually this recognition was made by again the famous Kantorovich. So here we will see how Kantorovich recognized it as an LP.

Monge's problem is about transporting soils (mined in several grounds places) into construction sites, each of which demands a certain amount of soils for construction purpose. For instance, let us consider an example illustrated in Fig. 2. Suppose there are three grounds places (marked in black squares) and four construction sites (marked in hallowed circles). For each ground, a certain amount of soils can be mined. Let s_i be the amount of soiled mined in ground i, $i = 1, 2, 3$. For simplicity, we assume that s_i's are normalized such that $s_1 + s_2 + s_3 = 1$. Let d_j indicate the amount of soils demanded at construction j, $j = 1, 2, 3, 4$. Assume that the total demand is the same as the total supply. Then, we have: $d_1 + d_2 + d_3 + d_4 = s_1 + s_2 + s_3 = 1$.

Now the goal of the problem is to find an optimal *coupling* such that the *transportation cost* is minimized. To figure out how to achieve this, we first need to understand how the transportation cost is determined. Here we assume that the cost is proportional to two factors: (i) distance between a ground and a construction site to which soils are sent from the ground; (ii) the amount

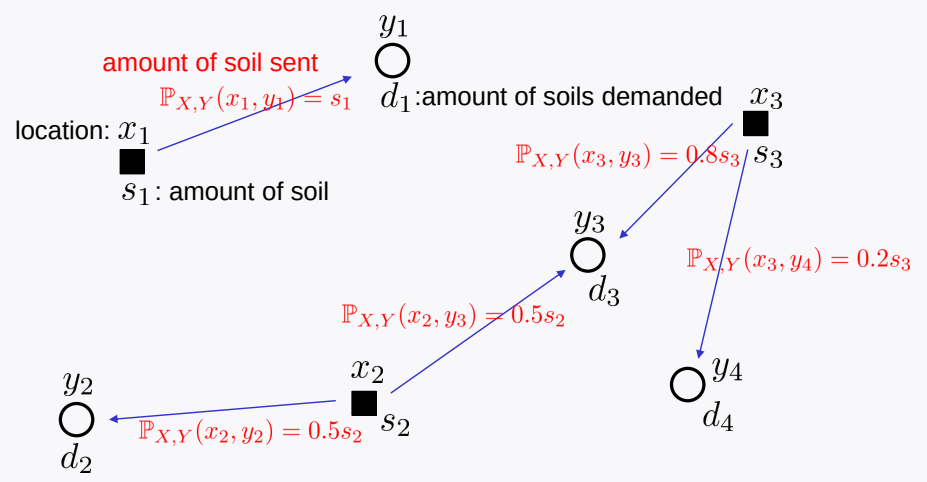


Figure 2: Monge's transportation problem.

of the soils sent. To quantify the distance, we need to define coordinates of locations of grounds and constructions. Let x_i and y_j denote location coordinates of ground i and construction j, respectively, where $i \in \{1, 2, 3\}$ and $j \in \{1, 2, 3, 4\}$. Then, the distance between ground i and construction j can be written as:

$$\mathsf{dist}(\text{ground } i, \text{construction } j) = \|x_i - y_j\| \tag{11}$$

where $\|\cdot\|$ denotes the Euclidean distance.

Now how to represent the amount of soils delivered from ground i to construction j? For ease of illustration, let us consider a particular *coupling* illustrated in Fig. 2. This coupling suggests that all the soils from ground 1 are transmitted only to construction 1. So the amount of soils must be s_1. Let's denote that by $\mathbb{P}_{X,Y}(x_1, y_1) = s_1$. Later you will see why I use this complicated-looking notation $\mathbb{P}_{X,Y}(\cdot, \cdot)$. Please be patient until we get to the point. For ground 2, the soils are split into two construction sites: construction 2 and 3. Let's assume the equal split. Then, we can represent the splitting by $\mathbb{P}_{X,Y}(x_2, y_2) = 0.5s_2$ and $\mathbb{P}_{X,Y}(x_2, y_3) = 0.5s_2$. Similarly for ground 3, the soils are split into construction 3 and 4, but with an asymmetric split, say 8:2. So we have $\mathbb{P}_{X,Y}(x_3, y_3) = 0.8s_3$ and $\mathbb{P}_{X,Y}(x_3, y_4) = 0.2s_3$. Notice that the soil allocation, determined by the values of $\mathbb{P}_{X,Y}(x_i, y_j)$'s, is the one that we can control over. So this is an *optimization variable*. It is a 12-dimensional vector in the example.

Next let's think about constraints posed in the problem. The constraints are two-folded: (i) all the soils mined in each ground should be transmitted to construction sites; (ii) the demands of all the constructions should be satisfied. In terms of mathematical notations, this means that:

$$\sum_{j=1}^{4} \mathbb{P}_{X,Y}(x_i, y_j) = s_i \qquad i = 1, 2, 3, \tag{12}$$

$$\sum_{i=1}^{3} \mathbb{P}_{X,Y}(x_i, y_j) = d_j \qquad j = 1, 2, 3, 4. \tag{13}$$

We can then write down the optimization problem as: Given (s_i, d_j)'s,

$$\begin{aligned} \min \sum_{i=1}^{3}\sum_{j=1}^{4} \underbrace{\mathbb{P}_{X,Y}(x_i, y_j)}_{\text{amount of soils}} \cdot \underbrace{\|x_i - y_j\|}_{\text{distance}} : \\ \sum_{j=1}^{4} \mathbb{P}_{X,Y}(x_i, y_j) = s_i \quad i = 1, 2, 3, \\ \sum_{i=1}^{3} \mathbb{P}_{X,Y}(x_i, y_j) = d_j \quad j = 1, 2, 3, 4. \end{aligned} \tag{14}$$

Notice that the objective function and all the functions that appear in the constraints are *affine* w.r.t. the optimization variable $\mathbb{P}_{X,Y}(x_i, y_j)$. Hence, it is an LP.

Wasserstein distance

If you think about the formula of (14), we can see that this can be succinctly represented with *probability distributions*. Remember that s_i's and d_j's are normalized: $s_1 + s_2 + s_3 = 1$ and $d_1 + d_2 + d_3 + d_4 = 1$. So one can view this as a valid probability distribution. For example, defining $\mathbb{P}_X(x_i) := s_i$, we see that $\mathbb{P}_X(x_i)$ is a probability distribution. Similarly, $\mathbb{P}_Y(y_j) := d_j$ is another valid probability distribution.

Keeping these in our mind, let us take a look at the constraints in the above optimization problem (14). What do the constraints remind you of? They remind you of the *total probability law*! Hence, one can view $\mathbb{P}_{X,Y}(x_i, y_j)$ as another valid probability distribution. This probabilistic viewpoint then allows us to simplify the optimization problem (14) as: Given $(\mathbb{P}_X, \mathbb{P}_Y)$,

$$W(\mathbb{P}_X, \mathbb{P}_Y) := \min_{\mathbb{P}_{X,Y}} \mathbb{E}\left[\|X - Y\|\right], \tag{15}$$

where the minimization is over all joint distributions $\mathbb{P}_{X,Y}$ which respect marginals:

$$\begin{aligned} \sum_{y_j} \mathbb{P}_{X,Y}(x_i, y_j) = \mathbb{P}_X(x_i) \quad \forall x_i, \\ \sum_{x_i} \mathbb{P}_{X,Y}(x_i, y_j) = \mathbb{P}_Y(y_j) \quad \forall y_j. \end{aligned} \tag{16}$$

Here $W(\mathbb{P}_X, \mathbb{P}_Y)$ is a function of $\mathbb{P}_X$ and $\mathbb{P}_Y$. So it can be interpreted as sort of *distance* between the two distributions. This succinct expression (15) was recognized by Kantorovich and other person, named Rubinstein. So it is called the *Kantorovich-Rubinstein distance*. Actually the distance measure was generalized later by incorporating an arbitrary pth-order exponent in $\|X - Y\|$ (i.e., $\|X - Y\|^p$). The general measure concerning $\|X - Y\|^p$ is called the pth-order Wasserstein distance. So $W(\mathbb{P}_X, \mathbb{P}_Y)$ is called the 1st-order Wasserstein distance.

It turns out the Wasserstein distance appears in many of the optimal transportation problems as a key measure, thus revolutionizing the field of transportation theory. Very interestingly, the Wasserstein distance played a crucial role in designing a famous machine learning model, called Generative Adversarial Networks (GANs), thus leading to the development of Wasserstein GAN that has been proved powerful in many applications. We will investigate details on this in Part III of this course.

Look ahead

Due to the interest of time, we stop here. We will continue to study the last example from the next lecture on. We will also do what we were planning to do for LP: Studying an LP relaxation technique which turns out to be very useful for some very difficult problems.

Lecture 5: Examples and LP Relaxation

Recap

Last time, we explored two historical examples, which can be be translated into LPs and played a big role in the fields like economics, operation research and transportation:

1. Kantorovich's plywood cutting problem;
2. Monge's transportation problem.

In the process, we also learned about a couple of techniques which allow us to translate problems into LPs, and also gained some insights as to how to recognize LPs. However, due to the interest of time, we could not complete the contents that we were planning to cover in the lecture.

Today's lecture

Today we will first complete the remaining action item: studying the last example,

3. Linear classification problem,

which is arguably the most classical problem that arises in machine learning. Next we will deal with another claim that I made in Lecture 3: Some very difficult problems can be solved via an LP relaxation technique. Specifically we will study a couple of examples in which the LP relaxation can provide the exact solutions in some cases.

Linear classification problem

The last example that we planned to touch upon is a very popular problem that arises in a wide variety of fields such as machine learning, artificial intelligence, finance, and real estates. In particular, it is the most classical and canonical problem in machine learning.

For illustration purpose, let us explain what the problem is for a very simple task setting: classifying legitimate emails against spam emails. Suppose there are two datasets. One dataset contains data points (also called *samples or examples* in the machine learning field) concerning *spam* emails. The other includes those concerning *legitimate* emails. Assume that each data point is represented by two *features*[1]: (i) frequency of keyword 1 (say, *dollar signs $$*); (ii) frequency of keyword 2 (say, *winner*). Then, each data point, say x_i, can be denoted by $x_i := (x_{i1}, x_{i2})$ where x_{i1} and x_{i2} indicate the two features: frequencies of keyword 1 and 2 contained in the ith email, respectively. See Fig. **??** for data points in two datasets, blue (legitimate) and red (spam) datasets. Here we are also given a *label* which indicates whether data point x_i is about a legitimate email ($y_i = 1$) or a spam email ($y_i = -1$). Assume that we have m such paired samples.

Now given these data points together with labels $\{(x_i, y_i)\}_{i=1}^m$, the goal of the linear classification problem is to find a *line* that separates two datasets. A line can be represented as a linear equation in the two-dimensional space: $a^T x - b = 0$. Note that $a^T x - b \geq 0$ is a half-space that

[1] In machine learning, the feature is a frequently used terminology which refers to a key component that well describes characteristics of data.

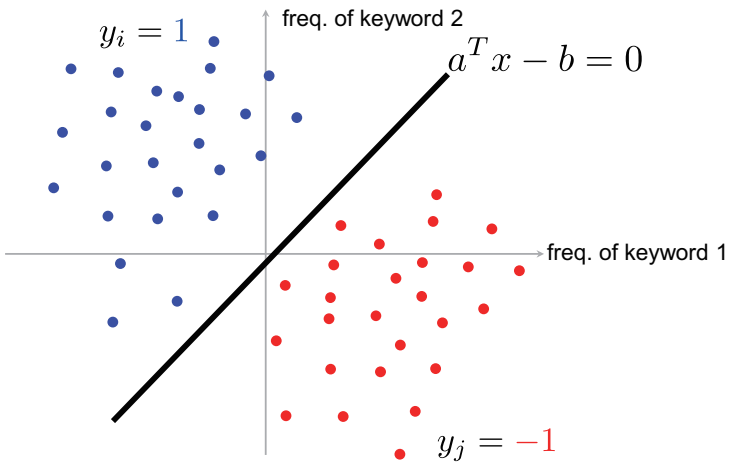


Figure 1: Two datasets: one for spam emails; the other for legitimate emails.

covers the region above the line (where the blue data points reside) while $a^T x - b \leq 0$ indicates a half-space spanning the bottom region (where the red data points reside). Hence, in order for a line to separate the two datasets, it must hold that:

$$y_i(a^T x_i - b) \geq 0 \quad i = 1, \ldots, m. \tag{1}$$

Here one thing to note is that a and b are the *optimization variable*, not (x_i, y_i)'s. You may be confused about (and possibly annoyed by) the notations because we have used x notation for the optimization variable. But here we use the x notation to indicate *data points*, which is a sort of convention in machine learning.

Notice in (**??**) that $(a, b) = (0, 0)$ always satisfies the constraint. However, it is obviously not of interest. Also one may want a *strict separability*, meaning that a strict inequality may be preferred in (**??**). These motivate us to consider the following slightly different constraint instead:

$$y_i(a^T x_i - b) \geq 1 \quad i = 1, \ldots, m. \tag{2}$$

Whenever, a strict inequality in (**??**), one can make the inequality (**??**) hold by properly scaling (a, b).

As long as the above constraints are satisfied for all data points (x_i, y_i)'s, then we are done, meaning that there is nothing to minimize or maximize. Hence, we can write the optimization problem as: Given $\{(x_i, y_i)\}_{i=1}^m$,

$$\begin{aligned} &\min_{a,b} 0 : \\ &y_i(a^T x_i - b) \geq 1 \quad i = 1, \ldots, m. \end{aligned} \tag{3}$$

Here we minimize a constant, say 0, since there is nothing to minimize. Observe that all the functions that appear are affine. Hence, it is an LP.

Non-separable case

Here you may wonder: What if datasets are *not linearly separable*, as illustrated in Fig. **??**? Notice in Fig. **??** that some red points reside near a cluster of blue points, and also some blue points are mingled with a majority of red points. Obviously there is no line that separates the

two datasets. Actually this non-separable case occurs in various tasks especially in computer vision. For instance, in the cat-dog classification problem, the boundary that separates cat dataset and dog dataset is usually highly non-linear.

One naive[2] yet natural way to handle this non-separable case is to introduce the concept of *margin*. For some outlier data points (x_i, y_i), we introduce margins, say $v_i \geq 0$, such that

$$y_i(a^T x_i - b) + v_i \geq 1 \quad i = 1, \ldots, m. \tag{4}$$

Whenever $y_i(a^T x_i - b)$ is strictly less than 1 (which is undesirable), we include a positive margin v_i so that the sum of them is greater than or equal to 1. Obviously the smaller the margin, the better the situation.

We can then set out our new goal as: minimizing the aggregated margins while respecting the above constraint (**??**). Hence, one can formulate the optimization problem as:

$$\begin{aligned} &\min_{a,b,v_i} \sum_{i=1}^{m} v_i : \\ &y_i(a^T x_i - b) + v_i \geq 1 \quad i = 1, \ldots, m, \\ &v_i \geq 0. \end{aligned} \tag{5}$$

Again this is still an LP.

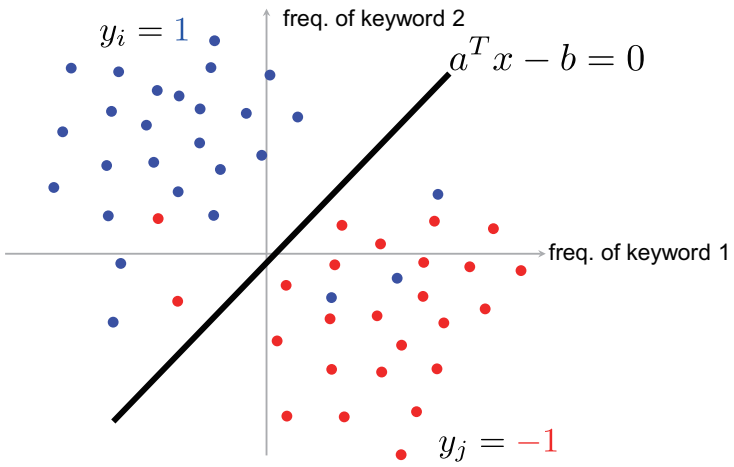


Figure 2: Non-separable case.

A class of difficult yet solvable problems (via LP relaxation)

Now what is next? Recall a claim that we made earlier: Some very difficult problems can be solved via an LP relaxation technique. So let us study the technique in the context of such

[2] A more powerful yet sophisticated way is to employ *deep neural networks* (see below for definition) that you may hear of very often. During the past decode, there has been a breakthrough in the history of machine learning. It has been shown that *deep neural networks* can well represent any *arbitrary* (possibly highly non-linear) functions with sort of reasonable computational complexity in view of current technologies. So one can use such network to implement a non-linear classifier to do much better. We will get to this point again in Part III of this course. *Neural networks* are systems with one or multiple layers in which each layer consists of an affine operation and an arbitrary (possibly non-linear) operation (called the activation function in the literature). Input and output in each layer are high-dimensional vectors and each component in the vectors is represented as a circle and called a neuron. The naming was originated from the fact that the structure looks like that of brain networks. Deep neural networks refer to the one with more than one layer.

difficult problem class. One such prominent class is: a class of *boolean problems*, which can be formally stated as below.

$$\begin{aligned} p^* := &\min w^T x : \\ &Ax - b \leq 0,\ Cx - e = 0, \\ &x_i \in \{0,1\},\ i = 1, \ldots, d. \end{aligned} \tag{6}$$

Here we see the last additional constraint, saying that the optimization variable is constrained to be *boolean*. We often use the following shorthand notation: $x \in \{0,1\}^d$. Actually there are many problems which can be formulated as above in the real world. To get some feeling, let us explore two very popular examples.

Example 1: Bipartite matching problem

The first problem is one of the fundamental problems in *combinatorial optimization*: the *bipartite matching problem*.

Let us start by explaining some terminologies required to understand what the problem is. Here one terminology that you may wonder about is *bipartite*. A very casual and non-precise definition is: a *graph with two parties*. In mathematics, what does it mean? To understand this, we need to first know about the concept of *graphs*. A graph, denoted by $\mathcal{G}$, is a collection of two sets: (i) a vertex (or node) set, denoted by $\mathcal{V}$; (ii) an edge set, denoted by $\mathcal{E}$. The vertex set includes many nodes indicating some objects of interest. The edge set includes many edges indicating some connections between two nodes. A *bipartite graph* is a special type of graph in which there are two *disjoint* sets such that: (i) there is no edge among nodes within each disjoint set; (ii) edges appear only across the disjoint sets. See Fig. **??** for an example. Here the vertex and edge sets are:

$$\begin{aligned} &\mathcal{V} = \{\text{person } 1, \ldots, \text{person } N, \text{task } 1, \ldots, \text{task } N\}; \\ &\mathcal{E} = \{(\text{person } 1, \text{task } 2), (\text{person } 2, \text{task } N), \ldots, (\text{person } N, \text{task } 1)\}. \end{aligned} \tag{7}$$

And the two disjoint sets are:

$$\begin{aligned} &\mathcal{V}_1 = \{\text{person } 1, \ldots, \text{person } N\}; \\ &\mathcal{V}_2 = \{\text{task } 1, \ldots, \text{task } N\}. \end{aligned} \tag{8}$$

We are now ready to define what the bipartite matching problem is. The problem is about *matching* N people to N tasks in an one-to-one fashion. Consider one particular matching illustrated in Fig. **??**. In this case, we have matchings for (person 1, task 2), (person 2, task N) and (person N, task 1). Here x_{ij} indicates whether or not person i is assigned with task j. So it is a binary value.

Now the goal of the problem is to find a matching such that the *total cost* is minimized. Here the cost is decided by some quantity assigned on an edge (called the *weight* in the literature). The weight is denoted by w_{ij} in which $(i,j) \in \mathcal{E}$. For instance, the cost for assigning person 1 to task 2 is w_{12}. Using this, we can then set the objective as:

$$\min_{x_{ij}} \sum_{i=1}^{N} \sum_{j=1}^{N} w_{ij} x_{ij}. \tag{9}$$

But there is a constraint here. The constraint is that the matching should be one-to-one mapping, i.e., each person must be assigned to one task and vice versa. This constraint can be expressed

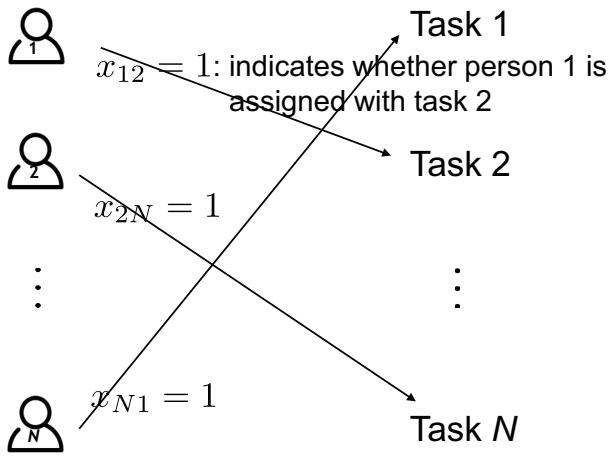


Figure 3: Bipartite matching problem.

as:

$$\begin{aligned} &\sum_{j=1}^{N} x_{ij} = 1 \ \text{(each person must be assigned to one task);} \\ &\sum_{i=1}^{N} x_{ij} = 1 \ \text{(each task must be assigned to one person).} \end{aligned} \tag{10}$$

Using this together with the above objective (**??**), we can then formulate the following optimization problem: Given w_{ij}'s,

$$\begin{aligned} &\min_{x_{ij}} \sum_{i=1}^{N} \sum_{j=1}^{N} w_{ij} x_{ij} : \\ &\sum_{j=1}^{N} x_{ij} = 1, \ \sum_{i=1}^{N} x_{ij} = 1, \ x_{ij} \in \{0, 1\}. \end{aligned} \tag{11}$$

Note that it belongs to the class (**??**), so it is an boolean problem.

Example 2: Shortest path problem

Another problem that I would like to mention is also a fundamental problem in combinatorial optimization: the *shortest path problem*. The problem is about finding a *path* from a source to a destination in a graph.

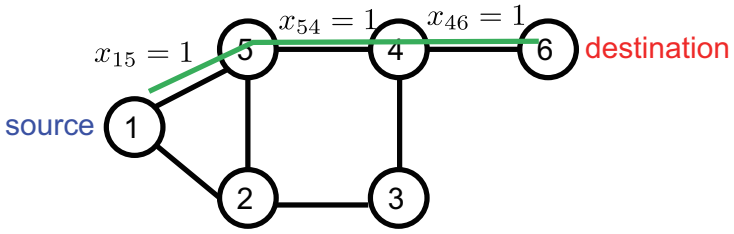


Figure 4: Shortest path problem.

For ease of illustration, let us consider an example in Fig. **??**. Here we have a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where:

$$\mathcal{V} = \{1, 2, 3, 4, 5, 6\};$$
$$\mathcal{E} = \{(1,2), (2,1), (1,5), (5,1), (2,3), (3,2), (2,5), (5,2), (3,4), (4,3), (4,5), (5,4), (4,6), (6,4)\}.$$

Here we consider a *bi-directed* graph (in which the edges are bi-directional), although the picture seems to suggest an *undirected* one. Let node 1 and node 6 be source and destination, respectively. A *path* is defined as a sequence of edges that connects the source to the destination. See an example path in Fig. **??**, marked in a green line: $(1,5) \to (5,4) \to (4,6)$. Let x_{ij} indicate whether the edge (i,j) is participated in the path. So in this example, $x_{15} = x_{54} = x_{46} = 1$ while all others are 0, i.e., $x_{51} = 0$, $x_{12} = 0$, $\ldots, x_{34} = 0, x_{64} = 0$. Notice that the path has a *direction*. So $x_{51} = x_{45} = x_{64} = 0$, since those are reverse to the direction of the path flow.

Now the goal of the problem is to find a path such that the total cost is minimized. It is assumed that the cost is decided again by the weight w_{ij} that comes with an edge (i,j). Hence, the objective can be stated as:

$$\min_{x_{ij}} \sum_{(i,j)\in\mathcal{E}} w_{ij} x_{ij}. \tag{12}$$

Here the constraint is that x_{ij}'s should be set to ensure a valid path, i.e., connecting the source to the destination. Now then the question is: how to check whether or not a path is valid? Actually the validation is a bit tricky. But if you think about the nature of the flow of a valid path, then you can come up with an easy way to check.

Consider the flow at the source node 1 in the example. One key observation is that the flow is just *outgoing*, i.e., there is no ingoing flow. So we have:

$$\begin{aligned} &\text{outgoing flow} - \text{ingoing flow} = 1 \\ \iff &\sum_{(1,j)\in\mathcal{E}} x_{1j} - \sum_{(j,1)\in\mathcal{E}} x_{j1} = 1 \end{aligned} \tag{13}$$

where $\sum_{(1,j)\in\mathcal{E}} x_{1j}$ indicates the entire flow that comes out of source 1 and $\sum_{(j,1)\in\mathcal{E}} x_{j1}$ denotes the aggregated flow that goes into source 1. On the other hand, at the destination, the situation is reversed:

$$\begin{aligned} &\text{outgoing flow} - \text{ingoing flow} = -1 \\ \iff &\sum_{(6,j)\in\mathcal{E}} x_{6j} - \sum_{(j,6)\in\mathcal{E}} x_{j6} = -1. \end{aligned} \tag{14}$$

For other node, say u (neither source nor destination), the flow is *just passing*, meaning that the flow coming in must go out. So we have:

$$\begin{aligned} &\text{outgoing flow} - \text{ingoing flow} = 0 \\ \iff &\sum_{(u,j)\in\mathcal{E}} x_{uj} - \sum_{(j,u)\in\mathcal{E}} x_{ju} = 0. \end{aligned} \tag{15}$$

Using all these, we can then formulate the optimization problem as:

$$\begin{aligned}
&\min_{x_{ij}} \sum_{(i,j)\in\mathcal{E}} w_{ij}x_{ij} : \\
&\sum_{(1,j)\in\mathcal{E}} x_{1j} - \sum_{(j,1)\in\mathcal{E}} x_{j1} = 1 \\
&\sum_{(6,j)\in\mathcal{E}} x_{6j} - \sum_{(j,6)\in\mathcal{E}} x_{j6} = -1 \\
&\sum_{(u,j)\in\mathcal{E}} x_{uj} - \sum_{(j,u)\in\mathcal{E}} x_{ju} = 0 \\
&x_{ij} \in \{0,1\} \ \ \forall (i,j) \in \mathcal{E}.
\end{aligned} \tag{16}$$

So we can now see that this is a boolean problem (**??**).

LP relaxation

Actually the boolean problem (**??**) is known to be notoriously difficult in general. In most cases, we need to search over all possible binary choices of x to figure out the optimal solution. So the complexity scales like 2^d (exponential to the dimension).

To deal with such difficult problems, people thought about a way to move forward. One natural way is to just ignore the binary value constraint (the very cause of making the problem intractable). This natural way is indeed the LP relaxation. Simply ignoring the binary value constraint in (**??**), we get:

$$\begin{aligned}
p^*_{\mathsf{LP}} := &\min w^T x : \\
&Ax - b \leq 0, \ Cx - e = 0, \\
&0 \leq x_i \leq 1, \ i = 1, \ldots, d
\end{aligned} \tag{17}$$

where x_i is now relaxed to be any real value $\in [0,1]$.

Since it is a more relaxed problem, we can do better, so in general,

$$p^* \geq p^*_{\mathsf{LP}}. \tag{18}$$

But very interestingly, it turns out that for some situations under the class of the bipartite matching problem and the shortest path problem, the optimal solution for the relaxed problem is binary: $x^*_{ij,\mathsf{LP}} \in \{0,1\}$, thus implying that $p^* = p^*_{\mathsf{LP}}$. We will not prove this here. Instead you will check this numerically in PS. If you are interested in further details, you may want to take a graph theory course offered in math and/or computer science departments.

Look ahead

So far we have studied the three historical examples which can be translated into LPs, as well as some boolean problems which can be solved via LP relaxation. Next time, we will cover the remaining part of what we were planning to do for LP: (i) Investigating efficient algorithms; (ii) Studying how to implement algorithms using software like CVX running in MATLAB.

EE523 Convex Optimization
KAIST, Spring 2019

March 19, 2019

Lecture 6: Algorithms and CVX Implementation

Recap

So far we have studied three important examples which can be translated into LPs: (1) Kantorovich's plywood cutting problem; (2) Monge's transportation problem; (3) the linear classification problem. We also investigated one class of very difficult problems which can be however solved via an LP relaxation technique: Boolean problems. As examples, we explored the following two: (4) the bipartite matching problem; (5) the shortest path problem.

Today's lecture

Today we are going to cover the remaining stuffs that we were planning to do: (i) Investigating efficient algorithms for LP; (ii) Studying how to solve LPs using software like CVX.

Algorithms for LP

There are three major algorithms for LP. The first is obviously the one that the Father of LP, Kantorovich, developed.

The second is a very famous and faster algorithm, called the *simplex algorithm*. The algorithm was developed in 1947 by an American mathematician, named George Dantzig. Actually some scientists and mathematicians in the West, especially at Berkeley (where Dantzig obtained PhD) and Stanford (where he was the Professor), claimed that the inventor of LP is Dantzig, not Kantorovich. The claim was based on the fact that the simplex algorithm is the first one that solves any LP in a finite number of steps (which was not revealed by Kantorovich) as well as the fact that the naming of LP was first used in print by Dantzig. However, many people including many Russians (of course) did not accept the claim, and perhaps more importantly, the Nobel Prize committee was silent on this.[1]

The last algorithm is a very generic algorithm, called the *interior point method*, which can be applied to general convex optimization problems beyond LP. The algorithm is based on the strong duality. Since the strong duality will be covered later in Part II, we will study the algorithm around at the time. In this lecture, we will focus on the simplex algorithm which is known to be the fastest LP solver even until now. Actually there is another reason why I would like to deal with the simplex algorithm in depth here. The reason is that the algorithm is very natural and so beautiful. You will check this soon.

Standard form for simplex algorithm

Remember the standard form of LP:

$$\min w^T x : \ Ax - b \leq 0, \ Cx - e = 0. \tag{1}$$

Actually the simplex algorithm relies on a different yet simpler equivalent form, called the

[1] Kantorovich's contribution for which the Nobel Prize was awarded was actually on the *optimal allocation of scarce resources*, which is not the invention of LP although highly related. If the committee had wanted to award a prize for LP, then Dantzig should have been included.

standard form of the simplex algorithm:

$$\begin{aligned} \max\, w^T x :\ & Ax \le b, \\ & x \ge 0. \end{aligned} \tag{2}$$

Later you will see why this form (2) helps to run the simplex algorithm. Please be patient until we get to that point. One can readily see that (2) belongs to the class of (1). The other direction turns out to be true, i.e., any form like (1) can be converted into the form like (2).

How to convert into the standard form?

To show the conversion from (1) to (2), we need to demonstrate four things. The first is to convert min to max. This can be done very easily by flipping the sign of the objective function. The second is to convert the equality constraint into inequality one(s). This is also immediate because:

$$Cx - e = 0 \iff Cx \le e,\ \ Cx \ge e.$$

The last is to ensure that all the optimization variables are non-negative, i.e., $x \ge 0$. This can be done in the following manner. Suppose there is no sign constraint on a variable, say x_1. Then, by introducing two new non-negative variables, say $x_2, x_3 \ge 0$, we can cover the case with:

$$x_1 = x_2 - x_3, \qquad x_2, x_3 \ge 0.$$

Here one important note is that using the equality $x_1 = x_2 - x_3$, we should replace all x_1's (that appear in other constraints if any) with $x_2 - x_3$, so that there is no x_1 in the final form.

Simplex algorithm: Convert into the slack form

We are now ready to describe how the algorithm works. Actually the precise description of the algorithm is very complicated although the idea is very simple and insightful. So we will focus only on grasping the key idea through the following example:

$$\begin{aligned} \max\, & 5x_1 + 4x_2 : \\ & 3x_1 + 5x_2 \le 78 \\ & 4x_1 + x_2 \le 36 \\ & x_1, x_2 \ge 0. \end{aligned} \tag{3}$$

The algorithm starts with converting into another form, called the *slack form*. In the slack form, two types of new variables are introduced. One is the target variable, usually denoted by z, which indicates the objective function itself:

$$z = 5x_1 + 4x_2.$$

The others are the slack variables, usually denoted by s_i's, which indicate the ones that make the inequality constraints equality ones. For instance, $3x_1 + 5x_2 \le 78$ can be equivalently written as:

$$3x_1 + 5x_2 + s_1 = 78, \qquad s_1 \ge 0.$$

With the target and slack variables, we can then re-write (3) as:

$$\max z : \tag{4}$$
$$z - 5x_1 - 4x_2 = 0 \tag{5}$$
$$3x_1 + 5x_2 + s_1 = 78 \tag{6}$$
$$4x_1 + x_2 + s_2 = 36 \tag{7}$$
$$x_1, x_2, s_1, s_2 \geq 0. \tag{8}$$

In the slack form for the simplex algorithm, one thing that we need to ensure is that the right hand sides in the translated equality constraints should be *non-negative*, as in the above example.

But then you may wonder what if the right hand side in one of the inequality constraints in (3) is *negative*? For example, suppose we have the following inequality instead:

$$-2x_1 - 4x_2 \leq -34.$$

In this case, the way to introduce a slack variable is slightly different. We first flip the sign of both sides to obtain:

$$2x_1 + 4x_2 \geq 34.$$

We then *subtract* a non-negative slack variable, say s_1, so that we obtain the following equality:

$$2x_1 + 4x_2 - s_1 = 34.$$

This way, we can ensure that all the right-hand-sides are non-negative.

Overall procedure

The simplex algorithm is sort of an iterative algorithm. So we have many iterations. Each iteration consists of the following operations:

1. Start with an initial feasible solution.
2. Perturb the solution along a direction that can maximize the target variable z.

A special note: If needed, we need to do some additional processes which can ease the above operations. You may not understand what this note means as of now. Don't worry. We will clarify what it means later on.

Once we obtain a newly perturbed solution, then we set it as another initial point in the next iteration, and again do perturbation along now a new z-maximizing direction in view of the new initial point. We repeat this procedure until any perturbation does not increase z further.

Iteration 1

First of all, how to set up an initial solution? The initial solution comes naturally from the two equality constraints that involve the two slack variables: (6) and (7). Notice that s_1 appears only in (6), similarly s_2 appears only in (7); on the other hand, (x_1, x_2) appear both in the two equations. So one natural feasible point is the one in which we set the both-appearing variables to zero, i.e., $x_1 = x_2 = 0$. This way, one can readily see that the following is a feasible solution:

$$(x_1, x_2, s_1, s_2) = (0, 0, 78, 36). \tag{9}$$

Notice in this case that $z = 0$ due to (5).

A natural question that arises next is then: Can we do better? To figure this out, we consider the equality constraint (5) that includes z of interest:

$$z = 5x_1 + 4x_2. \tag{10}$$

We see that increasing x_1 and/or x_2 (from the initial point $x_1 = x_2 = 0$) yields an increase in z. So one may wonder which direction to take to maximize z? There are three possible options that one can think of: (i) increasing x_1 only while maintaining $x_2 = 0$, i.e., $(x_1, x_2) = (\delta, 0)$ where $\delta \geq 0$; (ii) the other way around, i.e., $(x_1, x_2) = (0, \delta)$; (iii) increasing both x_1 and x_2, i.e., $(x_1, x_2) = (\delta_1, \delta_2)$ where $\delta_i \geq 0$. The simplex algorithm takes only the first two options. You may wonder why the last option is ignored - this will explored in depth in PS.

The first option looks the z-maximizing direction because the slope 5 placed in front of x_1 is larger than the slope 4 in front of x_2 in (10). However, it is not that clear if taking that direction is indeed the best way to go. The reason is that the maximum values of δ that we can push through can be different across distinct directions. So we need to investigate the two options carefully.

First consider $(x_1, x_2) = (\delta, 0)$. The constraints of (6) and (7) then give: $s_1 = 78 - 3\delta \geq 0$ and $s_2 = 36 - 4\delta \geq 0$, which in turn yields: $\delta = \min\{26, 9\} = 9$. So x_1 can be maximally set to 9 where $z = 45$. On the other hand, $(x_1, x_2) = (0, \delta)$ gives: $s_1 = 78 - 5\delta \geq 0$ and $s_2 = 36 - \delta \geq 0$. Hence, $\delta = \min\{\frac{78}{5}, 36\} = \frac{78}{5}$, which yields $z = 62.4$. So from this, we see that the second option is better, although the slope 4 is smaller than the other slope 5. This naturally motivates us to choose the following feasible point:

$$(x_1, x_2, s_1, s_2) = \left(0, \frac{78}{5}, 0, \frac{102}{5}\right), \tag{11}$$

where s_1 and s_2 are set according to the constraints of (6) and (7). The geometric picture for this is illustrated in Fig. 1. We move from $(x_1, x_2) = (0, 0)$ to $(x_1, x_2) = (0, \frac{78}{5})$.

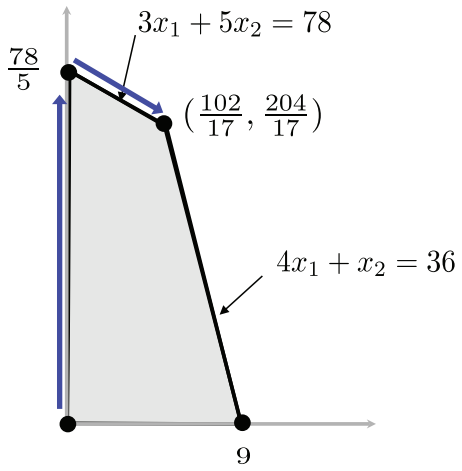


Figure 1: Geometric picture of simplex algorithm

Iteration 2

We first take the solution (11) as an initial feasible solution. Now the question is: Can we do better than this? To check this, let us ponder (11) again:

$$(x_1, x_2, s_1, s_2) = \left(0, \frac{78}{5}, 0, \frac{102}{5}\right). \tag{12}$$

Remember in the initial feasible point (9) that $(x_1, x_2) = (0, 0)$. On the other hand, in the second feasible point (12), $(x_1, s_1) = (0, 0)$. So in this case, one may now consider two possible options: (i) $(x_1, s_1) = (\delta, 0)$; (ii) $(x_1, s_1) = (0, \delta)$. To check which direction is better, we first need to ponder (5) to see how x_1 and s_1 affect z: $z - 5x_1 - 4x_2 = 0$. But there is a problem here. The problem is that it is difficult to see how s_1 affect z, since s_1 does not appear in the equation.

Here a very famous technique, called the *Gaussian elimination*, helps us to see the effect of s_1 upon z. Massaging (5) and (6) properly, we can cancel x_2 out to obtain:

$$\begin{array}{rl} & 5 \times [z - 5x_1 - 4x_2 = 0] \\ + & 4 \times [3x_1 + 5x_2 + s_1 = 78] \\ \hline \rightarrow & 5z - 13x_1 + 4s_1 = 312. \end{array} \tag{13}$$

This immediately rules out the second option: $(x_1, s_1) = (0, \delta)$, since increasing s_1 yields a *decrease* in z. So taking the first option $(x_1, s_1) = (\delta, 0)$ is the right way to go. Now the question is: How maximally can we set δ? To check this, let us ponder the constraints (6) and (7) again:

$$3x_1 + 5x_2 + s_1 = 78; \tag{14}$$

$$4x_1 + x_2 + s_2 = 36. \tag{15}$$

Here (14) looks okay because we can immediate see how x_2 is changed depending on (x_1, s_1) and this helps us to easily identify the limit of δ. On the other hand, the form like (15) is not desirable because in the form it is difficult to see how s_2 is changed depending on (x_1, s_1). Hence, it is not that simple to identify the limit of δ. Actually the following form is preferred instead: $?x_1 + ?s_1 + ?s_2 = ?$. Again the *Gaussian elimination* helps us to obtain such form:

$$\begin{array}{rl} & 5 \times [4x_1 + x_2 + s_2 = 36] \\ - & [3x_1 + 5x_2 + s_1 = 78] \\ \hline \rightarrow & 17x_1 - s_1 + 5s_2 = 102. \end{array} \tag{16}$$

Now with (16) and (13), we can re-write the optimization problem as:

$$\max z : \tag{17}$$

$$5z - 13x_1 + 4s_1 = 312 \tag{18}$$

$$3x_1 + 5x_2 + s_1 = 78 \tag{19}$$

$$17x_1 - s_1 + 5s_2 = 102 \tag{20}$$

$$x_1, x_2, s_1, s_2 \geq 0. \tag{21}$$

Remember that our second feasible point was:

$$(x_1, x_2, s_1, s_2) = \left(0, \frac{78}{5}, 0, \frac{102}{5}\right), \tag{22}$$

As mentioned earlier, we can immediately rule out the second option $(x_1, s_1) = (0, \delta)$. So taking the first option $(x_1, s_1) = (\delta, 0)$, we get: $5x_2 = 78 - 3\delta \geq 0$ and $5s_2 = 102 - 17\delta \geq 0$, which then yields: $\delta = \min\{26, \frac{102}{17}\} = \frac{102}{17}$. Hence, we obtain: $z = 78$. Since $z = 78$ is strictly larger than $z = 62.4$ (obtained under (22)), this motivates us to choose the following feasible point:

$$(x_1, x_2, s_1, s_2) = \left(\frac{102}{17}, \frac{204}{17}, 0, 0\right), \tag{23}$$

where (x_2, s_2) are set according to (19) and (20). The geometric picture for this is illustrated in Fig. 1. We move from $(x_1, x_2) = (0, \frac{78}{5})$ to $(x_1, x_2) = (\frac{102}{17}, \frac{204}{17})$.

Iteration 3

Again one can ask the same question: Can we do better? To check this, again ponder (23). We now have the following two options for perturbation: (i) $(s_1, s_2) = (\delta, 0)$; (ii) $(s_1, s_2) = (0, \delta)$, since $(s_1, s_2) = (0, 0)$ in the solution. To check which direction is better, again consider (18) to see how (s_1, s_2) affect z: $5z - 13x_1 + 4s_1 = 312$. Here it is difficult to see how s_2 affects z. Again use the Gaussian elimination to obtain the following where one can see the effect immediately:

$$\begin{aligned} & 17 \times [5z - 13x_1 + 4s_1 = 312] \\ - \quad & 13 \times [17x_1 - s_1 + 5s_2 = 102] \\ \hline \rightarrow \quad & 75z + 55s_1 + 65s_2 = 6630. \end{aligned} \tag{24}$$

We see from (24) that increasing (s_1, s_2) yields a *decrease* in z, meaning that any perturbation does not increase z further. Hence, we stop here, obtaining:

$$(x_1^*, x_2^*) = \left(\frac{102}{17}, \frac{204}{17}\right) \Longrightarrow z^* = 78. \tag{25}$$

This is how the simplex algorithm works - we stop when increasing such zero-variables does not increase z. It turns out this way of iteration enables us to achieve the optimal solution in a finite number steps. In many practical applications, it has been shown that the finite number of steps required is much less than the total number of vertices in the polytope formed by the constraints, meaning that the simplex algorithm arrives at the optimal point *very fast*.

Software implementation for LP

Now let me say a few words about software implementation before ending the part for LP. There are two popular softwares depending on platforms: (1) CVX (running in MATLAB); (2) CVXPY (running in Python). In fact, MATLAB is much more user-friendly and hence much easier to use although it requires a license. The good news is that the MATLAB license comes for free for any member in KAIST who can access to kftp within campus. So in this course, we will use CVX throughout. To download MATLAB, visit https://kftp.kaist.ac.kr/.

How to install CVX in MATLAB?

Installation of CVX is very simple. Download one of installation files uploaded on a course website (depending on your OS): (i) Windows 64-bit: "cvx-64w.zip"; (ii) Windows 32-bit: "cvx-32w.zip"; (iii) Mac 64-bit: "cvx-maci64.zip"; (iv) Linux 64-bit: "cvx-a64.zip". Next unpack the file to an empty direction, and then run `cvx_setup` from the MATLAB command line.

How to use CVX?

To give you a rough idea as to how to use CVX, let us give you a simple script of CVX implementation for Kantorovich's problem that we investigated in Lecture 4. Remember the standard form of Kantorovich's problem:

$$\min w^T x : \ Ax - b \leq 0 \tag{26}$$

```
2
3   d=3; % dimension of the optimization variable
4   w = [0;0;1];
5   A = [  -1,   0,  0;
6           1,   0,  0;
7           0,  -1,  0;
8           0,   1,  0;
9         -10, -20, -1;
10         10,  40, -1];
11  b = [0;1;0;1;0;50];
12
13  cvx_begin
14     variable x(d,1)
15     minimize( w'*x )
16     subject to
17         A*x <= b;
18  cvx_end
19
20
```

Figure 2: CVX implementation for Kantorovich's problem

where

$$w = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \; A = \begin{bmatrix} -1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \\ -10 & -20 & -1 \\ 10 & 40 & -1 \end{bmatrix}, \; b = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 50 \end{bmatrix}. \tag{27}$$

See Fig. 2 for CVX implementation. If we have an equality constraint like $Cx - e = 0$, we can then add a command like: `C*x==e`. Here the caveat is that we use equality symbols twice `==` to indicate an equality.

For other problems like Monge's problem and linear classification, you will have chances to do CVX implementation in PS. If you want to learn more about CVX syntax, consult the following website: http://web.cvxr.com/cvx/doc/.

Look ahead

So far we have studied the LP. Next time, we will move onto the second instance of convex optimization problems: the *Least-Squares problem*.

Lecture 7: Least-squares Problem

Recap

So far we have studied several stuffs: (1) the concept of convex optimization problems; (2) a categorization of such problems as summarized in Fig. 1; (3) why the convex optimization problems are tractable; (4) a bunch of important examples which can be translated into LPs and which can be solved via LP relaxation; (5) the simplex algorithm for LPs; (6) CVX implementation for LPs.

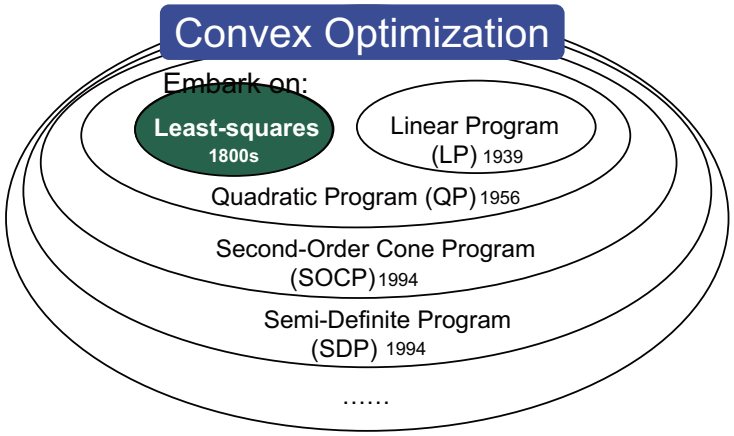


Figure 1: Hierarchy of convex optimization problems

Today's lecture

Today we are going to move onto the second instance of convex optimization problems: the *Least-squares* problem. Specifically in this lecture, we are going to cover three stuffs: (1) Will review what the least-squares problem is; (2) Will provide a geometric insight which can help us to understand what the least-squares solution means and therefore why the problem is important; (3) Will study one very important application in machine learning: the classification problem.

Review: Least-squares problem

Let us start by reviewing the least-squares problem that we studied in Lecture 1. The problem is formulated as:

$$\min \|Ax - b\|^2. \tag{1}$$

As mentioned earlier, one of the most important things that we can benefit from this problem is that it has the *closed-form solution*:

$$x^* = (A^T A)^{-1} A^T b. \tag{2}$$

Actually in Lecture 1, we could obtain this solution by simply looking for a solution that makes its gradient w.r.t. x as 0:

$$\nabla \|Ax^* - b\|^2 = 0. \tag{3}$$

But I never explained why (3) enables us to obtain the solution though. Now remember what we learned about *unconstrained convex minimization* in Lecture 3: If $f(x)$ is convex and differentiable at every point in $\mathbf{dom} f$, then $\nabla f(x^*) = 0$ is the optimality condition (i.e., the sufficient and necessary condition for x^* to be optimal). Applying this to the least-squares problem, we then see that (3) is indeed the optimality condition. This is because the objective function $\|Ax - b\|^2$ is convex in x (why? check this in PS3).

Dimensions of (x, A, b)

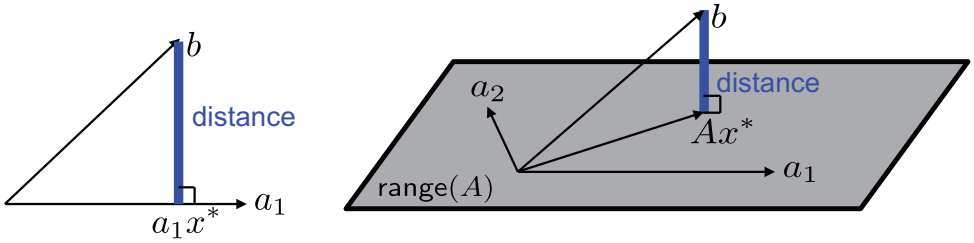
What about dimensions of (x, A, b)? Let d be the dimension of the optimization variable x. Then, $x \in \mathbf{R}^d$. Let $A := [a_1 \ \cdots \ a_d] \in \mathbf{R}^{m \times d}$. Then, $b \in \mathbf{R}^m$. We now have two cases depending on the values of m and d.

One is: $m < d$ (A is a *wide matrix*[1]). Suppose all the row vectors in A are linearly independent, i.e., $\mathsf{rank}(A) = m$, which is a typical case in practice. In this case, we have a larger number d of unknowns than the number m of linear equations in $Ax - b = 0$. This implies that there are infinitely many solutions that satisfy $Ax - b = 0$. So in this case, the optimal value $p^* = 0$, which is definitely not an interesting scenario.

The second case is: $m \geq d$ (A is a *tall matrix*). Suppose that b does not lie in the *range space* of A, $\mathsf{range}(A)$ (the space spanned by all the column vectors of A), which is a typical case in practice. In this case, obviously there is *no solution* that satisfies $Ax - b = 0$. But what we can say is that it has a solution that minimizes $\|Ax - b\|^2$ though, and this forms the basic idea behind the least-squares problem (that Gauss brought up in the 1800s). So what we are interested in is the second case: $m \geq d$.

Geometric insight

Now let me give you a geometric insight behind the least-squares problem. From this, you will see what the least-squares solution means, as well as why the problem is important accordingly.



(a) $d = 1$ (b) $d = 2$

Figure 2: Geometric insight of least-squares solutions

Let us first consider the simplest setting in which $d = 1$. In this case, A is simply a column vector and x is a scalar. Suppose we have two vectors a_1 and b, as illustrated in Fig. 2(a), in which b is not aligned with a_1 due to our assumption made earlier: $b \notin \mathsf{range}(A)$. Notice that $\|a_1 x - b\|^2$ is minimized when the vector $a_1 x - b$ (marked in the blue thick line) is *perpendicular* to the direction of a_1. So from this, one can interpret the least-squares solution as the *distance-minimizing solution*. The distance-minimizing solution is obviously what we want. So it is sort

[1] In fact, a majority of people use the terminology like a *fat matrix* instead. But Prof. Stephen Boyd at Stanford strongly recommended me to use a different terminology: a *wide matrix*. His rationale was that the wide matrix has sort of *positive* nuance, while the fat matrix looks *negative*. I respect his attitude for advocating *positive* aspects, so I use the "wide matrix" terminology.

of a good solution which well matches with our natural demand.

We can have the same interpretation for a slightly more general case, say $d = 2$. In this case, $A = [a_1 \ a_2] \in \mathbf{R}^{m\times 2}$. The vector Ax now lies in the *plane*, which is the range space of A: $Ax \in \mathsf{range}(A)$; see Fig. 2(b). Similarly $\|Ax - b\|^2$ is minimized when the vector $Ax - b$ (marked in the blue thick line) is perpendicular to the plane, as illustrated in Fig. 2(b). So again one can interpret Ax^* as the distance-minimizing solution.

An application: Classification problem

As mentioned earlier, the least-squares problem is a very popular and powerful problem which has played a significant role in the optimization field since the birth of the problem in the 1800s. It has been employed for addressing many important problems that arise in a wide variety of applications.

In this lecture, I would like to put a particular emphasis on one important application that arises in machine learning as well as that we have already investigated earlier: the *classification problem*.

Remember the classification example that we studied in Lecture 5: legitimate-vs-spam emails classification, in which we are given m data points $\{(x_i, y_i)\}_{i=1}^m$. Here x_i indicates a *feature vector*. In the example, we considered a two-dimensional case where $x_i := (x_{i1}, x_{i2})$ and (x_{i1}, x_{i2}) denote the frequencies of keywords 1 and 2 that appear in the ith email, respectively. Here y_i is a *label*, indicating an identity of the ith email: $y_i = +1$ (legitimate email), $y_i = -1$ (spam email).

For the above setting, we considered *linear* classifiers. Specifically, for the separable case, we formulated an LP which intends to find a line that separates two datasets (legitimate vs. spam). For the non-separable case (which is a typical case in practice), we formulated a slightly different LP which finds a line that minimizes the aggregated *margin*.

In this lecture, we will consider a different classifier which is based on the least-squares problem and therefore called: the *least-squares classifier*. The idea of the least-squares classifier is to find a *linear projection* that minimizes the aggregated *squared error*.

Least-squares classifier

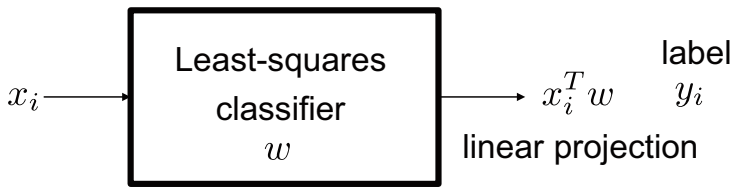


Figure 3: Block diagram of the least-squares classifier.

To see what the idea means, let us consider a block diagram for the classifier, illustrated in Fig. 3. The least-squares classifier is parameterized by a weight vector, say $w \in \mathbf{R}^d$. Given input x_i, it computes a linear projection onto the space spanned by the weight vector w; hence, it outputs $x_i^T w$. You may wonder about a sightly more general setting where we allow for having a bias term, like $x_1^T w + b$. It turns out one can deal with this case easily with a slight modification to the classifier. This will be explored in details in PS. The way to design w is as follows. Using the corresponding label y_i, we first compute its squared error: $\|x_i^T w - y_i\|^2$. Next compute the aggregated squared error with all of the m data points given. Finally we formulate

an optimization problem which minimizes the aggregation:

$$\min_{w \in \mathbf{R}^d} \sum_{i=1}^{m} \|x_i^T w - y_i\|^2. \tag{4}$$

Notice that the objective function is very similar to the one that we saw in Lecture 1. Yes, that is the objective function that Gauss came up with while addressing the astronomy problem. So we can use the same simplification trick that Gauss did, thus obtaining:

$$\begin{aligned} \sum_{i=1}^{m} \|x_i^T w - y_i\|^2 &= \left\| \begin{bmatrix} x_1^T w - y_1 \\ \vdots \\ x_m^T w - y_m \end{bmatrix} \right\|^2 \\ &= \left\| \begin{bmatrix} x_1^T \\ \vdots \\ x_m^T \end{bmatrix} w - \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \right\|^2. \end{aligned} \tag{5}$$

Letting $A := [x_1^T; \cdots; x_m^T]$ and $b := [y_1; \cdots; y_m]$, we can then re-write the optimization problem as:

$$\min_{w} \|Aw - b\|^2, \tag{6}$$

which is the least-squares problem. So we obtain w^* as:

$$w^* = (A^T A)^{-1} A^T b. \tag{7}$$

Testing classifiers on new unseen data

Here one natural question that arises is: Can the least-squares classifier (7) perform better than the linear classifier that we developed earlier using LP? To answer this question, first of all, we need to know what is a proper performance measure that one can use. One very popular and conventional approach in machine learning is to employ so called the *test error*. The test error is computed by testing a classifier on new *unseen data* called the *test data*.

To see what it means, consider the following setup. Given the classifier w^* designed as per (7), we input unseen test data, say x_{test}. The output is then $x_{\mathsf{test}}^T w^*$. Next we declare a legitimate email if $x_{\mathsf{test}}^T w^* \geq 0$; otherwise, declare a spam email, meaning that we take the sign of the output to obtain: $\hat{y}_{\mathsf{test}} = \mathsf{sign}(x_{\mathsf{test}}^T w)$. Comparing this to the ground-truth test data label y_{test}, we check if the classifier gives a correct answer (0) or not (1). Considering many such test data points, say m_{test} test data points, we compute the test error as the average of such measures:

$$\mathsf{TestError} = \frac{1}{m_{\mathsf{test}}} \sum_{i=1}^{m_{\mathsf{test}}} \mathbf{1}\{\hat{y}_{\mathsf{test},i} \neq y_{\mathsf{test},i}\} \tag{8}$$

where $\mathbf{1}\{\cdot\}$ denotes an indicator function which outputs 1 when the event $\{\cdot\}$ is true; 0 otherwise.

On a side note, we say that the *seen* data employed for training a classifier is called the *training data*. In this case, the training data are:

$$A := \begin{bmatrix} x_1^T \\ \vdots \\ x_m^T \end{bmatrix}, \; b := \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}.$$

Example: Evaluation on test dataset ($m_{\text{test}} = 1139 + 127$)

Here is an example that demonstrates the test error performance of a trained classifier tested on a test dataset. Suppose that the test dataset has two datasets: (1) legitimate dataset containing 1139 test data points; (2) spam dataset containing 127 test data points. Applying the trained classifier, we have 1120 correct and 19 wrong answers for 1139 legitimate emails. For 127 spam emails, we have 95 correct and 32 wrong answers. See Fig. 4.

	$\hat{y} = +1$	$\hat{y} = -1$
$y = +1$ (legitimate)	1120 true positive	19 misdetection
$y = -1$ (spam)	32 false positive (false alarm)	95

Figure 4: Test error performance of a trained classifier tested on a test dataset.

Then, the test error is computed as:

$$\text{TestError} = \frac{19 + 32}{1139 + 127} \approx 4\%. \tag{9}$$

Actually this error can be categorized into two types depending on what the ground truth is, and a different emphasis should be put on the two types of errors depending on applications. To explain what it means, let us first introduce some terminologies required to understand the two types of errors. The first is the *true positive* case which indicates the event $\{\hat{y} = +1 | y = +1\}$. The second is the *misdetection* case indicating $\{\hat{y} = -1 | y = +1\}$. The third is the *false positive (or false alarm)* case, which refers to $\{\hat{y} = +1 | y = -1\}$.

Now the first type of error is concerned about the misdetection case and therefore called the *misdetection error*: $\Pr\{\hat{y} = -1 | y = +1\}$. The second type of error then refers to the false positive case, so it is called the *false positive error*: $\Pr\{\hat{y} = +1 | y = -1\}$. In the above example in Fig. 4, the two types of error can be computed as:

$$\text{MisdetectionError} = \frac{19}{19 + 1120} \approx 1.7\%;$$
$$\text{FalsePositiveError} = \frac{32}{32 + 95} \approx 25\%.$$

Notice that the two errors are highly imbalanced; one is much smaller than the other. Actually if you think about it, it is sort of a *desired* situation. In reality, it is crucial to protect missing legitimate emails. In other words, we should be able to well declare legitimate emails if they are the cases, meaning that we should reduce the misdetection error as much as possible. In this case, it is around 1.7%, which is more or less okay.

On the other hand, we are sort of okay with declaring spam emails when they are actually legitimate emails, meaning that a moderate value of the false positive error is acceptable in reality. In this case, it is around 25%, which is more or less okay.

Linear classifier vs. least-squares classifier

Since we know about the test error which is a proper performance measure, we are now ready to compare performances of the two classifiers: the linear classifier and the least-squares classifier.

To this end, we first gather training dataset: $\{(x_i, y_i)\}_{i=1}^m$. We then use this to design the margin-based linear classifier (using LP) as well as the least-squares classifier (using (7)). Next we test the classifiers on test dataset $\{(x_{\mathsf{test},i}, y_{\mathsf{test},i})\}_{i=1}^{m_{\mathsf{test}}}$ to compute $\mathsf{TestError}_{\mathsf{linear}}$ and $\mathsf{TestError}_{\mathsf{LeastSquares}}$. This is how we compare performances. Now you may wonder which is better in terms of the test error measure. Don't worry. You will have a chance to check this in PS.

Regularization technique

Actually there is an issue in applying the least-squares classifier without any modification. In reality, a data point, say x, contains some *noise*. Data points are usually obtained from measurements made by humans or sensors. But humans and sensors are not perfect in reality, so x definitely contains some error. This error incurs an issue: Large values of w^* (designed as per (7)) can *boost up such noise*.

To avoid this, we somehow want to make those values small. One way to implement this is to minimize $\|w^*\|^2$. But obviously at the same time, we want to make $\|Aw - y\|^2$ small; otherwise, w^* would be always 0 - this is obviously what we do not want to get. This motivates people to come up with a natural idea, which is to *regulate the two objective at the same time*:

$$\min_{w \in \mathbf{R}^d} \|Aw - y\|^2 + \lambda \|w\|^2 \tag{10}$$

where $\lambda \geq 0$. Notice that for one extreme case of $\lambda = 0$, we obtain the conventional least-squared solution while for the other extreme case of $\lambda = \infty$, we get $w^* = 0$ in which we declare spam emails randomly with a priori probability that a randomly selected email is spam. This technique is called the *regularization* and λ is called the *regularization factor*.

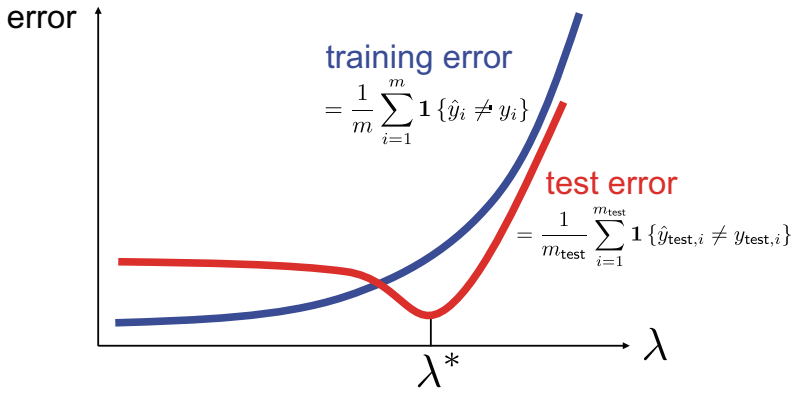


Figure 5: Training and test errors as a function of regularization factor λ.

Now you may wonder how performances vary in terms of the regularization factor λ. First consider the *training error* which is defined as:

$$\mathsf{TrainError} = \frac{1}{m} \sum_{i=1}^{m} \mathbf{1}\{\hat{y}_i \neq y_i\} \tag{11}$$

where $\hat{y}_i = \mathsf{sign}(x_i^T w^*)$. Obviously the training error is minimized at $\lambda = 0$ because in the case we focus only on the error factor induced by the training dataset. And it monotonically increases

with an increase in λ, since the higher λ, the more we regulate, penalizing more on the training error. So we will get something like a blue curve plotted in Fig. 5.

On the other hand, the situation is different about the test error, defined as:

$$\mathsf{TestError} = \frac{1}{m_{\mathsf{test}}} \sum_{i=1}^{m_{\mathsf{test}}} \mathbf{1}\left\{\hat{y}_{\mathsf{test},i} \neq y_{\mathsf{test},i}\right\}. \tag{12}$$

When $\lambda = 0$, the test error would be small but larger than the training error because the $\lambda = 0$ case focuses only on the training error. Increasing λ, we would have a regularization effect, so we expect the smaller test error. But if λ is too big, then the classifier would be close to $w^* = 0$ in which the test error would be obviously very large. So one can expect there is a sweet spot on λ that minimizes the test error. Hence, we may obtain something like a red curve plotted in Fig. 5. Actually this is indeed the case. Again you will have a chance to check this in PS.

How to solve the regularized problem?

Going back to the regularized least-squares problem (10), now how can we solve the problem? If you think about it, this is nothing but another least-squares problem. Why? Again applying the same Gauss's simplification trick, we obtain:

$$\|Aw - b\|^2 + \lambda\|w\|^2 = \left\| \begin{bmatrix} A \\ \sqrt{\lambda} I \end{bmatrix} w - \begin{bmatrix} b \\ 0 \end{bmatrix} \right\|^2 = \|A'w - b'\|^2$$

where $A' := [A; \lambda I]$ and $b' := [b; 0]$. Hence, we get:

$$\min_{w \in \mathbf{R}^d} \|A'w - b'\|^2. \tag{13}$$

So the solution would be:

$$w^* = (A'^T A')^{-1} A'^T b'. \tag{14}$$

Look ahead

Next time, we will study another application in which the least-squares problem has played a crucial role in a different field, the medical field: *Computed Tomography (CT)*.

Lecture 8: Computed Tomography (CT)

Recap

Last time, we have embarked on the second instance of convex optimization problems, Least Squares:

$$\min \|Ax - b\|^2 \tag{1}$$

where $x \in \mathbf{R}^d$, $A \in \mathbf{R}^{m \times d}$, $b \in \mathbf{R}^m$ and $m \geq d$. We then studied one very important machine-learning application that we had investigated in Lecture 5: the *classification problem*. Specifically we developed another linear classifier which can be designed via a least-squares problem. We also discussed a popular performance measure, called the *test error*, which is instrumental to make a fair comparison between distinct classifiers.

Today's lecture

Today we are going to study another application that arises in quite a different domain: the *medical* field. The application that we will investigate is: *Computed Tomography*, CT for short, that you may hear of. It turns out very interestingly, there is a mathematical principle behind the idea of CT and the principle is based on the least-squares problem.

X-rays

To see how the least-squares problem is related to CT, we need to understand the principle of CT. But to this end, we first need to understand the principle of X-rays which forms the basis of CT. So let us first study what X-rays is.

X-rays is a form of electromagnetic radiation. It was discovered in 1895 by a German physicist, named Wilhelm Röntgen. Actually the discovery opened up a new medical field, called the *radiology*. The radiology field is now a very well-known and well-established field, but there was no such field before the discovery of X-rays. In addition, the X-rays played a significant role in other areas beyond the medical field, like Physics and Chemistry. So the discovery won Röntgen the *first Nobel Prize* in Physics in 1901. Take a look at the very short 5-year gap between the discovery year 1895 and the award year 1901. It is a very rare case because it usually takes much longer time (around more than 10∼20 years) to receive a Nobel Prize since the discovery (or invention).

Principle of X-rays

The principle of X-ray was discovered through Röntgen's key observation made while he was working in his laboratory. While Röntgen was dealing with experimental tubes, he observed a type of *unidentified* radiation emanating from the tubes. Actually the naming "X-rays" was originated from the nature of the *unidentified* radiation, as "X" typically refers to the unknown. And he made an interesting observation about the mysterious radiation: When passing through an object, it absorbs photons and its energy (typically quantified as the *intensity*) is proportional to the number of photons absorbed: more photons (denser), the stronger intensity.

In fact, he did lots of experiments which support the observation, and he tested even on his wife's

hand, obtaining a scary picture that shows the bone structure of the hand[1]. It was a sort of the first historical medical X-ray image. The discovery together with such medical images made many people excited about the X-rays. In particular, people used the X-rays for the purpose of investigating the inside structure of interested objects (like human bodies) without drilling it, and this could open up a new medical field: *radiology*.

Limitations of X-rays

However, the X-rays has some limitations in figuring out a detailed structure of interested objects. Usually an object of interest is 3-dimensional. But the X-rays can yield only a projected 2-dimensional image, so this gives a challenge in identifying a complicated 3D object structure. For example, it is hard to spot tumors behind bones. Another clear example is illustrated in Fig. 1. Here a 2D image projected on the wall looks like a human's hand, but the actual 3D object is a rabbit. This implies that much of the key structure-related information can be lost while being projected. This clearly shows the limitations of X-rays.

Figure 1: An example in which the projected 2D image does not well represent a 3D structure.

Invention of Computed Tomography (CT)

Many people tried to solve such information-missing problem. A history was made in 1967 among such efforts. At the time, a smart way to address such problem was developed, named the *Computed Tomography (CT)*. The technique together with a computer-aided machine that implements the technique were invented by two people: one is a British electrical engineer, named Godfrey Hounsfield; the other is a South African-American physicist, named Allan Cormack. Actually this invention was not done in a cooperative manner - rather it was done independently. While Hounsfield's invention was slightly earlier, the credit was also fully given to Cormack, so they could be co-awarded the Nobel Prize in Physiology or Medicine in 1979 for the development of CT.

Idea of CT

Here is an idea of CT. In fact, the idea is very well reflected in the name. "Tomos" is a Greek word which means "projected section (or slice)"; "graphy" means "describe". So in words, it means "*describing* an object using *slices*." Details on the idea are the following:

[1]At that time, Röntgen had no idea of how detrimental X-rays is to human bodies. Perhaps, believe it or not, this is the reason why his wife passed away 6 years later since the discovery?

1. *Project* X-ray beams to an object from many different angles;
2. Calculate the intensities of the projected images (slices);
3. Use them to reconstruct (*describe*) the object.

A simple example

To explain what it means in detail, let us consider the following example in which an object of interest is comprised of four equal-sized grids - see Fig. 2. For illustrative purpose, here we consider a 2D object, although many of interested objects are 3D. Once you grasp the idea soon, you will understand that the idea can readily be extended to a 3D object case. The object has two black grids on left upper and right bottom parts, while having two white grids on right upper and left bottom parts. Suppose that X-rays absorbs no photon when passing through a black grid, i.e., the density is 0. Here we define the unit of the density as the number of photons *per unit length*. On the other hand, assume that X-rays absorbs something (say, the density is 1) when passing through a white grid.

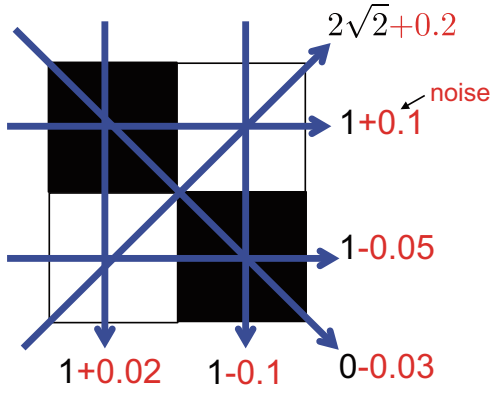


Figure 2: A simple grid example.

Now suppose we project a horizontal X-ray beam to the upper part of the object so that it passes through the two upper grids. Then, it would absorb nothing in the black grid zone while absorbing something in the white grid zone. Since the unit of the density that we define here is w.r.t. the unit length, the intensity of the X-ray beam would be proportional to the width of the white grid zone. For simplicity, assume that the width is 1 unit length. Then, the intensity would be 1. But there is always an error in measurement, say that the measurement noise here is 0.1 (marked in red in Fig. 2). We also project another horizontal X-ray beam to the bottom part of the object. We then measure the corresponding intensity, say $1 - 0.05$. Projecting two vertical beams, we get two measurements, say: $1 + 0.02$ and $1 - 0.1$.

Now shooting a top-to-bottom diagonal beam to the object, we would absorb nothing since it passes only through the black grids, so the measurement would be close to 0, say $0 - 0.03$. On the other hand, the other bottom-to-top diagonal beam would pass only through the white grids. Since the length of the passing zone is $2\sqrt{2}$, the intensity measurement would be close to $2\sqrt{2}$, say $2\sqrt{2} + 0.2$.

Actually what we want to figure out are the densities of the four grids, so let us denote those by unknown variables, say d_1 (top left), d_2 (top right), d_3 (bottom left), d_4 (bottom right). Using these notations, we can then express the above six measurements as the following linear equations:

$$
\begin{aligned}
d_1 + d_2 &= 1.1 \\
d_3 + d_4 &= 0.95 \\
d_1 + d_3 &= 1.02 \\
d_2 + d_4 &= 0.9 \\
\sqrt{2}d_1 + \sqrt{2}d_4 &= -0.03 \\
\sqrt{2}d_2 + \sqrt{2}d_3 &= 2.2.
\end{aligned} \tag{2}
$$

Least-squares problem formulation

Notice in (2) that we have six equations while having four unknowns. So there is *no solution* in general. Actually this is indeed the no-solution case. But we can invoke Gauss's idea to address this case. Yes, we can formulate it as a least-squares problem. Defining:

$$
A := \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ \sqrt{2} & 0 & 0 & \sqrt{2} \\ 0 & \sqrt{2} & \sqrt{2} & 0 \end{bmatrix}, \quad b := \begin{bmatrix} 1.1 \\ 0.95 \\ 1.02 \\ 0.9 \\ -0.03 \\ 2.2 \end{bmatrix},
$$

we can formulate it as:

$$
\min \|Ad - b\|^2. \tag{3}
$$

Then, the solution would be $d^* = (A^T A)^{-1} A^T b$. For the above example, we obtain:

$$
(d_1^*, d_2^*, d_3^*, d_4^*) = (0.1132, 0.8416, 0.8266, -0.0218),
$$

which looks making sense, as it is close to the ground truth $(0, 1, 1, 0)$.

A more realistic example

In fact, the example in Fig. 2 is too simple. In reality, an object is of an arbitrary shape and also the density of the object *continuously* changes over regions. To understand how to apply the idea into a more realistic object, let us consider another example in Fig. 3.

Here what we want to figure out is the density, say $d(x, y)$, which indicates the density at a coordinate (x, y). Suppose that we project to the object a X-ray beam (with a bottom-to-top diagonal direction), as illustrated in Fig. 3. Let t be the length of the beam trajectory from the starting point (x_0, y_0) at $t = 0$. Let θ be the angle of the beam in reference to the x-axis. Then, the coordinate $p(t)$ that the beam indicates when the beam length is t would be:

$$
p(t) = (x_0, y_0) + t(\cos\theta, \sin\theta). \tag{4}
$$

Remember that the intensity is: (density) $\times$ (length that the beam traverses). Here the density changes over the regions that the beam swipes. So it can be represented as $d(p(t))$. And the length of the beam w.r.t. the very small region where the density is almost constant can be represented by dt. So the intensity measurement would be:

$$
b = \int d(p(t))dt + v \tag{5}
$$

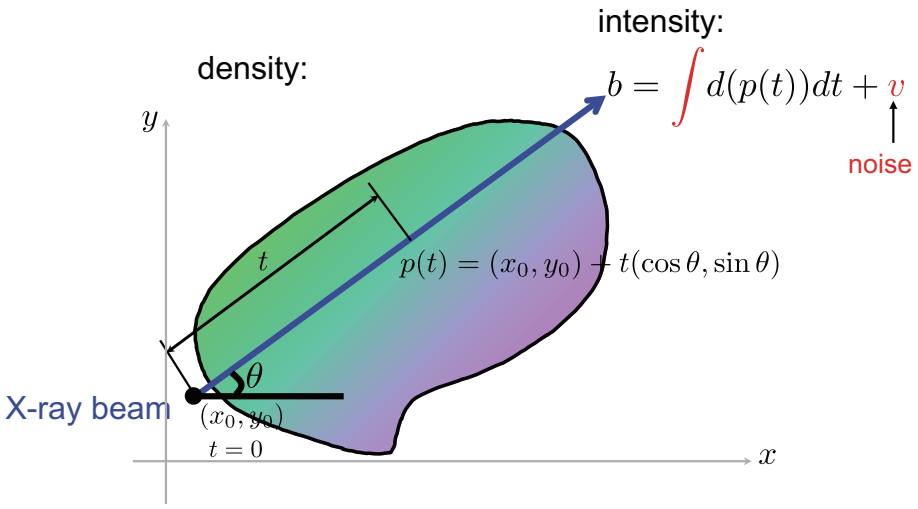


Figure 3: A more realistic example.

where v indicates a measurement noise.

Discretization

Here a question arises: How to estimate the density $d(p(t))$ of interest only from such measurement (5)? More specifically, one can ask: How is it related to the least-squares problem which does not deal with such integral-involved term? The idea is applying the *discretization* illustrated in Fig. 4. We make many minuscule grids in the space so that the density for each tiny grid can be assumed to be constant. Let d_i be the density of the ith grid. Denoting by a_i the length of the beam traversed at the ith grid, we can approximate the intensity absorbed through the ith grid as $a_i d_i$. Letting $\mathcal{S}_{\mathsf{beam}}$ the set of the indices of the grids that the beam travels, we can then approximate the aggregated intensity measured as:

$$b \approx \sum_{i \in \mathcal{S}_{\mathsf{beam}}} a_i d_i + v. \tag{6}$$

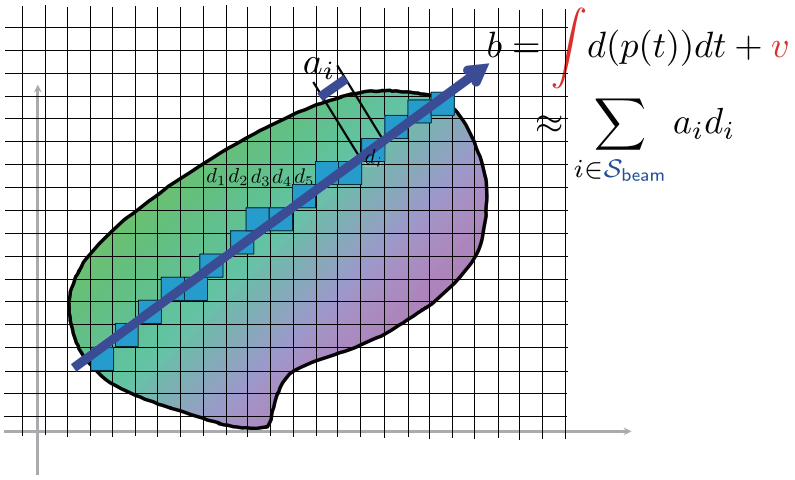


Figure 4: Discretization.

Now shooting many X-ray beams from many different angles, we obtain the following measurements:

$$\begin{aligned} b_1 &\approx \sum_{i\in\mathcal{S}_{\mathsf{beam1}}} a_i d_i + v_1 \\ b_2 &\approx \sum_{i\in\mathcal{S}_{\mathsf{beam2}}} a_i d_i + v_2 \\ &\vdots \\ b_m &\approx \sum_{i\in\mathcal{S}_{\mathsf{beam-m}}} a_i d_i + v_m. \end{aligned} \tag{7}$$

Least-squares problem formulation

Notice in (7) that we have m equations and the number of unknowns is the same as the number of grids that the object spans. With a sufficiently large number m of measurements (this is subject to our design), we can make m always larger than the number of unknowns. And for this setting, we can again apply Gauss's idea to formulate a least-squares problem as follows:

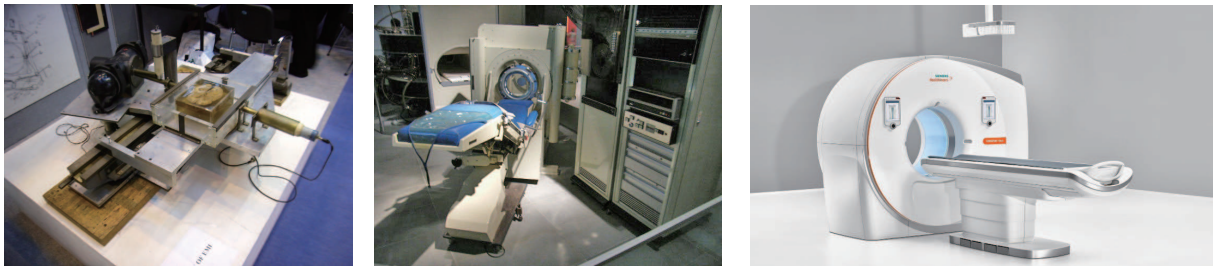
$$\min \|Ad - b\|^2 \tag{8}$$

where

$$A := \begin{bmatrix} \{a_i\}_{i\in\mathcal{S}_{\mathsf{beam1}}} \\ \{a_i\}_{i\in\mathcal{S}_{\mathsf{beam2}}} \\ \vdots \\ \{a_i\}_{i\in\mathcal{S}_{\mathsf{beam-m}}} \end{bmatrix}, \quad b := \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}.$$

History of CT scanners

This is the idea that Hounsfield thought of. While he mimicked Gauss's idea, I believe that the way to mimick is highly non-trivial. Applying this idea, Hounsfield could also develop a prototype CT scanner in 1971; see the far left picture in Fig. 5.



Prototype CT scanner | A historical EMI-scanner | CT scanner nowadays

Figure 5: History of CT scanners.

Remember that he was an *electrical engineer* - he was good enough to build an electrical computer-aided machine. The prototype supported $m = 160$ measurements (X-ray beams). The scanning time for each beam was a little over 5 mins. So the total scanning time was around 13 hours. Also the computation time for reconstructing an object with measurements

(solving the least-squares problem) was around 2.5 hours on a computer that he had. So it could not be commercialized as it took lots of time.

Fortunately, at that time, Hounsfield was working at a big and supportive company: EMI (Electric & Music Industries). While EMI was a music-record company, it was rich enough to invest some money to a field which has nothing to do with the music industries. Actually EMI was even going further. There was a rumour that with tons of money from the sales of *The Beatles* records in the 1960s, EMI helped fund the development of CT scanners. Anyhow the fact is that in the same year 1971, Hounsfield could develop the first commercial CT scanner with generous support from EMI, named the EMI-scanner - see the middle picture in Fig. 5. The performance of the scanner was remarkable relative to the prototype scanner: The scanning and reconstruction times were around 4 mins and 7 mins, respectively. So it could actually be commercialized.

CT scanners nowadays are beyond remarkable. For example, Siemens CT scanner (2017 model) in Fig. 5 takes only ~ 0.33 seconds for the whole process.

Look ahead

So far, we have studied two instances of convex optimization problems: LP and Least-Squares. Next time, we will study another instance which subsumes LP and Least-Squares as special cases: Quadratic Program.

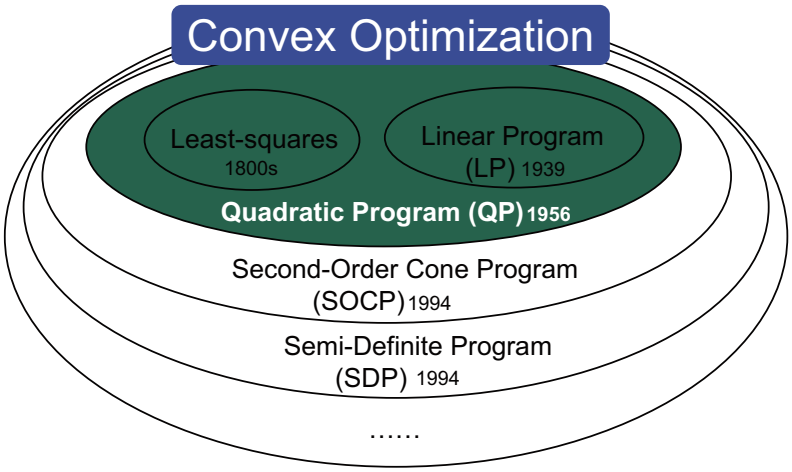


Figure 6: Hierarchy of convex optimization problems.

Lecture 9: Quadratic Program

Recap

For the past five lectures, we have studied two instances of convex optimization problems: LP and Least-Squares. For LP, we investigated a bunch of important examples which can be formulated as LPs or can be solved via LP relaxation. For least-squares, we explored two applications to demonstrate the power of the least-squares problem.

Today's lecture

Today we are going to study the follow-up instance that includes the LP and least-squares as special cases: Quadratic Program (QP). Specifically we are going to cover three stuffs. First we will study what QP is and verify that it indeed subsumes the LP and least-squares. Next, we will investigate a special case of QP in which there is a closed-form solution: *Constrained least-squares*. Finally, we will briefly discuss how to deal with general QP.

What is Quadratic Program?

The standard form of Quadratic Program (QP) is as follows:

$$\begin{aligned} \min\, & w^T x + x^T Q x : \\ & Ax - b \leq 0 \\ & Cx - e = 0 \end{aligned} \tag{1}$$

where $Q = Q^T \in \mathbf{R}^{d\times d} \succeq 0$ is a positive semi-definite (PSD) matrix[1] and (w, A, b, C, e) are of compatible size. Using the 2nd-order condition of convexity (check Problem 2 in PS2), one can readily show that QP is indeed a convex optimization problem:

$$\begin{aligned} \nabla^2(w^T x + x^T Q x) &= \nabla(w + 2Qx) \\ &= 2Q^T \succeq 0. \end{aligned}$$

Obviously QP includes LP as a special case in which $Q = 0$. To check whether it subsumes Least-Squares, consider:

$$\min \|Ax - b\|^2 = \min x^T (A^T A)x - 2b^T Ax + b^T b. \tag{2}$$

Notice that $A^T A$ is a PSD (why?) and $b^T b$ does not alter the optimal solution. Hence, the least-squares problem indeed belongs to QP.

Equality-constrained least-squares

As mentioned in the beginning, there is a special (yet important) case in which the closed-form solution exists. It turns out that the special case is the one in which the objective function

[1] We say that a *symmetric* matrix, say $Q = Q^T \in \mathbf{R}^{d\times d}$, is positive semi-definite if $v^T Q v \geq 0$, $\forall v \in \mathbf{R}^d$, i.e., all the eigenvalues of Q are non-negative. It is simply denoted by $Q \succeq 0$.

follows the one in the ordinary least-squares problem and we have only the *equality* constraint, which we call *equality-constrained least squares*:

$$\begin{aligned} \min \|Ax - b\|^2 : \\ Cx - e = 0 \end{aligned} \tag{3}$$

where $A \in \mathbf{R}^{m \times d}$ and $C \in \mathbf{R}^{p \times d}$.

Obviously we are interested in the case of $m \geq d$ (why? check Lecture 7). Depending on the values of p and d, we can now think of two cases: (i) $p > d$; (ii) $p \leq d$. The first is not an interesting case because in the case x^* is simply determined solely by the equality constraint (so it has nothing to do with minimizing the squared error) or there is no solution. Hence, the second case $p \leq d$ is of interest. Regarding the wide (or square) matrix C, without loss of generality, we assume that

$$\mathsf{rank}(C) = p. \tag{4}$$

Otherwise, one can eliminate *dependent* rows of C so that it has always full-rank. We also assume that

$$\mathsf{rank}\left(\begin{bmatrix} A \\ C \end{bmatrix} \right) = d, \tag{5}$$

meaning that all the columns of $[A; C]$ are linearly independent. Actually it is often the case in reality.

Closed-form solution

Under such case in which $m \geq d$, $p \leq d$, (4) and (5) hold: the closed-form solution for (3) reads:

$$x^* = d\text{-}\mathsf{Components}\left\{ \begin{bmatrix} 2A^TA & C^T \\ C & 0 \end{bmatrix}^{-1} \begin{bmatrix} 2A^Tb \\ e \end{bmatrix} \right\} \tag{6}$$

where d-$\mathsf{Components}(\cdot)$ indicates an operator that takes the first d components of $(\cdot)$. Notice that the inside of the operator is a $(d+p)$-dimensional vector. The proof of (6) consists of two parts:

1. Show that $\exists z \in \mathbf{R}^p$:

$$\begin{bmatrix} 2A^TA & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} x^* \\ z \end{bmatrix} = \begin{bmatrix} 2A^Tb \\ e \end{bmatrix}. \tag{7}$$

2. Show that

$$\begin{bmatrix} 2A^TA & C^T \\ C & 0 \end{bmatrix} \text{ is invertible.} \tag{8}$$

Proof of (7)

Actually the optimality condition that we learned in Lecture 3 for *constrained convex optimization* plays a crucial role to prove:

$$\nabla f(x^*)^T(x - x^*) \geq 0, \;\; \forall x : Cx = e. \tag{9}$$

Since x^* is obviously a feasible point, we have $Cx^* = e$. Now suppose we represent:

$$x = x^* + v. \tag{10}$$

Then, the vector v must satisfy $Cv = 0$, as $Cx = Cx^* = e$. So the optimality condition is equivalent to:

$$\nabla f(x^*)^T v \geq 0, \;\; \forall v : Cv = 0. \tag{11}$$

Here one important thing to note is that this optimality condition is equivalent to the following condition:

$$\nabla f(x^*)^T v = 0, \;\; \forall v : Cv = 0, \tag{12}$$

meaning that whenever $\nabla f(x^*)^T v \geq 0$, the equality must hold. Why? Check this in PS.

Here the condition (12) is equivalent to saying that: $v^T \nabla f(x^*) = 0, \;\; \forall v : v^T C^T = 0$. This implies that: $\nabla f(x^*) \in \mathsf{range}(C^T)$. If you are not quite convinced about this, check this in PS. Hence, we can say that:

$$\exists z \in \mathbf{R}^p : \;\; \nabla f(x^*) + C^T z = 0. \tag{13}$$

Since $\nabla f(x^*) = 2(A^T A)x^* - 2A^T b$ in the constrained least-squares problem of interest, we get:

$$\exists z : \;\; 2(A^T A)x^* - 2A^T b + C^T z = 0. \tag{14}$$

This together with $Cx^* - e = 0$ prove (7).

Proof of (8)

The proof idea is *by contradiction*. Suppose:

$$\begin{bmatrix} 2A^T A & C^T \\ C & 0 \end{bmatrix} \text{ is not invertible.} \tag{15}$$

Here not being invertible means that any column in the matrix in (15) can be expressed as a linear combination of the other columns of the matrix. This implies that:

$$\exists [\bar{x}; \bar{z}] \neq 0 : \;\; \begin{bmatrix} 2A^T A & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \bar{x} \\ \bar{z} \end{bmatrix} = 0. \tag{16}$$

From this, we have:

$$2A^T A\bar{x} + C^T \bar{z} = 0. \tag{17}$$

Now multiplying $\bar{x}^T$ to both sides from the left, we get:

$$2\bar{x}^T A^T A\bar{x} + \bar{x}^T C^T \bar{z} = 0. \tag{18}$$

Since $C\bar{x} = 0$ due to (16), $\bar{x}^T C^T = 0$. Applying this to (18), we get: $\|A\bar{x}\|^2 = 0$, which then yields: $A\bar{x} = 0$. This together with $C\bar{x} = 0$ gives:

$$\begin{bmatrix} A \\ C \end{bmatrix} \bar{x} = 0. \tag{19}$$

Now recall one of our assumptions made earlier (5). This implies that all the columns of $[A; C]$ are *linearly independent*. So (19) must imply that:

$$\bar{x} = 0. \tag{20}$$

Applying this to (17), we get:

$$C^T \bar{z} = 0. \tag{21}$$

Again recall the other assumption made earlier (4). This implies that all the rows of C (i.e., all the columns of C^T) are *linearly independent*. Hence,

$$\bar{z} = 0. \tag{22}$$

This together with (20) shows contradiction with (16), thus completing the proof (8).

Remark #1: KKT equations

Actually the key equations (7) that lead to the closed-form solution (6) are very famous ones which were investigated by three prominent mathematicians, whose last names are: Karush, Kuhn and Tucker[2]. So they are called the *KKT equations*.

$$\begin{aligned} \text{KKT equations: } & 2(A^T A)x^* - 2A^T b + C^T z = 0; \\ & Cx^* - e = 0. \end{aligned}$$

And the key matrix that appears in the left hand side of (8) is called the *KKT matrix*:

$$\text{KKT matrix}: \begin{bmatrix} 2A^T A & C^T \\ C & 0 \end{bmatrix}.$$

In fact, the KKT equations are part of the *KKT conditions*, which were derived as *necessary* conditions for optimality of *general* optimization problems[3]. The KKT conditions are very important conditions that form the basis of strong duality that we will study in Part II. So we will discuss more on this later.

Remark #2: Direct verification

Actually once we remember the KKT equations (7), then the proof of the optimality of x^* is very easy. In other words, one can easily verify that:

$$(7) \Longrightarrow \|Ax - b\|^2 \geq \|Ax^* - b\|^2, \ \ \forall x : Cx - e = 0.$$

So to prepare midterm and final exams, you may want to remember the KKT equations to ease related proofs.

Below we provide the direct verification:

$$\begin{aligned} \|Ax - b\|^2 &= \|(Ax - Ax^*) + (Ax^* - b)\|^2 \\ &= \|Ax - Ax^*\|^2 + \|Ax^* - b\|^2 - 2(Ax - Ax^*)^T(Ax^* - b) \\ &\overset{(a)}{=} \|Ax - Ax^*\|^2 + \|Ax^* - b\|^2 \\ &\geq \|Ax^* - b\|^2 \end{aligned}$$

[2]Kuhn is famous as a friend of John Nash, who received the Nobel Prize in economics for the game theory and is a model for the main character in the movie *Beautiful Mind* that you might watch. Tucker is famous as a PhD advisor of John Nash.

[3]The KKT conditions were publicized in a conference paper by Kuhn and Tucker in 1951. But later it was revealed that the same conditions were already derived in the *master thesis* by Karush in 1939.

The only thing that remains to complete the proof is to show (a); see below for the proof:

$$\begin{aligned} 2(Ax - Ax^*)^T(Ax^* - b) &= 2(x - x^*)^T A^T(Ax^* - b) \\ &\overset{(b)}{=} -(x - x^*)^T C^T z \\ &= -(Cx - Cx^*)^T z \\ &= -(e - e)^T z = 0 \end{aligned}$$

where (b) comes from the fact that $2A^TAx^* - 2A^Tb = -C^Tz$, which is the first among the KKT equations (7).

General QP

Recall the general QP which has the following standard form:

$$\begin{aligned} \min\, & w^Tx + x^TQx : \\ & Ax - b \leq 0 \\ & Cx - e = 0 \end{aligned} \tag{23}$$

where $Q = Q^T \succeq 0$ is a PSD matrix. Now how to solve the general QP? Unfortunately, there is no closed-form solution in general. So what we can do is to rely solely on the *optimality condition* that we learned: for optimal x^*,

$$\nabla f(x^*)(x - x^*) \geq 0, \;\; \forall x \in \mathcal{S} \tag{24}$$

where $\mathcal{S}$ denotes a feasible set.

As mentioned in Lecture 3, the optimality condition provides algorithmic insights via *strong duality*. So we will study how to solve the problem later when dealing with the strong duality in Part II.

Look ahead

So far, we have studied three instances of convex optimization problems: LP, Least-Squares and QP. For the upcoming two lectures, we will study two more instances that subsume all of the prior problems as special cases: Second-Order Cone Program (SOCP) and Semi-Definite Program (SDP).

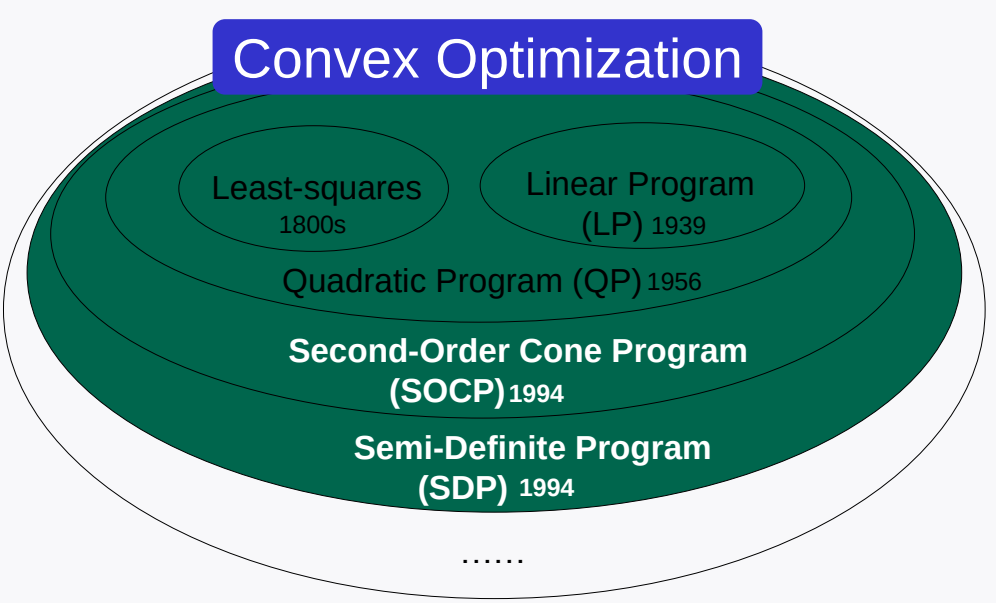


Figure 1: Hierarchy of convex optimization problems

Lecture 10: Second-Order Cone Program

Recap

So far we have studied three instances of convex optimization problems: LP, Least-Squares and QP. Last time we studied QP, investigating a special case of QP in which there is a closed-form solution: the *equality-constrained least-squares*. In particular I emphasized the KKT equations and KKT matrix that appear in the closed-form solution. I also mentioned that the KKT equations are part of the KKT conditions that we will study in depth in Part II.

Today's lecture

Today we are going to study the follow-up instance that includes the LP, Least-Squares and QP as special cases: Second-Order Cone Program (SOCP). Specifically we are going to cover three stuffs. First we will study what SOCP is and also verify that it indeed belongs to a convex optimization problem. Next, we will demonstrate that it subsumes LP and QP. Finally, we will discuss some applications in which SOCP can play a role and therefore one can see the power of the problem.

What is Second-Order Cone Program (SOCP)?

The standard form of Second-Order Cone Program (SOCP) is as follows:

$$\begin{aligned} \min\ & w^T x : \\ & \|A_i x - b_i\| \le c_i^T x + e_i, \quad i = 1, \dots, m, \\ & Fx = g \end{aligned} \tag{1}$$

where $A_i \in \mathbf{R}^{m_i \times d}$ and $F \in \mathbf{R}^{p \times d}$. Here the complicated-looking inequality constraint is the one that you have never seen thus far. Let us first verify that the problem belongs to a convex problem. To this end, we need to show that the following function is convex (why?):

$$\|A_i x - b_i\| - c_i^T x - e_i.$$

Notice that the latter term $-c_i^T x - e_i$ in the above is affine and also the inside term of the Euclidean norm is affine. Since convexity preserves under addition and affine transformation, it suffices to show that $\|x\|$ is convex. In the one-dimensional case, the function $\|x\|$ is "V"-shaped. So it looks like a convex function. It turns out this is indeed the case. Please prove it rigorously in PS.

We we call SOCP?

As you may guess, the naming comes from the never-seen inequality constraint:

$$\|A_i x - b_i\| \le c_i^T x + e_i. \tag{2}$$

To see the rationale behind the naming, let us consider a very simple setting which gives a hint: $A_i = I, b_i = 0, c_i = 0, e_i = t$. In this case, the constraint is simplified as: $\|x\| \le t$. Now consider a set of (x, t) which satisfies the constraint (2).

$$\mathcal{C} := \{(x, t) : \|x\| \le t\}. \tag{3}$$

Take a look at the shape of the set, illustrated in Fig. 1. It looks like an ice-cream *cone*. Also the norm that appears in the set is the Euclidean norm, which is the ℓ_2 norm. Hence, it is called the *second-order cone (SOC)*. Another names are quadratic cone, ice-cream cone or Lorentz cone.

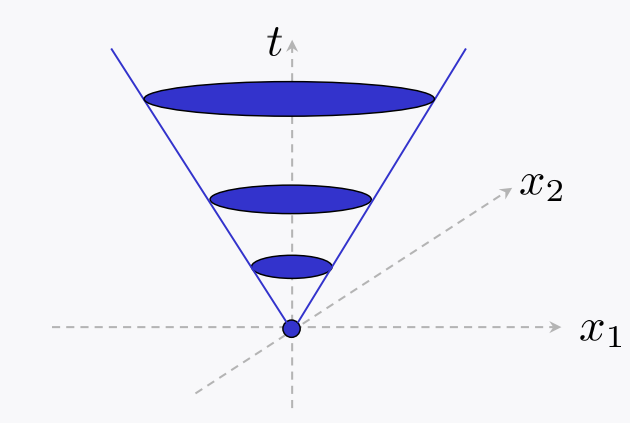


Figure 1: Picture of the second-order cone: $\{(x,t) : \|x\| \le t\}$.

Since the constraint (3) is a special case of the original constraint (2), you may still wonder why the problem (1) is called SOCP. Here a key observation is that the set of affine transformation of x is also a SOC:

$$(A_i x - b_i, c_i^T x + e_i) \in \mathcal{C}.$$

And the convexity preserves under affine transformation. Hence, one can also view the original constraint (2) as a SOC upto affine transformation (which does not alter the convexity property). So we can interpret the problem (1) as a SOC-constraint-based Program, which can be simply called SOCP.

Subsumes QP: QP $\rightarrow$ SOCP

Now let us show that the problem (1) includes LP and QP as special cases. One can immediately see the inclusion of LP by setting:

$$A_i = 0 \;\; \forall i = 1, \ldots, m,$$

in the original problem.

The proof of the inclusion of QP is slightly involved. To this end, we will show that QP can be cast into the form of SOCP. So let us start with the standard form of QP:

$$\begin{aligned} \min\, & w^T x + x^T Q x : \\ & Ax - b \le 0 \\ & Cx - e = 0 \end{aligned} \tag{4}$$

where $Q \in \mathbf{R}^{d\times d} \succeq 0$, $A \in \mathbf{R}^{m\times d}$ and $C \in \mathbf{R}^{p\times d}$. Here what is annoying is the quadratic term $x^T Q x$ that appears in the objective function. In an effort to translate the term into an affine term, let us first manipulate the matrix Q via eigenvalue value decomposition (EVD)[1]. Since Q is *symmetric*, one can always apply EVD to get:

$$Q = U \Lambda U^T$$

[1]If you are not familiar with EVD, then please take a look at the appendix of the linear algebra book uploaded on the course website: "Append_SVD_others.pdf".

where $U \in \mathbf{R}^{d\times d}$ is a unitary matrix (i.e., $U^TU = I$) and Λ is a diagonal matrix: $\Lambda = \mathsf{diag}(\lambda_1, \ldots, \lambda_d)$ where λ_i indicates the ith eigenvalue of Q. Now we define $y := Q^{1/2}x$ where

$$Q^{1/2} := U\Lambda^{1/2}U^T$$

where $\Lambda^{1/2} := \mathsf{diag}(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_d})$. This yields: $y^Ty = x^TQx$. Applying this to (4), we then get:

$$\begin{aligned} \min_{x,\, y}\ & w^Tx + y^Ty : \\ & Ax - b \leq 0 \\ & Cx - e = 0 \\ & y = Q^{1/2}x. \end{aligned} \tag{5}$$

While the newly introduced constraint $y = Q^{1/2}x$ is okay as it is *affine*, the quadratic term y^Ty in the objective is still problematic. To translate this into an affine term, we introduce a new variable, say t, such that

$$t \geq y^Ty. \tag{6}$$

Here the key observation is that minimizing t is equivalent to minimizing y^Ty, i.e., minimizing t, one can minimize y^Ty and vice versa. Hence, we can replace y^Ty in the objective with t while adding the constraint (6), thus obtaining:

$$\begin{aligned} \min_{x,\, y,\, t}\ & w^Tx + t : \\ & Ax - b \leq 0 \\ & Cx - e = 0 \\ & y = Q^{1/2}x \\ & y^Ty \leq t. \end{aligned} \tag{7}$$

Is $y^Ty \leq t$ a SOC constraint?

Now the question is: Is the newly introduced constraint $y^Ty \leq t$ is a SOC? At first glance, it looks not the case, as it can be represented as: $\|y\| \leq \sqrt{t}$. Notice that $\sqrt{t}$ is *not affine* in t. But it turns out it is the case with some modification. To see this, let us first manipulate it into: $4\|y\|^2 \leq 4t$. This is then equivalent to:

$$4\|y\|^2 + (t-1)^2 \leq (t+1)^2.$$

Observe in the above that we have *square* exponents in every term. So one can represent it as:

$$\left\| \begin{bmatrix} 2y \\ t-1 \end{bmatrix} \right\|^2 \leq (t+1)^2$$

Now dropping the square exponents in both sides and then using the definition (3) of SOC, we see that the set of $([2y; t-1], t+1)$ (affine transformation of the variables) is a SOC:

$$\left(\begin{bmatrix} 2y \\ t-1 \end{bmatrix}, t+1 \right) \in \mathcal{C}.$$

Hence, we obtain the following SOCP (from QP):

$$\begin{aligned} \min_{x,\, y,\, t} \; & w^T x + t : \\ & Ax - b \leq 0 \\ & Cx - e = 0 \\ & y = Q^{1/2} x \;\; \text{(affine)} \\ & \left\| \begin{bmatrix} 2y \\ t - 1 \end{bmatrix} \right\| \leq t + 1 \;\; \text{(SOC)}. \end{aligned} \tag{8}$$

Applications

Now why do we care about SOCP? The obvious reason is that it has many applications as LP and Least-Squares. In particular, it plays an important role in two specific settings.

The first setting is the one which represents *very practical scenarios* in which there is *uncertainty* in data and/or parameters. For example, in the legitimate-vs-spam email classification problem, data points can be viewed as sort of random quantities. It turns out that taking this probabilistic aspect into account, one can modify the original LP (that we formulated in Lecture 5) into a SOCP. In fact, such a modified LP is categorized into a broader class of LPs, called *Robust LP*, which covers all the probabilistic variations of LPs.

Another example is the least-squares problem with *uncertain* matrix A. For instance, in the CT application that we investigated in Lecture 8, the matrix A contains the length information of a beam trajectory for a small grid. Since the length is a *measured* quantity, it may contain some *measurement noise*, thus yielding some uncertainty. It turns out that taking this aspect, one can modify the original Least-Squares into a SOCP.

The second setting is the case in which optimization problems are formulated with Euclidean norms. Examples include: (i) distance-minimizing location planning in which one wants to locate a warehouse so as to serve many service locations while minimizing the transportation cost, which is usually proportional to the Euclidean distance; (ii) image denoising in which one wishes to remove the noise effect on the edges of an image while incorporating a sort of regularization term which involves an Euclidean norm; (iii) penalized Least-Squares in which one wants to minimize a noise effect while adding an Euclidean-norm-associated term (in the object function) which penalizes the noise effect.

Here we cannot cover all of the above applications due to the interest of time. Instead we are going to cover one of the most important applications in this lecture: Robust LP. Some of the other applications will be dealt with in PS.

Example of Robust LP: Chance Program (CP)

The application that I would like to put an emphasis on is a prominent example of Robust LP, called the *Chance Program (CP)*. Just for illustrative purpose, let us consider a very simple LP in which there is only one inequality:

$$\min w^T x : \quad a^T x \leq b. \tag{9}$$

In the legitimate-vs-spam email classification problem, here a, marked in *red*, indicates a data point. As mentioned earlier, such data point can be viewed as a *random* quantity. So in this case, a can be modeled as a *random vector*.

In an effort to deal with *uncertainty* that is induced by such random vector, one may want to

instead consider a *probabilistic* constraint which can be formulated as:

$$\Pr\left(a^T x \leq b\right) \geq 1 - \epsilon \tag{10}$$

for some small $\epsilon > 0$. Now then the question is: How to compute $\Pr(a^T x \leq b)$?

Gaussian approximation for $\Pr(a^T x \leq b)$

Actually the exact computation is almost impossible in reality as we have no idea of the probability distribution that the vector a is subject to. One way to go is to instead *approximate* the computation assuming that the vector a follows a well-known distribution in which the probability calculation is tractable. One such well-known distribution is the *Gaussian* distribution. In fact, the Gaussian distribution is not only computationally tractable, but it also well represents many practical settings which include the legitimate-vs-spam email classification problem.

So here we will use the Gaussian distribution to approximate the probability computation. Specifically assume that a follows the Gaussian distribution with:

$$a \sim \mathcal{N}(\bar{a}, K) \tag{11}$$

where $\bar{a}$ indicates the mean $\mathbb{E}[a]$ and K denotes the covariance matrix of a, defined as $K := \mathbb{E}[(a - \bar{a})(a - \bar{a})^T]$. Here the symbol "$\sim$" means "is distributed according to", and $\mathcal{N}$ denotes the Gaussian distribution.

Now consider a linear combination of a, $a^T x$, which is of our interest. Under the Gaussian assumption (11), $a^T x$ is also Gaussian:

$$a^T x \sim \mathcal{N}(\bar{a}^T x, x^T K x).$$

Why? Check in PS. Using this, we can then compute:

$$\begin{aligned} \Pr\left(a^T x \leq b\right) &= \Pr\left(\frac{a^T x - \bar{a}^T x}{\sqrt{x^T K x}} \leq \frac{b - \bar{a}^T x}{\sqrt{x^T K x}}\right) \\ &= \Phi\left(\frac{b - \bar{a}^T x}{\sqrt{x^T K x}}\right) \end{aligned} \tag{12}$$

where $\Phi(\cdot)$ indicates the cumulative distribution function (CDF) of the normal Gaussian distribution (with zero mean and variable 1): $\Phi(x) := \Pr(t \leq x)$ where $t \sim \mathcal{N}(0, 1)$; also see Fig. 2.

Applying (12) into (10), we get:

$$\Phi\left(\frac{b - \bar{a}^T x}{\sqrt{x^T K x}}\right) \geq 1 - \epsilon.$$

Since $\Phi(\cdot)$ is a non-decreasing one-to-one function (again see Fig. 2), we can invert the function to get:

$$\frac{b - \bar{a}^T x}{\sqrt{x^T K x}} \geq \Phi^{-1}(1 - \epsilon),$$

which in turns yields:

$$\Phi^{-1}(1 - \epsilon)\sqrt{x^T K x} \leq b - \bar{a}^T x. \tag{13}$$

CP $\rightarrow$ SOCP

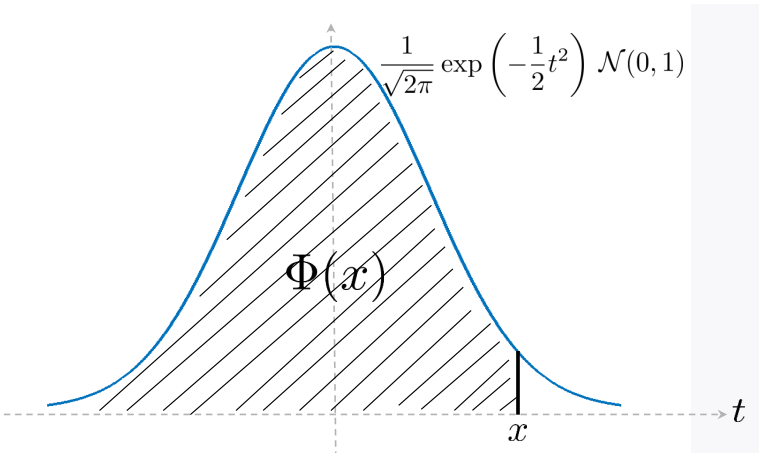


Figure 2: Cumulative density function (CDF) $\Phi(x)$ of the normal Gaussian distribution $\mathcal{N}(0,1)$.

Applying (13) to (9), we obtain:

$$\min w^T x: \quad \Phi^{-1}(1-\epsilon)\sqrt{x^T K x} \leq b - \bar{a}^T x. \tag{14}$$

Now the question is: Is the constraint in the above a SOC? To figure this out, let us simplify the constraint by letting $y := K^{1/2}x$. We then get:

$$\|y\| \leq \frac{b - \bar{a}^T x}{\Phi^{-1}(1-\epsilon)}.$$

Notice that the set of affine transformation of the optimization variables is a SOC:

$$\left(y, \frac{b - \bar{a}^T x}{\Phi^{-1}(1-\epsilon)}\right) \in \mathcal{C}.$$

Hence, we get the following SOCP (from CP):

$$\begin{aligned} &\min_{x,y} w^T x: \\ &\quad y = K^{1/2}x \ \ \text{(affine)} \\ &\quad \|y\| \leq \frac{b - \bar{a}^T x}{\Phi^{-1}(1-\epsilon)} \ \ \text{(SOC)}. \end{aligned} \tag{15}$$

How to solve SOCP?

Like QP, there is no closed-form solution for SOCP in general. So as mentioned earlier, we should rely on strong duality to gain some insights into algorithms. Hence, we will study in depth in Part II.

Look ahead

So far, we have studied four instances of convex optimization problems: LP, Least-Squares, QP and SOCP. Next time, we will study the final (from this course's point of view) instance that subsumes all of the prior problems as special cases: Semi-Definite Program (SDP).

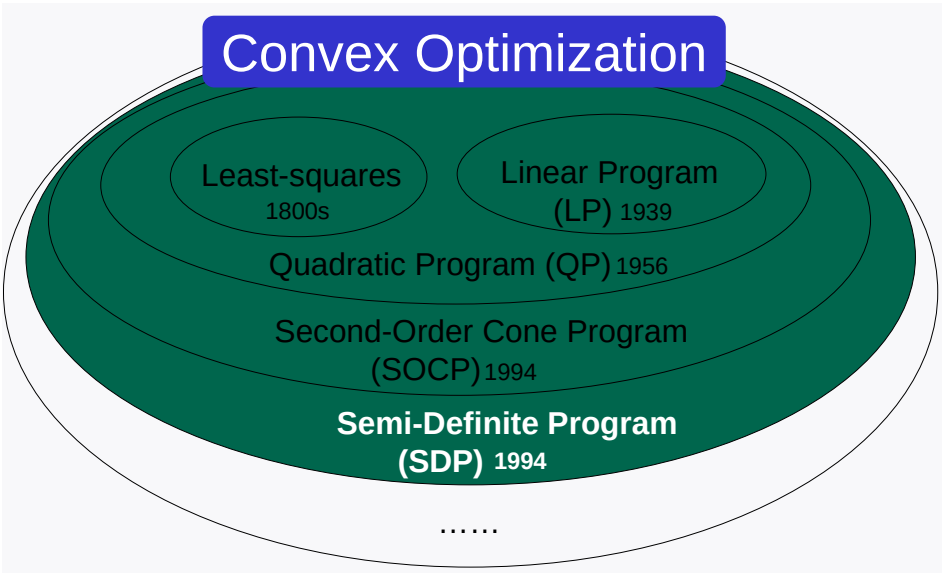


Figure 3: Hierarchy of convex optimization problems

Lecture 11: Semi-Definite Program

Recap

So far we have studied four instances of convex optimization problems: LP, Least-Squares, QP and SOCP. Last time we studied SOCP and figured out that the problem can play a role in very practical contexts in which there is *uncertainty* in data and/or parameters.

Today's lecture

Today we are going to study another instance that includes all of the prior problems as special cases, which is Semi-Definite Program (SDP). First we will study what SDP is and show that the feasible set in the problem is convex, thus proving that the problem indeed belongs to a convex optimization problem. Next, we will demonstrate that it subsumes LP, QP and SOCP. Finally, we will discuss some applications and will put a particular emphasis on a technique called *SDP relaxation*, which is proven to be very powerful for approximating non-convex optimization problems.

What is Semi-Definite Program (SDP)?

The standard form of Semi-Definite Program (SDP) is as follows:

$$\begin{aligned} \min\ & w^T x : \\ & G + x_1 F_1 + \cdots + x_d F_d \succeq 0 \\ & Cx = e \end{aligned} \tag{1}$$

where G, F_i's $\in \mathbf{R}^{k\times k}$ are *symmetric* matrices and $C \in \mathbf{R}^{p\times d}$. Here the inequality involves a bunch of *matrices* which are related with x in a *linear* manner. Hence, it is called the *Linear Matrix Inequality (LMI)*.

Why is the problem convex? To figure this out, we need to demonstrate that the following set induced by the inequality constraint is convex:

$$\mathcal{S} := \{x : G + x_1 F_1 + \cdots + x_d F_d \succeq 0\}.$$

Suppose that $x, y \in \mathcal{S}$. Fix $\lambda \in [0, 1]$. Now let us check if a linear combination $\lambda x + (1-\lambda)y$ is in the set $\mathcal{S}$. To this end, consider:

$$\begin{aligned} & G + (\lambda x_1 + (1-\lambda)y_1)F_1 + \cdots + (\lambda x_d + (1-\lambda)y_d)F_d \\ & = \lambda \underbrace{[G + x_1 F_1 + \cdots + x_d F_d]}_{\overset{(a)}{\succeq} 0} + (1-\lambda) \underbrace{[G + y_1 F_1 + \cdots + y_d F_d]}_{\overset{(b)}{\succeq} 0} \\ & \overset{(c)}{\succeq} 0 \end{aligned}$$

where (a) and (b) come from the hypothesis that $x, y \in \mathcal{S}$; and (c) follows from the fact that a convex combination of two PSD matrices is also PSD (why?). This implies that the linear

combination is also in the set: $\lambda x + (1-\lambda)y \in \mathcal{S}$. Hence, this proves the convexity of $\mathcal{S}$, thereby showing the convexity of the problem (**??**).

Subsumes LP and QP

It is straightforward to prove the inclusion of LP. Setting G and F_i's as *diagonal* matrices in the problem (**??**), one can reduce the problem to an LP. As for QP, it turns out that showing inclusion is almost equally difficult to showing the inclusion of SOCP. Hence, we will focus instead on proving the inclusion of SOCP.

Subsumes SOCP

In this section, we will demonstrate that SOCP can be cast into the form of SDP. So let us start with the standard form of SOCP:

$$\begin{aligned} &\min\ w^T x : \\ &\quad \|A_i x - b_i\| \le c_i^T x + e_i, \ \ i = 1, \ldots, m, \\ &\quad Fx = g \end{aligned} \tag{2}$$

where $A_i \in \mathbf{R}^{m_i \times d}$ and $F \in \mathbf{R}^{p \times d}$.

Manipulating the SOC constraint in (**??**), we get $(c_i^T x + e_i)^2 \ge \|A_i x - b_i\|^2$, which in turns gives:

$$(c_i^T x + e_i) - (A_i x - b_i)^T \{(c_i^T x + e_i) I\}^{-1} (A_i x - b_i) \ge 0. \tag{3}$$

Here a key observation is that the left-hand-side in (**??**) is the very famous *Schur Complement* of the matrix $(c_i^T x + e_i)I$. So one can use the prominent *Schur Complement Lemma* (formally stated below) to write down the SOC constraint as an LMI that we wish to represent as:

$$F_i(x) := \left[\begin{array}{cc} (c_i^T x + e_i) I & A_i x - b_i \\ (A_i x - b_i)^T & c_i^T x + e_i \end{array} \right] \succeq 0. \tag{4}$$

Note that $F_i(x)$ is symmetric and *affine* in x. Also one can show that all the matrices associated with x_i's are symmetric (why? check in PS). Hence, it is an LMI.

Schur Complement Lemma: Suppose $A \succ 0$. Then,

$$X = \left[\begin{array}{cc} A & B \\ B^T & C \end{array} \right] \succeq 0 \quad \Longleftrightarrow \quad S := C - B^T A^{-1} B \succeq 0. \tag{5}$$

Proof: Check in PS. ■

Notice in the standard form (**??**) of SDP that we have a *single* LMI, while here we have m such LMIs as per (**??**). But such multiple LMIs can be represented as a single LMI using the following trick:

$$F_1(x), \ldots, F_m(x) \succeq 0 \quad \Longleftrightarrow \quad F(x) := \left[\begin{array}{cccc} F_1(x) & 0 & \cdots & 0 \\ 0 & F_2(x) & \cdots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & \cdots & 0 & F_m(x) \end{array} \right] \succeq 0. \tag{6}$$

You may wonder why (**??**) holds - check in PS. Using this, one can rewrite the problem (**??**) as:

$$\begin{aligned} &\min\ w^T x : \\ &F(x) \succeq 0 \\ &Fx = g. \end{aligned} \tag{7}$$

Hence, we show that SOCP can be translated into SDP, thus proving the inclusion of SOCP.

Applications

Why do we care about SDP? Obviously it is because it has many applications. Particularly SDP plays a crucial role in *approximating* difficult non-convex optimization problems via a famous technique, called *SDP relaxation*. This will be explored further in the next section.

SDP is also useful for a variety of contexts in which the maximum eigenvalue of a matrix or the nuclear norm[1] are interested entities that we wish to minimize. One of the recent popular applications where such problems arise: *matrix completion* in which one wishes to identify missing entries of a matrix only from partially revealed entries (which are possibly noisy). Details will be dealt with in PS or exams.

Example of SDP relaxation: MAXCUT problem

In this section, we study one very well-known problem in which the SDP relaxation technique plays a role. The problem that we will investigate is: the MAXCUT problem.

The goal of the problem is to find a set that maximizes a cut. To understand what it means, we need to know about the concepts of a set and a cut. The context in which the problem is defined is a graph $\mathcal{G}$ which consists of a vertex set $\mathcal{V}$ and an edge set $\mathcal{G}$. Specifically, the problem is concerned about a *undirected* graph in which each edge does not have a direction, meaning that one edge in $\mathcal{E}$, say $(1,2)$, is the same as its counterpart $(2,1)$. For the example in Fig. **??**, the edge set reads:

$$\mathcal{E} = \{(1,2),(1,4),(1,5),(2,3),(2,4),(3,6),(4,5)\}.$$

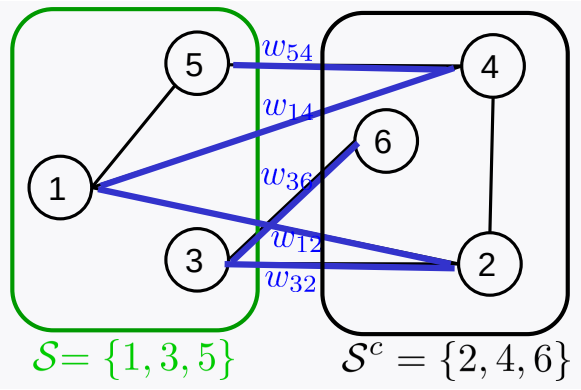


Figure 1: MAXCUT problem: Finding a set that maximizes a cut. In this example, the set $\mathcal{S} = \{1,3,5\}$ and the cut w.r.t. the set $\mathcal{S}$ is $w_{54} + w_{14} + w_{36} + w_{12} + w_{32}$.

Here what it means by a set $\mathcal{S}$ is a subset of the vertex set $\mathcal{V}$. For example, in Fig. **??**, the set can be $\mathcal{S} = \{1,3,5\} \subset \mathcal{V}$. The cut is defined as the aggregation of all the weights of the edges that come across the set $\mathcal{S}$ and its complement $\mathcal{S}^c$. In the example of Fig. **??**, the crossing edges are: $\{(5,4),(1,4),(3,6),(1,2),(3,2)\}$. Hence the cut w.r.t. the set $\mathcal{S}$ is:

$$\mathsf{Cut}(\mathcal{S}) = w_{54} + w_{14} + w_{36} + w_{12} + w_{32}, \tag{8}$$

[1]The nuclear norm, denoted by $\|A\|_*$, is defined as: $\|A\|_* := \sum_i \sigma_i(A)$ where $\sigma_i(A)$ indicates the ith singular value of A.

where w_{ij} denotes a weight associated with an edge $(i,j) \in \mathcal{E}$.

Optimization problem

To formulate an optimization problem that allows us to attain the above goal, we first need to come up with a proper optimization variable. Obviously the optimization variable should be a function of the choice of a set $\mathcal{S}$. Hence, we consider the following variable x_i such that it indicates whether node i is in the set $\mathcal{S}$:

$$x_i = \begin{cases} +1, & x \in \mathcal{S}; \\ -1, & \text{otherwise.} \end{cases} \tag{9}$$

Here a key observation is that when $x_i \neq x_j$, the edge (i,j) comes across the two sets $\mathcal{S}$ and $\mathcal{S}^c$, and hence, this should contribute to $\mathsf{Cut}(\mathcal{S})$ by the amount of w_{ij}. On the other hand, when $x_i = x_j$, there should be no contribution to the cut. This motivates us to formulate an optimization problem as:

$$\begin{aligned} \max_{x_i} \sum_{i,j} \frac{1}{2} w_{ij}(1 - x_i x_j) : \\ x_i^2 = 1, \quad i = 1, \ldots, d. \end{aligned} \tag{10}$$

Notice in the objective function that we get w_{ij} when $x_i \neq x_j$; 0 otherwise. The constraint $x_i^2 = 1$ respects the fact that x_i can be only either $+1$ or -1.

A translation technique: Lifting

Observe in the objective function in (**??**) that we have a *quadratic* term like $x_i x_j$. Also we have a *quadratic* equality-constraint. So these do not match with the standard form of any convex instance that we have studied thus far.

In an effort to translate such undesirable terms into favourable terms (e.g., *affine* terms), we introduce a well-known technique, called *lifting*. Here the lifting means raising a space that the optimization variable lives in. In the considered example, the optimization variables x_i's can be represented as a vector, like: $x := [x_1, \ldots, x_d]^T$. So the lifting in this context is to convert the vector x into a higher dimensional entity, say a matrix. For instance, one may introduce a new matrix, say X, such that its (i,j)-entry $[X]_{ij}$ is defined as:

$$X_{ij} := x_i x_j. \tag{11}$$

A more succinct way to represent this is: $X = xx^T$. Notice that the matrix X is then a rank-1 matrix and its eigenvalue is equal to $x^T x$ (Why? Think about the definition of the eigenvalue). So the change of variable induces the following constraints:

$$X_{ii} = 1, \quad X \succeq 0, \quad \mathsf{rank}(X) = 1. \tag{12}$$

Now with a new matrix variable X, the problem (**??**) can be rewritten as:

$$\begin{aligned} p^* := \max_{X} \sum_{i,j} \frac{1}{2} w_{ij}(1 - X_{ij}) : \\ X_{ii} = 1, \quad i = 1, \ldots, d \ \text{ (affine)} \\ X \succeq 0 \ \text{ (LMI)} \\ \mathsf{rank}(X) = 1 \quad \text{(rank constraint)} \ . \end{aligned} \tag{13}$$

Notice that $X \succeq 0$ is indeed an LMI. For example, for the $d = 2$ case,

$$X = \begin{bmatrix} X_{11} & X_{12} \\ X_{12} & X_{22} \end{bmatrix} = X_{11} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + X_{12} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} + X_{22} \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

is affine in X_{ij}'s and the associated matrices are all symmetric.

SDP relaxation

Notice in (**??**) that the objective function is affine in X_{ij}, the first equality constraint is affine, and the second inequality constraint is an LMI. However, it contains an undesirable constraint: $\mathsf{rank}(X) = 1$ (rank constraint). So it is not an SDP.

This is where a technique, called SDP relaxation, kicks in. The idea of the technique is simply to ignore such rank constraint. By ignoring the constraint, the search space in the optimization problem is broadened and hence it is indeed *relaxation*. Applying the technique, we get:

$$\begin{aligned} p^*_{\mathsf{SDP}} := \max_X \sum_{i,j} \frac{1}{2} w_{ij}(1 - X_{ij}) : \\ X_{ii} = 1, \quad i = 1, \ldots, d \ \text{ (affine)} \\ X \succeq 0 \ \text{ (LMI)}. \end{aligned} \tag{14}$$

Obviously $p^*_{\mathsf{SDP}} \geq p^*$, since it is relaxation for the *maximization* problem.

Interestingly, it turns out that in many cases, the gap between p^*_{SDP} and p^* is not so large. In some cases, the gap can be large. But it was shown by Nesterov (a Russian mathematician who has been playing an important role in the convex optimization field) in 1996 that the gap cannot be arbitrarily large. The worst-case bound was shown to be:

$$\frac{p^*_{\mathsf{SDP}} - p^*}{p^*_{\mathsf{SDP}}} \leq \frac{\pi}{2} - 1 \approx 0.571.$$

The proofs of these are out of the scope of the course. But instead you will have a chance to check some of them numerically in PS.

How to convert X^*_{SDP} into x^*?

You may wonder how to convert X^*_{SDP} (obtained from (**??**)) to x^*, as X^*_{SDP} may not be of the following desired form: $X^*_{\mathsf{SDP}} = xx^T$. In such an undesirable yet frequently-occuring case, one way to go is to apply a very well-known technique in statistics, called the *Principle Component Analysis (PCA)*. The way it works is as follows. We first do eigenvalue decomposition to get:

$$X^*_{\mathsf{SDP}} = U\mathsf{diag}(\lambda_1, \lambda_2, \ldots, \lambda_d)U^T \tag{15}$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \geq 0$. Here the number of non-zero eigenvalues determines the rank of the matrix. The idea is to take only the first (*principle*) largest eigenvalue λ_1 while ignoring others to approximate it as:

$$\tilde{X}^*_{\mathsf{SDP}} := U\mathsf{diag}(\lambda_1, 0, \ldots, 0)U^T. \tag{16}$$

This way, we can ensure $\mathsf{rank}(\tilde{X}^*_{\mathsf{SDP}}) = 1$, enabling us to obtain x^*.

How to solve general SDP?

Like QP and SOCP, there is no closed-form solution for SDP in general. So as mentioned earlier, we should rely on strong duality to gain insights into algorithms. Hence, we will study in depth in Part II.

Look ahead

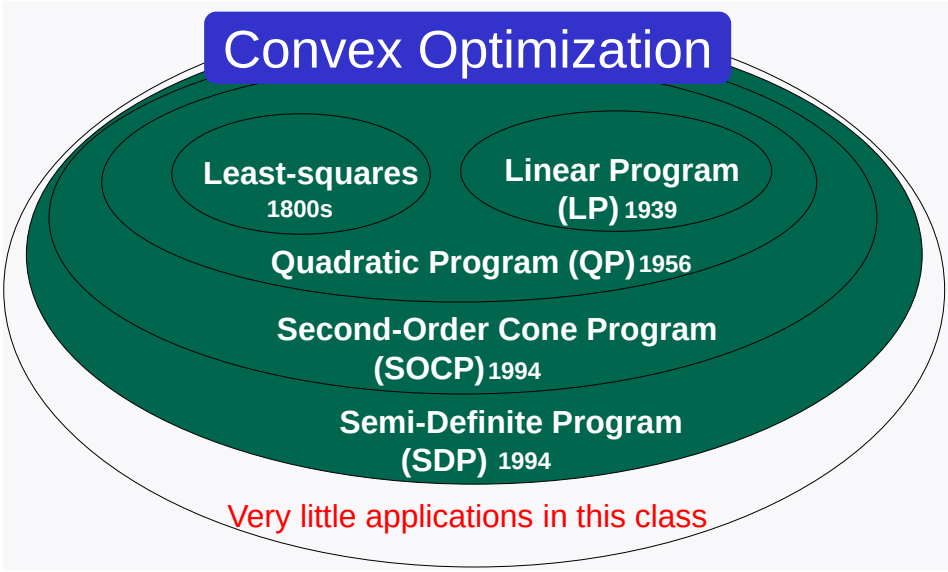


Figure 2: Hierarchy of convex optimization problems

So far, we have studied five instances of convex optimization problems: LP, Least-Squares, QP, SOCP and SDP. In fact, we have more instances which are convex but not belonging to the prior problems. However, there is very little application that such a class plays a role in. Hence, we stop here. Instead we will focus on studying algorithms for general QP, SOCP and SDP, which we have deferred as they are based on strong duality. So from next time, we will embark on Part II to start investigating strong duality.

Lecture 12: Strong Duality

Recap

We have thus far studied several classes of convex optimization problems: LP, Least-Squares, QP, SOCP and SDP. Actually we have one more well-known instance which is convex but not belonging to the prior classes. That is, *cone program*. In fact, understanding cone program requires lots of mathematical concepts, definitions and techniques, although there are very few applications of cone program. Too much overhead while gaining a little. Hence, we will not go further. Instead we will focus on studying what we have missed so far while investigating the previous convex instances. That is, algorithms for general QP, SOCP and SDP. The reason that we have deferred algorithm studies is that algorithms for the general settings are based on *strong duality* that we plan to cover in Part II. So from now on, we will move onto Part II to start investigating strong duality.

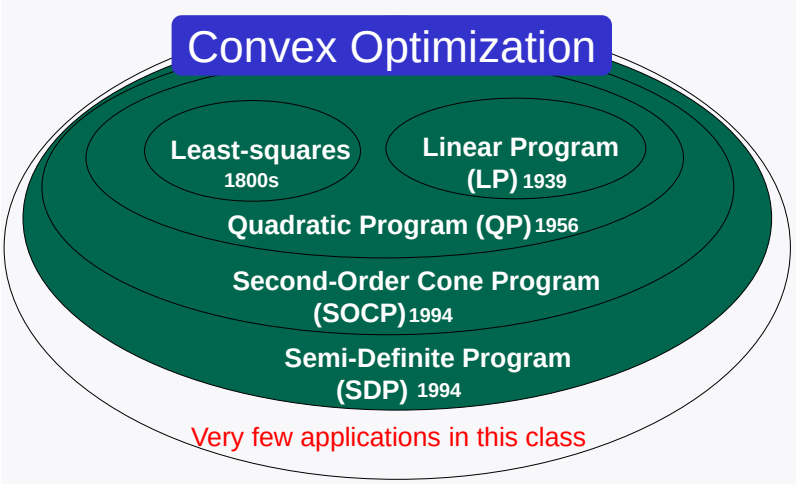


Figure 1: Hierarchy of convex optimization problems.

Today's lecture

Today we are going to cover three stuffs. Actually the strong duality is based on the concepts of primal and dual problems. So we will first study what the primal and dual problems are. Next, we will study what it means by strong duality. Finally, we will explore why the strong duality gives insights into the design of algorithms.

Primal & dual problems

Let us start by recalling the standard form of convex optimization:

$$\begin{aligned} &\min f(x) : f_i(x) \leq 0,\ i = 1, \ldots, m, \\ &\qquad Ax - b = 0, \end{aligned} \tag{1}$$

where $f(x)$ and $f_i(x)$'s are *convex* functions, $A \in \mathbf{R}^{p \times d}$ and $p \leq d$. Without loss of generality, assume that $\mathsf{rank}(A) = p$; otherwise, one can remove dependent rows in A to make it have full

rank. The primal problem is defined as a problem that we start with, and hence the above is the primal problem.

There is another problem which is intimately related to the primal problem, called the *dual* problem. But to explain what it means, we need to first know about a function, called the *Lagrange function*. The Lagrange function is denoted by $\mathcal{L}(x, \lambda, \nu)$. It takes three arguments. The second argument λ is a real-valued vector of size m, which coincides with the number of inequality constraints: $\lambda := [\lambda_1, \ldots, \lambda_m]^T$. The last argument ν (pronounced as "nu") is also a real-valued vector yet of different size p, which is the number of equality constraints: $\nu := [\nu_1, \ldots, \nu_p]^T$. The Lagrange function is defined as:

$$\mathcal{L}(x, \lambda, \nu) := f(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \nu^T (Ax - b). \tag{2}$$

Notice in the second summation term that $f_i(x)$ (that appears in the ith inequality constraint) is multiplied by with λ_i. Similarly the ith equality-constraint function is multiplied by with ν_i to form the last term $\nu^T(Ax - b)$. Hence, λ_i's and ν_i's are called *Lagrange multipliers*.

Are we now ready to define the dual problem? No. To explain what the dual problem is, we need to know about one more function, called the *Lagrange dual function*, or simply the *dual function*. Let us just use the simple version: the dual function. It is denoted by $g(\lambda, \nu)$ and defined as below.

$$\begin{aligned} g(\lambda, \nu) &:= \min_{x \in \mathbf{dom} f} \mathcal{L}(x, \lambda, \nu) \\ &= \min_{x \in \mathbf{dom} f} f(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \nu^T (Ax - b). \end{aligned} \tag{3}$$

Two things to note. The first and very important one is that the minimization here is over the *entire space* that x lies in w.r.t. $f(x)$: $\mathbf{dom} f$. Notice that the search space is *not limited to the feasible set* induced by inequality and equality constraints in the primal problem. The second thing to note is that $\mathcal{L}(x, \lambda, \nu)$ is not necessarily convex in x. When $\lambda_i \geq 0 \ \forall i$, $\mathcal{L}(x, \lambda, \nu)$ is simply a summation of convex and affine functions. So in this case, the function is convex and therefore, the minimum value is attained. However, λ_i's could be negative, as there is no sign constraint on λ in defining $\mathcal{L}(x, \lambda, \nu)$. This may render $\mathcal{L}(x, \lambda, \nu)$ concave (or affine) in x where $g(\lambda, \nu) = -\infty$.

We are now ready to define the dual problem of our primary interest. Observe in (3) that $g(\lambda, \nu)$ is a *pointwise minimum* of affine functions (in (λ, ν)) over all x's in $\mathbf{dom} f$. Hence, it is *concave* in (λ, ν) (Why? Think about what we proved in PS1: the *maximum* of convex functions is *convex*. More generally, one can prove that the maximum of *affine* functions is convex. Someone may still wonder about the above case (3) in which the minimum is over *continuous* values not over only a few candidates. Even in this case, one can readily prove the claim still holds. The proof is not that difficult - think about it). Hence, the maximum is always attained. The dual problem is an optimization problem that finds such maximum. So it is formulated as:

$$\text{(Dual problem)}: \max_{\lambda, \nu} \ g(\lambda, \nu) : \ \lambda \geq 0. \tag{4}$$

Here one thing to notice is that there is a constraint on λ ($\lambda \geq 0$) while there is none for ν. This together with the definition (3) of the dual function gives the following equivalent expression:

$$\max_{\lambda, \nu} \min_{x \in \mathbf{dom} f} f(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \nu^T (Ax - b) : \ \lambda \geq 0. \tag{5}$$

What strong duality means?

Here is a summary of the primal and dual problems:

$$\text{(Primal): } p^* := \min f(x): \ f_i(x) \le 0, \ i = 1, \dots, m, \ Ax - b = 0;$$

$$\text{(Dual): } \quad d^* := \max_{\lambda, \nu} \min_{x \in \mathbf{dom} f} f(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \nu^T (Ax - b): \ \lambda \ge 0.$$

Here we denote by p^* (or d^*) the optimal value for the primal (or dual) problem.

Using these, we can now state what strong duality is. What it means is that the optimal values of the two problems are equal:

$$\text{(Strong duality): } p^* = d^*. \tag{6}$$

It has been shown that in general, the optimal values are different, i.e., the strong duality does not hold. But interestingly it turns out that the strong duality (6) does hold for *convex* optimization of our interest, under a very mild condition[1]. We call this the *strong duality theorem*.

Now you may wonder why the strong duality theorem matter for developing algorithms. Of course, there is a reason. The reason is that when strong duality holds, one can derive necessary and sufficient conditions (in order for strong duality to hold), which provide algorithmic insights. So for the rest of this lecture, we will derive such conditions and will explain why the conditions shed lights as to how to design algorithms. In the next-next lecture, we will prove the strong duality theorem - please be patient until we get to the point.

Necessary conditions for strong duality to hold

Let us first focus on deriving necessary conditions. Suppose that strong duality holds $p^* = d^*$, and x^* and (λ^*, ν^*) are the optimal solutions of the primal and dual problems, respectively.

Since $f(x^*) = p^* = d^* = g(\lambda^*, \nu^*)$ under the hypothesis, we get:

$$\begin{aligned} f(x^*) &= g(\lambda^*, \nu^*) \\ &\overset{(a)}{=} \min_{x \in \mathbf{dom} f} f(x) + \sum_{i=1}^{m} \lambda_i^* f_i(x) + \nu^{*T}(Ax - b) \\ &\overset{(b)}{\le} f(x^*) + \sum_{i=1}^{m} \lambda_i^* f_i(x^*) + \nu^{*T}(Ax^* - b) \\ &\overset{(c)}{\le} f(x^*) \end{aligned} \tag{7}$$

where (a) is due to the definitions of the dual function (3) and the Lagrange function (2); (b) comes from the fact that x^* is a particular choice in view of the minimization problem in step (a); and (c) follows from the fact that $\lambda_i^* \ge 0$, $f_i(x^*) \le 0$, and $Ax^* - b = 0$ (since (x^*, λ^*) must be feasible points).

In the above, the left hand side and the right hand side are the same as $f(x^*)$, suggesting that the two inequalities in steps (b) and (c) are tight. From the tightness of the inequality (b), we see

[1]The mild condition says that there exists x such that strict inequality holds $f_i(x) < 0 \ \forall i$ subject to $Ax = b$. Actually the condition holds for almost all the problem instances that arise in reality. So one can say that strong duality *usually* holds for convex optimization. We will later discuss on this in detail.

that x^* indeed minimizes $\mathcal{L}(x, \lambda^*, \nu^*)$ over x. Since it is unconstrained, the optimality condition is that its gradient at x^* is zero:

$$\nabla_x \mathcal{L}(x^*, \lambda^*, \nu^*) = 0. \tag{8}$$

On the other hand, the tightness of the second inequality (c) implies:

$$\lambda_i^* f_i(x^*) = 0. \tag{9}$$

There is a name for this condition. It is called the *complementary slackness* condition. Why? The term $\lambda^* f_i(x^*)$ captures sort of *slack (gap)* between $d^* := g(\lambda^*, \nu^*)$ and $p^* := f(x^*)$; see the 1st, 3rd and 4th line in (7). The condition (9) implies:

$$\begin{aligned} f_i(x^*) < 0 &\implies \lambda_i^* = 0; \\ \lambda_i^* > 0 &\implies f_i(x^*) = 0, \end{aligned}$$

which says that whenever one of the inequality constraints is strict, the other inequality must be tight, i.e., both are sort of *complementary* in view of ensuing the equality.

The conditions (8) and (9) together with the constraints in the primal and dual problems then constitute the following necessary conditions for strong duality to hold:

$$\nabla_x \mathcal{L}(x^*, \lambda^*, \nu^*) = 0; \tag{10}$$
$$\lambda_i^* f_i(x^*) = 0; \tag{11}$$
$$f_i(x^*) \leq 0; \tag{12}$$
$$Ax^* - b = 0; \tag{13}$$
$$\lambda^* \geq 0. \tag{14}$$

where (12) and (13) come from the primal problem and (14) is from the dual problem.

In fact, these conditions (10)$\sim$(14) coincide with the ones that I mentioned in Lecture 9 while deriving the closed-form solution for the equality-constrained least-squares problem. These are the *KKT conditions*! Remember that the KKT conditions are not limited to convex optimization, but are intended for general convex & non-convex optimization problems. These are necessary conditions for a solution to be optimal for general optimization problems.

KKT conditions are also sufficient for strong duality to hold

Interestingly the KKT conditions are also sufficient for strong duality to hold under (x^*, λ^*, ν^*):

$$\text{KKT conditions} \implies p^* = f(x^*) = g(\lambda^*, \nu^*) = d^*. \tag{15}$$

Let us first show that $f(x^*) = g(\lambda^*, \nu^*)$. To this end, recall the definition of the Lagrange function (2) to obtain:

$$\begin{aligned} \mathcal{L}(x^*, \lambda^*, \nu^*) &:= f(x^*) + \sum_{i=1}^{m} \lambda_i^* f_i(x^*) + \nu^{*T}(Ax^* - b) \\ &= f(x^*) \end{aligned} \tag{16}$$

where the second equality follows from (11) and (13). On the other hand, from the definition of

the dual function (3), we get:

$$\begin{aligned} g(\lambda^*, \nu^*) &:= \min_{x \in \mathbf{dom} f} \mathcal{L}(x, \lambda^*, \nu^*) \\ &= \min_{x \in \mathbf{dom} f} f(x) + \sum_{i=1}^{m} \lambda_i^* f_i(x) + \nu^{*T}(Ax - b) \\ &\overset{(a)}{=} f(x^*) + \sum_{i=1}^{m} \lambda_i^* f_i(x^*) + \nu^{*T}(Ax^* - b) \\ &= \mathcal{L}(x^*, \lambda^*, \nu^*) \end{aligned} \tag{17}$$

where (a) comes from the fact that the condition (10) in the unconstrained minimization suggests that x^* minimizes $\mathcal{L}(x, \lambda^*, \nu^*)$. This together with (16) gives:

$$f(x^*) = g(\lambda^*, \nu^*). \tag{18}$$

Since p^* is the optimal value in the primal *minimization* problem, $p^* \leq f(x^*)$. Also $d^* \geq g(\lambda^*, \nu^*)$ as it is the optimal value in the dual *maximization* problem. These together with (18) yield:

$$p^* \leq d^*. \tag{19}$$

Now we will below prove that

$$p^* \geq d^* \tag{20}$$

to complete the proof of sufficiency of the KKT conditions for ensuring strong duality. To prove (20), consider a primal optimal point, say $\tilde{x}$, that achieves p^*. Also consider a dual optimal point, say $(\tilde{\lambda}, \tilde{\nu})$, that achieve d^*. Then,

$$\begin{aligned} p^* &= f(\tilde{x}) \\ &\overset{(a)}{\geq} f(\tilde{x}) + \sum_{i=1}^{m} \tilde{\lambda}_i f_i(\tilde{x}) + \tilde{\nu}^T(A\tilde{x} - b) \\ &\geq \min_{x \in \mathbf{dom} f} f(x) + \sum_{i=1}^{m} \tilde{\lambda}_i f_i(x) + \tilde{\nu}^T(Ax - b) \\ &\overset{(b)}{=} g(\tilde{\lambda}, \tilde{\nu}) \\ &= d^* \end{aligned} \tag{21}$$

where (a) follows from the fact that $\tilde{\lambda}_i \geq 0$, $f_i(\tilde{x}) \leq 0$ and $A\tilde{x} - b = 0$ (since $(\tilde{x}, \tilde{\lambda}, \tilde{\nu})$ must be feasible points); and (b) is due to the definition (3) of the dual function.

Actually the relationship between p^* and d^* stated in (20) is called the *weak duality*. It turns out the weak duality holds for any optimization problem (including non-convex optimization), and this will be explored further in a later lecture.

Look ahead

Now what can we do with the KKT conditions for developing algorithms? Next time, we will study details on this, and will demonstrate that the conditions indeed play a crucial role in designing algorithms.

Lecture 13: Algorithms

Recap

Last time, we embarked on Part II and started investigating strong duality which I claimed several times that it provides a detailed guideline as to how to design algorithms. The strong duality relies on the concept of primal and dual problems:

$$\text{(Primal): } p^* := \min f(x) : \ f_i(x) \leq 0, \ i = 1, \ldots, m, \ Ax - b = 0;$$

$$\text{(Dual): } \quad d^* := \max_{\lambda, \nu} \min_{x \in \mathbf{dom} f} f(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \nu^T (Ax - b) : \ \lambda \geq 0.$$

Using these, we studied what strong duality means:

$$\text{(Strong duality): } p^* = d^*. \tag{1}$$

We then stated (yet without proof) that strong duality (1) holds for *convex* optimization of our interest, under a mild condition. Next we derived necessary and sufficient conditions (KKT conditions) in order for strong duality to hold under a feasible point of (x^*, λ^*, ν^*):

$$\nabla_x \mathcal{L}(x^*, \lambda^*, \nu^*) = 0; \tag{2}$$

$$\lambda_i^* f_i(x^*) = 0; \tag{3}$$

$$f_i(x^*) \leq 0; \tag{4}$$

$$Ax^* - b = 0; \tag{5}$$

$$\lambda^* \geq 0. \tag{6}$$

Lastly we claimed that these KKT conditions give algorithmic insights.

Today's lecture

Today we are going to study details as to why that is the case. Specifically we will support the claim while investigating the following three problem settings. The first is a somewhat special yet prominent problem setting where we already saw the KKT conditions (in Lecture 9): the equality-constrained least-squares problem. In this case, we will demonstrate that the KKT conditions indeed recover the KKT equations which led to a concrete closed-form solution. The second is a broader setting which however has still only the equality constraints. In this setting, we will show that the KKT conditions can be solved via the gradient decent algorithm that we studied in Lecture 3. The last is a general setting which also contains inequality constraints. We will introduce one very powerful algorithm, called the *interior point method*, which can approximately implement the KKT conditions and therefore can approach the optimum with a reasonably small performance gap to the optimality.

Equality-constrained Least-Squares

Recall the equality-constrained least-squares problem:

$$\min \|Ax - b\|^2 : Cx - e = 0.$$

Let us verify that the KKT conditions (2)∼(6) recover the KKT equations. We first simplify the KKT conditions, tailoring for this equality-constrained setting:

$$\nabla_x \mathcal{L}(x^*, \nu^*) = 0; \tag{7}$$
$$Cx^* - e = 0. \tag{8}$$

Taking a derivative of the Lagrange function

$$\mathcal{L}(x, \nu) = \|Ax - b\|^2 + \nu^T (Cx - e)$$

w.r.t. x and setting it to 0, we see that (7) reads:

$$\nabla_x \mathcal{L}(x^*, \nu^*) = 2A^T Ax^* - 2A^T b + C^T \nu^* = 0. \tag{9}$$

This together with the equality constraint yields the KKT equations:

$$2A^T Ax^* - 2A^T b + C^T \nu^* = 0; \tag{10}$$
$$Cx^* - e = 0. \tag{11}$$

Compared to the setting in Lecture 9 where we first investigated the KKT equations, the only distinction here is that we used a different notation ν^* instead of z.

Equality-constrained convex optimization

Now what about for general convex optimization problems? It turns out that solving the KKT conditions, one can develop some algorithms. In particular, for *equality*-constrained optimization problems, one can come up with a simple algorithm.

So let us first consider such equality-constrained setting:

$$\min f(x) : \ Ax - b = 0. \tag{12}$$

Under this setting, the KKT conditions (2)∼(6) read:

$$\nabla_x \mathcal{L}(x^*, \nu^*) = 0; \tag{13}$$
$$Ax^* - b = 0. \tag{14}$$

Recalling $\mathcal{L}(x, \nu) = f(x) + \nu^T (Ax - b)$, we see that

$$Ax^* - b = 0 \iff \nabla_\nu \mathcal{L}(x^*, \nu^*) = 0. \tag{15}$$

Since strong duality holds $p^* = d^*$ in the convex optimization setting (due to the *strong duality theorem*), it suffices to develop an algorithm that achieves the optimal value in the *dual* problem. So we focus on:

$$\begin{aligned} d^* &:= \max_\nu g(\nu) \\ &= \max_\nu \min_{x \in \mathbf{dom} f} \mathcal{L}(x, \nu). \end{aligned}$$

Here one can make the following observations: (i) $\mathcal{L}(x, \nu)$ is *convex* in x; (ii) $\min_{x \in \mathbf{dom} f} \mathcal{L}(x, \nu)$ is *concave* in ν; (iii) $\min_{x \in \mathbf{dom} f} \mathcal{L}(x, \nu)$ is *unconstrained* (w.r.t. x); (iv) $\max_v \min_{x \in \mathbf{dom} f} \mathcal{L}(x, \nu)$ is *unconstrained* (w.r.t. ν). Remember in Lecture 3 that the optimal condition for *unconstrained* minimization (or maximization) is that the gradient evaluated at the optimal point must be 0. More specifically, the optimality condition for the inner minimization problem is: given a ν,

$$\nabla_x \mathcal{L}(x^*(\nu), \nu) = 0 \tag{16}$$

where $x^*(\nu) := \arg\min_{x \in \mathbf{dom} f} \mathcal{L}(x, \nu)$. The optimality condition for the outer maximization problem is:

$$\nabla_\nu \mathcal{L}(x^*(\nu^*), \nu^*) = 0 \tag{17}$$

where $\mathcal{L}(x^*(\nu), \nu) = \min_{x \in \mathbf{dom} f} \mathcal{L}(x, \nu)$. Letting $x^* := x^*(\nu^*)$, the two conditions (16), (17) yield:

$$\nabla_x \mathcal{L}(x^*, \nu^*) = 0;$$
$$\nabla_\nu \mathcal{L}(x^*, \nu^*) = 0.$$

Since these coincide with the KKT conditions ((13), (14), and (15)), it suffices to find a point (x^*, ν^*) such that the gradients are zeros.

Gradient decent algorithm

In Lecture 3, we studied one popular algorithm which allows us to find a point whose gradient is 0. That was: the *gradient decent algorithm*. So we can use the same algorithm. The only distinction here is that we have two points (x, ν) to optimize over and so we have two corresponding gradients to compute. Below is how a modified algorithm works.

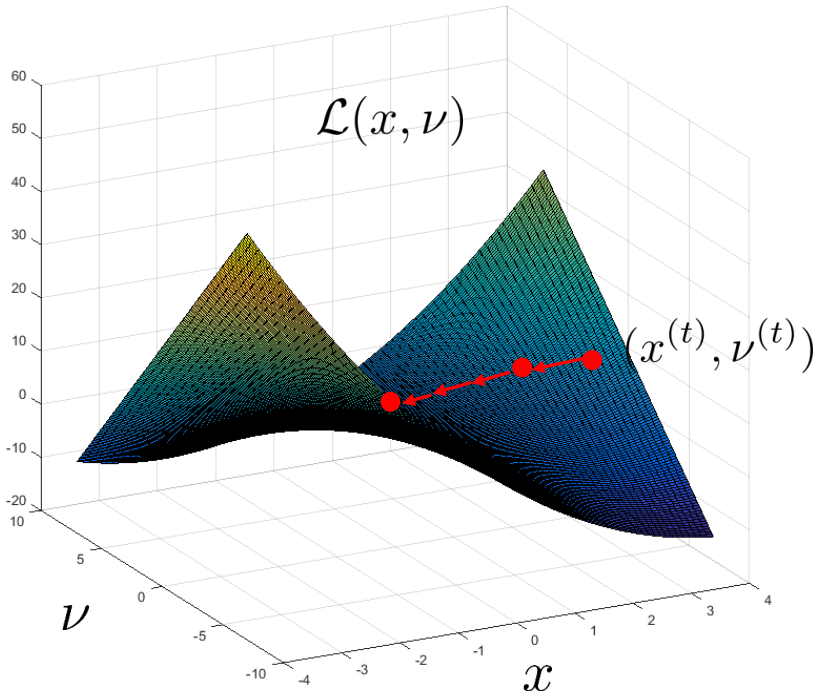


Figure 1: Gradient decent algorithm for equality-constrained optimization.

Let $(x^{(t)}, \nu^{(t)})$ be the estimates at the tth iteration; see Fig. 1. First we compute a gradient w.r.t. x at the point: $\nabla_x \mathcal{L}(x^{(t)}, \nu^{(t)})$. Since $\mathcal{L}(x, \nu)$ is convex in x (see Fig. 1 as well), we should move the point to the opposite direction (in reference to the gradient) so as to approach the optimal solution. So we update $x^{(t)}$ as:

$$x^{(t+1)} \leftarrow x^{(t)} - \alpha^{(t)} \nabla_x \mathcal{L}(x^{(t)}, \nu^{(t)})$$

where $\alpha^{(t)} > 0$ indicates a step size, which is usually set as a decaying function like $\alpha^{(t)} = \frac{1}{2^t}$.

Next, we compute a gradient w.r.t. ν: $\nabla_\nu \mathcal{L}(x^{(t)}, \nu^{(t)})$. Since $\min_x \mathcal{L}(x, \nu) (=: \mathcal{L}(x^*(\nu), \nu))$ is concave in ν (see the bottom part of the curve in Fig. 1 where the minimum of $\mathcal{L}(x, \nu)$ is attained over x), we should move the point to the *same* direction (in reference to the gradient) so as to approach the optimal solution. So we update $\nu^{(t)}$ as:

$$\nu^{(t+1)} \leftarrow \nu^{(t)} + \beta^{(t)} \nabla_\nu \mathcal{L}(x^{(t)}, \nu^{(t)})$$

where $\beta^{(t)} > 0$ indicates another step size, which is not necessarily the same as $\alpha^{(t)}$.

We repeat the above procedures until $(x^{(t)}, \nu^{(t)})$ converges. It turns out: as $t \to \infty$, it actually converges:

$$(x^{(t)}, \nu^{(t)}) \longrightarrow (x^*, \nu^*), \tag{18}$$

as long as the step sizes $(\alpha^{(t)}, \beta^{(t)})$ are properly chosen (like decaying functions). As in Lecture 3, we will not touch upon the convergence proof, as it is out of the scope of this course.

Interior point method

Now what about for general convex optimization settings which also involve *inequality* constraints? It turns out this is a bit challenging case. It is not that simple to solve the KKT conditions (2)∼(6) directly. Instead there are algorithms which can *approximate* the KKT conditions. One such very popular algorithm is the *interior point method*.

The idea of the interior point method is to take the following two steps:

1. *Approximate* the primal problem into an *equality*-constrained optimization.
2. Apply equality-constrained-tailored algorithms (like the gradient decent algoritm explained earlier) to the approximated optimization.

Since the method is based on an *approximation* trick, one may wonder how the performance of such approach is far from optimality. It turns out that with a proper approximation trick (that we will explain soon), we can achieve the optimal solution with a small gap to the optimality. To see this, let us first investigate what the approximation trick is.

Approximation trick

Recall the standard form of general convex optimization problem including inequality constraints:

$$\begin{aligned} \min f(x) : f_i(x) \leq 0,\ i = 1, \ldots, m, \\ Ax - b = 0, \end{aligned} \tag{19}$$

where $f(x)$ and $f_i(x)$'s are convex functions, $A \in \mathbf{R}^{p \times d}$ and $p \leq d$.

Now how to handle the inequality constraints? Here what we wish to do is to somehow merge them with the objective function $f(x)$ so that we have only equality constraints. To this end, we can set up a specific goal as:

$$\begin{aligned} f_i(x) \leq 0 &\longrightarrow \text{ Make no change to } f(x); \\ f_i(x) > 0 &\longrightarrow \text{ Make the optimization problem infeasible.} \end{aligned}$$

As an effort to implement such goal, we consider a function, called the *barrier function*, which is defined as:

$$I_-(z) = \begin{cases} 0, & z \leq 0; \\ \infty, & z > 0. \end{cases} \tag{20}$$

Now inputting $f_i(x)$ to the barrier function as an argument, we get:

$$I_-(f_i(x)) = \begin{cases} 0, & f_i(x) \le 0; \\ \infty, & f_i(x) > 0. \end{cases}$$

This then motivates the following natural idea: Adding $I_-(f_i(x))$ to the objective function $f(x)$. This leads to the following reformulated problem:

$$\min f(x) + \sum_{i=1}^{m} I_-(f_i(x)) : \ Ax - b = 0. \tag{21}$$

Notice that when $f_i(x) \le 0$, the objective function is unchanged; on the other hand, whenever $f_i(x) > 0$ for some i, $f(x)$ takes infinity, making the problem infeasible.

A surrogate of the barrier function

In the reformulated problem (21), however, there is one critical issue. The issue comes from the fact that $I_-(\cdot)$ is *not differentiable*. Notice that the first KKT condition (2) includes the *gradient* term. So it requires *differentiability* of the barrier functions as they appear in the Lagrange function. However, since I_- is not differentiable, we cannot implement the KKT conditions.

To resolve such critical issue, we consider a *surrogate* of the barrier function, which is differentiable and well approximates the barrier function. One very-well known surrogate is the *logarithmic* barrier function defined as:

$$\mathsf{LB}(z) := -\mu \log(-z), \quad \mu > 0. \tag{22}$$

Note that the function is indeed differentiable, and also it well approximates the barrier function for a small μ. See Fig. 2. Moreover, it is *convex* in z, and hence, we can maintain the objective function as convex.

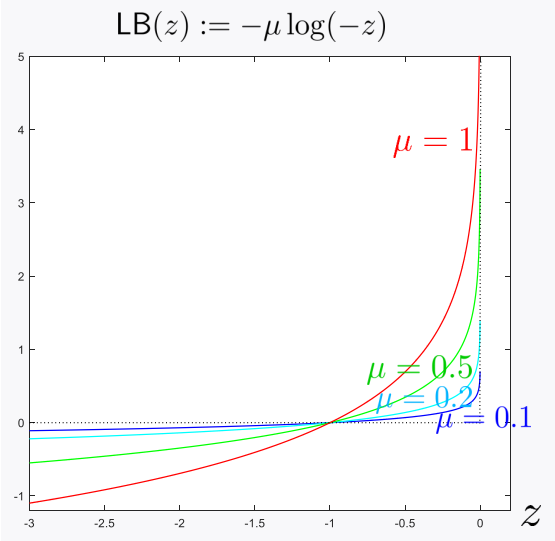


Figure 2: Logarithmic barrier function.

Approximated convex optimization

Replacing the barrier function with the logarithmic barrier function in (21), we can then approximate (21) as:

$$\min f(x) - \mu \sum_{i=1}^{m} \log(-f_i(x)) : \ Ax - b = 0. \tag{23}$$

There is one caveat here. The caveat is that the search space of x should be:

$$\{x : f_1(x) < 0, \ldots, f_m(x) < 0, \ \ Ax - b = 0\}. \tag{24}$$

This is because the equality $f_i(x) = 0$ for some i makes the logarithmic barrier function blow up. So we assume that the set in (24) is not empty. Actually this suggested the naming of "interior point method" as the method searches over interior points.

Since the approximated optimization (23) contains only the *equality* constraint, we can apply exactly the same approach that we took for the earlier equality-constrained setting. In other words, we first compute the Lagrange function:

$$\mathcal{L}(x, \nu) = f(x) - \mu \sum_{i=1}^{m} \log(-f_i(x)) + \nu^T (Ax - b). \tag{25}$$

We then try to find (x^*, ν^*) such that the following KKT conditions are satisfied:

$$\begin{aligned} \nabla_x \mathcal{L}(x^*, \nu^*) &= 0; \\ \nabla_\nu \mathcal{L}(x^*, \nu^*) &= 0. \end{aligned} \tag{26}$$

Again one can use the gradient decent algorithm to solve this.

Performance gap to the optimality

Once we employ the interior point method that is based on the *approximated* optimization (23), one natural question that one can ask is: How far is the performance of such approximation approach from optimality?

To figure this out, we first consider (x^*, ν^*) such that the KKT conditions (26) hold. At this point, we get $f(x^*)$ and obviously $f(x^*) \geq p^*$ as x^* (that satisfies (26) intended for the approximated optimization) is not necessarily the optimal solution of the original non-approximated optimization. So the performance gap can be quantified as $f(x^*) - p^*$. The gap depends obviously on μ, which is a control parameter which adjusts the closeness to the barrier function. Let us figure how it varies over μ.

Starting with strong duality, we get:

$$\begin{aligned} p^* &= d^* \\ &= \max_{\lambda \geq 0, \nu} g(\lambda, \nu) \\ &\overset{(a)}{\geq} g(\lambda^*, \nu^*) \\ &\overset{(b)}{=} \min f(x) + \sum_{i=1}^{m} \lambda_i^* f_i(x) + \nu^{*T}(Ax - b) \end{aligned} \tag{27}$$

where (a) follows from the fact that ν^* is a feasible point that satisfies (26) (not necessarily the one that maximizes $g(\lambda, \nu)$), and λ^* is another particular feasible point which will be detailed soon; and (b) is due to the definition of the dual function.

Now remember that (x^*, ν^*) is a point that satisfies (26), and hence:

$$\nabla_x \mathcal{L}(x^*, \nu^*) = \nabla f(x^*) + \sum_{i=1}^{m} \frac{-\mu}{f_i(x^*)} \nabla f_i(x^*) + A^T \nu^* = 0. \tag{28}$$

From this, we see that the particular point λ_i^* can be interpreted as:

$$\lambda_i^* = \frac{-\mu}{f_i(x^*)}. \tag{29}$$

With this interpretation, the condition (28) implies that x^* is a minimizer of the optimization in step (b) in (27). Hence, applying this to (27), we get:

$$\begin{aligned} p^* &\geq \min f(x) + \sum_{i=1}^{m} \lambda_i^* f_i(x) + \nu^{*T}(Ax - b) \\ &= f(x^*) + \sum_{i=1}^{m} \frac{-\mu}{f_i(x^*)} f_i(x^*) + \nu^{*T}(Ax^* - b) \\ &\overset{(c)}{=} f(x^*) - m\mu \end{aligned} \tag{30}$$

where (c) comes from $Ax^* - b = 0$ (since x^* must be a feasible point).

Hence, we obtain an upper bound of the gap as:

$$f(x^*) - p^* \leq m\mu.$$

Observe that the gap is at most $m\mu$, so we can almost achieve the optimality for small μ.

A special note on the choice of μ: One may want to set μ arbitrarily small to ensure almost optimal performance. In practice, however, this is not suggested. The reason is that the KKT conditions (26) are implemented via an algorithm (like the gradient decent) whose convergence speed is significantly affected by μ. The smaller μ, the slower the speed. Hence, in practice, the choice should be carefully made taking such tradeoff relationship into consideration.

Look ahead

So far we have studied what strong duality is, and derived the KKT conditions, which are necessary and sufficient conditions for strong duality to hold in convex optimization in which we claimed that the strong duality indeed holds. We also demonstrated that the KKT conditions provide detailed guidelines as to how to design algorithms. Next time, we will prove the *strong duality theorem* which we only stated without proof.

Lecture 14: Proof of Strong Duality Theorem

Recap

During the past lectures, we have investigated strong duality. First of all, in order to understand what it means, we studied the concept of primal and dual problems:

$$\begin{aligned}&\text{(Primal): } p^* := \min f(x): \ f_i(x) \le 0, \ i = 1, \dots, m, \ Ax - b = 0;\\ &\text{(Dual): } \quad d^* := \max_{\lambda,\nu} g(\lambda, \nu): \ \lambda \ge 0.\end{aligned}$$

Using these, we stated the strong duality:

$$\text{(Strong duality): } p^* = d^*. \tag{1}$$

We then argued that strong duality (1) holds for *convex* optimization of our interest, under a mild condition. The mild condition was:

$$\exists x : f_1(x) < 0, \dots, f_m(x) < 0, Ax - b = 0. \tag{2}$$

Next we derived necessary and sufficient conditions (KKT conditions) in order for strong duality to hold. Last time, we found that the KKT conditions indeed give algorithmic insights.

Today's lecture

Today we are going to prove the strong duality theorem which we deferred proving earlier. Actually the proof is not that easy. Not only the proof takes many non-trivial steps together with a bunch of ideas, but it also requires some important theorem which we did not learn about. So we will prove it step-by-step so that you guys can easily grasp how the proof goes on. Specifically we will investigate from simple to general cases: (i) *unconstrained* case; (ii) *equality*-constrained case; (iii) *inequality*-constrained case; (iv) *general* case (including both equality-&-inequality constraints). Throughout the proof, we will assume that p^* is finite. Otherwise, $p^* = -\infty$, which is definitely not an interested scenario.

Unconstrained optimization

This is a very trivial case. In this case, the primal and dual problems are:

$$\begin{aligned}p^* &:= \min f(x);\\ d^* &:= \max g.\end{aligned}$$

Since d^* is simply g, we get:

$$\begin{aligned}d^* = g : &\overset{(a)}{=} \min_{x \in \mathbf{dom} f} \mathcal{L}(x)\\ &= \min_{x \in \mathbf{dom} f} f(x) = p^*,\end{aligned}$$

where (a) is due to the definition of the dual function.

Equality-constrained optimization

Consider the primal and dual problems:

$$p^* := \min f(x) : Ax - b = 0;$$
$$d^* := \max_{\nu} g(\nu),$$

where $A \in \mathbf{R}^{p \times d}$ and $p \leq d$. Without loss of generality, assume that $\mathsf{rank}(A) = \min(p, d) = p$. Why? Otherwise, one can remove dependent rows of A to make it full-ranked. We will prove the strong duality by showing the following two:

$$p^* \geq d^*; \tag{3}$$
$$p^* \leq d^*. \tag{4}$$

In fact, (3) is what we proved earlier in Lecture 12 for a more general context. In the sequel, we will review the proof, as you may not remember details (or just for warming up).

Review of the proof of (3): $p^* \geq d^*$

Suppose that a feasible point in the primal problem, say x^*, achieves p^*; similarly, another feasible point in the dual problem, say ν^*, achieves d^*. Using the fact that (x^*, ν^*) are the minimizer and maximizer of the primal and dual problems respectively, we get:

$$\begin{aligned} p^* &= f(x^*) \\ &\overset{(a)}{=} f(x^*) + \nu^{*T}(Ax^* - b) \\ &\geq \min_{x \in \mathbf{dom} f} f(x) + \nu^{*T}(Ax - b) \\ &\overset{(b)}{=} g(\nu^*) \\ &= d^* \end{aligned}$$

where (a) follows from $Ax^* - b = 0$ for a feasible point x^*; and (b) comes from the definition of the dual function.

Proof of (4): $p^* \leq d^*$

In fact, the proof of this is not that straightforward. It relies on some trick which is based on a smartly-manipulated set (that you will see soon), as well as a well-known theorem, concerning the role of a hyperplane[1] when there are two disjoint convex sets. As of now, you may have no idea of what I am talking about. Don't worry. This will be clearer soon - please be patient.

Let us start by defining such smartly-manipulated set:

$$\mathcal{S} := \{(v, t) \in \mathbf{R}^{p+1} : \exists x \text{ such that } f(x) \leq t, Ax - b = v\}. \tag{5}$$

There are three properties for the set $\mathcal{S}$, which play a crucial role in proving (4). The first is that the set $\mathcal{S}$ contains the optimal point p^* of the primal problem when $v = 0$, i.e., $(0, p^*) \in \mathcal{S}$. Also this point is on the *boundary* of the set. Why? Suppose that $(0, p^*)$ is not on the boundary, i.e., it is strictly inside $\mathcal{S}$. Then, there exists some arbitrarily small $\epsilon > 0$ such that another point $(0, p^* - \epsilon)$ is in $\mathcal{S}$, which contradicts with the fact that p^* is the optimal value.

The second property is that the set $\mathcal{S}$ is convex. The proof of this is straightforward. Suppose $(v_1, t_1), (v_2, t_2) \in \mathcal{S}$. Then, this together with the definition (5) of the set $\mathcal{S}$ yields: there exist

[1]For those who do not remember the definition of the hyperplane, here I repeat the definition. A hyperplane is a linear subspace whose dimension is one less than that of its ambient space.

some points, say x_1 and x_2, such that

$$f(x_1) \leq t_1, \quad Ax_1 - b = v_1;$$
$$f(x_2) \leq t_2, \quad Ax_2 - b = v_2.$$

Applying an λ-weighted linear combination to the above, we get: for $\lambda \in [0, 1]$,

$$\lambda v_1 + (1-\lambda)v_2 = A(\lambda x_1 + (1-\lambda)x_2) - b;$$
$$\lambda t_1 + (1-\lambda)t_2 \geq \lambda f(x_1) + (1-\lambda)f(x_2)$$
$$\overset{(a)}{\geq} f(\lambda x_1 + (1-\lambda)x_2)$$

where (a) follows from the convexity of $f(x)$. This implies that there exists $x = \lambda x_1 + (1-\lambda)x_2 \in \mathbf{dom} f$ such that:

$$f(x) \leq \lambda t_1 + (1-\lambda)t_2;$$
$$Ax - b = \lambda v_1 + (1-\lambda)v_2,$$

which in turns yields that $\lambda(v_1, t_1) + (1-\lambda)(v_2, t_2) \in \mathcal{S}$, thus proving the convexity of $\mathcal{S}$.

The last property that I would like to emphasize is that:

$$(v, t) \in \mathcal{S} \implies (v, t') \in \mathcal{S}, \quad \forall t' \geq t. \tag{6}$$

This is obvious, since $f(x) \leq t$ implies that $f(x) \leq t'$ for $t' \geq t$. For instance, any point $(0, t') \in \mathcal{S}$ for $t' \geq p^*$, as illustrated with a blue line in Fig. 1.

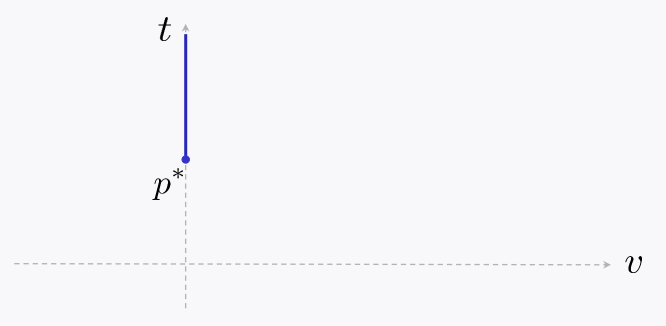


Figure 1: The set $\mathcal{S} := \{(v, t) \in \mathbf{R}^{p+1} : \exists x \text{ such that } f(x) \leq t, Ax - b = v\}$ contains the point $(0, p^*)$ as well as any point $(0, t')$ where $t' \geq p^*$.

Using the second and third properties mentioned above, one can imagine how the set looks like. Since it is convex and also any point *above the boundary* is in the set, the boundary of the set would be *bowl-shaped*, as illustrated in Fig. 2.

We are now ready to introduce a well-known theorem regarding a hyperplane, so called the *separating hyperplane theorem*. The theorem says: If there are two disjoint convex sets, then there exists a hyperplane which separates the two convex sets. Intuitively this makes sense. Why? Think about two disjoint circles in a 2-dimensional space, which are obviously convex. Then, there must be a line somewhere in between the two circles, which separates the two. Actually the proof of this trivially-looking theorem is non-trivial. So we will investigate this in PS. But don't worry - you will be given some hints while proving.

Now you may then wonder why such theorem kicks in. The reason is that the theorem allows us to come up with a hyperplane which passes through the boundary point $(0, p^*)$ while separating

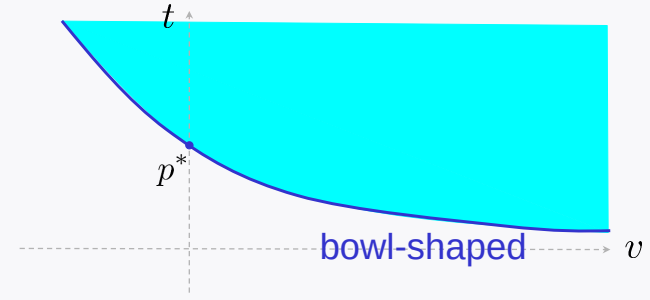


Figure 2: The boundary (marked in the blue curve) of the *convex* set $\mathcal{S} := \{(v,t) \in \mathbf{R}^{p+1} : \exists x \text{ such that } f(x) \leq t, Ax - b = v\}$ is of a *bowl shape*.

the set $\mathcal{S}$ from another disjoint convex set, and this will help us to prove (4) in the end. Why does the theorem ensure the existence of such hyperplane? To see this, consider another set, say $\mathcal{S}'$, defined as:

$$\mathcal{S}' := \{(0,s) \in \mathbf{R}^{p+1} : s < p^*\}. \tag{7}$$

Obviously this is convex (as it is just a line) and disjoint with $\mathcal{S}$. Now using the separating hyperplane theorem, we can then say that there exists a hyperplane which separates $\mathcal{S}$ from $\mathcal{S}'$ while passing through the boundary point $(0, p^*)$. Let $[\nu; \mu] \in \mathbf{R}^{p+1}$ be the supporting vector of the hyperplane. Then, the hyperplane is represented as:

$$\begin{bmatrix} \nu \\ \mu \end{bmatrix}^T \left(\begin{bmatrix} v \\ t \end{bmatrix} - \begin{bmatrix} 0 \\ p^* \end{bmatrix} \right) = 0. \tag{8}$$

Why? The separating hyperplane theorem says that whenever $(v,t) \in \mathcal{S}$, it always lies in the right-hand-side space in reference to the hyperplane, i.e.,

$$(v,t) \in \mathcal{S} \implies \begin{bmatrix} \nu \\ \mu \end{bmatrix}^T \left(\begin{bmatrix} v \\ t \end{bmatrix} - \begin{bmatrix} 0 \\ p^* \end{bmatrix} \right) \geq 0. \tag{9}$$

Also see Fig. 3 to help your understanding.

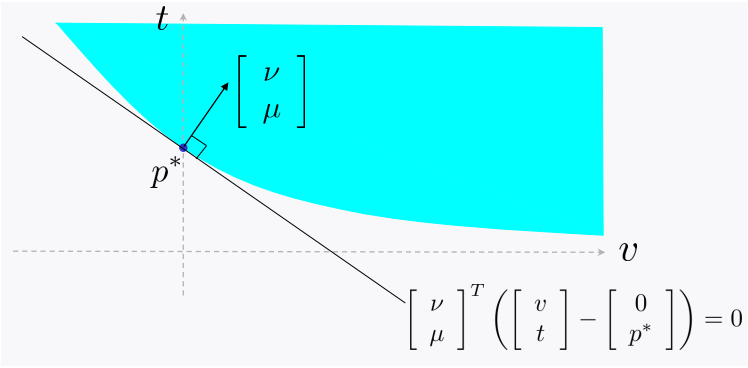


Figure 3: There exists a hyperplane that passes through $(0, p^*)$ in the set $\mathcal{S}$ while separating $\mathcal{S}$ from another disjoint convex set.

Using (9), we then see:

$$\mu p^* \leq \nu^T v + \mu t \quad \forall (v,t) \in \mathcal{S}. \tag{10}$$

One important thing to notice here is that $\mu \geq 0$. To prove this, suppose $\mu < 0$. Then, one can make $t \to \infty$. Observe that such $(v, t) = (v, \infty)$ is still in $\mathcal{S}$ due to the third property (6) of $\mathcal{S}$. But this yields a contradiction with (10) as:

$$\mu p^* \leq \nu^T v + \mu t = -\infty.$$

Notice that for finite p^* (which we assumed), μp^* is also finite, which can never go below $-\infty$. Another thing to notice is that:

$$\mu \neq 0. \tag{11}$$

We will prove this in the subsequent section. This together with $\mu \geq 0$ gives: $\mu > 0$. So we can divide both sides in (10) by $\mu > 0$ to obtain:

$$p^* \leq \left(\frac{\nu}{\mu}\right)^T v + t \quad \forall (v, t) \in \mathcal{S}. \tag{12}$$

Recall the definition of the interested set: $\mathcal{S} := \{(v, t) \in \mathbf{R}^{p+1} : \exists x \text{ such that } f(x) \leq t, Ax - b = v\}$. So for some (v, t), one can think of a point, say $x^*(v, t)$, such that:

$$\begin{aligned} x^*(v, t) &= \arg \min_{x: f(x) \leq t, Ax - b = v} t + \left(\frac{\nu}{\mu}\right)^T v \\ &\overset{(a)}{=} \arg \min_{x: f(x) \leq t, Ax - b = v} f(x) + \left(\frac{\nu}{\mu}\right)^T (Ax - b) \end{aligned} \tag{13}$$

where (a) follows from the equality constraint $Ax - b = v$ and the inequality constraint $f(x) \leq t$ (note that minimizing t is equivalent to minimizing $f(x)$). Notice in the above that (i) $f(x) \leq t$ becomes unconstrained by taking $t \to \infty$; and (ii) v is of our choice while respecting $v = Ax - b$. Hence, there exist v^* and $x^*(v^*, \infty)$ such that $f(x) + (\frac{\nu}{\mu})^T (Ax - b)$ is minimized. Putting such $x^*(v^*, \infty)$ to (12), we get:

$$\begin{aligned} p^* &\leq f(x^*(v^*, \infty)) + \left(\frac{\nu}{\mu}\right)^T (Ax^*(v^*, \infty) - b) \\ &= \min_x f(x) + \left(\frac{\nu}{\mu}\right)^T (Ax - b) \\ &\overset{(a)}{=} g\left(\frac{\nu}{\mu}\right) \\ &\overset{(b)}{\leq} d^*, \end{aligned}$$

where (a) is due to the definition of the dual function and (b) comes from the definition of d^*. This completes the proof of (4).

Proof of (11): $\mu \neq 0$

The proof idea is by contradiction. Suppose $\mu = 0$. Then, (10) implies:

$$\nu^T v \geq 0 \quad \forall (v, t) \in \mathcal{S}.$$

Since such v is in the set $\mathcal{S}$, i.e., $v = Ax - b$, we get:

$$\nu^T (Ax - b) \geq 0 \qquad \text{for some } x \text{ such that } Ax - b = v. \tag{14}$$

Two things to note. The first is that $\nu \neq 0$. This is obvious, as otherwise $(\nu, \mu) = 0$, and this implies that there does not exist a hyperplane that passes through $(0, p^*)$ in $\mathcal{S}$ while separating $\mathcal{S}$ from another disjoint convex set, which violates the separating hyperplane theorem. The second is that $Ax - b = v$ can take an *arbitrary* direction as v is of our choice and A has full rank due to $\mathsf{rank}(A) = p$. So we can choose x such that $Ax - b$ points to an arbitrary direction. This implies that there exists some point, say x', such that the direction of $Ax' - b$ is somewhat *opposite* to ν so that:

$$\nu^T(Ax' - b) < 0.$$

This contradicts with (14), thus completing the proof of (11).

Inequality-constrained optimization

For illustrative purpose, consider a simple case in which there is only one inequality constraint:

$$\begin{aligned} p^* &:= \min f(x) : f_1(x) \leq 0; \\ d^* &:= \max_{\lambda \geq 0} g(\lambda). \end{aligned}$$

It turns out that one can readily extend the proof tailored for this special case (to be presented soon) to a general case. So let us focus on this simple setting.

Again like the equality-constrained case, one can easily show that $p^* \geq d^*$. The proof is almost same. Please check this by yourself. So it suffices to prove that

$$p^* \leq d^*. \tag{15}$$

Proof of (15): $p^* \leq d^*$

Like the equality-constrained case, let us start by defining a set, which is defined similarly to (5):

$$\mathcal{S} = \{(u, t) \in \mathbf{R}^2 : \exists x \text{ such that } f_1(x) \leq u, f(x) \leq t\}. \tag{16}$$

One distinction is that we now have u which sets an upper bound on $f_1(x)$. Similar to (5), the

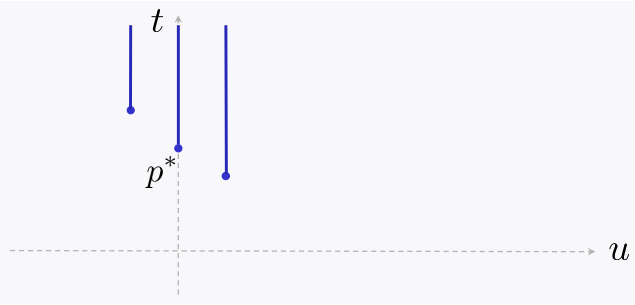


Figure 4: The set $\mathcal{S} = \{(u, t) \in \mathbf{R}^2 : \exists x \text{ such that } f_1(x) \leq u, f(x) \leq t\}$ contains the point $(0, p^*)$ as well as any point $(0, t')$ where $t' \geq p^*$. Also for $u > 0$, the minimum t in the set $\mathcal{S}$ is smaller than or equal to p^*; when $u < 0$, the minimum t is larger than or equal to p^*.

set $\mathcal{S}$ defined in (16) has the three properties: (i) It contains the boundary point $(0, p^*)$; (ii) $\mathcal{S}$ is convex; (iii) any point (u', t') where $u' \geq u$ and $t' \geq t$ is also in $\mathcal{S}$ whenever $(u, t) \in \mathcal{S}$. The first and third are obvious. The second property is also obvious if we think about a picture, as

illustrated in Fig. 4. Consider a case in which $u > 0$. In this case, the minimum $t \in \mathcal{S}$ would be *smaller than or equal to* p^*, since this is a *more relaxed scenario* relative to $u = 0$ (Why?). On the other hand, when $u < 0$, the minimum $t \in \mathcal{S}$ would be *larger than or equal to* p^*, as it is a *more constrained scenario*. With this argument, one can image a shape of the set $\mathcal{S}$ like the one in Fig. 5 (an cian colored region). So one can conjecture that it is convex. It turns out it is indeed the case. Check this rigorously in PS.

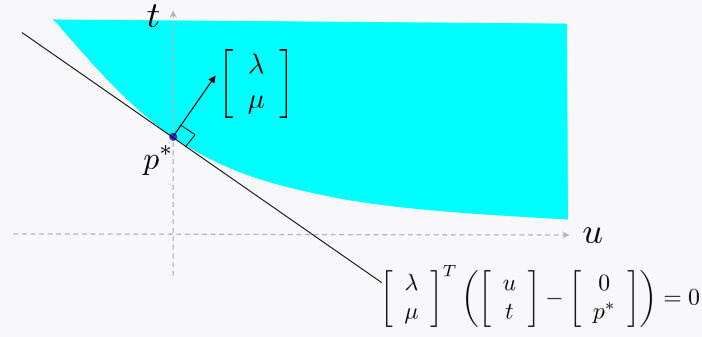


Figure 5: There exists a hyperplane (a line in this example) that passes through $(0, p^*)$ in the set $\mathcal{S} = \{(u,t) \in \mathbf{R}^2 : \exists x \text{ such that } f_1(x) \le u, f(x) \le t\}$.

Now again using the *separating hyperplane theorem*, there exists $[\lambda; \mu] \in \mathbf{R}^2 \neq 0$ such that

$$(u,t) \in \mathcal{S} \implies \begin{bmatrix} \lambda \\ \mu \end{bmatrix}^T \left(\begin{bmatrix} u \\ t \end{bmatrix} - \begin{bmatrix} 0 \\ p^* \end{bmatrix} \right) \ge 0. \tag{17}$$

Looking at the hyperplane in Fig. 5, we see that the supporting vector w.r.t. the hyperplane has a *positive* direction. Hence, one may conjecture that

$$\lambda \ge 0, \quad \mu \ge 0. \tag{18}$$

It turns out this is indeed the case. Check this in PS.

From (17), we get:

$$\mu p^* \le \lambda u + \mu t \quad \forall (u,t) \in \mathcal{S}. \tag{19}$$

Like the equality-constrained case, it turns out:

$$\mu \neq 0 \tag{20}$$

where the proof is given in the next section. This together with (18) gives $\mu > 0$. Now dividing both sides in (19) by $\mu > 0$, we obtain:

$$p^* \le \frac{\lambda}{\mu} u + t \quad \forall (u,t) \in \mathcal{S}. \tag{21}$$

Recall the definition of the interested set: $\mathcal{S} := \{(u,t) \in \mathbf{R}^2 : \exists x \text{ such that } f_1(x) \le u, f(x) \le t\}$. So for some (u,t), one can think of a point, say $x^*(u,t)$ such that:

$$\begin{aligned} x^*(u,t) &= \arg \min_{x: f_1(x) \le u, f(x) \le t} t + \frac{\lambda}{\mu} u \\ &\overset{(a)}{=} \arg \min_{x: f_1(x) \le u, f(x) \le t} f(x) + \frac{\lambda}{\mu} f_1(x) \end{aligned} \tag{22}$$

where (a) follows from the inequality constraints: $f_1(x) \leq u$ and $f(x) \leq t$. Notice in the above that $f_1(x) \leq u$ and $f(x) \leq t$ become unconstrained as $(u,t) \to (\infty,\infty)$. Hence, $x^*(\infty,\infty)$ is a minimizer of $f(x) + \frac{\lambda}{\mu} f_1(x)$. Putting the minimizer $x^*(\infty,\infty)$ to (21), we get:

$$\begin{aligned} p^* &\leq f(x^*(\infty,\infty)) + \frac{\lambda}{\mu} f_1(x^*(\infty,\infty)) \\ &= \min_x f(x) + \frac{\lambda}{\mu} f_1(x) \\ &= g\left(\frac{\lambda}{\mu}\right) \\ &\leq d^*. \end{aligned}$$

This completes the proof.

Proof of (20): $\mu \neq 0$

Again the proof idea is by contradiction. Suppose $\mu = 0$. Then, (19) implies that:

$$\lambda u \geq 0 \quad \forall (u,t) \in \mathcal{S}.$$

Applying the same argument as in (22) to the above, we get:

$$\min_x \lambda f_1(x) \geq 0. \tag{23}$$

One thing to notice is that λ (which was claimed to be non-negative in (18)) is actually strictly positive: $\lambda > 0$. Why? Otherwise, $(\lambda, \mu) = 0$ and this contradicts with the *separating hyperplane theorem*. Next, consider $f_1(x)$. Actually we have never used the mild condition (2) thus far. This is where the mild condition kicks in. Due to the condition, there exists a point, say $\bar{x}$, such that $f_1(\bar{x}) < 0$. This together with $\lambda > 0$ yields:

$$\lambda f_1(\bar{x}) < 0,$$

which contradicts with (23). This completes the proof.

General optimization

Lastly consider a general convex optimization problem which has both equality-&-inequality constraints:

$$\begin{aligned} p^* := \min f(x) : f_i(x) \leq 0 \;\; i = 1, \ldots, m, \\ Ax - b = 0; \\ d^* := \max_{\lambda \geq 0, \nu} g(\lambda, \nu), \end{aligned}$$

where $A \in \mathbf{R}^{p \times d}$, $p \leq d$ and $\mathsf{rank}(A) = p$. Here a good news is that using the techniques that we learned so far, one can prove $p^* = d^*$ in this general case as well. The only thing that we need to worry about is that there must be lots of dirty notations, so the procedure would look very much complicated. But the key procedure follows simply a combination of the ideas that we studied while investigating the above simpler cases. So we would not repeat those here. Instead you will have a chance to prove it in PS.

Look ahead

In Part I, we studied a variety of convex optimization problems: all the problems in Fig. 6. But in Part I, we did not learn how to design generic algorithms that can be applied to arbitrary scenarios. To this end, during the past lectures, we learned about strong duality and studied a generic algorithm building upon such strong duality: the *interior point method*. And today we proved the *strong duality theorem* that we deferred proving earlier. So we are essentially done with the convex optimization story.

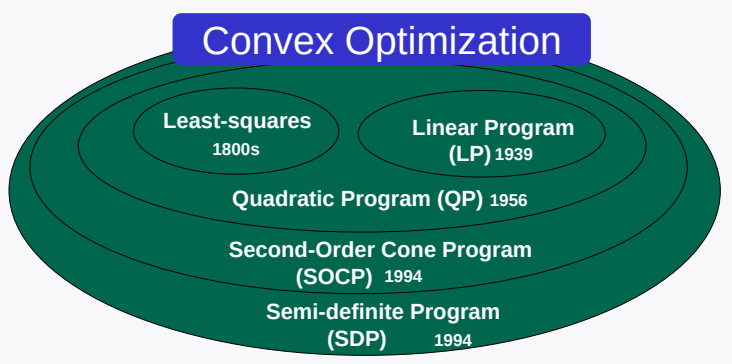


Figure 6: Convex optimization problems that we studied in Part I.

Now what is next? The end of the course? Of course not. We are just in the right middle of the course. So we may want to ask some interesting questions that can spark future studies. One very natural follow-up question is: What about for *non-convex* optimization? Can the techniques that we have learned w.r.t. convex optimization problems help address the general case? Very fortunately, it is indeed the case. It turns out those techniques can help *approximating* optimal solutions in general problems. Actually in order to understand what it means, we need to study another important theory, so called *weak duality*. So next time, we will study weak duality.

Lecture 15: Weak Duality

Recap

In Part I, we investigated many instances of convex optimization problems, ranging from LP, to Least-Squares, QP, SOCP and all the way up to SDP. See Fig. 1. But in Part I, the algorithm part was not complete. We studied only two algorithms: the simplex algorithm and gradient decent algorithm, which can be applied only to specific problem settings: some classes of QP. In other words, we did not learn how to design *generic* algorithms intended for arbitrary settings under the convex problem class. To this end, during the past lectures, we learned about strong duality and studied a generic algorithm based on the strong duality. That was, the *interior point method*. Last time, we proved the *strong duality theorem* which forms the basis of the interior point method. With all of these, we could finally end the convex optimization story!

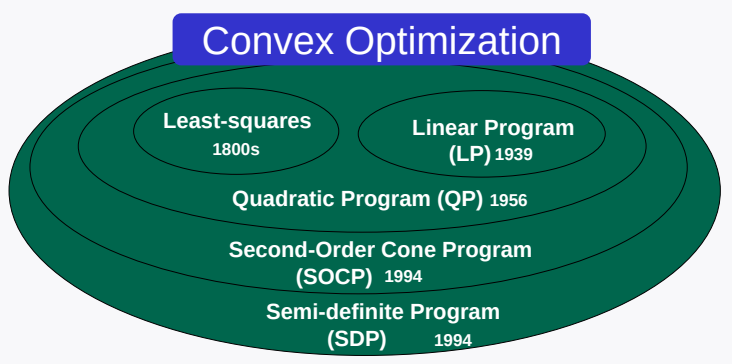


Figure 1: Convex optimization problems that we studied in Part I.

Now what is next? One of the natural questions that one can think of is: What if optimization problems of interest are *non-convex*? Can the techniques that we have studied so far while investigating the convex optimization problems help saying something about non-convex optimization problems? It turns out the answer is yes! Interestingly it has been shown that the techniques can help *approximating* optimal solutions of such non-convex optimization. In fact, in order to understand what it means, we need to study another important theory. That is, *weak duality*.

Today's lecture

So today we are going to study weak duality. Specifically we will cover the following three stuffs: (i) Will study what weak duality means and prove it; (ii) Will explore why it helps approximating non-convex optimization problems; (iii) Will investigate how far the approximated solution is from optimality.

Primal & dual problems

Let us start by recalling the standard form of general optimization problems that we investigated

in Lecture 2:

$$\begin{aligned} \min f(x) : f_i(x) \leq 0,\ i = 1, \ldots, m; \\ h_i(x) = 0,\ i = 1, \ldots, p, \end{aligned} \tag{1}$$

where $(f(x), f_i(x), h_i(x))$ are arbitrary functions, not necessarily *convex* or *affine* functions. Since we start with the above problem, let us say that the problem is *primal*.

In view of the primal problem, the Lagrange function is defined as:

$$\mathcal{L}(x, \lambda, \nu) := f(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x) \tag{2}$$

where λ_i's and ν_i's are Lagrange multipliers. And the dual function is defined as:

$$g(\lambda, \nu) := \min_{x \in \mathbf{dom} f} \mathcal{L}(x, \lambda, \nu). \tag{3}$$

Using this dual function, we can then formulate the dual problem as:

$$\text{(Dual problem): } \max_{\lambda, \nu}\ g(\lambda, \nu) :\ \lambda \geq 0. \tag{4}$$

What does weak duality mean?

Here is a summary of the primal and dual problems:

$$\begin{aligned} &\text{(Primal): } p^* := \min f(x) :\ f_i(x) \leq 0,\ i = 1, \ldots, m,\ h_i(x) = 0,\ i = 1, \ldots, p; \\ &\text{(Dual): } \quad d^* := \max_{\lambda, \nu} g(\lambda, \nu) :\ \lambda \geq 0. \end{aligned}$$

Using these, we can now state what weak duality is. What it means is that the optimal values of the two problems are of the following relationship:

$$\text{(Weak duality): } p^* \geq d^*. \tag{5}$$

Here the most critical point that I would like to emphasize is that the weak duality (5) hold for *any* optimization problems which include non-convex optimization, i.e., no matter what the function types of $(f(x), f_i(x), h_i(x))$ are. We call this the *weak duality theorem*.

Proof of the weak duality theorem (5): $p^* \geq d^*$

As you may notice, we already saw the weak duality (5) before. Actually we saw twice; one in Lecture 12, and the other in Lecture 14. But we proved it only for a specific context: the *convex optimization problem* context. It turns out the proof is not tailored for such convexity condition, i.e., it can carry over to non-convex optimization problems as well. Let us verify that it is indeed the case below.

Suppose that a feasible point in the primal problem, say x^*, achieves p^*; similarly, another feasible point in the dual problem, say (λ^*, ν^*), achieves d^*. Using the fact that x^* and (λ^*, ν^*)

are the minimizer and maximizer of the primal and dual problems respectively, we get:

$$\begin{aligned} p^* &= f(x^*) \\ &\overset{(a)}{\geq} f(x^*) + \sum_{i=1}^{m} \lambda_i^* f_i(x^*) + \sum_{i=1}^{p} \nu_i^* h_i(x^*) \\ &\geq \min_{x \in \mathbf{dom} f} f(x) + \sum_{i=1}^{m} \lambda_i^* f_i(x) + \sum_{i=1}^{p} \nu_i^* h_i(x) \\ &\overset{(b)}{=} g(\lambda^*, \nu^*) \\ &= d^* \end{aligned} \tag{6}$$

where (a) follows from the fact that $f_i(x^*) \leq 0, \lambda_i^* \geq 0$ and $h_i(x^*) = 0$ for a feasible point (x^*, λ^*, ν^*); and (b) comes from the definition of the dual function.

Why weak duality matters?

Now you may wonder why the weak duality theorem is important. The reason is that the dual problem is *always convex* no matter what the optimization type is. Take a careful look at the procedures in (6) again; we never used anything about function types of $(f(x), f_i(x), h_i(x))$. The convexity of the problem then allows us to simply focus on the dual problem (which is convex and hence tractable). By solving the tractable dual problem, we can then obtain an *approximated* solution of the original non-convex primal problem. Assuming that we can solve the dual problem exactly, we achieve d^*, thus ensuring that the performance gap to the optimality (which is p^*) is at most $p^* - d^*$.

For the rest of this lecture, we will prove that the dual problem is indeed convex. We will then discuss how good the approximated solution is for interested non-convex optimization problems.

Proof: Dual problem is convex

Let us prove that the dual problem is always convex no matter what the function types of $(f(x), f_i(x), h_i(x))$ are. Let us start by considering the dual function:

$$g(\lambda, \nu) = \min_{x \in \mathbf{dom} f} f(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x).$$

Two key observations. The first is that for any types of the functions $(f(x), f_i(x), h_i(x))$,

$$f(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x)$$

is *affine* in (λ, ν). The second observation is that taking the *minimum* of any affine functions, we obtain a *concave* function (Why? Remember what we proved in PS and even in midterm). Hence,

$$g(\lambda, \nu) \text{ is always } \textit{concave} \text{ in } (\lambda, \nu),$$

no matter what the function types of $(f(x), f_i(x), h_i(x))$ are. Therefore, the dual problem which maximizes such concave function is *convex optimization*.

How to solve dual problem?

As mentioned earlier, the weak duality and the fact that the dual problem is convex motivate us to focus on solving the dual problem to obtain an approximate solution. So let us first discuss how to solve such dual problem:

$$\text{(Dual) } d^* := \max_{\lambda,\nu} g(\lambda,\nu) : \ \lambda \geq 0.$$

Notice that the dual problem has an inequality constraint. So one cannot rely simply on the gradient decent algorithm. We should instead employ another more sophisticated algorithm. One such algorithm that we studied in Lecture 13 is: the *interior point method*. Remember that it takes the following two procedures:

1. Approximate the problem into an unconstrained problem via a logarithmic barrier function, defined as:

$$\mathsf{LB}(z) := -\mu \log(-z), \text{ for some } \mu > 0.$$

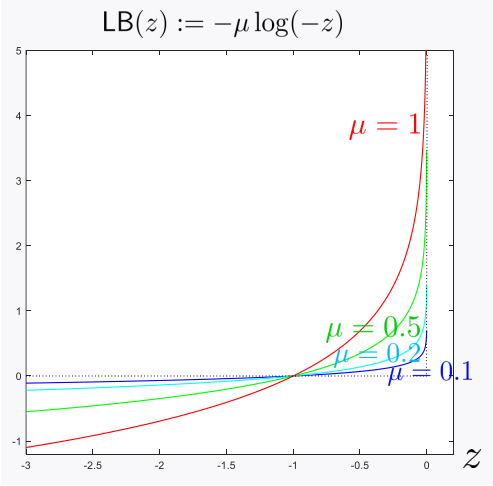


Figure 2: Logarithmic barrier function.

   Specifically, the approximated unconstrained problem is:

$$\max_{\lambda,\nu} g(\lambda,\nu) + \mu \sum_{i=1}^{m} \log \lambda_i. \tag{7}$$

2. Use the *gradient decent algorithm* to find a point, say $(\tilde{\lambda}, \tilde{\nu})$, such that it has zero gradients of the function in (7) at the point:

$$\begin{aligned} \nabla_\lambda \left(g(\lambda,\nu) + \mu \sum_{i=1}^{m} \log \lambda_i \right) \Big|_{\lambda=\tilde{\lambda},\nu=\tilde{\nu}} &= 0; \\ \nabla_\nu \left(g(\lambda,\nu) + \mu \sum_{i=1}^{m} \log \lambda_i \right) \Big|_{\lambda=\tilde{\lambda},\nu=\tilde{\nu}} &= 0. \end{aligned} \tag{8}$$

Performance of the interior point method

Remember that the performance of the interior point method depends on the control parameter μ that appears in the logarithmic barrier function. To see how it depends, consider a zero-gradient point $(\tilde{\lambda}, \tilde{\nu})$. Then, the dual function evaluated at the point should be larger than or equal to d^*, as the dual problem is about *maximization*. Hence, we get:

$$d^* \geq \quad g(\tilde{\lambda}, \tilde{\nu}),$$

Also remember what we analysed w.r.t. the gap in Lecture 13. We showed that the gap is upper-bounded by $m\mu$:

$$d^* - g(\tilde{\lambda}, \tilde{\nu}) \leq m\mu.$$

So for a sufficient small μ, we can say that:

$$g(\tilde{\lambda}, \tilde{\nu}) \approx d^*$$

Performance gap to optimality p^*

So under a choice of sufficiently small μ, the gap to the optimal value p^* in the primal problem would be:

$$\text{Gap} \approx p^* - d^*.$$

We have a terminology which indicates such gap. Since the gap is w.r.t. the *dual* problem, it is called the *duality gap*.

Let us introduce another naming which refers to such approximation technique based on the weak duality. We call such technique: *Lagrange relaxation*. Whenever we study a relaxation technique, we need to worry about how good such technique is. So one can ask: How far is the bound due to the Lagrange relaxation from the optimal value p^*?

How good is Lagrange relaxation?

Here we intend to address the question by comparing to other relaxation techniques that we investigated earlier. The first relaxation technique is the one that we studied in Lecture 5 in the context of LP. That was LP relaxation. The second is the one that we studied in Lecture 11 for a different problem context. That was SDP relaxation. So we are interested particularly in how good the Lagrange relaxation is relative to LP and SDP relaxation techniques.

Lagrange relaxation vs. LP relaxation

First let us compare to LP relaxation. Actually such relaxation arose in a very difficult problem class, called the *Boolean problems*. A more general problem including the Boolean problems as a special case is the *integer program* wherein optimization variables are constrained to be integers. So let us make a comparison under the integer program. It turns out that for such problems:

Lagrange relaxation is *at least as good as* LP relaxation.

This means that the bound due to Lagrange relaxation is closer (or equally close) to the optimal value p^* relative to the bound due to LP relaxation, showing the power of Lagrange relaxation. But an interesting result is that for the specialized Boolean problems:

Lagrange relaxation has *the same performance as* that of LP relaxation.

Lagrange relaxation vs. SDP relaxation

Now what if we compare to SDP relaxation? It turns out that in general,

Lagrange relaxation is *at least as good as* SDP relaxation.

But another very interesting result comes in for a variety of important and classical problem instances including the MAXCUT problem that we studied in Lecture 11. It turns out that for such problem instances:

Lagrange relaxation has *the same performance as* that of SDP relaxation.

Look ahead

During upcoming lectures, we will try to support the above claims. Specifically, for the Boolean problems, we will first study how to implement Lagrange relaxation. We will then compare the performance of such method to LP relaxation. Next, for the MAXCUT problem, we will do the same thing, but now making a comparison w.r.t. SDP relaxation.

Lecture 16: Lagrange Relaxation for Boolean problems

Recap

Last time we moved onto *non-convex* optimization problems. At the beginning of the last lecture, I told you that the techniques that we have learned so far w.r.t. convex optimization problems play a crucial role in solving non-convex optimization problems. To understand what it means, we needed to study another important theory regarding duality. That is, the *weak duality theorem*, which says:

$$p^* \geq d^* \text{ holds for } \textit{any} \text{ optimization problem.} \tag{1}$$

Here p^* and d^* indicate the optimal values of the primal and dual problems, respectively. One important thing that we proved here is that the dual problem is *always convex* no matter what types of the functions that appear in the optimization problem are. This motivates us to focus on solving the *tractable* dual problem, and actually one can solve it using the interior point method. Assuming that we use a sufficiently small control parameter μ that appears in the logarithmic barrier function employed for the interior point method, we can achieve d^* approximately, which in turn ensures the performance gap to the optimality to be roughly $p^* - d^*$. This gap is called the *duality gap*, and the approximation technique that leads to the gap is called *Lagrange relaxation*. At the end of the last lecture, I mentioned about the performance of such relaxation technique especially in comparison to other relaxation techniques that we learned in Part I, which are *LP relaxation* and *SDP relaxation*. Specifically I claimed that in general the Lagrange relaxation is at least as good as the other techniques, meaning that it is better than or equal to others.

Today's lecture

Today we are going to support the claim w.r.t. LP relaxation. To this end, we will cover the following three stuffs. First of all, we will review the problem context in which LP relaxation plays a powerful role. That is, the Boolean problems, which are highly non-convex and known to be quite difficult in general. Next we will study how to employ Lagrange relaxation to solve such difficult Boolean problems. Finally we will compare the performance of such approach to LP relaxation.

Review of Boolean problems

Remember that we discussed the Boolean problems in the context of LP. So the problems are basically on top of LP, but one special constraint is added. That is, the optimization variable x takes *discrete* values, like binary values. So the problem has the following standard form:

$$\begin{aligned} p^* := \min\, & w^T x : \\ & Ax - b \leq 0,\ Cx - e = 0, \\ & x_i \in \{0,1\},\ i = 1, \ldots, d. \end{aligned} \tag{2}$$

Here the last additional constraint means that the optimization variable is constrained to be *boolean*.

Equivalent form

Before studying how to apply Lagrange relaxation, let us first convert the above standard form into another with which one can play around easily. Notice that the boolean constraint in the above is not of the *inequality* or *equality* constraint form that we are familiar with. So we may want to convert it into such familiar constraint. Let us start by simplifying the optimization problem. For notational simplicity, we replace the equality constraint, $Cx - e = 0$, with two equivalent inequality constraints: $Cx - e \leq 0$ and $Cx - e \geq 0$. This way, it suffices to focus on the inequality constraint form. Also the boolean constraint can be equivalently expressed as the following *equality constraint*:

$$x_i(x_i - 1) = 0 \;\; i = 1, \ldots, d.$$

Taking this expression as well as notational simplification, we can then write (2) as:

$$\begin{aligned} p^* := \min\, & w^T x : \\ & Ax - b \leq 0, \\ & x_i(x_i - 1) = 0, \; i = 1, \ldots, d. \end{aligned} \tag{3}$$

Lagrange function

Now let us think about how to apply Lagrange relaxation. To this end, first consider the Lagrange function which is defined as:

$$\begin{aligned} \mathcal{L}(x, \lambda, \nu) &= w^T x + \lambda^T (Ax - b) + \sum_{i=1}^{d} \nu_i x_i (x_i - 1) \\ &= (w + A^T \lambda - \nu)^T x - \lambda^T b + \sum_{i=1}^{d} \nu_i x_i^2 \\ &= (w + A^T \lambda - \nu)^T x - \lambda^T b + x^T \mathsf{diag}(\nu_1, \ldots, \nu_d) x \\ &= (w + A^T \lambda - \nu)^T x - \lambda^T b + x^T D_\nu x \end{aligned} \tag{4}$$

where the last equality follows from the definition of $D_\nu := \mathsf{diag}(\nu_1, \ldots, \nu_d)$.

Dual function

Next consider the dual function:

$$g(\lambda, \nu) = \min_x x^T D_\nu x + (w + A^T \lambda - \nu)^T x - \lambda^T b. \tag{5}$$

Here $g(\lambda, \nu)$ seems about a *QP*, as it contains a *quadratic* term $x^T D_\nu x$ and D_ν is *symmetric*. But it is not clear whether the associated optimization problem is indeed a QP. Why? D_ν is not necessarily positive semi-definite (PSD).

Actually D_ν is what we can optimize over in view of the *dual problem* which takes ν (together with λ) as an optimization variable. So one can think of an easy case where the problem is indeed a QP and the solution can be derived easily. One such easy case is:

$$\text{Case I: } D_\nu \succ 0. \tag{6}$$

In this case, D_ν is positive definite, so the optimization problem in (5) is an *unconstrained QP*. So one can solve it by finding x^* such that

$$\nabla_x \mathcal{L}(x^*, \lambda, \nu) = 2D_\nu x^* + w + A^T \lambda - \nu = 0. \tag{7}$$

Hence, the minimizer and the optimal value are:

$$\begin{aligned} x^* &= -\frac{1}{2} D_\nu^{-1}(w + A^T\lambda - \nu); \\ g(\lambda,\nu) = \mathcal{L}(x^*,\lambda,\nu) &= -\frac{1}{4}(w + A^T\lambda - \nu)^T D_\nu^{-1}(w + A^T\lambda - \nu) - \lambda^T b. \end{aligned} \tag{8}$$

But the easy case (6) is actually not quite satisfactory in view of the dual problem that we will formulate soon. The reason is that (6) would serve as an inequality constraint in the dual problem, but the constraint is of a *strict* inequality form, which is not compatible with the standard form of convex optimization problems. Hence, we wish to consider the following slightly more relaxed case:

$$\text{Case II: } D_\nu \succeq 0. \tag{9}$$

In this case, D_ν is PSD, so the optimization problem in (5) is still a QP. But there is a caveat here. The caveat is that D_ν is *not necessarily invertible*. So there may be no solution x^* such that (7) holds. Here we can think of two subcases depending on whether the solution exists.

The first is the no-solution case:

$$\text{Case II-1: } w + A^T\lambda - \nu \notin \mathsf{range}(D_\nu). \tag{10}$$

In this case, there is no stationary point of x^* that satisfies (7). Hence, $\mathcal{L}(x,\lambda,\nu)$ is *unbounded below*, i.e., $g(\lambda,\nu) = -\infty$. This is definitely not an interested case. The dual problem (that we will formulate soon) is about *maximization*, so $g(\lambda,\nu) = -\infty$ is definitely not the one that we want to achieve. The other is the case in which there is indeed a solution:

$$\text{Case II-2: } w + A^T\lambda - \nu \in \mathsf{range}(D_\nu). \tag{11}$$

This is definitely of our interest. Under this case, the minimizer and the optimal value can be exactly of the same form of (8), as long as we admit the generalized definition of D_ν^{-1}:

$$[D_\nu^{-1}]_{ii} := \begin{cases} \nu_i^{-1} & \text{if } \nu_i \neq 0; \\ 0 & \text{if } \nu_i = 0. \end{cases} \tag{12}$$

On the other hand, one may want to consider the following last case:

$$\text{Case III: } D_\nu \not\succeq 0. \tag{13}$$

This means that there exists $\nu_i < 0$ for some i. This is obviously not an interested case. Notice in (4) that by setting $x_i = \infty$ for such i, we get $\mathcal{L}(x,\lambda,\nu) = -\infty$, which in turn leads to $g(\lambda,\nu) = -\infty$. Again this is not what we want to achieve.

Dual problem

Now focusing on the second case (9) in which $g(\lambda,\nu)$ is finite and we have (non-strict) inequality constraints, we obtain the dual problem as:

$$\begin{aligned} \max_{\lambda\geq 0,\nu\geq 0} g(\lambda,\nu) = \max_{\lambda\geq 0,\nu\geq 0} & -\frac{1}{4}(w + A^T\lambda - \nu)^T D_\nu^{-1}(w + A^T\lambda - \nu) - \lambda^T b : \\ & w + A^T\lambda - \nu \in \mathsf{range}(D_\nu), \end{aligned} \tag{14}$$

where the constraint $\nu \geq 0$ comes from $D_\nu \succeq 0$ (9) and $g(\lambda,\nu)$ is due to (8). As mentioned earlier, the definition of D_ν^{-1} is subject to (12).

Notice in the above that the first term in the objective function (marked in red) is not compatible with the standard form of any convex optimization problem that we studied earlier. So let us convert it into a form that we are familiar with. To this end, we first introduce a new optimization variable, say t, such that

$$t \leq -\frac{1}{4}(w + A^T\lambda - \nu)^T D_\nu^{-1}(w + A^T\lambda - \nu).$$

What it means here is that by maximizing t, one should make the RHS larger, and also by maximizing the RHS, one can set a lower bound on t larger. So this implies that maximizing t is equivalent to maximizing the RHS. Hence, with this new variable t, one can write (14) as:

$$\begin{aligned} \max_{\lambda\geq 0,\nu\geq 0} g(\lambda,\nu) = \max_{\lambda\geq 0,\nu\geq 0,t} \; & t - \lambda^T b : \\ & w + A^T\lambda - \nu \in \mathsf{range}(D_\nu); \\ & t \leq -\frac{1}{4}(w + A^T\lambda - \nu)^T D_\nu^{-1}(w + A^T\lambda - \nu). \end{aligned} \tag{15}$$

The newly-introduced inequality constraint here can be alternatively written as:

$$-t - (w + A^T\lambda - \nu)^T (4D_\nu)^{-1}(w + A^T\lambda - \nu) \geq 0. \tag{16}$$

This is actually the one that we are familiar with. This reminds you of the Schur Complement Lemma! See below for the statement.

Schur Complement Lemma: Suppose $A \succ 0$. Then,

$$X = \begin{bmatrix} A & B \\ B^T & C \end{bmatrix} \succeq 0 \quad \Longleftrightarrow \quad S := C - B^T A^{-1} B \succeq 0. \qquad \blacksquare \tag{17}$$

In view of (16) and (17), we see that (A, B, C) correspond to $4D_\nu$, $w + A^T\lambda - \nu$ and $-t$, respectively. But the above lemma is not perfectly instrumental to the setting of our interest, as it requires the invertibility of A while D_ν may not be invertible. Actually there is a generalized version of Schur Complement Lemma which deals with the case $A \succeq 0$, formally stated below:

Generalized Schur Complement Lemma: Suppose $A \succeq 0$. Then,

$$X = \begin{bmatrix} A & B \\ B^T & C \end{bmatrix} \succeq 0 \quad \Longleftrightarrow \quad S := C - B^T A^\dagger B \succeq 0; \;\; Bv \in \mathsf{range}(A) \;\; \forall v, \tag{18}$$

where $A^\dagger := U\Sigma^{-1}U^T$ when $A = U\Sigma U^T$. Here Σ^{-1} is subject to the definition of (12). $\blacksquare$

As per the above generalized lemma, as long as we define D_ν^{-1} as the one in (12), then the two constraints in (15) are equivalent to:

$$\begin{bmatrix} 4D_\nu & w + A^T\lambda - \nu \\ (w + A^T\lambda - \nu)^T & -t \end{bmatrix} \succeq 0. \tag{19}$$

We will not delve into details on the proof of this generalized lemma - that is too much distraction. But don't worry. You will have a chance to check it in PS.

Taking this conversion, we can then write (15) as:

$$\begin{aligned} d^* := \max_{\lambda,\nu,t} \; & t - \lambda^T b : \\ & \begin{bmatrix} 4D_\nu & w + A^T\lambda - \nu \\ (w + A^T\lambda - \nu)^T & -t \end{bmatrix} \succeq 0, \\ & \lambda \geq 0, \nu \geq 0. \end{aligned} \tag{20}$$

Notice that all the inequalities are the ones that we are familiar with. These are all LMIs! So this problem belongs to an SDP. Hence, one can rely on CVX to solve the problem.

LP relaxation & its dual

We are now ready to compare the performance of Lagrange relaxation (which we formulated as (20)) to that of LP relaxation. To this end, we first recall the problem that we obtained due to LP relaxation (in Lecture 5):

$$\begin{aligned} p^*_{\mathsf{LP}} := \min\, & w^T x : \\ & Ax - b \leq 0, \\ & 0 \leq x \leq 1, \end{aligned} \tag{21}$$

where x_i is now relaxed to be any real value $\in [0, 1]$.

Looking at (20) and (21), making a comparison seems not that simple. One is about *maximization*, while the other is about *minimization*. To ease comparison, we can invoke one important theorem that we learned during past lectures. That is, the *strong duality theorem*. The optimization problem (21) is an LP (a convex optimization problem). Also it satisfies the mild condition, as obviously there exists x such that strict inequality holds. Hence, the dual problem has exactly the same solution as p^*_{LP}, so alternatively one can consider the dual problem if it helps making a comparison. It turns out that the dual problem is of the following form and it indeed eases the comparison.

$$\begin{aligned} d^*_{\mathsf{LP}} := \max_{\lambda,\nu,t}\, & t - \lambda^T b : \\ & w + A^T\lambda + \nu \geq 0, \\ & -t - \nu^T 1 \geq 0, \\ & \lambda \geq 0, \nu \geq 0. \end{aligned} \tag{22}$$

We will prove (22) once we make a comparison. Please be patient.

Lagrange relaxation vs. LP relaxation

Looking at (20) and (22), we see that the two problems are very much similar. Actually such similarity can help us to show that

$$d^*_{\mathsf{LP}} = d^*. \tag{23}$$

In general non-convex optimization problems where optimization variables are constrained to be *integers*, it is known that

$$d^*_{\mathsf{LP}} \leq d^* \leq p^*. \tag{24}$$

where the second inequality is due to the weak duality theorem (1). Note that the bound due to Lagrange relaxation is closer (or equally close) to the optimality p^* relative to the bound due to LP relaxation, meaning that Lagrange relaxation is better than or equal to LP relaxation. In this particular *Boolean problem settings*, however, the equality holds interestingly. We will not prove it here. But you will have a chance to prove it in PS.

Proof of (22)

First consider the Lagrange function:

$$\begin{aligned} \mathcal{L}(x, \lambda, \nu, \mu) &= w^T x + \lambda^T (Ax - b) + \nu^T (x - 1) - \mu^T x \\ &= (w + A^T\lambda + \nu - \mu)^T x - \lambda^T b - \nu^T 1. \end{aligned}$$

Here we use a Lagrange multiplier ν for the inequality constraint of $x - 1 \leq 0$, and use μ for the other inequality constraint of $-x \leq 0$. The dual function is then:

$$g(\lambda, \nu, \mu) = \min_x (w + A^T\lambda + \nu - \mu)^T x - \lambda^T b - \nu^T 1.$$

The key observation that one can make here is that if $w + A^T\lambda + \nu - \mu \neq 0$, then one can always choose some x such that $(w + A^T\lambda + \nu - \mu)^T x < 0$. Why? There is no constraint in choosing x. So by scaling up such x infinitely, we can obtain $g(\lambda, \nu) = -\infty$. So we get:

$$g(\lambda, \nu, \mu) = \begin{cases} -\lambda^T b - \nu^T 1, & \text{if } w + A^T\lambda + \nu - \mu = 0; \\ -\infty, & \text{otherwise.} \end{cases}$$

Notice that the condition $w + A^T\lambda + \nu - \mu = 0$ together with $\mu \geq 0$ (the constraint that would appear in the dual problem) yields $w + A^T\lambda + \nu \geq 0$. Hence, the dual problem can be formulated as:

$$\begin{aligned} \max_{\lambda,\nu} & -\lambda^T b - \nu^T 1 : \\ & w + A^T\lambda + \nu \geq 0, \\ & \lambda \geq 0, \nu \geq 0. \end{aligned} \tag{25}$$

Now introducing a new variable t such that

$$t \leq -\nu^T 1,$$

we can write (25) as:

$$\begin{aligned} \max_{\lambda,\nu,t} & \; t - \lambda^T b : \\ & w + A^T\lambda + \nu \geq 0, \\ & -t - \nu^T 1 \geq 0, \\ & \lambda \geq 0, \nu \geq 0. \end{aligned} \tag{26}$$

Look ahead

In this lecture, we compared Lagrange relaxation with LP relaxation in the context of Boolean problems. Next time, we will do similar studies for the purpose of comparing to SDP relaxation. Specifically, we will study how to implement Lagrange relaxation in the context of the MAXCUT problem. We will then make a comparison to SDP relaxation.

EE523 Convex Optimization
KAIST, Spring 2019

May 2, 2019

Lecture 17: Lagrange Relaxation for the MAXCUT Problem

Recap

Last time we demonstrated the power of Lagrange relaxation in the context of Boolean problems, by proving that it yields the same performance as that offered by LP relaxation which is known to be powerful for the Boolean problems. Specifically we implemented Lagrange relaxation by formulating the dual problem of a Boolean problem as a tractable SDP. We also formulated a dual problem of the LP relaxation, which provides the same performance as the primal problem due to strong duality. We then compared them to show that both yield the same performance.

Today's lecture

Today we will do the same thing, but now by making a comparison to SDP relaxation for a different problem context. Specifically we will cover the following three stuffs. First of all, we will review the problem context in which SDP relaxation played a role. That is, the MAXCUT problem, which is highly non-convex and known to be notoriously difficult to solve. Next we will study how to employ Lagrange relaxation to solve such difficult problem. Finally we will compare the performance of such approach to SDP relaxation.

Review of the MAXCUT problem

Let us start by reviewing the goal of the MAXCUT problem. The goal of the problem is to find a set that maximizes a cut. See Fig. 1 to refresh your memory.

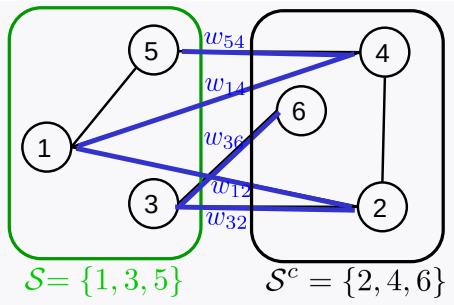


Figure 1: MAXCUT problem: Finding a set that maximizes a cut. In this example, the set $\mathcal{S} = \{1,3,5\}$ and the cut w.r.t. the set $\mathcal{S}$ is $w_{54} + w_{14} + w_{36} + w_{12} + w_{32}$. Here w_{ij} denotes a weight associated with an edge $(i,j) \in \mathcal{E}$.

To formulate an optimization problem, we introduced an optimization variable x_i which indicates whether node i is in a candidate set $\mathcal{S}$:

$$x_i = \begin{cases} +1, & x \in \mathcal{S}; \\ -1, & \text{otherwise.} \end{cases} \tag{1}$$

We then made a key observation: When $x_i \neq x_j$, the edge (i,j) comes across the two sets $\mathcal{S}$ and $\mathcal{S}^c$, and hence, this should contribute to a cut by the amount of w_{ij}; On the other hand,

when $x_i = x_j$, there should be no contribution to the cut. This naturally yielded the following optimization problem:

$$\begin{aligned} \max_{x_i} \sum_{(i,j)\in\mathcal{E}} \frac{1}{2} w_{ij}(1 - x_i x_j) : \\ x_i^2 = 1, \quad i = 1, \ldots, d, \end{aligned} \tag{2}$$

where d denotes the number of nodes in the graph and the constraint $x_i^2 = 1$ respects the fact that x_i can be only either $+1$ or -1.

A simplified form

For simplification, let us multiply the objective function by 2. Since we are familiar with a *minimization* problem, let us also change a sign in the objective to convert the problem into the following *minimization* problem:

$$\begin{aligned} \text{(Primal): } p^* := \min_{x_i} \sum_{(i,j)\in\mathcal{E}} w_{ij}(x_i x_j - 1) : \\ x_i^2 = 1, \ i = 1, \ldots, d. \end{aligned} \tag{3}$$

Let us say that this is the primal problem that we start with.

Lagrange function

Now how to implement Lagrange relaxation for the primal problem (3)? To this end, first consider the Lagrange function:

$$\begin{aligned} \mathcal{L}(x, \nu) &= \sum_{(i,j)\in\mathcal{E}} w_{ij}(x_i x_j - 1) + \sum_{i=1}^{d} \nu_i (1 - x_i^2) \\ &= -\sum_{(i,j)\in\mathcal{E}} w_{ij} + \nu^T 1 + \sum_{(i,j)\in\mathcal{E}} w_{ij} x_i x_j - \sum_{i=1}^{d} \nu_i x_i^2 \end{aligned} \tag{4}$$

where $\nu \in \mathbf{R}^d$ denotes a Lagrange multiplier that is associated with the equality constraints.

Notice that the Lagrange function (4) contains some complicated-looking terms, which are *summation* terms. One way to succinctly represent such dirty terms is to rely on *matrix* and *vector* notations. To apply this way, consider a matrix, say W, which is defined as:

$$W := \frac{1}{2} \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1d} \\ w_{21} & w_{22} & \cdots & w_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d1} & w_{d2} & \cdots & w_{dd} \end{bmatrix} \in \mathbf{R}^{d\times d}. \tag{5}$$

Note that the matrix W is *symmetric*, i.e., $W = W^T$, as we consider an *undirected* graph for the MAXCUT problem. We can also set $w_{ij} = 0$ when $(i, j) \notin \mathcal{E}$. So the diagonal entries must be zero: $w_{ij} = 0$.

With this W notation, we can then represent the summation terms in (4) as:

$$\begin{aligned}
\sum_{(i,j)\in\mathcal{E}} w_{ij} &\overset{(a)}{=} \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d} w_{ij} \\
&= [1,1,\ldots,1]\left(\frac{1}{2}\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1d} \\ w_{21} & w_{22} & \cdots & w_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d1} & w_{d2} & \cdots & w_{dd} \end{bmatrix}\right)\begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \\
&= 1^T W 1; \\
\sum_{(i,j)\in\mathcal{E}} w_{ij}x_i x_j &\overset{(b)}{=} \sum_{i=1}^{d} x_i \sum_{j=1}^{d} \frac{w_{ij}}{2} x_j \\
&= [x_1, x_2, \ldots, x_d]\left(\frac{1}{2}\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1d} \\ w_{21} & w_{22} & \cdots & w_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d1} & w_{d2} & \cdots & w_{dd} \end{bmatrix}\right)\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}
\end{aligned}$$

where (a) and (b) are due to the fact that we set $w_{ij} = 0$ when $(i,j) \notin \mathcal{E}$ and that $\sum_{i=1}^{d}\sum_{j=1}^{d} w_{ij}$ counts double.

Applying the above into (4), we then get:

$$\mathcal{L}(x,\nu) = -1^T W 1 + \nu^T 1 + x^T W x - x^T D_\nu x \tag{6}$$

where $D_\nu := \mathsf{diag}(\nu_1, \nu_2, \ldots, \nu_d)$.

Dual function

Next consider the dual function:

$$g(\nu) = \min_x x^T (W - D_\nu) x - 1^T W 1 + \nu^T 1. \tag{7}$$

Here $g(\nu)$ looks like a *QP*, as it contains a *quadratic* term $x^T(W - D_\nu)x$ and $W - D_\nu$ is *symmetric*. But as we encountered in the last lecture (Lecture 16), it is not clear whether the associated optimization problem is indeed a QP. The reason is that $W - D_\nu$ is not necessarily positive semi-definite (PSD). So let us consider two cases depending on whether $W - D_\nu$ is PSD or not.

The first case is:

$$\text{Case I: } W - D_\nu \succeq 0 \text{ (PSD)}. \tag{8}$$

In this case, the optimization problem in (7) is an *unconstrained QP*. So one can solve it by finding x^* such that

$$\nabla_x \mathcal{L}(x^*, \nu) = 2(W - D_\nu)x^* = 0. \tag{9}$$

Hence, the optimal value simply reads:

$$g(\nu) = \mathcal{L}(x^*, \nu) = -1^T W 1 + \nu^T 1. \tag{10}$$

On the other hand, the second case is:

$$\text{Case II: } W - D_\nu \not\succeq 0 \text{ (not PSD)}. \tag{11}$$

This non-PSD condition implies that there exists $u \in \mathbf{R}^d$ such that

$$u^T(W - D_\nu)u < 0.$$

Here by scaling up such u infinitely, we get $\mathcal{L}(x, \nu) = -\infty$, which in turn leads to $g(\nu) = -\infty$. Obviously this is not what we want to achieve, so we can ignore the second case.

Dual problem

Now focusing on the first case (8) in which $g(\nu)$ is finite and we have the PSD constraint of $W - D_\nu \succeq 0$, we can then obtain the dual problem as:

$$\begin{aligned} d^* &:= \max_\nu g(\nu) \\ &= \max_\nu -1^T W 1 + \nu^T 1 : \\ &\qquad W - D_\nu \succeq 0, \end{aligned} \tag{12}$$

where $g(\nu)$ is due to (10). Notice that this problem is the one that we are familiar with. That is, an SDP! Hence, one can simply rely on CVX to solve the problem.

Recall: SDP relaxation

To make a comparison to SDP relaxation, let us review what the optimization problem due to SDP relaxation was. To this end, first recall the primal problem (3) that we started with.

$$\begin{aligned} p^* := \min_x & \sum_{(i,j)\in\mathcal{E}} w_{ij} x_i x_j - 1^T W 1 : \\ & x_i^2 = 1,\ i = 1, \ldots, d. \end{aligned} \tag{13}$$

Here we represent the problem using matrix-vector notations, as an effort to ease comparison with the Lagrange-relaxed problem (12), represented with such notations.

In Lecture 11, we employed a technique, called the *lifting*, to enable SDP relaxation. The idea of lifting is to raise a vector space that the optimization variable lives in, into a *matrix* space. To apply this idea, we introduced a new matrix X such that its (i, j)-entry $[X]_{ij}$ is defined as:

$$X_{ij} := x_i x_j. \tag{14}$$

A more succinct way to represent this was: $X = xx^T$. This then yielded the following constraints:

$$X_{ii} = 1, \quad X \succeq 0, \quad \mathsf{rank}(X) = 1. \tag{15}$$

Dropping the last *rank* constraint was essentially the key idea of SDP relaxation. So the SDP-relaxed problem was:

$$\begin{aligned} p^*_{\mathsf{SDP}} := \min_X & \sum_{i,j} w_{ij} X_{ij} - 1^T W 1 : \\ & X_{ii} = 1, \quad i = 1, \ldots, d, \\ & X \succeq 0. \end{aligned} \tag{16}$$

Lagrange function of the SDP-relaxed optimization problem (16)

Looking at (12) and (16), a comparison seems not that straightforward. The problem (12) is about *maximization*, while the problem (16) is about *minimization*. So as we did in the last

lecture, to ease comparison, let us apply the strong duality theorem to obtain the dual problem of (16), which would be definitely about *maximization*.

To this end, first consider the Lagrange function w.r.t. (16). But there is a problem in formulating the Lagrange function. The problem comes from the inequality constraint form in (16). Why? The inequality is now w.r.t. a symmetric *matrix*, not a scalar or a vector. Actually we never formulated a Lagrange function w.r.t. an optimization problem which involves such *matrix-associated inequality*. So we do not know what is a proper *Lagrange multiplier* for such problem.

The Lagrange multiplier, usually denoted by λ_i, is the one that is multiplied to a component that appears in the LHS in the ith inequality constraint. But in the interested problem (16), the component is the symmetric matrix $(-X)$ which contains the number d^2 of entries $-X_{ij}$'s. So a natural way to think about a Lagrange multiplier in this matrix-associated context is that the Lagrange multiplier is also a *symmetric matrix* that contains the same number d^2 of entries. It turns out this way of defining a Lagrange multiplier enables the Lagrange function to play exactly the same role as that in the scalar-or-vector-associated problem context. Here what it means by the role includes ensuring duality-related theorems like the strong duality and weak duality theorems. You may wonder why. Don't worry. You will have a chance to check this in PS.

Let Z be such symmetric matrix that plays a role as a Lagrange multiplier. Then, the inequality-related term that appears in the Lagrange function w.r.t. (16) should read:

$$\sum_{i=1}^{d}\sum_{j=1}^{d} Z_{ij}(-X_{ij}) = -\sum_{i=1}^{d}\sum_{j=1}^{d} Z_{ij}X_{ij}. \tag{17}$$

Here $-X_{ij}$ represents just a single component of $-X$ that appears in the LHS in the inequality constraint in (16). Actually there is a succinct way to represent the above summation term (17). The way is to exploit a very popular operation used in linear algebra: $\mathsf{trace}(\cdot)$. The $\mathsf{trace}(\cdot)$ takes a square matrix as an input argument, so it is defined as: for a square matrix $A \in \mathbf{R}^{n\times n}$,

$$\mathsf{trace}(A) := \sum_{i=1}^{n} A_{ii} \tag{18}$$

where A_{ii} indicates the (i,i) diagonal entry of A. Using the trace notation, we can then rewrite (17) as:

$$\begin{aligned} -\sum_{i=1}^{d}\sum_{j=1}^{d} Z_{ij}X_{ij} &\overset{(a)}{=} -\sum_{i=1}^{d}\sum_{j=1}^{d} Z_{ij}X_{ji} \\ &\overset{(b)}{=} -\sum_{i=1}^{d}[ZX]_{ii} \\ &\overset{(c)}{=} -\mathsf{trace}(ZX) \end{aligned} \tag{19}$$

where (a) follows from $X_{ij} = X_{ji}$ (X is *symmetric*); (b) comes from the definition of matrix multiplication and the fact that $[ZX]_{ii}$ indicates the (i,i) entry of ZX; and (c) is due to the definition of $\mathsf{trace}(\cdot)$.

With such Z (Lagrange multiplier) as well as (19), we can then define the Lagrange function as:

$$\begin{aligned}\mathcal{L}(X, Z, \nu) &= -1^T W 1 + \sum_{(i,j)\in\mathcal{E}} w_{ij} X_{ij} - \mathsf{trace}(ZX) + \sum_{i=1}^{d} \nu_i (1 - X_{ii}) \\ &\overset{(a)}{=} -1^T W 1 + \nu^T 1 + \mathsf{trace}(WX) - \mathsf{trace}(ZX) - \sum_{i=1}^{d} \nu_i X_{ii} \\ &\overset{(b)}{=} -1^T W 1 + \nu^T 1 + \mathsf{trace}(WX) - \mathsf{trace}(ZX) - \mathsf{trace}(D_\nu X) \\ &\overset{(c)}{=} -1^T W 1 + \nu^T 1 + \mathsf{trace}((W - D_\nu - Z)X).\end{aligned}$$

where (a) is due to $\sum_{(i,j)\in\mathcal{E}} w_{ij} X_{ij} = \mathsf{trace}(WX)$ (Why?); (b) follows from $\sum_{i=1}^{d} \nu_i X_{ii} = \mathsf{trace}(D_\nu X)$; and (c) comes from the *linearity* of $\mathsf{trace}(\cdot)$ (Why?).

Dual problem of the SDP-relaxed optimization problem (16)

The dual function is then:

$$g(Z, \nu) = \min_X \mathsf{trace}((W - D_\nu - Z)X) - 1^T W 1 + \nu^T 1.$$

The key observation that one can make here is that if $W - D_\nu - Z \neq 0$, then one can always choose some X such that $\mathcal{L}(X, Z, \nu) = -\infty$. So it is not an interested case. Here is why. Such case implies that:

$$\exists w_{ij} - [D_\nu]_{ij} - Z_{ij} \neq 0 \text{ for some } (i, j).$$

We can then set:

$$\text{Can set } X_{ij} = -\mathsf{sign}(w_{ij} - [D_\nu]_{ij} - Z_{ij}) \times \infty$$

for such (i, j) while setting others as 0. This setting then leads to: $g(Z, \nu) = -\infty$. This is definitely not an interested case.

Considering the above, we can then get:

$$g(Z, \nu) = \begin{cases} -1^T W 1 + \nu^T 1, & \text{if } W - D_\nu - Z = 0; \\ -\infty, & \text{otherwise.} \end{cases}$$

The condition $W - D_\nu - Z = 0$ is equivalent to $W - D_\nu = Z$. Here the inequality-related Lagrange multiplier Z should respect a constraint in the dual problem. In the scalar or vector inequality constraint case, such constraint was simply $\lambda \geq 0$. It turns out the proper constraint on a symmetric matrix Z is: $Z \succeq 0$, as you may expect. Again here what it means by "proper constraint" is the one which ensures the same role as $\lambda \geq 0$, like duality-related theorems. Please be patient until you would figure this out in a crystal clear manner in PS.

The condition $W - D_\nu = Z$ together with $Z \succeq 0$ (the constraint that would appear in the dual problem) yields $W - D_\nu \succeq 0$. Hence, the dual problem is formulated as:

$$\begin{aligned} d^*_{\mathsf{SDP}} := \max_\nu \; & -1^T W 1 + \nu^T 1 : \\ & W - D_\nu \succeq 0. \end{aligned} \tag{20}$$

Lagrange relaxation vs. LP relaxation

Looking at (12) and (20), we see that the two problems are identical. Hence, we can conclude that Lagrange relaxation yields exactly the same performance as that by SDP relaxation, i.e., $d^* = d^*_{\mathsf{SDP}}$.

Summary of Part I and Part II

We have thus far studied lots of stuffs for both convex and non-convex optimization problems. In Part I, we have investigated many instances of convex optimization problems together with some algorithms (like the simplex algorithm and gradient decent algorithm) that can be applied to certain settings. One critical thing that we missed is about the development of *generic algorithms* which can be applied to arbitrary settings under the class.

To fill up the missing part, in Part II, we studied an important concept about duality: *strong duality*. We studied what it means and then figured out that it provides algorithm insights for convex optimization. Actually it leads to a famous algorithm, called the interior point method. With strong duality, we could complete the convex optimization story.

We then moved onto *non-convex* optimization problems. What we could figure out is that another important theory regarding duality helps approximating optimal solutions of non-convex optimization. That is, the *weak duality theorem*. We also figured out that the approximation technique based on such weak duality (called Lagrange relaxation) yields the best approximation performances at least for the settings in which LP and SDP relaxation are known to be powerful.

All of the above form the contents of Part I and Part II.

Outline of Part III

Now what is next? A natural follow-up question is: What can do we further with the techniques that we have learned so far? It turns out we can do something crucial in a wide variety of research fields. One such field that is quite trending these days is: *Machine Learning*.

So in Part III, we are going to study how the optimization techniques that we learned can play a role in such trending field. Specifically we are going to investigate two very important learning frameworks that arise in the field: (i) *supervised learning* in which training data contain *label* information (like the identity of emails among spam vs. legitimate emails); (ii) *unsupervised learning* in which such label information is not available. In supervised learning, we will explore one very particular yet powerful technique, called *deep learning*. In this context, you will see some role of convex optimization. In unsupervised learning, we will explore one very popular network architecture: *Generative Adversarial Networks (GANs)*. In this context, you will see a powerful role of *duality theorems* that we learned in Part II.

Lecture 18: Supervised Learning and Optimization

Recap

Last time we ended Part II. In Part II, we focused on the study of the two important theorems regarding duality: (i) strong duality theorem; (ii) weak duality theorem. The strong duality theorem provided algorithmic insights, thus leading to the interior point method, which can be applied to generic settings that we have investigated in Part I. The weak duality theorem helped us to approximate optimal solutions for non-convex optimization problems, which are intractable in general.

At the end of the last lecture, we then claimed that what we have learned so far are quite instrumental to addressing important issues that arise in a recent trending research field: *Machine Learning*. So the goal of Part III is to support this claim.

Today's lecture

Today we will start investigating the field of machine learning and roles of optimization in the field. Specifically we will cover the following three stuffs. First of all, we will study what *machine learning* means and what the mission of the field is. We will then investigate one very popular and classical way to achieve the mission:

Supervised Learning.

Lastly we will explore how optimization techniques are related to supervised learning. It turns out there are two very popular optimization frameworks that implement supervised learning. So we will also investigate them accordingly.

Machine learning

Let us start by investigating what machine learning means. Machine learning is about an algorithm which is defined to be a set of instructions that a computer system can execute. Formally speaking, machine learning is the study of *algorithms* with which one can train a computer system so that it can perform a specific task of interest. Pictorially, it means the following; see Fig. 1.

Here the entity that we are interested in building up is a computer system, which is definitely a *machine*. Since it is a system (i.e., a function), it has input and output. The input, usually denoted by x, indicates information which is employed to perform a task of interest. The output, usually denoted by y, indicates a task result. For instance, if a task of interest is legitimate-emails filtering against spam emails that we studied in Part I, then x could be *features* (e.g., frequencies of some keywords that appear in an email), and y is an email entity, e.g., $y = +1$ indicates a legitimate email while $y = -1$ denotes a spam email. Or if an interested task is cat-vs-dog classification, then x could be *image-pixel values* and y is a binary value indicating whether the fed image is a cat (say $y = 1$) or a dog ($y = 0$).

Machine learning is about designing *algorithms*, wherein the main role is to train (teach) the computer system (machine) so that it can perform such task well. In the process of designing

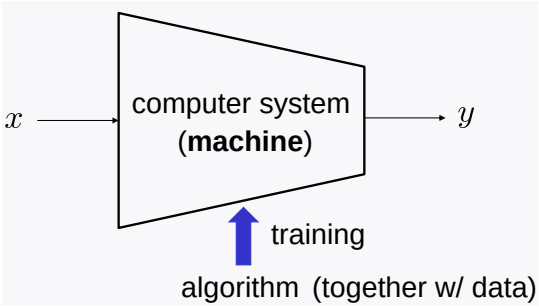


Figure 1: Machine learning is the study of algorithms which provide a set of explicit instructions to a computer system (machine) so that it can perform a specific task of interest. Let x be an input which indicates information employed to perform a task. Let y be an output to denote a task result.

algorithms, we use something, called *data*.

Why called "Machine Learning"?

Here you may want to ask about the naming of machine learning. One can easily see the rationale if one can change a viewpoint. From a *machine's perspective*, one can say that a *machine learns* the task from data. Hence, it is called *machine learning*, ML for short.

This naming was originated in 1959 of course from the Father of the field: Arthur Lee Samuel. See Fig. 2.

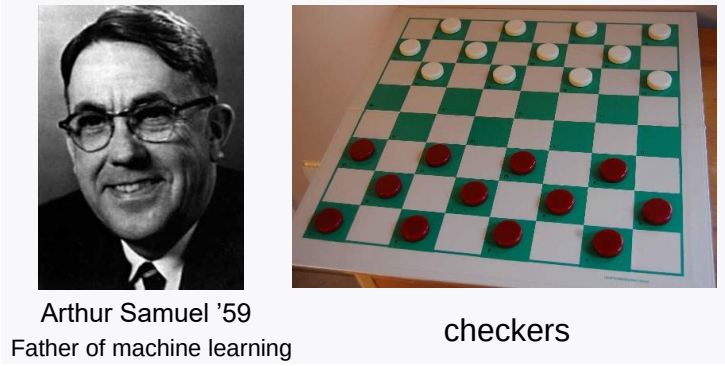


Figure 2: Arthur Lee Samuel is an American pioneer in the field of artificial intelligence, known mostly as the Father of machine learning. He found the field in the process of developing computer checkers which later formed the basis of AlphaGo.

He is actually one of the pioneers in the broader field of *Artificial Intelligence* (AI) which includes machine learning as a sub-field. The AI field is the study of creating *intelligence* by *machines*, unlike the *natural intelligence* displayed by intelligent beings like humans and animals. Since the ML field is about building up a human-like machine, it is definitely a sub-field of AI.

In fact, Arthur Samuel found the ML field in the process of developing a human-like computer player for a board game, called *checkers*; see the right figure in Fig. 1. He proposed many algorithms and ideas while developing computer checkers. It turns out very interestingly, those algorithms could form the basis of *AlphaGo*, a computer program for the board game Go which defeated one of the 9-dan professional players, Lee Sedol, for the first time, with 4 wins out of

5 games in 2016.

Mission of machine learning

As mentioned earlier, ML is a sub-field of AI. So its end-mission is obviously *achieving artificial intelligence*. So in view of the block diagram in Fig. 1, one can say that the goal of ML is to design an algorithm (a trainer from a machine's perspective) so that the trained machine *behaves like intelligence beings*.

Supervised learning

There are some methodologies which help us to achieve the goal of ML. One specific yet very popular method is the one called:

Supervised Learning.

What supervised learning means is *learning* a function $f(\cdot)$ (indicating a functional of the machine) with the help of a *supervisor*. See Fig. 3.

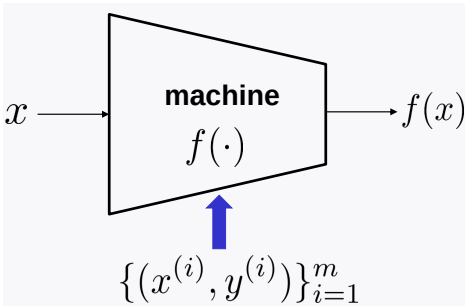


Figure 3: Supervised Learning: Design a computer system (machine) $f(\cdot)$ with the help of a supervisor which offers input-output pair samples, called a training dataset $\{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$.

What the supervisor means in this context is the one who provides input-output samples. Obviously the input-output samples form *data* employed for training the machine, usually denoted by:

$$\{(x^{(i)}, y^{(i)})\}_{i=1}^{m}, \tag{1}$$

where $(x^{(i)}, y^{(i)})$ indicates the ith input-output sample (or called a *training sample* or an *example*) and m denotes the number of samples. Using this notation (1), one can say that supervised learning is to:

$$\text{Estimate } f(\cdot) \text{ using the training samples } \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}. \tag{2}$$

Optimization

A common way to estimate $f(\cdot)$ is through *optimization*. Actually this is exactly how the optimization techniques that we learned so far are related to supervised learning.

In view of the goal (2) of supervised learning, what we want is:

$$y^{(i)} \approx f(x^{(i)}), \quad \forall i \in \{1, \ldots, m\}.$$

A natural question that arises is then: How to quantify closeness between the two quantities: $y^{(i)}$ and $f(x^{(i)})$? One very common way that has been used in the field is to employ a function, called a *loss* function, usually denoted by:

$$\ell(y^{(i)}, f(x^{(i)})). \tag{3}$$

One obvious property that the loss function $\ell(\cdot, \cdot)$ should have is that it should be small when the two arguments are close, while being zero when the two are identical. One very popular loss function that you saw earlier is: the squared-error function, introduced by the Father of optimization, Gauss: $\|y^{(i)} - f(x^{(i)})\|^2$.

Using such loss function (3), one can then formulate an optimization problem as follows:

$$\min_{f(\cdot)} \sum_{i=1}^{m} \ell(y^{(i)}, f(x^{(i)})). \tag{4}$$

Actually this is not of the conventional optimization problem structure that you are familiar with. In (4), there is *no optimization variable*. Instead we have a different quantity that we can optimize over: *the function* $f(\cdot)$.

We never saw this type of optimization, called *function optimization*. How to deal with such function optimization? There is one very prominent approach in the field. The approach is to represent the function $f(\cdot)$ with *parameters (or called weights)*, denoted by w, and then consider the weights as an optimization variable. Taking this approach, one can then translate the problem (4) into:

$$\min_{w} \sum_{i=1}^{m} \ell(y^{(i)}, f_w(x^{(i)})) \tag{5}$$

where $f_w(x^{(i)})$ denotes the function $f(x^{(i)})$, parameterized by w.

Now the question of interest is: How is the optimization problem (5) related to convex optimization problems that we have thus far learned about? To see this, we need to check whether or not the objective function $\ell(y^{(i)}, f_w(x^{(i)}))$ is convex in the optimization variable w. Obviously the convexity depends on how we define the two functions: (i) $f_w(x^{(i)})$ w.r.t. w; (ii) the loss function $\ell(\cdot, \cdot)$.

Introduction of Neural Networks

How to define the two functions? Around at the same time when the ML field was founded, one architecture was suggested for the first function $f_{(\cdot)}(x)$ in the context of simple binary classifiers in which y takes one among the two options only. The architecture is called:

$$\textit{Perceptron},$$

and was invented in 1957 by one of the pioneers in the AI field, named Frank Rosenblatt. Interestingly, Frank Rosenblatt was a *psychologist*. So he was interested in how brains of intelligent beings work and his study on brains led him to come up with the perceptron which is inspired by the brain structure and therefore gave significant insights into *neural networks* that many of you guys heard of.

How brains work

Here are details on how the brain structure inspired the architecture of the perceptron. Inside a brain, there are many *electrically excitable cells*, called *neurons*; see Fig. 4.

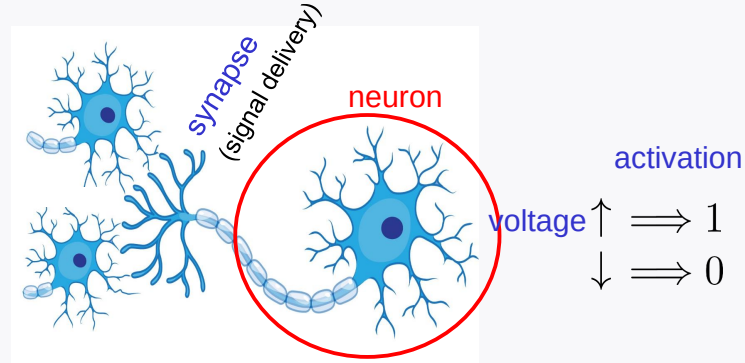


Figure 4: Neurons are electrically excitable cells and are connected through synapses.

Here a red-circled one indicates a neuron. So the figure shows three neurons in total. There are three major properties about neurons that led to the architecture of perceptron.

The first is that a neuron is an *electrical* quantity, so it has a *voltage*. The second property is that neurons are connected with each other through mediums, called *synapses*. So the main role of synapses is to deliver electrical voltage signals across neurons. Depending on the connectivity strength level of a synapse, a voltage signal from one neuron to another can increase or decrease. The last is that a neuron takes a particular action, called *activation*. Depending on its voltage level, it generates an all-or-nothing pulse signal. For instance, if its voltage level is above a certain threshold, then it generates an impulse signal with a certain magnitude, say 1; otherwise, it produces nothing.

Perceptron

The above three properties about neurons led Frank Rosenblatt to propose the architecture of perceptron, as illustrated in Fig. 5.

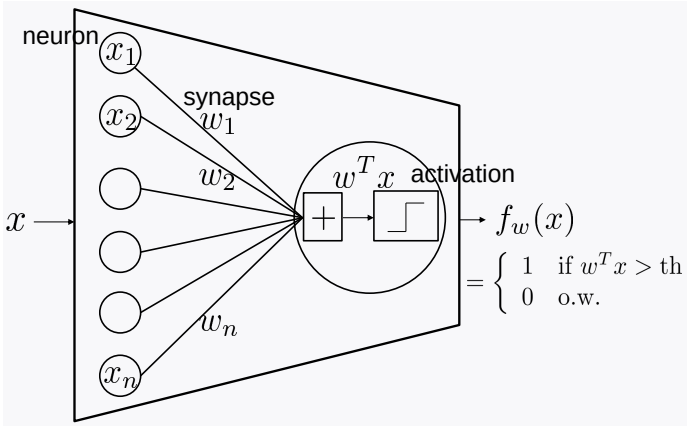


Figure 5: An architecture of perceptron.

Let x be an n-dimensional real-valued signal: $x := [x_1, x_2, \ldots, x_n]^T$. Suppose each component x_i is distributed to each neuron, and let x_i indicates a voltage level of the ith neuron. The voltage signal x_i is then delivered through a synapse to another neuron (placed on the right in the figure, indicated by a big circle). Remember that the voltage level can increase or decrease depending on the connectivity strength of a synapse. To capture this, a weight, say w_i, is

multiplied to x_i so $w_i x_i$ is a delivered voltage signal at the terminal neuron. Based on another observation that people made on neurons that the voltage level at the terminal neuron increases with more connected neurons, Rosenblatt introduced an adder which simply aggregates all the voltage signals coming from many neurons, so he modeled the voltage signal at the terminal neuron as:

$$w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = w^T x. \tag{6}$$

Lastly in an effort to mimic the *activation*, he modeled the output signal as

$$f_w(x) = \begin{cases} 1 & \text{if } w^T x > \text{th}, \\ 0 & \text{o.w.} \end{cases} \tag{7}$$

where "th" indicates a certain threshold level. It can also be simply denoted as

$$f_w(x) = \mathbf{1}\{w^T x > \text{th}\}. \tag{8}$$

Activation functions

Taking the percentron architecture in Fig. 5, one can then formulate the optimization problem (5) as:

$$\min_w \sum_{i=1}^m \ell(y^{(i)}, \mathbf{1}\{w^T x^{(i)} > \text{th}\}). \tag{9}$$

This is an initial optimization problem that people developed, inspired by perceptron. However, people immediately figured out there is an issue in solving this optimization. The issue comes from the fact that the objective function contains an indicator function, so it is *not differentiable*. Why not being differentiable problematic then? Remember the algorithms that we learned in the past: gradient decent algorithm and the interior point method. All of them involve *derivatives* operations. So the non-differentiability of the objective function does not allow to employ such algorithms.

What can we do then? One typical way that people have taken in the field is to *approximate* the activation function. There are many ways for approximation. From below, we will investigate two approaches.

Least-Squares classifier

The first approach is a very naive one, which is simply taking the input as an output. This is called *linear approximation*:

$$f_w(x) = w^T x. \tag{10}$$

A good thing about this approximation is that the linear function is differentiable and also convex. For this function, people thought of also a very well-known loss function, which is the squared-error function: $\ell(y, \hat{y}) = \|y - \hat{y}\|^2$. This then leads to the following optimization problem:

$$\min_w \sum_{i=1}^m \|w^T x^{(i)} - y^{(i)}\|^2. \tag{11}$$

Notice that this is the *Least-Squares* optimization problem that we studied earlier:

$$\min_w \|Aw - b\|^2 \tag{12}$$

where

$$A := \begin{bmatrix} x^{(1)T} \\ \vdots \\ x^{(m)T} \end{bmatrix}, \; b := \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix}.$$

Logistic regression

The second approach is to take a more accurate approximation. To this end, it takes sort of a *smooth* transition from 0 to 1:

$$f_w(x) = \frac{1}{1 + e^{-w^T x}}. \tag{13}$$

Notice that $f_w(x) \approx 0$ when $w^T x$ is very small; it then grows exponentially with an increase in $w^T x$; later grows logarithmically; and finally saturates as 1 when $w^T x$ is very large. Actually the function (13) is a very popular one used in statistics, called the *logistic*[1] function. There is another name for the function, which is the *sigmoid*[2] function.

There are two good things about the logistic function. First it is differentiable. The second is that it can play a role as the *probability* for the output in the binary classifier, e.g., $\Pr(y = 1)$ where y denotes the ground-truth label in the binary classifier. So it is very much interpretable.

For this function, people came up with a loss function, which turns out to be *optimal in some sense* and expressed as:

$$\ell(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}). \tag{14}$$

This function is called *cross-entropy* and the rationale behind the naming will be explained later (next lecture). Taking the logistic function together with the cross-entropy loss function, we can then formulate the problem (5) as:

$$\min_w \sum_{i=1}^{m} -y^{(i)} \log \frac{1}{1 + e^{-w^T x^{(i)}}} - (1 - y^{(i)}) \log \frac{e^{-w^T x^{(i)}}}{1 + e^{-w^T x^{(i)}}}. \tag{15}$$

It turns out this optimization problem is *convex* and provides the *optimal* performance in some sense. The name of this classifier is *logistic regression*.

activation	loss	classifier
$f_w(x) = w^T x$	$\ell(y, \hat{y}) = \|y - \hat{y}\|^2$	Least-Squares
$f_w(x) = \frac{1}{1 + e^{-w^T x}}$	$\ell(y, \hat{y}) = -y \log \hat{y}$ $-(1 - y) \log(1 - \hat{y})$	Logistic regression

Figure 6: Two popular classifiers.

Look ahead

Next time, we will show that the logistic regression is indeed a *convex* classifier. We will also study in what sense the logistic regression is *optimal* and will prove the optimality.

[1] The word *logistic* comes from a Greek word which means a slow growth, like a logarithmic growth.

[2] Sigmoid means resembling the lower-case Greek letter sigma, S-shaped.

Lecture 19: Logistic regression

Recap

Last time we embarked on Part III which deals with application topics of optimization techniques. We investigated one such topic (that arises in the context of a trending field, machine learning) where optimization techniques play a significant role. That is, supervised learning, which serves as a powerful and classical methodology in achieving the goal of ML. The goal of supervised learning is to estimate a function $f_w(\cdot)$ of a computer system (machine) using input-output samples: $\{(x^{(i)}, y^{(i)})\}_{i=1}^m$ where m denotes the number of training samples. Here w indicates a collection of parameters which serve to implement the function $f_w(\cdot)$. For the function $f_w(\cdot)$, we studied one specific yet historical architecture, inspired by brains' neural networks of intelligent beings: *perceptron*. It first takes a linear operation with an input, say x, to compute $w^T x$. It then passes it to an activation function to yield an output, say $\hat{y} := f_w(x)$; see Fig. 1.

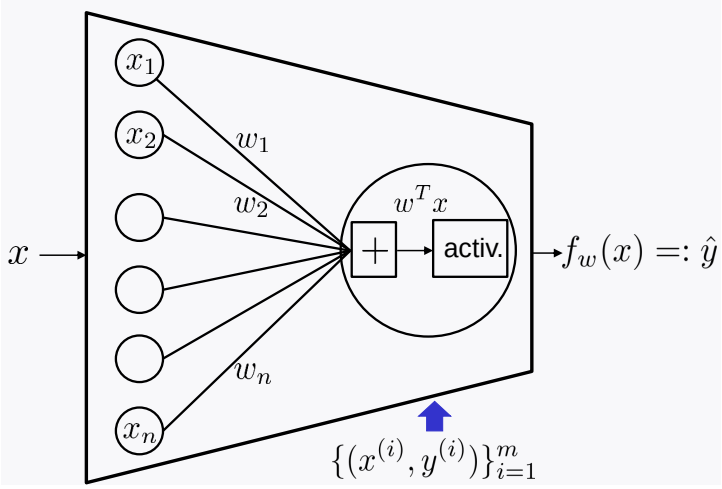


Figure 1: An architecture of Perceptron.

We then formulated an optimization problem accordingly:

$$\min_w \sum_{i=1}^m \ell(y^{(i)}, f_w(x^{(i)})) \tag{1}$$

where $\ell(y^{(i)}, f_w(x^{(i)}))$ indicates a loss function which plays a role to quantify closeness between the ground truth label $y^{(i)}$ and its estimate $f_w(x^{(i)})$. Since the impulse-shaped activation function that Frank Rosenblatts introduced initially is not differentiable and thus inapplicable to algorithms, we considered two other functions: (i) the linear function $f_w(x) = w^T x$; and (ii) the logistic function $f_w(x) = \frac{1}{1+e^{-w^T x}}$. Taking the linear activation function together with the squared-error loss function, we obtained the Least-Squares classifier.

Taking the logistic function together with a different loss function, called the *cross-entropy loss*,

$$\ell_{\mathsf{entropy}}(y, \hat{y}) = -y \log \hat{y} - (1-y) \log(1-\hat{y}), \tag{2}$$

we obtained a classifier, named *logistic regression*. We then claimed that the logistic regression is formulated as a *convex* optimization problem. We also claimed that the classifier based on the cross-entropy loss (2) yields the *optimal* performance in *some sense*.

Today's lecture

Today we will prove this claim. Specifically what we are going to do are three folded. First, we will show that logistic regression is indeed a *convex* classifier. Next, we will investigate *in what sense* the classifier is *optimal*. We will then prove the optimality. We will also discuss how to solve the optimization problem of logistic regression.

Optimization problem of logistic regression

Taking the logistic function and the cross-entropy loss (2), we obtain the logistic regression optimization problem:

$$\begin{aligned}
&\min_w \sum_{i=1}^m \ell_{\mathsf{entropy}}(y^{(i)}, f_w(x^{(i)})) \\
&= \min_w \sum_{i=1}^m \ell_{\mathsf{entropy}}\left(y^{(i)}, \frac{1}{1+e^{-w^T x^{(i)}}}\right) \\
&= \min_w \sum_{i=1}^m -y^{(i)} \log \frac{1}{1+e^{-w^T x^{(i)}}} - (1-y^{(i)}) \log \frac{e^{-w^T x^{(i)}}}{1+e^{-w^T x^{(i)}}}.
\end{aligned} \tag{3}$$

Proof of convexity

Let us first prove that the optimization problem (3) is indeed convex. Since convexity preserves under addition, it suffices to prove:

$$\begin{aligned}
&\text{(i)} - \log \frac{1}{1+e^{-w^T x}} \text{ is convex in } w; \\
&\text{(ii)} - \log \frac{e^{-w^T x}}{1+e^{-w^T x}} \text{ is convex in } w.
\end{aligned}$$

Since the second function in the above can be represented as the sum of an affine function and the first function:

$$-\log \frac{e^{-w^T x}}{1+e^{-w^T x}} = w^T x - \log \frac{1}{1+e^{-w^T x}},$$

it suffices to prove the convexity of the first function.

Notice that the first function can be rewritten as:

$$-\log \frac{1}{1+e^{-w^T x}} = \log(1+e^{-w^T x}). \tag{4}$$

Taking a derivative of the RHS formula in (4) w.r.t. w, we get:

$$\nabla_w \log(1+e^{-w^T x}) = \frac{-x e^{-w^T x}}{1+e^{-w^T x}}.$$

This is due to a chain rule of derivatives. Taking another derivative of the above, we obtain a Hessian as follows:

$$\begin{aligned}\nabla_w^2 \log(1+e^{-w^T x}) &= \nabla_w \left(\frac{-xe^{-w^T x}}{1+e^{-w^T x}} \right) \\ &\overset{(a)}{=} \frac{xx^T e^{-w^T x}(1+e^{-w^T x}) - xx^T e^{-w^T x} e^{-w^T x}}{(1+e^{-w^T x})^2} \\ &= \frac{xx^T e^{-w^T x}}{(1+e^{-w^T x})^2} \\ &\succeq 0\end{aligned} \tag{5}$$

where (a) is due to the derivative rule of a quotient of two functions. As the Hessian is PSD, we prove the convexity.

In what sense logistic regression optimal?

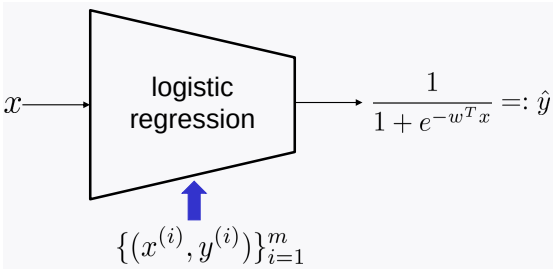


Figure 2: Logistic regression

Notice that the range of the output $\hat{y}$ of the logistic regression classifier is in between 0 and 1:

$$0 \leq \hat{y} \leq 1.$$

Hence, one can interpret this as a *probability* quantity. Then, what would $\hat{y}$ be in order to ensure a *good* classifier? Since we are interested in classifying whether $y = 1$ or $y = 0$ given x, a good classifier should yield $\hat{y}$ to well serve as sort of *critical* information that helps classification. One such critical information *of probability type* would be: $\Pr(y = 1|x)$. Hence, one very natural wish that one can imagine for ensuring a good classifier is:

$$\hat{y} = \Pr(y = 1|x). \tag{6}$$

We are now ready to explain in which sense we are talking about optimality. Here what we mean by *optimality* is:

$$\text{Logistic regression is optimal } \textit{in a sense of achieving the wish} : \ \hat{y} = \Pr(y = 1|x). \tag{7}$$

Then, a question that arises is: How to achieve the wish? Unfortunately, there is no good direct way to achieve the wish. One direct way is to *empirically estimate* $\Pr(y = 1|x)$ from the training samples $\{(x^{(i)}, y^{(i)})\}_{i=1}^m$ which are given in the problem. But there is a significant problem in performing an *empirical estimation*. To see this, let us consider a case in which we wish to estimate $\Pr(y = 1|x = (0.5, 0.2))$ where x is a particular two-dimensional vector, e.g., $x = (0.5, 0.2)$. To come up with a good empirical estimation of such quantity, we need a

sufficiently large number of training samples in which $y^{(i)} = 1$ and $x^{(i)} = (0.5, 0.2)$. But in many applications of interest, x is a *real-valued* quantity. So in general, there would no such training sample. Therefore, such empirical estimation is not a viable way.

Fortunately, there is an *indirect yet viable* approach to achieve the wish (6). This is based on the following natural expectation:

$$\text{If } \hat{y} = \Pr(y = 1|x), \text{ then such } \hat{y} \text{ would well explain the statistics of } \{(x^{(i)}, y^{(i)})\}_{i=1}^m. \tag{8}$$

Here you may wonder what it means by "well explain the statistics of training data". What that means is that it makes the occurrence of such training data *highly likely (probable)*, i.e., it makes the following probability maximized:

$$\begin{aligned} &\Pr\left(\{(x^{(i)}, y^{(i)})\}_{i=1}^m\}\right) \\ =&\Pr\left(\{(x^{(1)}, y^{(1)})\} \cap \cdots \cap \{(x^{(m)}, y^{(m)})\}\right). \end{aligned} \tag{9}$$

Here the key observation is that once we assume that $\hat{y} = \Pr(y = 1|x)$, then the probability (9) of training data being occurred is a function of $\hat{y}$. Hence the probability is also a function of w, as $\hat{y}$ depends on how we choose w. Therefore, one can say that the optimal w is the one that maximizes such probability (9):

$$w^* := \arg\max_w \Pr\left(\{(x^{(1)}, y^{(1)})\} \cap \cdots \cap \{(x^{(m)}, y^{(m)})\}\right), \tag{10}$$

assuming that we set $\hat{y} = \Pr(y = 1|x)$.

As mentioned earlier, logistic regression yields such optimal w under a mild assumption. Here the mild assumption is that samples are independent across the samples:

$$\{(x^{(i)}, y^{(i)})\}_{i=1}^m \text{ are independently.} \tag{11}$$

Proof of optimality

Consider the probability of interest (9):

$$\begin{aligned} &\Pr\left(\{(x^{(1)}, y^{(1)})\} \cap \cdots \cap \{(x^{(m)}, y^{(m)})\}\right) \\ &\overset{(a)}{=} \prod_{i=1}^m \mathbb{P}\left(x^{(i)}, y^{(i)}\right) \\ &\overset{(b)}{=} \prod_{i=1}^m \mathbb{P}(x^{(i)}) \mathbb{P}\left(y^{(i)}|x^{(i)}\right) \end{aligned} \tag{12}$$

where (a) comes from the mild assumption (11); and (b) follows from the definition of conditional probability. Here $\mathbb{P}(x^{(i)}, y^{(i)})$ denotes the probability distribution of input and output of the system:

$$\mathbb{P}(x^{(i)}, y^{(i)}) := \Pr(X = x^{(i)}, Y = y^{(i)}) \tag{13}$$

where X and Y indicate random variables of the input and the output, respectively.

Recall the setting (6) that we assumed:

$$\hat{y} = \Pr(y = 1|x).$$

This implies that:

$$\begin{aligned} y = 1: &\quad \mathbb{P}(y|x) = \hat{y} \\ y = 0: &\quad \mathbb{P}(y|x) = 1 - \hat{y}. \end{aligned}$$

Hence, one can represent $\mathbb{P}(y|x)$ as:

$$\mathbb{P}(y|x) = \hat{y}^y (1-\hat{y})^{1-y}.$$

Now using the notations of $(x^{(i)}, y^{(i)})$ and $\hat{y}^{(i)}$, we then get:

$$\mathbb{P}\left(y^{(i)}|x^{(i)}\right) = (\hat{y}^{(i)})^{y^{(i)}} (1-\hat{y}^{(i)})^{1-y^{(i)}}.$$

Plugging this into (12), we get:

$$\begin{aligned} &\Pr\left(\{(x^{(1)}, y^{(1)})\} \cap \cdots \cap \{(x^{(m)}, y^{(m)})\}\right) \\ &= \prod_{i=1}^{m} \mathbb{P}(x^{(i)}) \prod_{i=1}^{m} (\hat{y}^{(i)})^{y^{(i)}} (1-\hat{y}^{(i)})^{1-y^{(i)}}. \end{aligned} \tag{14}$$

This together with (10) yields:

$$\begin{aligned} w^* &:= \arg\max_w \prod_{i=1}^{m} \mathbb{P}(x^{(i)}) \prod_{i=1}^{m} (\hat{y}^{(i)})^{y^{(i)}} (1-\hat{y}^{(i)})^{1-y^{(i)}} \\ &\overset{(a)}{=} \arg\max_w \prod_{i=1}^{m} (\hat{y}^{(i)})^{y^{(i)}} (1-\hat{y}^{(i)})^{1-y^{(i)}} \\ &\overset{(b)}{=} \arg\max_w \sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)}) \\ &\overset{(c)}{=} \arg\min_w \sum_{i=1}^{m} -y^{(i)} \log \hat{y}^{(i)} - (1-y^{(i)}) \log(1-\hat{y}^{(i)}) \end{aligned} \tag{15}$$

where (a) follows from the fact that $\prod_{i=1}^{m} \mathbb{P}(x^{(i)})$ is irrelevant to w; (b) comes from the fact that $\log(\cdot)$ is a non-decreasing function and $\prod_{i=1}^{m} (\hat{y}^{(i)})^{y^{(i)}} (1-\hat{y}^{(i)})^{1-y^{(i)}}$ is positive; and (c) is due to changing a sign of the objective while replacing max with min.

Notice that the term inside the summation in the last equality in (15) is the cross entropy loss: $\ell_{\mathsf{entropy}}(y^{(i)}, \hat{y}^{(i)})$. This implies that the optimization problem for logistic regression yields the optimal w.

Remarks on cross-entropy loss (2)

Before moving onto the next topic about how to solve the optimization problem (3), let me say a few words about why the loss function (2) is called the *cross-entropy* loss. Actually this comes from the definition of *cross entropy*, which is a measure used in the field of *information theory*. The cross entropy is defined w.r.t. two random variables. For simplicity, let us consider two binary random variables, say $X \sim \mathsf{Bern}(p)$ and $Y \sim \mathsf{Bern}(q)$ where $X \sim \mathsf{Bern}(p)$ indicates a binary random variable with $p = \Pr(X = 1)$. For such two random variables, the cross entropy is defined as:

$$H(p, q) := -p \log q - (1-p) \log(1-q). \tag{16}$$

Notice that the formula of (2) is exactly the same as the term inside summation in (15), except for having different notations. Hence, it is called the *cross entropy loss*.

Then, you may now wonder why $H(p,q)$ in (16) is called *cross entropy*. Of course, there is a rationale. The rationale comes from the following fact:

$$H(p,q) \geq H(p) := -p\log p - (1-p)\log(1-p) \tag{17}$$

where $H(p)$ is a very-well known quantity in information theory, named *entropy (or Shannon entropy)*. One can actually prove the inequality in (17) using Jensen's inequality that you saw in PS1. Also one can verify that the equality holds when $p = q$. We will not prove this here. But don't worry. You will have a chance to check this in PS. So from this, one can interpret $H(p,q)$ as an *entropic*-measure of discrepancy a*cross* distributions. Hence, it is called *cross entropy*.

How to solve logistic regression?

Now let us get back to the main stream of this lecture. How to solve the optimization problem (3) for logistic regression? Let $J(w)$ be the objective function normalized by the number m of samples:

$$J(w) := \frac{1}{m}\sum_{i=1}^{m} -y^{(i)}\log\hat{y}^{(i)} - (1-y^{(i)})\log(1-\hat{y}^{(i)}). \tag{18}$$

Remember in the beginning of this lecture that we proved the convexity of the objective function. Also the optimization problem (3) is unconstrained. Hence, we can use the gradient decent algorithm to find the optimal solution. Specifically one can take the following update rule:

$$w^{(t+1)} \leftarrow w^{(t)} - \alpha^{(t)}\nabla J(w^{(t)})$$

where $w^{(t)}$ denotes the estimate of weights at the tth iteration, and $\alpha^{(t)}$ indicates a stepsize (or called a learning rate in the ML field). The gradient of the normalized objective function can be computed as:

$$\nabla J(w) = \frac{1}{m}\sum_{i=1}^{m} -y^{(i)}\frac{\nabla\hat{y}^{(i)}}{\hat{y}^{(i)}} + (1-y^{(i)})\frac{\nabla\hat{y}^{(i)}}{1-\hat{y}^{(i)}}. \tag{19}$$

Here the gradient of $y^{(i)}$ (marked in red) can be expressed as:

$$\nabla\hat{y}^{(i)} = \frac{x^{(i)}e^{-w^Tx^{(i)}}}{(1+e^{-w^Tx^{(i)}})^2} = x^{(i)}\hat{y}^{(i)}(1-\hat{y}^{(i)})$$

Plugging this to (19), we get:

$$\begin{aligned}\nabla J(w) &= \frac{1}{m}\sum_{i=1}^{m} x^{(i)}\left\{-y^{(i)}(1-\hat{y}^{(i)}) + (1-y^{(i)})\hat{y}^{(i)}\right\} \\ &= \frac{1}{m}\sum_{i=1}^{m} x^{(i)}\left\{\hat{y}^{(i)} - y^{(i)}\right\}.\end{aligned}$$

Notice that $\nabla J(w)$ is simple to calculate.

AI boomed in the 1960s but ...

As we verified above, the logistic regression is the best binary classifier in a sense of maximizing the likelihood of training data, assuming that the overall architecture is based on the perceptron. But as you can see, the perceptron architecture is somewhat restricted. So you may guess that the performance of the logistic regression based on the restricted architecture may not be that good in many applications. It turns out this is the case. Actually it was already verified in 1969 by two pioneers in the AI field, named Marvin Minsky and Seymour Papert; see Fig. 3.

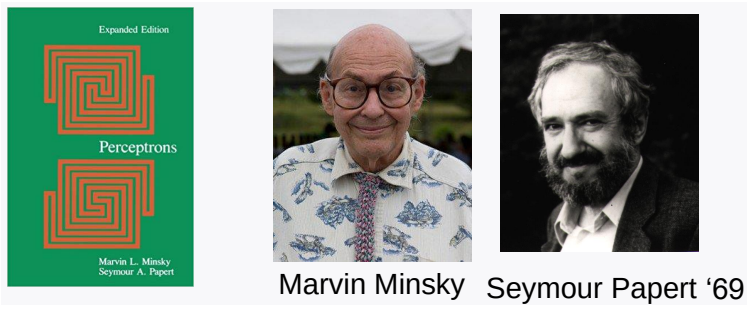


Figure 3: Two AI pioneers, Minsky & Papert, demonstrated limitations of the architecture of Perceptron in a book titled "Perceptrons". Unfortunately, this led to a very long depression period in the AI field, called the *AI winter*.

They published a book, titled "Perceptrons", where they demonstrated that the Perceptron architecture cannot implement even very simple functions, like XOR. Their results made many people at that time disappointed. This finally led to a very long depression period of the AI field, called the AI winter.

AI revived in 2012

The AI winter had continued until recently. However, a big event happened in 2012, enabling the AI field to revive. The big event was the winning of ImageNet recognition competition by the following three people: Geoffrey Hinton (a very well-known figure in the AI field, known as the Godfather of deep learning) and his two PhD students (Alex Krizhevsky and Ilya Sutskever); see Fig. 4.

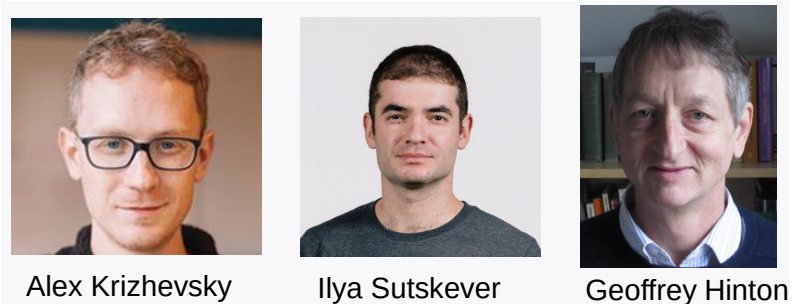


Figure 4: A giant in the AI field, Geoffrey Hinton, together with his PhD students, Alex Krizhevsky and Ilya Sutskever, developed a deep neural network, named AlexNet, intended for image classification. The AlexNet achieved almost *human-level recognition performances*, which were never attained earlier. This won them the ImageNet competition in 2012. More importantly, this recognition anchored the start of *deep learning revolution*!

They built a perceptron-like neural network but which consists of many layers, called a deep

neural network (DNN). They then showed that their DNN, which they named AlexNet, could achieve almost *human-level recognition performances*, which were astonishing at the time. This actually enabled *deep learning revolution* in the AI field.

During upcoming lectures

In the next lectures, we will investigate deep neural networks (DNNs) and the role of optimization in that context. Specifically we will cover the following four stuffs. Firs of all, we will study the architecture of DNNs. We will then formulate a corresponding optimization problem. Next, we will explore an efficient way of solving the problem. Lastly we will briefly discuss why DNNs offer great performances.

EE523 Convex Optimization
KAIST, Spring 2019

May 14, 2019

Lecture 20: Deep learning

Recap

During the past two lectures, we have studied one of application topics of optimization: supervised learning. The goal of supervised learning is to learn a functional of a machine of interest using training samples. We considered a specific yet brain-inspired architecture for the function structure: *perceptron*; see Fig. 1. Taking a linear activation function together with a squared-error loss, we obtained the least-squares classifier. Taking a logistic function together with a cross-entropy loss, we obtained logistic regression. We then proved that logistic regression is *optimal* in a sense of maximizing the likelihood of training samples.

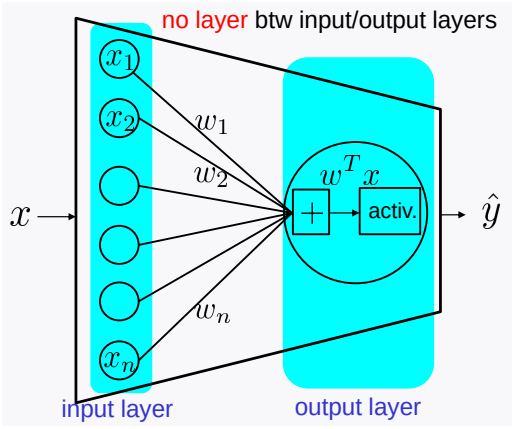


Figure 1: An architecture of Perceptron.

At the end of the last lecture, I mentioned briefly about how the AI field has evolved with research on *neural networks*. While one of the first neural networks, perceptron, enabled the AI revolution in the 1960s, the boom ended shortly after publication of a book by Minsky & Papert, titled "Perceptrons". The book criticized limitations of the perceptron architecture, and this freezed the passion of many people working on the AI field, thus leading to the AI winter.

Fortunately, the AI field boomed again in 2012 by Hinton & his group members. They developed a neural network (which they called AlexNet) that demonstrated human-level performances of image recognition, thus making people excited about the field again. AlexNet is based on a *deep neural network* architecture (DNN for short).

Today's lecture

Today we are going to start investigating deep neural networks (DNNs). Specifically we will cover the following four stuffs. First, we will study what DNNs mean and the network architecture. We will then investigate how DNNs were proposed, i.e., who the inventor was, as well as, what the motivation was. Next, we will discuss why DNNs were recently appreciated, in other words, why they were not appreciated during the AI winter. Finally we will formulate a corresponding optimization problem to start talking about connection to the interest (optimization) of this

course.

Terminologies

Let us start by recalling the perceptron architecture, illustrated in Fig. 1. Before defining the deep neural network (DNN), we need to introduce a couple of terminologies. The first is the *input layer*. We say that a collection of neurons which take input x is the input layer. Similarly, the *output layer* is defined as a collection of the output neuron(s) y. A *shallow* neural network is defined as a network which consists of only input and output layers, i.e., there is no intermediate layer between the two, like the perceptron in Fig. 1.

Definition of deep neural networks (DNNs)

On the other hand, we say that a neural network is *deep* if it has at least one intermediate layer between input and output layers. Such in-between-placed layer is called a *hidden layer*. So a *deep neural network* is defined as a network which contains hidden layer(s).

DNN architecture

Here are details on how such DNN looks like. For illustrative purpose, let us explain the architecture with a simple setting in which there is only one hidden layer, named a 2-layer neural network in the field[1]; also see Fig. 2.

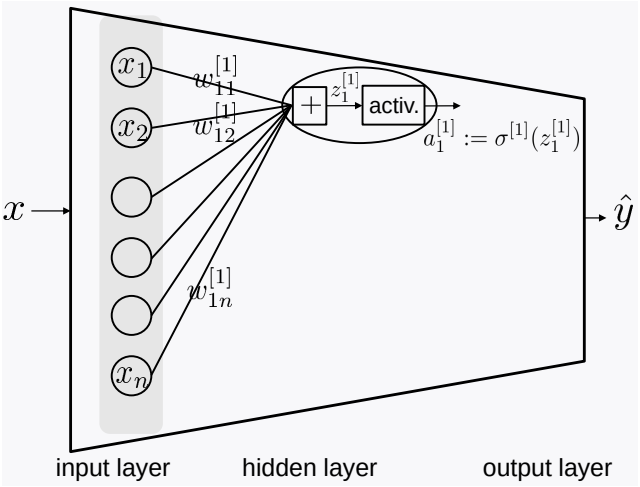


Figure 2: Operation at a neuron in the hidden layer.

Let us consider an operation at the first neuron in the hidden layer. The operation is exactly the same as the operation that we saw earlier at the output layer. First it takes a linear combination to yield:

$$z_1^{[1]} := w_{11}^{[1]} x_1 + w_{12}^{[1]} x_2 + \cdots + w_{1n}^{[1]} x_n \tag{1}$$

where $w_{1j}^{[1]}$ indicates a weight associated with x_j and the 1st neuron in the (1st) hidden layer.

[1]Someone may argue that this is a *3-layer* neural network as it has input/hidden/output layers. But the convention in the deep learning field, adopted by many of the pioneers in the field, is that the number of layers is counted as the total number of layers minus 1. Perhaps the reason is that they wanted to say that the shallow neural network is simply a single-layer neural network, although it actually has two input/output layers. While I believe that this way of defining a network is confusing, we will adopt this convention as it has already been so widely used.

Here the upper-script $(\cdot)^{[1]}$ denotes the indexing of hidden layers. In general, a bias term, say $b_1^{[1]}$, can be added into (1). But for illustrative simplicity, we will drop all the bias terms throughout. Next the linear combination is passed onto an activation function, so we get:

$$a_1^{[1]} := \sigma^{[1]}(z_1^{[1]}) \tag{2}$$

where $\sigma^{[1]}(\cdot)$ indicates an activation function employed in the 1st hidden layer. Usually we are allowed to use different activation functions across different layers, while the same activation function applies within the same layer by convention. Applying the same operation to the other neurons in the hidden layer, we obtain a picture like the one in Fig. 3.

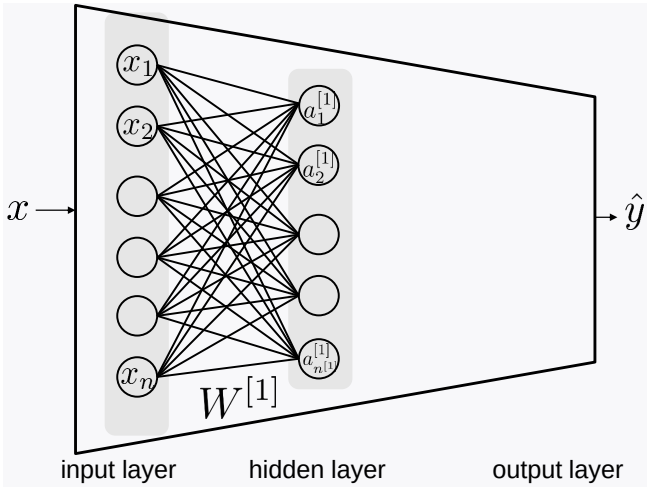


Figure 3: Architecture of the hidden layer.

For notational simplicity, we introduce a matrix, say $W^{[1]}$, which aggregates all the weight components associated with the input layer and the hidden layer:

$$W^{[1]} := \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} & \cdots & w_{1n}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} & \cdots & w_{2n}^{[1]} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n^{[1]}1}^{[1]} & w_{n^{[1]}2}^{[1]} & \cdots & w_{n^{[1]}n}^{[1]} \end{bmatrix} \in \mathbf{R}^{n^{[1]} \times n}, \tag{3}$$

where $n^{[1]}$ denotes the number of neurons in the (1st) hidden layer. Using this matrix notation, we can then represent the output of the hidden layer as:

$$a^{[1]} = \sigma^{[1]}\left(W^{[1]}x\right) \in \mathbf{R}^{n^{[1]}}, \tag{4}$$

where $\sigma^{[1]}(\cdot)$ indicates a *component-wise* function.

Applying the same operation into the one between the hidden and output layers, we obtain a picture like the one in Fig. 4. Using another matrix notation, say $W^{[2]} \in \mathbf{R}^{1 \times n^{[1]}}$, we can then represent the output in the output layer as:

$$\hat{y} = \sigma^{[2]}\left(W^{[2]}a^{[1]}\right) \in \mathbf{R}, \tag{5}$$

where $\sigma^{[2]}$ indicates an activation function at the output layer, which can possibly be different from $\sigma^{[1]}(\cdot)$, as mentioned earlier.

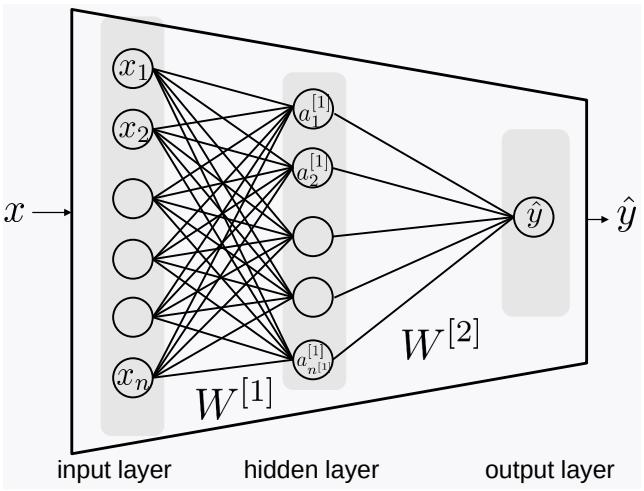


Figure 4: Architecture of an one-hidden-layer DNN, or called a 2-layer DNN.

DNN architecture in general

In general, L-hidden-layer DNN (or called $(L+1)$-layer DNN) can be expressed as:

$$\begin{aligned} a^{[1]} &= \sigma^{[1]}\left(W^{[1]}x\right) \in \mathbf{R}^{n^{[1]}}, \\ a^{[2]} &= \sigma^{[2]}\left(W^{[2]}a^{[1]}\right) \in \mathbf{R}^{n^{[2]}}, \\ &\vdots \\ a^{[L]} &= \sigma^{[L]}\left(W^{[L]}a^{[L-1]}\right) \in \mathbf{R}^{n^{[L]}}, \\ \hat{y} &= \sigma^{[L+1]}\left(W^{[L+1]}a^{[L]}\right) \in \mathbf{R}, \end{aligned} \tag{6}$$

where $a^{[i]} \in \mathbf{R}^{n^{[i]}}$ indicates the output of the ith hidden layer; $\sigma^{[i]}(\cdot)$ denotes the component-wise activation function at the ith hidden layer; and $W^{[i]} \in \mathbf{R}^{n^{[i]} \times n^{[i-1]}}$ denotes the weight matrix associated with the ith hidden layer and $(i-1)$th hidden layer. Here for notational consistency, one can define the 0th hidden layer as the input layer; $(L+1)$th hidden layer as the output layer, and hence, $n^{[0]} := n$ and $n^{[L+1]} := 1$.

How DNNs are proposed

Now let me explain how such DNN expressed in (6) was proposed. The first DNN was proposed in 1965 by an Ukrainian mathematician, named Alexey Ivakhnenko; see the first left picture in Fig. 6. He noticed that the perceptron architecture is too simple to represent a somewhat complex system. So he believed that a proper architecture should incorporate much more neurons as well as capture much higher connectivities across neurons. Obviously the most complex structure is the one in which each neuron is connected with all of the other neurons. But it was not that clear to him as to whether such complex structure is indeed the case in biological networks for brains of intelligent beings.

He was trying to gain some insights into this from another field: *evolution* in biology. In particular, he was inspired by *genetic natural selection in evolution*. What he was inspired is that a *complex species* is a consequence of evolution through many *generations* by *natural*

genetic selection. He then made an analogy between such evolution process and the process of a complex system (machine) of interest. Specifically he came up with operations/entities in a complex system which correspond to *species, generation and selection* that appear in the evolution process. See Fig. 5.

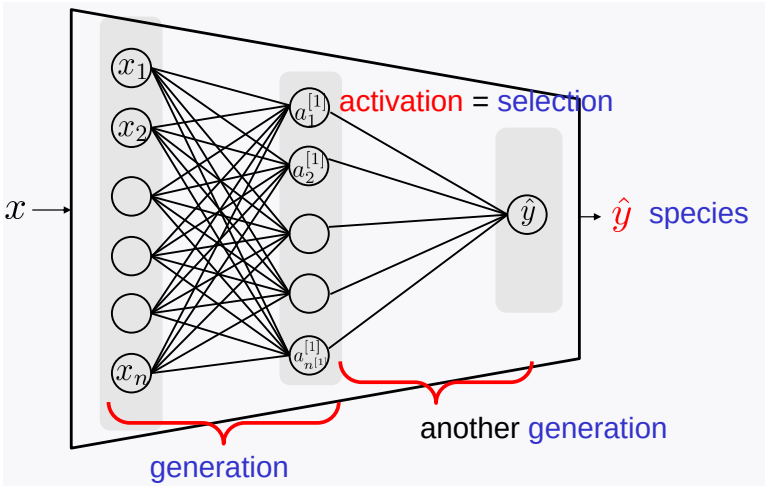


Figure 5: Analogy between the evolution process (by genetic natural selection in biology) and the process of a complex system (machine).

First of all, the *species* was mapped to the output $\hat{y}$ in an interested system. The *generation* was interpreted as the process that occurs in between two consecutive layers in the system. So the process with two generations yields a 2-layer neural network. Lastly the *selection* was captured by the *activation* process in the system. These analogies naturally led to the DNN architecture illustrated in Fig. 5.

Some pioneering efforts on DNNs

Initially, the DNN architecture in Fig. 5 was investigated in depth by only a few people in the field. One of the reasons was that there was no theoretical basis which supports that such architecture can represent any arbitrary complex functional of a system. The architecture was based solely on the hypothesis. There was no proof. Even worse, it was not that simple to do *experimental verification* because the technology of the day was so immature that the time required to train a DNN was very long from days to weeks. Nonetheless, there were some people who studied this architecture in depth. Here we list three of them below.

The first is obviously the inventor of the DNN architecture: Alexey Ivakhnenko. One of his great achievements in this field was to propose a 7-hidden-layer DNN in 1971. The second pioneer is a Japanese computer scientist, named Kunihiko Fukushima; see the middle picture in Fig. 6. He developed a specially-structured DNN intended for *pattern recognition* in computer vision in 1980. That was actually the first *convolutional neural network* (CNN), which is now known as the most famous and widely-used DNN in the computer vision field. The third pioneer is a French computer scientist, which is now very famous in the deep learning field and also a winner of the 2018 Turing Award (considered as the Nobel Prize in computer science). In 1989, he trained a CNN for the purpose of recognizing handwritten ZIP codes on mails. This development played a role to vitalize the deep learning field because the trained CNN actually worked very well and so was commercialized. Nonetheless, it was not enough to enable the deep learning revolution. One of the reasons was that the training time required *3 days* with the

Figure 6: Three pioneers in deep learning.

technology of the day.

Not appreciated much in early days

There were more critical reasons as to why the DNN architecture was not appreciated much at that time. These are two-folded. The first reason is concerning *performances*. The DNN-based algorithms at the time were easily outperformed by much simpler approaches. One of the simpler approaches was Support Vector Machines (SVMs), which is sort of a variant of the margin-based LP classifier that we learned in Part I. Remember that the margin-based LP classifier is very simple and runs very fast. On the other hand, the DNN architecture was relatively much more complex, yet even worse, the performance was not better. The second reason is about model complexity. The DNN model was so complex considering the technology of the day, so it required very long training time, from days to weeks.

Why appreciated nowadays?

But as many of you know, the DNN is greatly appreciated nowadays. Why is that? As I mentioned earlier, this is mainly due to the recent big event by Hinton[2] and his PhD students who demonstrated that DNNs can achieve human-level recognition performances. Then, how that happened? There are two technology breakthroughs that enabled the deep learning revolution; see Fig. 7.

The first breakthrough is the *advent of big data!* Nowadays we are living in the big data era. There are tons of data that are floating in the cyber-world. So it is possible to gather lots of training data. One such huge dataset gathered for the purpose of image cognition was ImageNet. The dataset was created in 2009 by a computer-vision team at Stanford, led by a young professor, Fei-Fei Li. It turned out this dataset played a crucial role for Hinton's team to demonstrate the power of DNN by offering a sufficiently large number of training samples enough to learn a complex model.

The second breakthrough is the supply of very fast and not-so-expensive Graphic Processing Units (GPUs). The major company that provided such GPUs is NVIDIA. It turned out GPUs offered great computational power to reduce training time of DNNs significantly.

An optimization problem

[2]By the way, he is also a co-winner of the 2018 Turing Award with Yann LeCun and another giant in the field, named Yoshua Bengio.

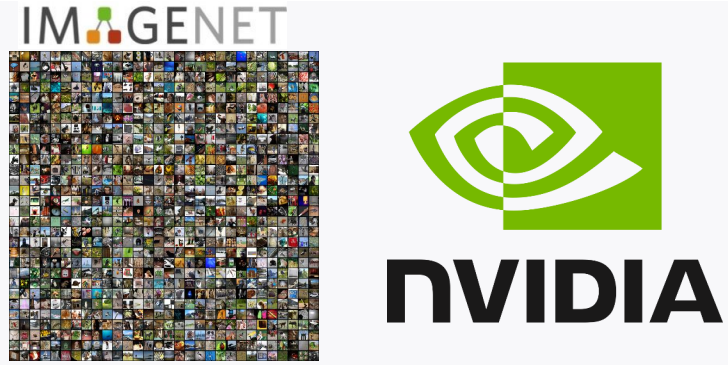


Figure 7: Two technology breakthroughs that enabled the deep learning revolution.

Now let us connect the DNN architecture to the optimization of this course's interest. For illustrative purpose, let us formulate an optimization problem for the simple two-layer DNN in Fig. 4. The optimization problem based on the DNN reads:

$$\min_{w} \sum_{i=1}^{m} \frac{1}{m} \ell(y^{(i)}, \hat{y}^{(i)}) \tag{7}$$

where:

$$\begin{aligned} \hat{y}^{(i)} &= \sigma^{[2]} \left(W^{[2]} a^{[1],(i)} \right), \\ a^{[1],(i)} &= \sigma^{[1]} \left(W^{[1]} x^{(i)} \right), \\ w &= (W^{[1]}, W^{[2]}). \end{aligned}$$

Optimal loss function

Suppose we use the logistic function for

$$\sigma^{[2]}(z) = \sigma(z) := \frac{1}{1+e^{-z}}.$$

Then, using exactly the same argument that we made in the previous lecture (Lecture 19), one can show that the optimal loss is *cross-entropy loss*:

$$\ell^*(y, \hat{y}) = \ell_{\text{entropy}}(y, \hat{y}) = -y \log y - (1-y) \log(1-\hat{y}). \tag{8}$$

Is it convex?

Pugging the cross-entropy loss into (7), we can then get:

$$\arg\min_{w} \frac{1}{m} \sum_{i=1}^{m} -y^{(i)} \log \hat{y}^{(i)} - (1-y^{(i)}) \log(1-\hat{y}^{(i)}) \tag{9}$$

where $\hat{y}^{(i)} = \sigma\left(W^{[2]} a^{[1],(i)}\right)$. Now a natural question that arises is: how to solve the problem? We are familiar with solving only *convex* optimization problems. So the question of our interest

is: Is the objective function *convex*? Obviously it depends on how to choose an activation function in the hidden layer: $\sigma^{[1]}(\cdot)$.

Look ahead

It turns out there is a very well-known and powerful activation function for $\sigma^{[1]}(\cdot)$. Unfortunately, under the choice of such function, the optimization problem (9) was shown to be *non-convex*. But there is a good news. That is, we have a way to handle such non-convex problem. Next time, we will study details on such way.

Lecture 21: Deep learning II

Recap

Last time we studied about the architecture of DNNs which have been shown to be quite powerful in recent years. We also discussed a brief history of DNNs; who the inventor was; what the motivation was; why they were not appreciated until recently; what led to the deep learning revolution. We then formulated an optimization problem for DNNs to start talking about connection to optimization topics of this course's interest. Here is the optimization problem intended for a 2-layer DNN:

$$\arg \min_{w=(W^{[1]},W^{[2]})} \frac{1}{m} \sum_{i=1}^{m} -y^{(i)} \log \hat{y}^{(i)} - (1-y^{(i)}) \log(1-\hat{y}^{(i)}) \tag{1}$$

where

$$\hat{y}^{(i)} = \sigma\left(W^{[2]} a^{[1],(i)}\right); \quad a^{[1],(i)} = \sigma^{[1]}\left(W^{[1]} x^{(i)}\right).$$

Here $\sigma(\cdot)$ indicates a component-wise logistic function defined as:

$$\sigma(z) := \frac{1}{1+e^{-z}}. \tag{2}$$

On the other hand, $\sigma^{[1]}(\cdot)$ denotes a possibly-different activation function used at the hidden layer.

At the end of the last lecture, we claimed that for a widely-used $\sigma^{[1]}(\cdot)$, the objective function in (1) is *non-convex*, which is intractable in general. We also claimed that fortunately, there is a good way to address such non-convex optimization problem.

Today's lecture

Today we are going to support these claims. Specifically what we are going to do are four folded. First of all, we will study what the widely-used activation function is nowadays. We will then check that the objective function is not convex. Next we will investigate what the good way is. Finally we will discuss how to solve the optimization problem in detail.

Widely-used activation function

Consider an operation that occurs at one neuron in the hidden layer; see Fig. 1. As mentioned earlier, the DNN architecture takes the *perception architecture* as the basic operation unit. So the basic operation consists of two procedures. First we do a linear operation by aggregating weighted signals coming from neurons in the preceding layer, thus yielding an output, say z. The output z is then passed onto an activation function, so we get an output, say a.

The activation function which has been widely used during recent years is a function named the *Rectified Linear Unit*, simply called ReLU. The functional is very simple; it simply bypasses the input if it is non-negative; yields 0 otherwise:

$$a = \begin{cases} z, & \text{if } z \geq 0; \\ 0, & \text{if } z < 0. \end{cases} \tag{3}$$

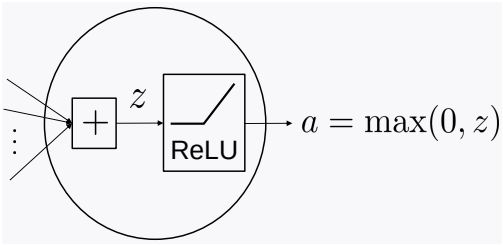


Figure 1: Rectified Linear Unit (ReLU).

So it can be represented as:

$$a = \max(0, z). \tag{4}$$

A brief history of ReLU

Actually the ReLU function was introduced in very early days in a variety of fields, not limited to the deep learning field. It appeared even in Fukushima's 1980 paper on convolutional neural networks (CNNs).

But the function was not frequently used in DNNs until recently. Instead more interpretable activation functions like the logistic function were widely used. Another popular activation function was a shifted version of the logistic function, called the tanh function:

$$\begin{aligned} \tanh(z) &:= \frac{e^z - e^{-z}}{e^z + e^{-z}} \\ &= \frac{1}{1 + e^{-2z}} - \frac{e^{-2z}}{1 + e^{-2z}} \\ &= \sigma(2z) - (1 - \sigma(2z)) \\ &= 2\sigma(2z) - 1. \end{aligned} \tag{5}$$

Note that $\tanh(z) = 2\sigma(2z) - 1$, so the range of the function is shifted from $(0, 1)$ to $(-1, 1)$.

A common rule of thumb that had been applied in the deep learning field until recently was to use the logistic function only at the output layer while taking the tanh function at all of the other neurons placed in hidden layers. Many empirical results have demonstrated that such rule of thumb always yields a better or equal performance, as compared to the other alternative which takes the logistic function at all places. There is no theoretical justification on this. But it looks more or less making-sense. The reason is that taking the tanh function at hidden neurons broadens the output range, thus yielding a more degree of freedom relative to the one by the logistic function.

ReLU became prevalent since 2011

A recent big wave arose in the domain of activation functions. In 2011, one of the deep learning heroes, named Yoshua Bengio (see the left picture in Fig. 2), together with his group members, Xavier Glorot (PhD student) and Antoine Bordes (postdoc), demonstrated via extensive simulation results that ReLU enables *faster and more effective training* of DNNs, compared to the logistic and/or tanh functions. This also was empirically confirmed by numerous practitioners on many datasets. Hence, ReLU now acts as a default activation function in hidden layers.

They also provided some intuitions as to why that is the case. One intuition is that ReLU better mimicks how brains of intelligent beings work. A report by neuroscientists says that *only a few*

Figure 2: Yoshua Bengio (left) is one of the giants in the deep learning field, and also a co-recipient of the 2018 Turing Award. One of his main achievements is to demonstrate the power of ReLU activation function, thus popularizing the use of the function in the deep learning society. Seppo Linnainmaa (right) is the inventor of *backpropagation* which serves as an efficient way of computing gradients in DNNs.

percentages of the neurons in human brains are *activated* even during active brain activities. This is somewhat consistent with a consequence of taking ReLU, as that way leads many of the neurons to be simply set to 0 when their values take negative.

Another explanation is sort of technical one, being tailored for a particular yet popular learning algorithm. One of the popular training algorithms employed in the field is based on computation of *gradients* of the objective function w.r.t. weights (optimization variables). Dynamics of gradients for the logistic or tanh functions are somewhat limited. They are close to 0 for large or small z. Why? Think about the shape of the function. It takes some meaningful gradient only when z is in a small range. On the other hand, the gradient of ReLU does not vanish even when z is very large. Notice that the gradient of ReLU reads:

$$\frac{d\mathsf{ReLU}(z)}{dz} = \begin{cases} 1, & \text{if } z > 0; \\ 0, & \text{if } z < 0; \\ \mathsf{undefined}, & \text{if } z = 0. \end{cases} \tag{6}$$

Note that the gradient takes 1 even when z is very large. So they believed that this *non-vanishing gradient effect* yields a better training, which I also agreed with more or less.

Remark on ReLU

As you may see from (6), there is an issue in computing the gradient of ReLU. The issue is that the function is *not differentiable*. The gradient is undefined at $z = 0$. But it turns out this is not a big deal in reality. In reality, the event $z = 0$ rarely happens. Actually the exactly-zero-event never happened - it has a *measure-zero-event*. So there is no problem to use in practice, although it is indeed *mathematically problematic*.

Convex vs. non-convex?

Let us get back to the optimization problem now when taking the ReLU activation function:

$$\arg \min_{w=(W^{[1]},W^{[2]})} \frac{1}{m} \sum_{i=1}^{m} -y^{(i)} \log \hat{y}^{(i)} - (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \tag{7}$$

where

$$\hat{y}^{(i)} = \sigma\left(W^{[2]} \max(0, W^{[1]} x^{(i)})\right).$$

The question of this course's interest is: Is the objective function *convex*? As claimed earlier, the answer is no. The objective function is *non-convex* in general. Actually the proof of this is a bit involved, as it may include a very complicated Hessian calculation. So this may distract us from the main stream of the contents of this lecture. So we omit the proof here. But don't worry. Of course you will have a chance to prove this in PS.

A way to handle such non-convex problem

As mentioned earlier, there is a good way to handle such non-convex optimization problem. The way is inspired by the observation made by numerous practitioners working on the field. Many *experimental* results by them revealed that in most cases:

$$\textit{Any local minimum is the global minimum.} \tag{8}$$

What this means is that in many of the practical settings, there is *no spurious local minimum*. As you may conjecture, this is not a mathematically correct statement. Actually this was proven to be *mathematically wrong*, meaning that there are counter-examples in which there are *spurious local minima*. But it was also empirically shown that those counter-examples rarely happen in many of the working DNNs. In fact, we are still very much lacking in our understanding on this. In other words, currently we have no idea what is the necessary/sufficient condition for (8) to hold.

Nonetheless, in many realistic scenarios, (8) was observed. So many people believe that in most interested cases, that is the case, e.g., the landscape of the objective function for a DNN-based optimization problem looks like the one in Fig. 3.

Figure 3: Landscape of the objective function in optimization for DNNs.

Note in Fig. 3 that there is no spurious local minimum. We have only the global minimum or saddle points[1]. This observation made through many experimental results suggested a good guideline in practice. That is, simply to find any minimum and then take it as a solution, no matter what the type of an optimization problem is. The question of interest is then: How to find a minimum? One very popular way is to apply the *gradient decent algorithm* which we are very much familiar with! Of course, the gradient decent algorithm may lead us to stuck at some saddle point which we do not want to arrive at. But the good news in practice is that it

[1]Remember the definition of a *saddle* point. A saddle point is defined as a point which is both local minimum and local maximum.

is extremely rare to stuck at a saddle point when there are minima. Actually it is even difficult to arrive at a saddle point even if we wish to do so. Hence, a general rule of thumb is to simply apply the gradient decent algorithm no matter what and whatsoever.

Gradient decent algorithm

What is the gradient decent algorithm in the interested optimization problem below?

$$\arg \min_{w=(W^{[1]},W^{[2]})} \underbrace{\frac{1}{m}\sum_{i=1}^{m} -y^{(i)} \log \hat{y}^{(i)} - (1-y^{(i)}) \log(1-\hat{y}^{(i)})}_{=:J(w)} \tag{9}$$

where $\hat{y}^{(i)} = \sigma\left(W^{[2]} \max(0, W^{[1]} x^{(i)})\right)$.

The algorithm is to iterate the following procedure: for the tth iteration, the new $(t+1)$th estimate for weights takes:

$$w^{(t+1)} \leftarrow w^{(t)} - \alpha^{(t)} \nabla_w J(w^{(t)})$$

where $\alpha^{(t)}$ indicates a stepsize (a learning rate). Since w is a collection of $(W^{[1]}, W^{[2]})$, the detailed procedure is:

$$W^{[2],(t+1)} \leftarrow W^{[2],(t)} - \alpha^{(t)} \nabla_{W^{[2]}} J(w^{(t)});$$
$$W^{[1],(t+1)} \leftarrow W^{[1],(t)} - \alpha^{(t)} \nabla_{W^{[1]}} J(w^{(t)}).$$

As you can see here, there are multiple weight update procedures - in this case, two procedures. Actually these multiple procedures yielded a critical concern in early days of the DNN research. The reason is that it requires computationally heavy calculations. Especially when a DNN has many layers, it raises definitely a critical computational concern. So some people tried to address this problem in early days to come up with an efficient way of computing such many gradients, called:

Backpropagation.

Backpropagation

The same *backpropagation* algorithm was *independently* developed by a bunch of research groups including Hinton's development in 1986 together with his colleagues, David Rumelhart & Ronald Williams. But the first invention was much earlier. It was around when the book of *Perceptrons* was published - that was 1970! In 1970, a Finnish mathematician (as well as a computer scientist), named Seppo Linnainmaa (see the right picture in Fig. 2), invented the algorithm.

The idea of the algorithm is very natural although it involves some complicatedly-looking math equations. Perhaps this may be one of the reasons that there were several independent yet same inventions. The idea is to:

> Successively compute gradients in a *backward* manner by using a chain rule for derivatives.

For illustrative purpose, let us first explain how the algorithm works for a simple single-example setting ($m = 1$). We will then extend it to the general case.

Backpropagation in action: $m = 1$

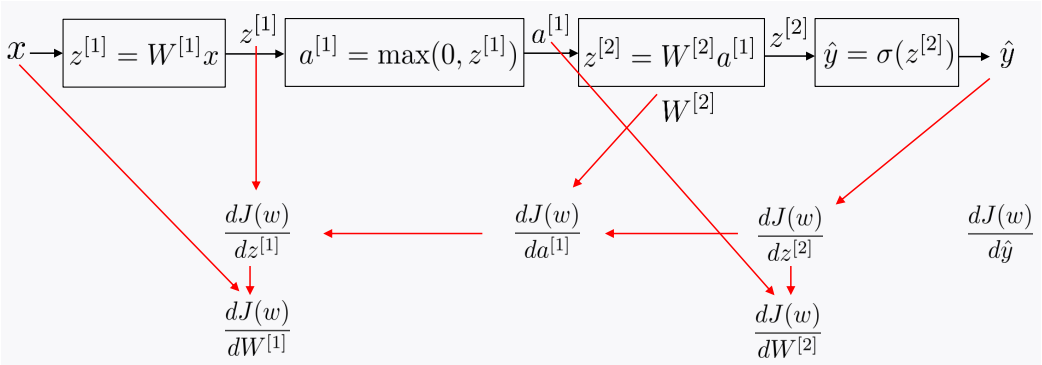


Figure 4: Illustration of the *backpropagation algorithm*: $m = 1$.

The illustration of the algorithm can be streamlined with the help of some picture which visualizes paths of signals. One such path is the *forward* path; see the top row in Fig. 4. The input signal x passes through the hidden layer to yield $z^{[1]}$ and then $a^{[1]}$. Similarly we get $z^{[2]}$ and then $\hat{y} := a^{[2]}$.

The *backpropagation* algorithm starts from *backward*. Consider the gradient of the objective function $J(w)$ w.r.t. the *last* output signal $\hat{y}$: $\frac{dJ(w)}{d\hat{y}}$. To ease algorithm illustration, we will use the notation of $\frac{d}{d\hat{y}}$ instead of $\nabla_{\hat{y}}$. The reason is that the algorithm is based on the *chain rule* for derivatives, so the notation $\frac{d}{d\hat{y}}$ helps us to better understand how the algorithm works, relative to $\nabla_{\hat{y}}$. Since $J(w)$ is simply the cross-entropy loss for $m = 1$, the gradient reads:

$$\frac{dJ(w)}{d\hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}. \tag{10}$$

Notice that one can easily compute this, since y is given in the problem and $\hat{y}$ is available once we compute the forward path.

Next, we consider the gradient of $J(w)$ now w.r.t. the *second-last* signal $z^{[2]}$: $\frac{dJ(w)}{dz^{[2]}}$. This is where the idea of the *chain rule* kicks in. Using the chain rule, we get:

$$\begin{aligned} \frac{dJ(w)}{dz^{[2]}} &= \frac{dJ(w)}{d\hat{y}} \frac{d\hat{y}}{dz^{[2]}} \\ &\overset{(a)}{=} \left(-\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}\right) \hat{y}(1-\hat{y}) \\ &= \hat{y} - y. \end{aligned} \tag{11}$$

where (a) follows from (10) (already computed earlier) as well as the fact that the gradient of the logistic function can simply be expressed as:

$$\begin{aligned} \frac{d\sigma(z)}{dz} &= \frac{d}{dz}\left(\frac{1}{1+e^{-z}}\right) = \frac{e^{-z}}{(1+e^{-z})^2} \\ &= \frac{1}{1+e^{-z}} \cdot \frac{e^{-z}}{1+e^{-z}} \\ &= \sigma(z)(1-\sigma(z)). \end{aligned} \tag{12}$$

From (11), we can now compute one of the interested gradients: $\frac{dJ(w)}{dW^{[2]}}$. Again using the chain

rule, we get:

$$\begin{aligned}\frac{dJ(w)}{dW^{[2]}} &= \frac{dJ(w)}{dz^{[2]}}\frac{dz^{[2]}}{dW^{[2]}} \\ &\overset{(a)}{=} \frac{dJ(w)}{dz^{[2]}}a^{[1]T}\end{aligned} \tag{13}$$

where (a) comes from $z^{[2]} = W^{[2]}a^{[1]}$ (why taking a transpose to get $a^{[1]T}$?). Notice that this can be computed from $\frac{dJ(w)}{dz^{[2]}}$ (already computed from (11)) and the knowledge of $a^{[1]}$. To indicate how it can be computed, we draw red-lined flows in Fig. 4.

You may now grab the idea of how the backpropagation algorithm works! We compute gradients w.r.t. from the last output signal ($\hat{y}$) to the inner signals ($z^{[2]}, W^{[2]}$), all the way *back* to ($z^{[1]}, W^{[1]}$). Did you get it? For those who did not get this yet, let me repeat.

We next consider the gradient of $J(w)$ now w.r.t. the *third*-last signal $a^{[1]}$: $\frac{dJ(w)}{da^{[1]}}$. Again using the chain rule, we obtain:

$$\begin{aligned}\frac{dJ(w)}{da^{[1]}} &= \frac{dJ(w)}{dz^{[2]}}\frac{dz^{[2]}}{da^{[1]}} \\ &\overset{(a)}{=} W^{[2]T}\frac{dJ(w)}{dz^{[2]}}\end{aligned} \tag{14}$$

where (a) is due to $z^{[2]} = W^{[2]}a^{[1]}$. Here you may be very confused about how the last equality comes up. Why we take a transpose for $W^{[2]}$? Why we first have $W^{[2]T}$, followed by $\frac{dJ(w)}{dz^{[2]}}$? Why not the other way around? Remember the rule of thumb that I mentioned in past lectures. First of all, you need to check what the dimension of the final result is. In this case, the final result is $\frac{dJ(w)}{da^{[1]}}$. The dimension should be exactly the same as that of $a^{[1]}$, so $\frac{dJ(w)}{da^{[1]}} \in \mathbf{R}^{n^{[1]}}$. Next think about the dimension of $\frac{dJ(w)}{dz^{[2]}}$. It should be $\frac{dJ(w)}{dz^{[2]}} \in \mathbf{R}^{n^{[2]}}$. Why? This suggests that $\frac{dJ(w)}{dz^{[2]}}$ should come *after* $W^{[2]T}$. Otherwise, dimensions do not match - a syntax error occurs! Now why taking a transpose for $W^{[2]}$? Again this is due to dimension matching. With the transpose, we can make sure that the dimension of the end result is $n^{[1]} \times 1$, which is what we want.

Again the key observation in (14) is that $\frac{dJ(w)}{da^{[1]}}$ can be computed from $\frac{dJ(w)}{dz^{[2]}}$ (which we already obtained from (11)) and the knowledge of $W^{[2]}$. See the knowledge path marked with red lines in Fig. 4.

We can next do the same thing for $\frac{dJ(w)}{dz^{[1]}}$ and $\frac{dJ(w)}{dW^{[1]}}$. Using the chain rule, we get:

$$\begin{aligned}\frac{dJ(w)}{dz^{[1]}} &= \frac{dJ(w)}{da^{[1]}}\frac{da^{[1]}}{dz^{[1]}} \\ &\overset{(a)}{=} \frac{dJ(w)}{da^{[1]}} .* \mathbf{1}\{z^{[1]} \geq 0\}\end{aligned} \tag{15}$$

where (a) follows from (6): $\frac{d\mathsf{ReLU}(z)}{dz} = \mathbf{1}\{z \geq 0\}$. Actually this is not quite correct mathematically, since the ReLU function is not differentiable at $z = 0$. But since it is okay to ignore such rare event in practice, we simply assume that the gradient is 1 at $z = 0$. Here the symbol $.*$ indicates the component-wise multiplication (MATLAB notation), not the normal multiplication. You can also easily think that it should be the component-wise multiplication, since otherwise dimensions do not match. Next we get:

$$\begin{aligned}\frac{dJ(w)}{dW^{[1]}} &= \frac{dJ(w)}{dz^{[1]}}\frac{dz^{[1]}}{dW^{[1]}} \\ &\overset{(a)}{=} \frac{dJ(w)}{dz^{[1]}}x^T\end{aligned} \tag{16}$$

where (a) is due to $z^{[1]} = W^{[1]}x$.

Here is a summary of all the important gradients that we derived in a *backward* manner:

$$\frac{dJ(w)}{dz^{[2]}} = \hat{y} - y; \tag{17}$$
$$\frac{dJ(w)}{dW^{[2]}} = \frac{dJ(w)}{dz^{[2]}} a^{[1]T}; \tag{18}$$
$$\frac{dJ(w)}{da^{[1]}} = W^{[2]T} \frac{dJ(w)}{dz^{[2]}}; \tag{19}$$
$$\frac{dJ(w)}{dz^{[1]}} = \frac{dJ(w)}{da^{[1]}} .* \mathbf{1}\{z^{[1]} \geq 0\}; \tag{20}$$
$$\frac{dJ(w)}{dW^{[1]}} = \frac{dJ(w)}{dz^{[1]}} x^T. \tag{21}$$

To run the gradient decent algorithm, what we need to use are (18) and (21). But the other gradients are still important because they serve as *bridges* to compute the interested gradients ((18) and (21)) in the end.

Backpropagation for general m

Now what about for the general m case? The idea is exactly same. The only distinction is that we now need to incorporate all the examples in computing gradients. It turns out *matrix* notations help us to derive such generalized gradients. Let

$$\begin{aligned}
Y &:= \begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(m)} \end{bmatrix} \in \mathbf{R}^{1\times m};\\
\hat{Y} &:= \begin{bmatrix} \hat{y}^{(1)} & \hat{y}^{(2)} & \cdots & \hat{y}^{(m)} \end{bmatrix} \in \mathbf{R}^{1\times m};\\
A^{[1]} &:= \begin{bmatrix} a^{[1],(1)} & a^{[1],(2)} & \cdots & a^{[1],(m)} \end{bmatrix} \in \mathbf{R}^{n^{[1]}\times m};\\
Z^{[1]} &:= \begin{bmatrix} z^{[1],(1)} & z^{[1],(2)} & \cdots & z^{[1],(m)} \end{bmatrix} \in \mathbf{R}^{n^{[1]}\times m};\\
Z^{[2]} &:= \begin{bmatrix} z^{[2],(1)} & z^{[2],(2)} & \cdots & z^{[2],(m)} \end{bmatrix} \in \mathbf{R}^{n^{[2]}\times m};\\
X &:= \begin{bmatrix} x^{(1)} & x^{(2)} & \cdots & x^{(m)} \end{bmatrix} \in \mathbf{R}^{n\times m}.
\end{aligned} \tag{22}$$

Using these matrix notations, one can readily show that the important corresponding gradients for the general m case are:

$$\frac{dJ(w)}{dZ^{[2]}} = \frac{1}{m}(\hat{Y} - Y); \tag{23}$$
$$\frac{dJ(w)}{dW^{[2]}} = \frac{dJ(w)}{dZ^{[2]}} A^{[1]T}; \tag{24}$$
$$\frac{dJ(w)}{dA^{[1]}} = W^{[2]T} \frac{dJ(w)}{dZ^{[2]}}; \tag{25}$$
$$\frac{dJ(w)}{dA^{[1]}} = \frac{dJ(w)}{dA^{[1]}} .* \mathbf{1}\{Z^{[1]} \geq 0\}; \tag{26}$$
$$\frac{dJ(w)}{dW^{[1]}} = \frac{dJ(w)}{dZ^{[1]}} X^T. \tag{27}$$

Note that these are exactly the same as those in the $m = 1$ case except two things. One is that we have now all the capital letters, which is obvious. The second thing is that we have the factor of $\frac{1}{m}$ in the first gradient (23). Why? You will have a chance to think about this while proving the above in PS.

Look ahead

This is the end of the supervised learning part. Actually there may be more contents that may be of your interest. But we stop here due to the interest of time. Obviously we cannot cover all the contents. If you are interested in more on supervised learning, then I strongly recommend you to take an online *deep learning* course on *Coursera*, taught by Prof. Andrew Ng at Stanford.

Next time, we will move onto the next application topic of optimization: *unsupervised learning*. Specifically we will study one of very popular machine-learning frameworks for unsupervised learning, called *Generative Adversarial Networks* (GANs for short). It turns out the duality theorems that we learned in Part II play a crucial role to understand the GANs. We will cover details from the next lecture on.

EE523 Convex Optimization
KAIST, Spring 2019

May 21, 2019

Lecture 22: Unsupervised learning: Generative models

Recap

During the past four lectures, we have studied some basic and trending contents on supervised learning. The goal of supervised learning is to estimate a functional $f(\cdot)$ of an interested computer system (machine) from input-output samples, as illustrated in Fig. 1.

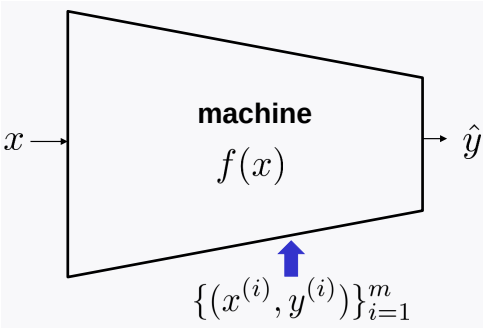


Figure 1: Supervised learning: Learning the functional $f(\cdot)$ of an interested system from data $\{(x^{(i)}, y^{(i)})\}_{i=1}^m$.

In an effort to translate a *function* optimization problem (a natural formulation of supervised learning) into a parameter-based optimization problem that we are familiar with, we expressed the function with parameters (or called weights) assuming a certain architecture of the system.

The certain architecture was: *Perceptron*. Taking the linear activation function together with squared-error loss, we obtained the Least-Squares classifier. Taking the logistic function together with cross-entropy loss, we obtained logistic regression. We then proved that logistic regression is optimal in a sense of maximizing the likelihood of training data.

We next considered the Deep Neural Networks (DNNs) architecture for $f(\cdot)$, which has been shown to be more expressive. Since there is no theoretical basis on the choice of activation functions in the DNN context, we investigated only a rule-of-thumb which is common to use in the field: Taking ReLU at all hidden neurons while taking the logistic function at the output layer. We have a theoretical justification only on the choice of a loss function: cross-entropy loss. We have also learned that in many of the interested settings, optimization problems for DNNs have no spurious local minima, although the problems are highly non-convex. This motivated the use of the gradient decent algorithm for such problems. Lastly we studied an efficient way of computing gradients: *backpropagation*, or simply called *backprop*.

Today's lecture

Now what is next? Actually we face one critical challenge in supervised learning. The challenge is that it is not that easy to collect *labeled* data in many realistic situations. In general, gathering labeled data is very expensive, as it usually requires extensive human-labour-based annotations. So people wish to do something without such labeled data. Then, a natural question that arises is: What can we do *only* with $\{x^{(i)}\}_{i=1}^m$?

This is where the concept of *unsupervised learning* kicks in. Unsupervised learning is an algorithm of learning something about the data $\{x^{(i)}\}_{i=1}^m$ without a particular task in mind. You may then ask: What is *something*? There are a few candidates for such something to learn in the field. Depending on target candidates, there are different unsupervised learning methods.

Today we will start investigating details on these. Specifically we are going to cover the following four stuffs. First of all, we will study what such candidates for something to learn are. We will then investigate what the corresponding unsupervised learning methods are. Next we will focus on arguably the most prominent and fundamental learning method among them: *Generative models*. Finally, we will connect this to optimization of this course's interest, by formulating an optimization problem for generative models.

Candidates for something to learn

There are three candidates for something to learn, from simple to complex. The first candidate, which is perhaps the simplest, is the *basic structure* of data. For instance, when $\{x_i\}_{i=1}^m$ indicates users/customers data, such basic structures could be *membership* of individuals, community type, gender type, or race type. For products-related data, it could be *abnormal (defect)* vs *normal* information. The second candidate is the one that we learned about in Part I, which is *features*: expressive (and/or compressed) components that well describe characteristics of data. The last is a sort of the most complex yet most fundamental information: the *probability distribution* of data, which allows us to create data as we wish.

Three unsupervised learning methods

Depending on which candidate that we focus on, we have three different unsupervised learning methods. The first is *clustering*, which serves to identify the basic structures of data. You may hear of k-means, k-nearest neighbors, community-detection, or anomaly-detection algorithms. All of these belong to this category. The second is *feature learning* (or called *representation learning*), which allows us to extract some well-representative features. You may hear of autoencoder, matrix factorization, principal component analysis, or dictionary learning, all of which can be categorized into this class. The last is *generative models*, which enable us to create arbitrary examples that well mimick real data. This is actually the most famous unsupervised learning method, which has received a particularly significant attention in the field nowadays. So in this course, we are going to focus on this method.

Why generative models prominent?

Before explaining details on generative models, let me say a few words about why generative models are most prominent in the field. We list three reasons below which I believe major.

The first reason is somewhat related to a famous quote by one of my heroes, Richard Feynman; see Fig. 2. Right before he died in 1988, he left an intriguing quote in his blackboard: *"What I cannot create, I do not understand."* What this quote implies in the context of *unsupervised learning* is that creating convincing examples of data is a *necessary condition* for complete understanding. In this regard, a generative model serves a very important role as it enables us to create arbitrary yet plausible examples that mimick real data.

The second reason is related to a recent breakthrough made in the history of the AI field by a very young research scientist, named Ian Goodfellow; see Fig. 3. He was a bachelor and master student at Stanford, working with Andrew Ng. But he moved to the University of Montreal for PhD study to join Joshua Bengio's group. During his PhD, he could develop a powerful generative model, which he named *"Generative Adversarial Networks (GANs)"*. The GANs are

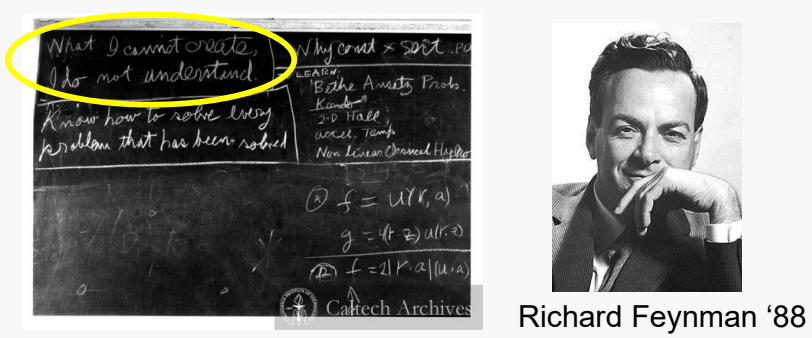


Figure 2: Richard Feynman left a quote on the relationship between *understanding* and *creating* on a blackboard around right before he died in 1988. The quote says, "What I cannot create, I do not understand." What this quote suggests is that being able to create convincing examples of data is a strong evidence of having understood it.

shown to be extremely instrumental in a wide variety of applications, even not limited to the AI field. Such applications include: image creation, human image synthesis, image inpainting, coloring, super-resolution image synthesis, speech synthesis, style transfer, robot navigation, to name a few. Since it works pretty well, as of May 16 2019 (a few days ago), the state of California is even considering a bill that would ban the use of GANs to make fake pornography without the consent of the people depicted. So the GANs have played a crucial role to popularize generative models.

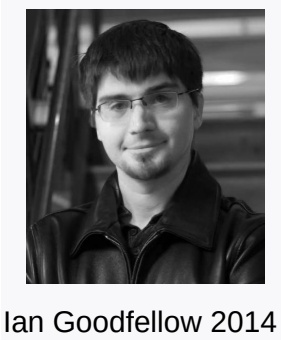


Figure 3: Ian Goodfellow, a young yet big figure in the modern AI field. He is best known as the inventor of the Generative Adversarial Networks (GANs), which made a big wave in the history of the AI field.

The third reason is related to *optimization* of this course's interest. Actually the GANs borrow very interesting ideas from *optimization*, thus making many optimization experts excited about the generative models. In particular, the duality theorems that we studied in Part II play a crucial role to understand the GANs as well as many GAN variants.

Generative models

Now let us dive into details on generative models. As you may easily guess, a generative model is defined as a mathematical model which allows us to generate *fake* data which has a *similar distribution* as that of *real* data. See Fig. 4 for a pictorial representation. The model parameters are learned via real data so that the learned model outputs fake data that resemble real data.

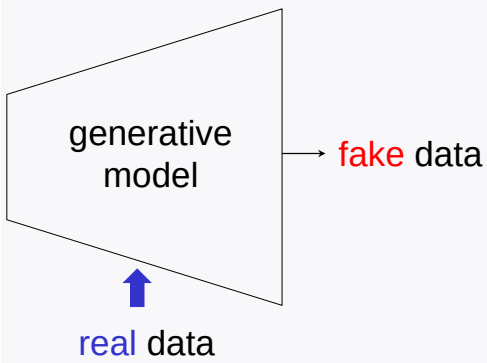


Figure 4: A generative model is the one that generates *fake* data which resembles *real* data. Here what *resembling* means in a mathematical language is that it has a *similar distribution*.

Here an input signal can be either an *arbitrary random* signal or a specifically synthesized signal that forms the skeleton of fake data. The type of the input depends on applications of interest - this will be detailed later on.

Remarks on generative models

In fact, the problem of designing a generative model is one of the most important problems in *statistics*, so it has been a classical age-old problem in that field. This is because the major goal of the field of statistics is to figure out (or estimate) the probability distribution of data that arise in the real world (that we call real data), and the generative model plays a role as a underlying framework in achieving the goal. Actually the model can do even more - it provides a concrete function block (called the *generator* in the field) which can create real-like fake data. There is a very popular name in statistics that indicates such problem, that is the *density estimation problem*. Here the density refers to the probability distribution.

As you may guess from the second reason that I mentioned above regarding why generative models are prominent, this problem was not that popular in the AI field until very recently, precisely 2014 when the GANs were invented.

How to formulate an optimization problem?

Now let us relate generative models to optimization of our interest. As mentioned earlier, we can feed some input signal (that we call *fake input*) which one can arbitrarily synthesize. Common ways employed in the field to generate them are to use Gaussian or uniform distributions. Since it is an *input* signal, we may wish to use a conventional "x" notation. So let us use $x \in \mathbf{R}^k$ to denote a fake input where k indicates a dimension of the signal.

Notice that this has a conflict with real data notation $\{x^{(i)}\}_{i=1}^m$. To avoid the conflict, let us use a different notation, say $\{y^{(i)}\}_{i=1}^m$, to denote real data. Please don't be confused with labeled data - these are not labels. In fact, the convention in the machine learning field is to use a notation z to indicate a fake input while maintaining real data notation as $\{x^{(i)}\}_{i=1}^m$. This may be another way to go; perhaps this is the way that you should take while writing papers. Anyhow let us take the first unorthodox yet reasonable option for this course.

Let $\hat{y} \in \mathbf{R}^n$ be a fake output. Considering m examples, let $\{(x^{(i)}, \hat{y}^{(i)})\}_{i=1}^m$ be such fake input-output m pairs and let $\{y^{(i)}\}_{i=1}^m$ be m real data examples. See Fig. 5.

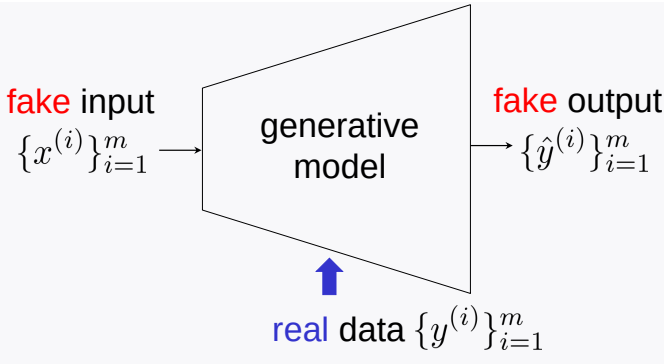


Figure 5: Problem formulation for generative models.

Goal

Let $G(\cdot)$ be a function of the generative model. Then, the goal of the generative model can be stated as: Designing $G(\cdot)$ such that

$$\{\hat{y}^{(i)}\}_{i=1}^m \approx \{y^{(i)}\}_{i=1}^m \textit{ in distribution.}$$

Here what does it mean by "in distribution"? To make it clear, we need to quantify closeness between two distributions. One natural yet prominent approach employed in the statistics field is to take the following two steps:

1. Compute empirical distributions or estimate distributions from $\{y^{(i)}\}_{i=1}^m$ and $\{(x^{(i)}, \hat{y}^{(i)})\}_{i=1}^m$. Let such distributions be:

   $$\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}}$$

   for real and fake data, respectively.

2. Next employ a well-known *divergence measure* in statistics which can serve to quantify the closeness of two distributions. Let $\mathsf{D}(\cdot, \cdot)$ be one such divergence measure. Then, the similarity between $\mathbb{Q}_Y$ and $\mathbb{Q}_{\hat{Y}}$ can be quantified as:

   $$\mathsf{D}(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}}).$$

Taking the above natural approach, one can concretely state the goal as: Designing $G(\cdot)$ such that

$$\mathsf{D}(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}}) \text{ is minimized.}$$

Optimization under the approach

Hence, under the approach, one can formulate an optimization problem as:

$$\min_{G(\cdot)} \mathsf{D}(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}}). \tag{1}$$

As you may easily notice, there are some issues in solving the above problem (1). There are three major issues.

The first is that it is *function optimization* which we are not familiar with. Notice that the optimization is over the function $G(\cdot)$. Second, the objective function $\mathsf{D}(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}})$ is a very complicated function of the knob $G(\cdot)$. Note that $\mathbb{Q}_{\hat{Y}}$ is a function of $G(\cdot)$, as $\hat{y} = G(x)$. So the objective is a twice folded composite function of $G(\cdot)$. The last is perhaps the most fundamental issue. It is not clear as to how to choose a divergence measure $D(\cdot, \cdot)$.

Look ahead

It turns out there are some ways to address the above issues. Very interestingly, one such way leads to an optimization problem for GANs! So next time, we will study what that way is, and then will take the way to derive an optimization problem for GANs.

Lecture 23: Generative Adversarial Networks (GANs)

Recap

Last time we started investigating *unsupervised learning*. The goal of unsupervised learning is to learn something about the data, which we newly denoted by $\{y^{(i)}\}_{i=1}^m$, instead of $\{x^{(i)}\}_{i=1}^m$. Depending on target candidates for something to learn, there are a few unsupervised learning methods. Among them, we explored one very prominent method, which is *generative models*. We formulated an optimization problem for generative models:

$$\min_{G(\cdot)} \mathsf{D}(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}}), \tag{1}$$

where $\mathbb{Q}_Y$ and $\mathbb{Q}_{\hat{Y}}$ indicate the empirical distributions (or the estimates of the true distributions) for real and fake data, respectively; $G(\cdot)$ denotes the function of a generative model; $\mathsf{D}(\cdot,\cdot)$ is a divergence measure. We then encountered a couple of issues that arise in the problem: (i) it is a *function optimization* which we are not familiar with; (ii) the objective is a very complicated function of $G(\cdot)$; (iii) not clear as to how to choose $\mathsf{D}(\cdot,\cdot)$.

At the end of the last lecture, I claimed that there are some ways to address such issues, and very interestingly, one such way leads to an optimization problem for a recently-developed powerful generative model, named Generative Adversarial Networks (GANs).

Today's lecture

Today we are going to explore details on the GANs. Specifically what we are going to do are three-folded. First we will investigate what that way leading to the GANs is. We will then take the way to derive an optimization problem for the GANs. Lastly we will demonstrate that the GANs indeed address the issues: (i) the GAN optimization problem is tractable; (ii) the problem can also be expressed as that in (1).

What is the way to address the issues?

Remember one challenge that we faced in the optimization problem (1): $\mathsf{D}(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}})$ is a complicated function of $G(\cdot)$. To address this, Ian Goodfellow, the inventor of GANs, took an *indirect way* to represent $\mathsf{D}(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}})$. He was thinking how $\mathsf{D}(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}})$ should behave, and then based on his observation, he was able to come up with an *indirect way* of mimicking the behaviour. It turned out the way led him to explicitly compute $\mathsf{D}(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}})$. Below are details.

How $\mathsf{D}(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}})$ should behave?

First of all, Goodfellow imagined how $\mathsf{D}(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}})$ should behave. What he imagined is that if one can *easily discriminate* real data y from fake data $\hat{y}$, then the divergence must be *large*; otherwise, it should be small. This naturally motivated him to:

Interpret $\mathsf{D}(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}})$ as a measure that can quantify *the ability to discriminate*.

In order to express such ability, he believed there must be something that plays a role of discriminating. So he introduced an entity that can play such role, and named it:

Discriminator.

Goodfellow considered a simple *binary-output* discriminator which takes as an input, either real data y or fake data $\hat{y}$. He then wanted to design $D(\cdot)$ such that $D(\cdot)$ well *approximates* the probability that the input $(\cdot)$ is *real* data:

$$D(\cdot) \approx \Pr((\cdot) = \text{real data}).$$

Noticing that

$$\Pr(y = \text{real}) = 1;$$
$$\Pr(\hat{y} = \text{real}) = 0,$$

he wanted to design $D(\cdot)$ such that:

$$D(y) \text{ is as large as possible, close to 1;}$$
$$D(\hat{y}) \text{ is as small as possible, close to 0.}$$

See Fig. 1.

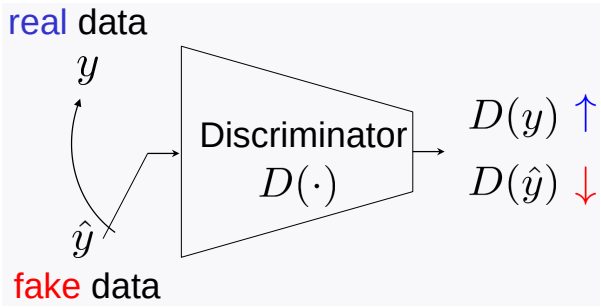


Figure 1: Discriminator wishes to output $D(\cdot)$ such that $D(y)$ is as large as possible while $D(\hat{y})$ is as small as possible.

How to quantity the ability to discriminate?

Keeping the picture in Fig. 1 in his mind, he then wanted to quantify the ability to discriminate. To this end, he first observed that if $D(\cdot)$ can easily discriminate, then we should have:

$$D(y) \uparrow; \;\; 1 - D(\hat{y}) \uparrow.$$

One naive way to capture the ability is simply *adding* the above two terms. But Goodfellow did not take the naive way. Instead he took the following *logarithmic* summation:

$$\log D(y) + \log(1 - D(\hat{y})). \tag{2}$$

Actually I was wondering why he took this particular way, as his paper does not mention about the rationale behind the choice. In NIPS 2016, he gave a tutorial on GANs, mentioning that the problem formulation was inspired by a paper published in AISTATS 2010:

$$http://proceedings.mlr.press/v9/gutmann10a.html$$

Reading the paper, I realized that the logarithmic summation (2) comes from Eq. (3) in the paper.

Making the particular choice, the ability to discriminate for m examples can be quantified as:

$$\frac{1}{m}\sum_{i=1}^{m}\log D(y^{(i)}) + \log(1 - D(\hat{y}^{(i)})). \tag{3}$$

A two-player game

Goodfellow then imagined a *two-player game* in which Player 1, Discriminator $D(\cdot)$, wishes to maximize the quantified ability (3), while Player 2, Generator $G(\cdot)$, wants to minimize (3). See Fig. 2.

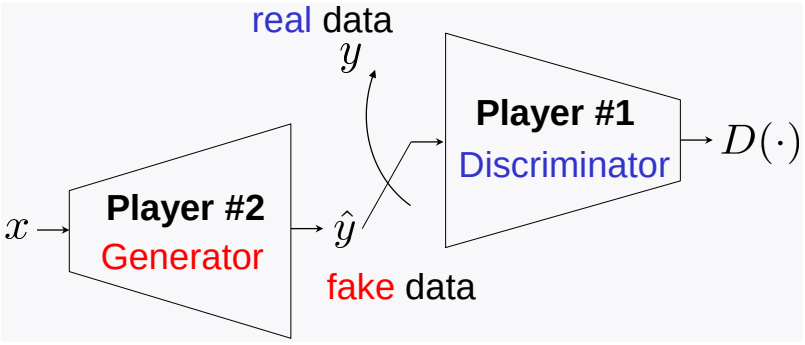


Figure 2: A two-player game.

Optimization for GANs

This naturally motivated him to formulate the following min max optimization problem:

$$\min_{G(\cdot)}\max_{D(\cdot)}\frac{1}{m}\sum_{i=1}^{m}\log D(y^{(i)}) + \log(1 - D(\hat{y}^{(i)})). \tag{4}$$

You may wonder why not max min. That may be another way to go, but Goodfellow made the above choice. Actually there is a reason why he took the way. This will be clearer soon. Here notice that the optimization is over the two *functions* of $D(\cdot)$ and $G(\cdot)$, meaning that it is still a *function* optimization. Luckily the year of 2014 (when the GAN paper was published) was after the starting point of the deep learning revolution, the year of 2012. Also at that time, Goodfellow was a PhD student of Joshua Bengio, one of the giants in the deep learning field. So he was very much aware of the power of neural networks:

"*Deep neural networks can well represent any arbitrary function.*"

This motivated him to parameterize the two functions with DNNs, which in turn led to the following optimization problem:

$$\min_{G(\cdot)\in\mathcal{N}}\max_{D(\cdot)\in\mathcal{N}}\frac{1}{m}\sum_{i=1}^{m}\log D(y^{(i)}) + \log(1 - D(\hat{y}^{(i)})), \tag{5}$$

where $\mathcal{N}$ denotes a set of DNN-architecture-based functions. This is exactly the optimization problem for GANs!

Related to original optimization?

Remember what I mentioned earlier: the way leading to the GAN optimization is an indirect way of solving the original optimization problem:

$$\min_{G(\cdot)} \mathsf{D}(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}}). \tag{6}$$

Then, a natural question that arises is: How are the two problems (5) and (6) related? It turns out these are very much related. This is exactly where the choice of min max (instead of max min) plays the role; the other choice cannot establish a connection - it was a smart choice. It has been shown that assuming that deep neural networks can represent any arbitrary function, the GAN optimization (5) can be translated into the original optimization form (6). We will prove this below.

Simplification & manipulation

Let us start by simplifying the GAN optimization (5). Since we assume that $\mathcal{N}$ can represent any arbitrary function, the problem (5) becomes *unconstrained*:

$$\min_{G(\cdot)} \max_{D(\cdot)} \frac{1}{m} \sum_{i=1}^{m} \log D(y^{(i)}) + \log(1 - D(\hat{y}^{(i)})). \tag{7}$$

Notice that the objective is a function of $D(\cdot)$, and the two functions $D(\cdot)$'s appear but with *different* arguments: one is $y^{(i)}$, marked in blue; the other is $\hat{y}^{(i)}$, marked in red. So in the current form (7), the inner (max) optimization problem is not quite tractable to solve. In an attempt to make it tractable, let us express it in a different manner using the following notations.

Define a random vector Y which takes one of the m real examples with probability $\frac{1}{m}$ (uniform distribution):

$$Y \in \{y^{(1)}, \dots, y^{(m)}\} =: \mathcal{Y}; \;\; \mathbb{Q}_Y(y^{(i)}) = \frac{1}{m}, \;\; i = 1, 2, \dots, m,$$

where $\mathbb{Q}_Y$ indicates the probability distribution of Y. Similarly define $\hat{Y}$ for fake examples:

$$\hat{Y} \in \{\hat{y}^{(1)}, \dots, \hat{y}^{(m)}\} =: \hat{\mathcal{Y}}; \;\; \mathbb{Q}_{\hat{Y}}(\hat{y}^{(i)}) = \frac{1}{m}, \;\; i = 1, 2, \dots, m,$$

where $\mathbb{Q}_{\hat{Y}}$ indicates the probability distribution of $\hat{Y}$. Using these notations, one can then rewrite the problem (7) as:

$$\min_{G(\cdot)} \max_{D(\cdot)} \sum_{i=1}^{m} \mathbb{Q}_Y(y^{(i)}) \log D(y^{(i)}) + \mathbb{Q}_{\hat{Y}}(\hat{y}^{(i)}) \log(1 - D(\hat{y}^{(i)})). \tag{8}$$

Still we have different arguments in the two $D(\cdot)$ functions.

To address this, let us introduce another quantity. Let $z \in \mathcal{Y} \cup \hat{\mathcal{Y}}$. Newly define $\mathbb{Q}_Y(\cdot)$ and $\mathbb{Q}_{\hat{Y}}(\cdot)$ such that:

$$\mathbb{Q}_Y(z) := 0 \text{ if } z \in \hat{\mathcal{Y}} \setminus \mathcal{Y}; \tag{9}$$

$$\mathbb{Q}_{\hat{Y}}(z) := 0 \text{ if } z \in \mathcal{Y} \setminus \hat{\mathcal{Y}}. \tag{10}$$

Using the z notation, one can then rewrite the problem (8) as:

$$\min_{G(\cdot)} \max_{D(\cdot)} \sum_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mathbb{Q}_Y(z) \log D(z) + \mathbb{Q}_{\hat{Y}}(z) \log(1 - D(z)). \tag{11}$$

We now see that the same arguments appear in the two $D(\cdot)$ functions.

Solving the inner optimization problem

We are ready to solve the inner optimization problem in (11). Key observations are: $\log D(z)$ is concave in $D(\cdot)$; $\log(1 - D(z))$ is concave in $D(\cdot)$; and therefore, the objective function is concave in $D(\cdot)$. This implies that the objective has the *unique maximum* in the function space $D(\cdot)$. Hence, one can find such maximum by searching for the one in which the derivative is zero. Taking a derivative and setting it to zero, we get:

$$\text{Derivative} = \sum_z \left[\frac{\mathbb{Q}_Y(z)}{D^*(z)} - \frac{\mathbb{Q}_{\hat{Y}}(z)}{1 - D^*(z)} \right] = 0.$$

Hence, we get:

$$D^*(z) = \frac{\mathbb{Q}_Y(z)}{\mathbb{Q}_Y(z) + \mathbb{Q}_{\hat{Y}}(z)}. \tag{12}$$

Plugging this into (11), we obtain:

$$\min_{G(\cdot)} \sum_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mathbb{Q}_Y(z) \log \frac{\mathbb{Q}_Y(z)}{\mathbb{Q}_Y(z) + \mathbb{Q}_{\hat{Y}}(z)} + \mathbb{Q}_{\hat{Y}}(z) \log \frac{\mathbb{Q}_{\hat{Y}}(z)}{\mathbb{Q}_Y(z) + \mathbb{Q}_{\hat{Y}}(z)}. \tag{13}$$

Jensen-Shannon divergence

Let us massage the objective function in (13) to express it as:

$$\min_{G(\cdot)} \underbrace{\sum_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mathbb{Q}_Y(z) \log \frac{\mathbb{Q}_Y(z)}{\frac{\mathbb{Q}_Y(z) + \mathbb{Q}_{\hat{Y}}(z)}{2}} + \mathbb{Q}_{\hat{Y}}(z) \log \frac{\mathbb{Q}_{\hat{Y}}(z)}{\frac{\mathbb{Q}_Y(z) + \mathbb{Q}_{\hat{Y}}(z)}{2}}}_{} - 2 \log 2. \tag{14}$$

It turns out the above underbraced term can be expressed with a well-known divergence measure in statistics, called *Jensen-Shannon divergence*[1]: for any two distributions, say p and q,

$$\mathsf{JSD}(p, q) := \frac{1}{2} \sum_z p(z) \log \frac{p(z)}{\frac{p(z)+q(z)}{2}} + \frac{1}{2} \sum_z q(z) \log \frac{q(z)}{\frac{p(z)+q(z)}{2}}. \tag{15}$$

This is indeed a valid divergence measure, i.e., it is non-negative, being equal to zero if and only if $p = q$. We will not prove this here, but you will have a chance to prove in PS.

Equivalent form

Using the divergence, one can then rewrite the problem (14) as:

$$\min_{G(\cdot)} 2\mathsf{JSD}(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}}) - 2 \log 2. \tag{16}$$

Hence, we get:

$$G^*_{\mathsf{GAN}} = \arg\min_{G(\cdot)} \mathsf{JSD}(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}}). \tag{17}$$

[1] One may guess that this is the divergence that Johan Jensen (the inventor of Jensen's inequality) and Claude Shannon (the Father of Information Theory) developed. But it is not the case. Johan Jensen died in 1925 when Claude Shannon was a child, so there was no collaboration between the two. Actually it was invented much later days in 1991 by a Taiwanese information theorist, named Jianhua Lin.

We see that this indeed belongs to the original optimization form (6) if one makes a choice for $\mathsf{D}(\cdot,\cdot)$ as: $\mathsf{D}(\cdot,\cdot) = \mathsf{JSD}(\cdot,\cdot)$.

An issue

The above derivation (17) was initially done by Ian Goodfellow in his seminal paper. Perhaps he might be happy about his derivation, as it is very simple and insightful, connecting the GAN to the beauty of statistics. In fact, I was surprised how the two-player game led to a very insightful JSD-based optimization problem. But Goodfellow missed something critical in the derivation.

Instead another person figured out that something almost immediately. That person is a computer scientist (as well as a mathematician), named Léon Bottou; see Fig. 3. He is a long

Figure 3: Léon Bottou is the inventor of Wasserstain GANs. He is another big figure in the AI field. He was a professor in NYU, but now moved to Facebook AI Research (FAIR).

collaborator of Yann LeCun. He is very strong at math and stats. So he could figure out that there is a critical issue in the GAN optimization (17), which comes from some undesirable property of JSD. More importantly, he knew how to address the issue. In the course of addressing the issue, he could develop a variant of GAN, which he called:

$$Wasserstein\ GAN.$$

For the remaining lectures

For the remaining two lectures, we will figure out what that critical issue is. We will then investigate how Bottou came up with Wasserstein GAN.

Lecture 24: Wasserstein GAN

Recap

Last time we formulated the optimization problem for GANs: Given the data $\{y^{(i)}\}_{i=1}^m$,

$$\min_{G(\cdot)\in\mathcal{N}} \max_{D(\cdot)\in\mathcal{N}} \frac{1}{m}\sum_{i=1}^{m} \log D(y^{(i)}) + \log(1 - D(\hat{y}^{(i)})) \tag{1}$$

where $G(\cdot)$ and $D(\cdot)$ indicate the functions of the generator and the discriminator, respectively; and $\mathcal{N}$ denotes a set of DNN-based functions. We then showed that the GAN optimization belongs to a very generic divergence-based optimization problem (an age-old problem in statistics), assuming that DNNs can represent any arbitrary function.

At the end of the last lecture, however, I claimed that there is a critical issue in the GAN optimization, and this is what Bottou figured out. I also claimed that in the course of addressing the issue, Bottou came up with a variant of GANs, which he named: Wasserstein GAN (WGAN for short).

Today's lecture

Supporting these claims form the contents of this lecture. Specifically we will cover the following three stuffs. First of all, we will figure out what the critical issue that arises in GANs is. We will then investigate how Bottou addressed the issue. Lastly we will discuss how Bottou's way is related to an optimization problem for WGAN.

What is the critical issue that arises in the GAN optimization?

Recall in the GAN optimization that the optimal generator $G^*(\cdot)$ reads:

$$G^* = \arg\min_{G(\cdot)} \mathsf{JSD}(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}}) \tag{2}$$

where $\mathbb{Q}_Y$ and $\mathbb{Q}_{\hat{Y}}$ indicate the empirical distributions of $Y \in \{y^{(1)}, \ldots, y^{(m)}\} =: \mathcal{Y}$ and $\hat{Y} \in \{\hat{y}^{(1)}, \ldots, \hat{y}^{(m)}\} =: \hat{\mathcal{Y}}$ respectively, and

$$\mathsf{JSD}(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}}) = \frac{1}{2}\sum_{z\in\mathcal{Y}\cup\hat{\mathcal{Y}}} \mathbb{Q}_Y(z)\log\frac{\mathbb{Q}_Y(z)}{\frac{\mathbb{Q}_Y(z)+\mathbb{Q}_{\hat{Y}}(z)}{2}} + \mathbb{Q}_{\hat{Y}}(z)\log\frac{\mathbb{Q}_{\hat{Y}}(z)}{\frac{\mathbb{Q}_Y(z)+\mathbb{Q}_{\hat{Y}}(z)}{2}}. \tag{3}$$

Here z indicates a dummy variable which takes an element either from $\mathcal{Y}$ or from $\hat{\mathcal{Y}}$. One key observation that one can make here is that in almost all practically-relevant settings, fake and real samples are *different* with each other:

$$\mathcal{Y} \cap \hat{\mathcal{Y}} = \emptyset. \tag{4}$$

Why? Imagine an image-data setting in which the dimension of data is usually very large, e.g., the dimension of an image in ImageNet is: $256 \times 256 \times 3 = 196,608$. In this setting, the probability that any fake image is exactly the same as one of the real images is almost 0, so it is

a *measure-zero event*. This (4) together with the definitions of $\mathbb{Q}_Y$ and $\mathbb{Q}_{\hat{Y}}$ (that we established in Lecture 23) then yields:

$$\frac{\mathbb{Q}_Y(z)}{\frac{\mathbb{Q}_Y(z)+\mathbb{Q}_{\hat{Y}}(z)}{2}} = \begin{cases} \frac{\frac{1}{m}}{\frac{\frac{1}{m}+0}{2}} = 2, & \text{if } z \in \mathcal{Y}; \\ \frac{0}{\frac{\frac{1}{m}+0}{2}} = 0, & \text{if } z \in \hat{\mathcal{Y}}; \end{cases}$$
$$\frac{\mathbb{Q}_{\hat{Y}}(z)}{\frac{\mathbb{Q}_Y(z)+\mathbb{Q}_{\hat{Y}}(z)}{2}} = \begin{cases} \frac{0}{\frac{\frac{1}{m}+0}{2}} = 0, & \text{if } z \in \mathcal{Y}; \\ \frac{\frac{1}{m}}{\frac{0+\frac{1}{m}}{2}} = 2, & \text{if } z \in \hat{\mathcal{Y}}. \end{cases} \tag{5}$$

Plugging this into (3), we get:

$$\begin{aligned} \mathsf{JSD}(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}}) &= \frac{1}{2}\sum_{z\in\mathcal{Y}} \mathbb{Q}_Y(z) \log \frac{\mathbb{Q}_Y(z)}{\frac{\mathbb{Q}_Y(z)+\mathbb{Q}_{\hat{Y}}(z)}{2}} + \frac{1}{2}\sum_{z\in\hat{\mathcal{Y}}} \mathbb{Q}_{\hat{Y}}(z) \log \frac{\mathbb{Q}_{\hat{Y}}(z)}{\frac{\mathbb{Q}_Y(z)+\mathbb{Q}_{\hat{Y}}(z)}{2}} \\ &= \frac{1}{2}\sum_{z\in\mathcal{Y}} \mathbb{Q}_Y(z) \log 2 + \frac{1}{2}\sum_{z\in\hat{\mathcal{Y}}} \mathbb{Q}_{\hat{Y}}(z) \log 2 \\ &= \log 2 \end{aligned} \tag{6}$$

where the last equality comes from $\sum_{z\in\mathcal{Y}} \mathbb{Q}_Y(z) = 1$ and $\sum_{z\in\hat{\mathcal{Y}}} \mathbb{Q}_{\hat{Y}}(z)$.

From (6), we can now see the critical issue:

$$\mathsf{JSD}(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}}) \text{ is } \textit{irrelevant} \text{ of how we choose } G(\cdot),$$

meaning that

$$G^* = \arg\min_{G(\cdot)} \mathsf{JSD}(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}}) = \arg\min_{G(\cdot)} \log 2 \quad \text{could be anything.} \tag{7}$$

This implies that we may arrive at a very stupid solution from the JSD-based optimization (2), as any $G(\cdot)$ can be optimal.

But wait! You may see that something weird is happening. Why? We already knew that GANs are working well in practice. This suggests that the phenomena observed by many practitioners look inconsistent with the theory due to the above simple derivation (7). Any mistake in the above derivation? Or something wrong in simulations done by many practitioners? Or something else? It turns out the answer is "something else". Remember in the GAN optimization (1) that $G(\cdot)$ and $D(\cdot)$ should be *DNN-based functions*, not arbitrary functions. So precisely speaking, the optimal generator should read:

$$G^*_{\mathsf{GAN}} := \arg \min_{G(\cdot)\in\mathcal{N}} \max_{D(\cdot)\in\mathcal{N}} \frac{1}{m}\sum_{i=1}^{m} \log D(y^{(i)}) + \log(1 - D(\hat{y}^{(i)})).$$

In practice, DNN is not perfectly expressive, and hence:

$$G^*_{\mathsf{GAN}} \neq G^*.$$

This is the reason why there is inconsistency between the theory and the practice. There are some groups of people (including Prof. Sanjeev Arora at Princeton, a brilliant theoretical computer scientist) who have been investigating why the GANs with DNN-function constraints lead to good performances. Nonetheless, as of now, no clear understanding on this.

Motivated the use of the Wasserstein distance!

The critical issue, reflected in (7), motivated Bottou to reconsider the generic divergence-based optimization problem:

$$\min_{G(\cdot)} \mathsf{D}(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}}) \tag{8}$$

where $\mathsf{D}(\cdot,\cdot)$ is of our design choice. He knew that there are some good divergence measures which do no yield the critical issue (7). One of the measures that he chose was the 1st order Wasserstein distance that we studied in Part I. This led him to obtain:

$$\min_{G(\cdot)} W(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}}) \tag{9}$$

where

$$\begin{aligned} W(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}}) &= \min_{\mathbb{Q}_{Y,\hat{Y}}} \mathbb{E}[\|Y - \hat{Y}\|] \\ &= \min_{\mathbb{Q}_{Y,\hat{Y}}} \sum_{i=1}^{m}\sum_{j=1}^{m} \mathbb{Q}_{Y,\hat{Y}}(y^{(i)}, \hat{y}^{(j)}) \|y^{(i)} - \hat{y}^{(j)}\|. \end{aligned} \tag{10}$$

Notice that $\|y^{(i)} - \hat{y}^{(j)}\|$ placed inside the doubled summation (marked in blue) depends on the values of $\{\hat{y}^{(i)}\}_{i=1}^m$ themselves. Hence, we can readily see that the objective is indeed a function of $G(\cdot)$ which directly controls $\{\hat{y}^{(i)}\}_{i=1}^m$.

How to solve the Wasserstein-distance-based optimization?

Replacing $\mathsf{D}(\cdot,\cdot)$ with the Wasserstein distance in (8), Bottou then wrote the optimization problem (9) as:

$$\begin{aligned} \min_{G(\cdot)} \min_{\mathbb{Q}_{Y,\hat{Y}}} & \sum_{i=1}^{m}\sum_{j=1}^{m} \mathbb{Q}_{Y,\hat{Y}}(y^{(i)}, \hat{y}^{(j)}) \|y^{(i)} - \hat{y}^{(j)}\| : \\ & \sum_{j=1}^{m} \mathbb{Q}_{Y,\hat{Y}}(y^{(i)}, \hat{y}^{(j)}) = \mathbb{Q}_Y(y^{(i)}), \quad i = 1, \ldots, m; \\ & \sum_{i=1}^{m} \mathbb{Q}_{Y,\hat{Y}}(y^{(i)}, \hat{y}^{(j)}) = \mathbb{Q}_{\hat{Y}}(\hat{y}^{(j)}), \quad j = 1, \ldots, m. \end{aligned} \tag{11}$$

Consider the inner optimization problem in (11). This is the problem that we are familiar with: LP! We know how to solve an LP using the simplex algorithm. Then, no problem? Unfortunately, that is not the case. There are some issues in solving the above problem. Two major issues.

The first is that there is no closed-form solution for LP, so this gives a challenge in finding $G^*(\cdot)$ in the end. The second issue is a more critical one. Notice in practice that the number of optimization variables, which are m^2 of $\mathbb{Q}_{Y,\hat{Y}}(y^{(i)}, \hat{y}^{(j)})$, in the inner problem is huge. In the big data era, m is typically an order of more than thousands or million, or even billion. So m^2 is typically a huge number. Even if we use a very fast algorithm, like the simplex algorithm, it would take very long time. So it is computationally very expensive.

Bottou immediately recognized the issues. More importantly, he knew how to address the issues. The idea is to rely on the Father of LP: Kantorovich! In fact, Kantorovich already established the strong duality theorem for this Wasserstein-distance-based LP, called Kantorovich duality or Kantorovich-Rubinstein duality. What Kantorovich showed is that the dual problem of the

Wasserstein-distance-based LP has exactly the same solution as that of the primal problem[1], and more importantly, the dual problem is computationally much more efficient. This is what Bottou already knew. So he simply applied the Kantorovich duality to come up with an optimization problem, which is now known as the WGAN optimization. We will describe the Kantorovich duality in detail below.

Notational simplification

Let us consider the inner optimization problem in (11). For notational simplification, we employ the dummy variable z that we introduced earlier: $z \in \mathcal{Y} \cup \hat{\mathcal{Y}}$. Let us use z and $\hat{z}$ to indicate $y^{(i)}$ and $\hat{y}^{(j)}$, respectively. Using this notation, we can then rewrite the inner optimization problem as:

$$\begin{aligned} \min_{\mathbb{Q}_{Y,\hat{Y}}} & \sum_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \sum_{\hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mathbb{Q}_{Y,\hat{Y}}(z, \hat{z}) \|z - \hat{z}\| : \\ & \sum_{\hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mathbb{Q}_{Y,\hat{Y}}(z, \hat{z}) = \mathbb{Q}_Y(z) \;\; \forall z \in \mathcal{Y} \cup \hat{\mathcal{Y}}; \\ & \sum_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mathbb{Q}_{Y,\hat{Y}}(z, \hat{z}) = \mathbb{Q}_{\hat{Y}}(\hat{z}) \;\; \forall \hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}. \end{aligned} \tag{12}$$

For further notational simplification, let

$$x(z, \hat{z}) := \mathbb{Q}_{Y,\hat{Y}}(z, \hat{z}) \geq 0. \tag{13}$$

Then, the problem can be rewritten as:

$$\begin{aligned} \min_{x(z,\hat{z})} & \sum_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \sum_{\hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \|z - \hat{z}\| x(z, \hat{z}) : \\ & - x(z, \hat{z}) \leq 0, \;\; \forall z, \hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}; \\ & \sum_{\hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}} x(z, \hat{z}) = \mathbb{Q}_Y(z) \;\; \forall z \in \mathcal{Y} \cup \hat{\mathcal{Y}}; \\ & \sum_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} x(z, \hat{z}) = \mathbb{Q}_{\hat{Y}}(\hat{z}) \;\; \forall \hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}. \end{aligned} \tag{14}$$

Lagrange function, dual function & dual problem

In an effort to derive the dual problem, we first consider the Lagrange function:

$$\begin{aligned} \mathcal{L}(x, \lambda, \nu, \mu) = & \sum_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \sum_{\hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \|z - \hat{z}\| x(z, \hat{z}) - \sum_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \sum_{\hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \lambda(z, \hat{z}) x(z, \hat{z}) \\ & + \sum_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \nu(z) \left(\mathbb{Q}_Y(z) - \sum_{\hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}} x(z, \hat{z}) \right) + \sum_{\hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mu(\hat{z}) \left(\mathbb{Q}_{\hat{Y}}(\hat{z}) - \sum_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} x(z, \hat{z}) \right) \\ = & \sum_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \sum_{\hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}} (\|z - \hat{z}\| - \lambda(z, \hat{z}) - \nu(z) - \mu(\hat{z})) x(z, \hat{z}) + \sum_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \nu(z) \mathbb{Q}_Y(z) + \sum_{\hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mu(\hat{z}) \mathbb{Q}_{\hat{Y}}(\hat{z}) \end{aligned} \tag{15}$$

[1]This is what we already know. But at that time, the strong duality theorem for convex optimization was not established yet. In fact, the Kantorovich duality formed the basis of the strong duality theorem for generic convex problems.

where (λ, ν, μ) are Lagrange multipliers. Notice that the multiplication factors associated with $x(z, \hat{z})$'s in the double summation term in the above last equation (marked in red) should be zeros:

$$\|z - \hat{z}\| - \lambda(z, \hat{z}) - \nu(z) - \mu(\hat{z}) = 0 \quad \forall z, \hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}. \tag{16}$$

Otherwise, one can set $x(z, \hat{z}) = \infty$ (or $-\infty$) depending on the sign of a non-zero such term while setting $x(z, \hat{z}) = 0$ for the other terms. This then yields $\mathcal{L}(x, \lambda, \nu, \mu) = -\infty$, and hence $g(\lambda, \nu, \mu) = -\infty$. Obviously this is not an interested case. Hence, applying (16), we derive the dual function as:

$$g(\lambda, \nu, \mu) = \sum_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \nu(z) \mathbb{Q}_Y(z) + \sum_{\hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mu(\hat{z}) \mathbb{Q}_{\hat{Y}}(\hat{z}). \tag{17}$$

Now notice that $\lambda(z, \hat{z}) \geq 0$ is a constraint that appears in the dual problem. This together with (16) then yields:

$$\|z - \hat{z}\| - \nu(z) - \mu(\hat{z}) = \lambda(z, \hat{z}) \geq 0 \quad \forall z, \hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}. \tag{18}$$

Using this, we can then formulate the dual problem as:

$$\begin{aligned} d^* := \max_{\nu, \mu} & \sum_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \nu(z) \mathbb{Q}_Y(z) + \sum_{\hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mu(\hat{z}) \mathbb{Q}_{\hat{Y}}(\hat{z}) : \\ & \nu(z) + \mu(\hat{z}) \leq \|z - \hat{z}\| \quad \forall z, \hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}. \end{aligned} \tag{19}$$

How to deal with two functions in the optimization?

How to solve the dual problem (19)? Actually it is not that simple. One may think of the following native approach: Searching for all the possible functions of $\nu(\cdot)$ and $\mu(\cdot)$ in finding the maximum. Then, it would have similar complexity as that of solving the primal problem!

Of course, Kantorovich did not take that naive approach. Instead he came up with a very interesting and smart idea. The idea is to translate the problem (19) with *two* functions ($\nu(\cdot)$ and $\mu(\cdot)$) that one can control over, into an equivalent problem but with *only one* function, say $\psi(\cdot)$. It turns out the idea led Kantorovich to come up with the following equivalent problem:

$$\begin{aligned} d^{**} := \max_{\psi} & \sum_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mathbb{Q}_Y(z) \psi(z) - \sum_{\hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mathbb{Q}_{\hat{Y}}(\hat{z}) \psi(\hat{z}) : \\ & |\psi(z) - \psi(\hat{z})| \leq \|z - \hat{z}\| \quad \forall z, \hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}. \end{aligned} \tag{20}$$

Notice in the translated problem that we have only one function to optimize over, so now the complexity is significantly reduced relative to that of (19).

Relying on the proof of $d^* = d^{**}$ together with the outer optimization problem (w.r.t. $G(\cdot)$), Bottou was able to derive a simpler form of an optimization problem, which is now known as the WGAN optimization.

Look ahead

Next time, we will prove that d^* is indeed d^{**}. We will then demonstrate that this proof leads to the WGAN optimization.

June 4, 2019

Lecture 25: Wasserstein GAN II

Recap

Last time we figured out there is a critical issue in GANs: $\mathsf{JSD}(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}})$ is *irrelevant* of $G(\cdot)$, which in turns suggests that the optimal G^* could be anything - this is definitely not what we want. In an effort to address this issue, we considered the 1st-order Wasserstein distance which does not have such undesirable property:

$$\min_{G(\cdot)} W(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}}) \tag{1}$$

where $\mathbb{Q}_Y$ and $\mathbb{Q}_{\hat{Y}}$ indicate the empirical distributions of $Y \in \{y^{(1)}, \ldots, y^{(m)}\} =: \mathcal{Y}$ and $\hat{Y} \in \{\hat{y}^{(1)}, \ldots, \hat{y}^{(m)}\} =: \hat{\mathcal{Y}}$, respectively. We then checked that the objective function in (1) is a sensitive function of $G(\cdot)$.

Since the inner optimization in (1) involves so many optimization variables (whose number scales at m^2) and hence is not computationally tractable (which is often the case in the current big data era), we started considering the dual problem which is known to be computationally tractable due to the Kantorovich duality. Introducing the $z \in \mathcal{Y} \cup \hat{\mathcal{Y}}$ notation, we expressed the dual problem as:

$$\begin{aligned} d^* := \max_{\nu,\mu} \sum_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mathbb{Q}_Y(z)\nu(z) + \sum_{\hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mathbb{Q}_{\hat{Y}}(\hat{z})\mu(\hat{z}) : \\ \nu(z) + \mu(\hat{z}) \le \|z - \hat{z}\| \quad \forall z, \hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}} \end{aligned} \tag{2}$$

where $\nu(z)$ and $\mu(z)$ denote Lagrange multipliers w.r.t. the marginal-distribution-associated equality constraints, one for $\mathbb{Q}_Y$ and the other for $\mathbb{Q}_{\hat{Y}}$.

At the end of the last lecture, we claimed that the above problem is equivalent to the following simpler optimization problem containing only one function variable:

$$\begin{aligned} d^{**} := \max_{\psi} \sum_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mathbb{Q}_Y(z)\psi(z) - \sum_{\hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mathbb{Q}_{\hat{Y}}(\hat{z})\psi(\hat{z}) : \\ |\psi(z) - \psi(\hat{z})| \le \|z - \hat{z}\| \quad \forall z, \hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}. \end{aligned} \tag{3}$$

Today's lecture

Today we will prove the above claim and then derive the WGAN optimization accordingly. Specifically what we are going to cover are three-folded. We will first prove $d^* = d^{**}$. We will then use the claim to derive an optimization for WGAN. Finally we will discuss on the optimality of the Wasserstein distance for divergence-measure based optimization problems.

Proof of $d^* \ge d^{**}$

We will show $d^* \ge d^{**}$ and $d^* \le d^{**}$ to complete the proof. First let us prove the former.

Consider:

$$\begin{aligned} d^{**} &:= \max_{\psi} \sum_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mathbb{Q}_Y(z)\psi(z) - \sum_{\hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mathbb{Q}_{\hat{Y}}(\hat{z})\psi(\hat{z}) \\ &\leq \max_{\psi} \max_{\mu} \sum_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mathbb{Q}_Y(z)\psi(z) + \sum_{\hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mathbb{Q}_{\hat{Y}}(\hat{z})\mu(\hat{z}) \\ &= \max_{\nu} \max_{\mu} \sum_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mathbb{Q}_Y(z)\nu(z) + \sum_{\hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mathbb{Q}_{\hat{Y}}(\hat{z})\mu(\hat{z}) \end{aligned} \tag{4}$$

where the inequality follows from the fact that $-\psi(\hat{z})$ (next to $\mathbb{Q}_{\hat{Y}}(\hat{z})$ in the first equation) can be interpreted as a particular choice among general functions represented by $\mu(\cdot)$; and the last equality comes from a change of function variable from $\psi(\cdot)$ to $\nu(\cdot)$.

On the other hand, with the new function notations, one can represent the constraint in (3) as:

$$|\nu(z) + \mu(\hat{z})| \leq \|z - \hat{z}\| \quad \forall z, \hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}. \tag{5}$$

Since the RHS in the above is non-negative, the constraint implies that:

$$\nu(z) + \mu(\hat{z}) \leq \|z - \hat{z}\| \quad \forall z, \hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}. \tag{6}$$

This together with (4) then yields:

$$d^* \geq d^{**}. \tag{7}$$

Proof of $d^* \leq d^{**}$

Moving the $\nu(z)$ term in the constraint of (2) to the RHS, we get:

$$\mu(\hat{z}) \leq -\nu(z) + \|z - \hat{z}\|. \tag{8}$$

Since this holds $\forall z, \hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}$, we obtain:

$$\begin{aligned} \mu(\hat{z}) &\leq \min_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} -\nu(z) + \|z - \hat{z}\| \\ &\leq -\nu(\hat{z}) \end{aligned} \tag{9}$$

where the second inequality comes form choosing the optimization variable z as $\hat{z}$. We now define:

$$-\psi(\hat{z}) := \min_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} -\nu(z) + \|z - \hat{z}\|. \tag{10}$$

This definition together with the two inequalities in (9) gives:

$$\begin{aligned} \nu(z) &\leq \psi(z) \quad \forall z \in \mathcal{Y} \cup \hat{\mathcal{Y}}; \\ \mu(\hat{z}) &\leq -\psi(\hat{z}) \quad \forall \hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}. \end{aligned} \tag{11}$$

Applying these into (2), we get:

$$d^* \leq \max_{\psi} \sum_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mathbb{Q}_Y(z)\psi(z) - \sum_{\hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mathbb{Q}_{\hat{Y}}(\hat{z})\psi(\hat{z}). \tag{12}$$

This coincides with the objective function in the other interested optimization problem (3). And it turns out that the definition (10) incurs a constraint which matches with the one in (3).

To see this, consider:

$$\begin{aligned} \psi(z) &= \max_{t \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \nu(t) - \|t - z\| \qquad \forall z \in \mathcal{Y} \cup \hat{\mathcal{Y}}; \\ -\psi(\hat{z}) &= \min_{t' \in \mathcal{Y} \cup \hat{\mathcal{Y}}} -\nu(t') + \|t' - \hat{z}\| \quad \forall \hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}. \end{aligned} \tag{13}$$

This comes simply from the definition (10). Adding the above two, we get: $\forall z, \hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}$,

$$\begin{aligned} \psi(z) - \psi(\hat{z}) &= \max_{t \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \nu(t) - \|t - z\| + \min_{t' \in \mathcal{Y} \cup \hat{\mathcal{Y}}} -\nu(t') + \|t' - \hat{z}\| \\ &\overset{(a)}{\leq} \max_{t \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \nu(t) - \|t - z\| - \nu(t) + \|t - \hat{z}\| \\ &\overset{(b)}{\leq} \|z - \hat{z}\| \end{aligned} \tag{14}$$

where (a) follows from choosing $t' = t$ in the minimization part; and (b) comes from the triangular inequality:

$$\|t - \hat{z}\| \leq \|t - z\| + \|z - \hat{z}\|.$$

Swapping the roles of z and $\hat{z}$ in (14), one can also get: $\forall z, \hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}$,

$$\psi(\hat{z}) - \psi(z) \leq \|z - \hat{z}\|. \tag{15}$$

This together with (14) then yields:

$$|\psi(\hat{z}) - \psi(z)| \leq \|z - \hat{z}\| \quad \forall z, \hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}. \tag{16}$$

Using this and (12), we can conclude that:

$$d^* \leq d^{**}. \tag{17}$$

1-Lipschitz constraint

Notice the constraint in (3). Actually this is a very well-known constraint in math and stats, called the 1-Lipschitz constraint. It comes from the definition of an 1-Lipschitz function. We say that a function $f(\cdot)$ is 1-Lip if

$$|f(x_1) - f(x_2)| \leq \|x_1 - x_2\| \quad \forall x_1, x_2. \tag{18}$$

Using this definition, one can therefore say that the function $\psi(\cdot)$ in (3) is 1-Lip. Applying this to (3), we can obtain a simpler expression for the optimization problem as:

$$\max_{\psi(\cdot):\ \text{1-Lip}} \sum_{z \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mathbb{Q}_Y(z)\psi(z) - \sum_{\hat{z} \in \mathcal{Y} \cup \hat{\mathcal{Y}}} \mathbb{Q}_{\hat{Y}}(\hat{z})\psi(\hat{z}). \tag{19}$$

Wasserstein GAN

Recall the definitions of $\mathbb{Q}_Y$ and $\mathbb{Q}_{\hat{Y}}$:

$$\begin{aligned} \mathbb{Q}_Y(z) &= \begin{cases} \frac{1}{m}, & \text{if } z \in \mathcal{Y}; \\ 0, & \text{if } z \in \hat{\mathcal{Y}} \setminus \mathcal{Y}; \end{cases} \\ \mathbb{Q}_{\hat{Y}}(\hat{z}) &= \begin{cases} \frac{1}{m}, & \text{if } \hat{z} \in \hat{\mathcal{Y}}; \\ 0, & \text{if } \hat{z} \in \mathcal{Y} \setminus \hat{\mathcal{Y}}. \end{cases} \end{aligned}$$

Also in the original optimization (1), we have the outer minimization over $G(\cdot)$. Taking all of these into consideration, we can translate the original Wasserstein-distance-based optimization into:

$$\min_{G(\cdot)} \max_{\psi(\cdot):\ 1\text{-Lip}} \frac{1}{m}\sum_{i=1}^{m}\psi(y^{(i)}) - \frac{1}{m}\sum_{i=1}^{m}\psi(\hat{y}^{(i)}). \tag{20}$$

Obviously this is a function optimization problem. Hence, as Goodfellow did, Bottou employed neural networks for $G(\cdot)$ and $\psi(\cdot)$ to approximate the optimization problem (20) as:

$$\min_{G(\cdot)\in\mathcal{N}} \max_{\psi(\cdot)\in\mathcal{N}:\ 1\text{-Lip}} \frac{1}{m}\sum_{i=1}^{m}\psi(y^{(i)}) - \frac{1}{m}\sum_{i=1}^{m}\psi(\hat{y}^{(i)}) \tag{21}$$

where $\mathcal{N}$ indicates a set of DNN-based functions. This is exactly the optimization problem for WGAN! It turns out the WGAN works pretty well, outperforming the original GAN by Goodfellow in many application scenarios. So as of now, it is the state of the art - many GAN variants that work best for some applications are based on the WGAN.

A fundamental question

Recall the generic divergence-based optimization problem:

$$\min_{G(\cdot)} \mathsf{D}(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}}).$$

Obviously the WGAN optimization (1) belongs to the above generic problem. Then, one very natural question that arises is: Is the Wasserstein distance the best choice for $\mathsf{D}(\cdot,\cdot)$? In other words,

$$\mathsf{D}^*(\cdot,\cdot) = W(\cdot,\cdot)? \tag{22}$$

A special case

In an effort to address this question, a few groups including my group have investigated a special setting in which the optimal generator G^* is known. The special setting is so called the *Gaussian linear-generator setting* wherein the data $\{y^{(i)}\}_{i=1}^m$ follows a Gaussian distribution, say:

$$y^{(i)} \sim \mathcal{N}(0, \mathbf{K}_Y) \quad \text{where } \mathbf{K}_Y = U\Lambda U^T, \tag{23}$$

and the generator is subject to a linear operation:

$$\hat{y}^{(i)} := G(x^{(i)}) = Gx^{(i)} \ \text{ where } G \in \mathbf{R}^{n\times k}. \tag{24}$$

Here one natural assumption that one can make on the distribution of $x^{(i)}$ is:

$$x^{(i)} \sim \mathcal{N}(0, \mathbf{I}), \tag{25}$$

as this way suggests that fake samples are also Gaussian, which coincides with the same type of distribution as that of real samples:

$$\hat{y}^{(i)} \sim \mathcal{N}(0, \mathbf{E}[(Gx^{(i)})(Gx^{(i)})^T]) = \mathcal{N}(0, GG^T). \tag{26}$$

Fortunately, under the above Gaussian setting, the optimal G^* is well-known. Here what it means by being optimal is in a sense of maximizing the likelihood of the data, as we adopted

while discussing on the optimality of a cross-entropy loss function in Lecture 19. It turns out the optimal G^* is the one that performs Principle Component Analysis (PCA):

$$\mathbb{E}[\hat{y}^{(i)}\hat{y}^{(i)T}] = G^*G^{*T} = U\mathsf{diag}(\lambda_1, \ldots, \lambda_k, 0, \ldots, 0)U^T \tag{27}$$

where $(\lambda_1, \ldots, \lambda_k)$ denote the k principal (largest) eigenvalues[1] of $\mathbf{K}_Y = U\Lambda U^T$. In a usual setting in which $k < n$, the PCA solution looks making sense. The rank of GG^T is limited by k, so it may not fully represent $\mathbf{K}_Y$ as the rank of $\mathbf{K}_Y$ can be n. In this case, what one can do for the best is to make GG^T as close as possible to K_Y. One such natural way is to take the k largest eigenvalues of $\mathbf{K}_Y$ to form a covariance matrix. It turns out it is the best way in a sense of maximizing the likelihood of the data. The proof of this will be explored in PS.

Now under the special Gaussian linear-generator setting, one can ask the fundamental question (22):

$$G^* = G^*_{\mathsf{WGAN}} = \arg\min_{G \in \mathbf{R}^{n\times k}} W(\mathbb{Q}_Y, \mathbb{Q}_{\hat{Y}})? \tag{28}$$

Interestingly it turns out the answer is yes! Actually the proof is not that short. So due to the interest of time, we will not prove it here; if you are interested, you can consult with me.

However, the answer holds under such special Gaussian setting. So you may wonder if that is the case also under general settings in which the data distribution is *arbitrary*. Unfortunately it has been unanswered - it is an open question. I believe this is one of the fundamental and intriguing questions in the context of the GAN-based framework. Someone may believe that the answer depends on what distribution of data we consider. This may be the case, but even this was not answered. So any progress on this will be interested.

Closing

Now let's conclude the course. In Part I, we investigated several instances of convex optimization problems, ranging from LP, Least-Squares, QP, SOCP, and all the way up to SDP. We studied how such problems are categorized, as well as how to formulate some real-world problems into such specialized problems via some translation techniques possibly aided by matrix-vector notations. For some certain settings including LP, unconstrained optimization and equality-constrained QP, we also studied how to solve the problems explicitly.

In Part II, we studied two important theorems regarding duality: (1) strong duality theorem; (2) weak duality theorem. With the strong duality theorem, we came up with a generic algorithm which provides detailed guidelines as to how to solve arbitrary convex optimization problems: the interior point method. With the weak duality theorem, we investigated a certain yet powerful method, called Lagrange relaxation, which can provide reasonably-good approximation solutions for a variety of *non-convex* problems.

In Part III, we explored one trending application where optimization tools that we learned play central roles: Machine learning. In particular, we studied two certain yet popular methodologies of machine learning: (1) supervised learning; (2) unsupervised learning. For supervised learning, we put an emphasis on deep learning, which is based on deep neural network architectures which received significant attention recently. We saw that the optimization tools and concepts that we learned are instrumental particularly to choosing objective functions as well as gaining algorithmic insights. As for unsupervised learning, we investigated the most fundamental learning method, called generative models, and then studied one specific yet powerful framework for

[1]Here one may ask how to compute such principal eigenvalues when $\mathbf{K}_Y$ is unknown, which is the case in reality. In this case, the optimal way is to compute an *empirical* covariance matrix $S := \frac{1}{m}\sum_{i=1}^m y^{(i)}y^{(i)T}$ and then to take the k largest eigenvalues of S.

such generative models, named GANs. In this context, we observed that the duality theorems play a crucial role in enabling a practical implementation for the state-of-the-art GAN, which is WGAN.

It is no doubt that tools for convex optimization are very powerful - the usefulness has already been proved by many researchers working on a wide variety of fields. While this course put an emphasis on a particular application (machine learning), it is shown to have much broader applicability. So I hope you would find all of these useful in your own research field.