L14_2017W

October 1, 2017

1 Introduction to Optimization

1.0.1 CPSC 340: Machine Learning and Data Mining

1.1 Optimization: introduction

definition

- An "optimization problem" refers to maximizing or minimizing a function.
- There are many, many types of optimization problems.

$$\min_x (x-a)^2 + b$$

Interpretation: minimize the function $f(x) = (x-a)^2 + b$ with respect to x and return the minimum value x. So in this case

$$f^\star = \min_x (x-a)^2 + b$$

minimizer vs. minimum

- Sometimes you are also interested in the **minimizer** rather than the **minimum** (plural **minima**). This is written as

$$x^\star = \arg\min_x (x-a)^2 + b$$

maximization vs. minimization The same goes for **maximizer** and **maximum** and $f^* = \max f(x)$ and $x^* = \arg\max f(x)$.

- The thing being maximized/minimized is called the **objective function**. Sometimes (primarily when minimizing) it's also called the **loss function** or **cost function**.
- Sometimes the result is not defined:

$$\max_x (x-a)^2 + b$$

is ∞. And

$$\arg\max_x (x-a)^2 + b$$

is undefined.

dimensionalities

- The function you are optimizing should be a scalar-valued function, but the inputs need not be scalar-valued. For example:

$$\min_{\mathbf{x}} ||\mathbf{x}||_2^2$$

is a reasonable thing to write.

Continuous vs. discrete

- Some optimization problems are **continuous** and others are **discrete**. In CPSC 340 we will only consider continuous optimization. But you probably saw discrete optimization in other courses (e.g., dynamic programming, Dijkstra's algorithm for shortest path in a graph). Also, fitting a decision tree can be thought of as discrete optimization, where your decision variables are the features to split on and the split thresholds.
- Mixed problems also exist, which contain both continuous and discrete variables. We will touch on this later in CPSC 340 with L0-regularization.

Domains

- An optimization problem may come with a domain. In the examples above, we've assumed the domain $x \in \mathbb{R}$. But sometimes we restrict ourselves. For example $x \geq 0$, etc.
- These are sometimes also called "bounds". Like $0 \leq x \leq 1$ means x is bounded below by 0 and bounded above by 1.
- Note: you'll always see things like $x \geq 0$ and never $x > 0$. Why? Consider this

$$\min_x x \quad \text{s.t.} \quad x > 0$$

What's the solution?

- BTW "s.t." means "such that". Or maybe "subject to". Depending on your mood.

Constraints: introduction

- An optimization problem may come with one or more **constraints** (but, typically, only one objective! If you've seen pareto fronts in economics class, that's a multi-objective situation).
- Example:

$$\min_x (x-a)^2 + b \quad \text{s.t.} \quad \sin(x) \leq 0.5$$

- If you want, you can think of bounds/domains as a special case of constraints.
- The above is called an **inequality constraint**. Another type of constraint an an **equality constraint**. For example:

$$\min_x (x-a)^2 + b \quad \text{s.t.} \quad \sin(x) = 0.5$$

These are very different beasts!!!! Roughly speaking,

- **equality** constraints tend to reduce the dimensionality of the search space (restricting us to a subspace)
- **inequality** constraints typically do not reduce the dimensionality but just cut the space "in half"

Inequality constraints show up in CPSC 340. For example in non-negative matrix factorization (NMF). They are also a way of thinking about regularization.

Constraints: feasibility

- Not all constraints change the solution. For example, the constraint $\sin(x) \leq 5$ is a vacuous constraint... it is true for any $x \in \mathbb{R}$.
- Some constraints cannot be satisfied. In this case we say the problem is **infeasible**. For example $\sin(x) \geq 5$.
- Sometimes a single constraint on its own isn't enough to make a problem infeasible, but their combination does so. For example having two constraints $x \geq 5$ and $x \leq 3$.

Constraints: active vs. not active

- Constraints may or may not be **active** at the solution. This is different from whether or not that are satisfiable (above). Example:

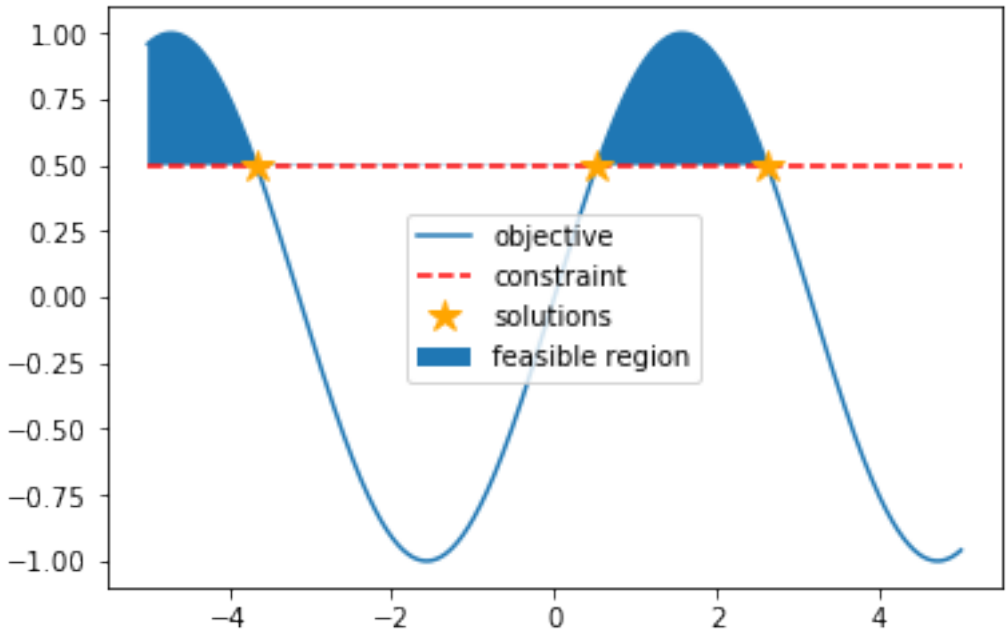
$$\min_x \; x^2$$
$$\text{s.t.}$$
$$|x| \geq 1$$
$$x^4 \leq 3$$

- the unconstrained minimizer is $x = 0$. The constrained minimizer is $x = 1$. So we say the first constraint is active at the solution. However, the second constraint is not active; if you removed it, the solution would be unchanged.
- in my mind, an active constraint is like a wall that I run into. I'd like to go further down the hill but I can't.
- key point: the objective function determines which of the constraints are active.

Example: $\min_x \; \sin(x)$ s.t. $\sin(x) \geq 0.5$

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline

        x = np.linspace(-5,5,1000)
        y = np.sin(x)
        plt.plot(x, y, label="objective")
        plt.plot(x, 0*x+0.5, color="r", linestyle="--", label="constraint")
        plt.fill_between(x[y>=0.5], 0.5, y[y>=0.5], label="feasible region")
        plt.plot([-7*np.pi/6, np.pi/6, 5*np.pi/6], 0.5+np.zeros(3), '*', color="orange", markers
        plt.legend()
        plt.show()
```



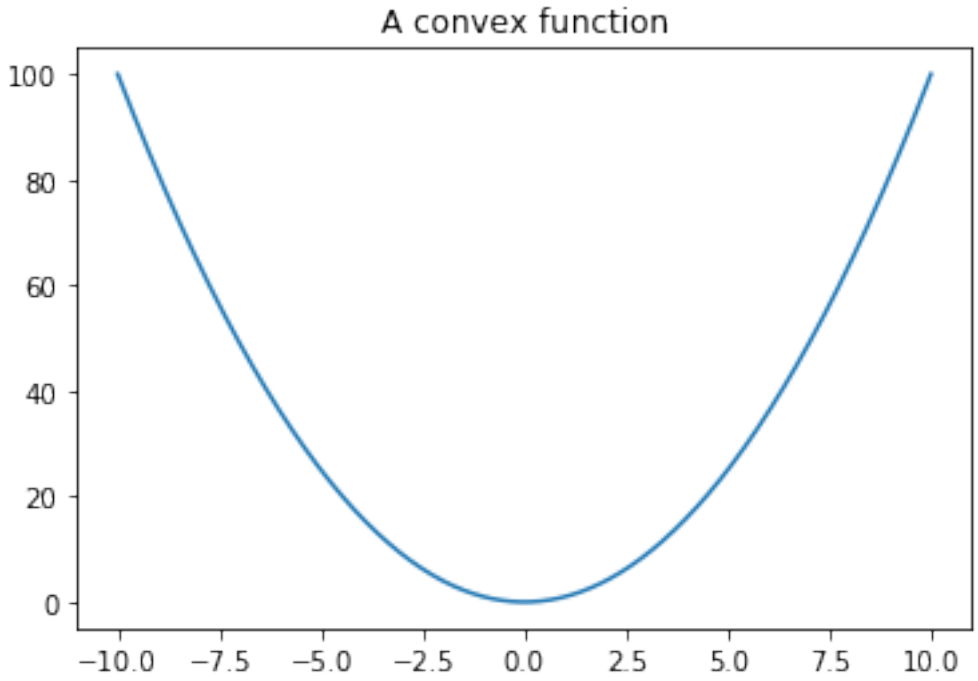
Convex functions

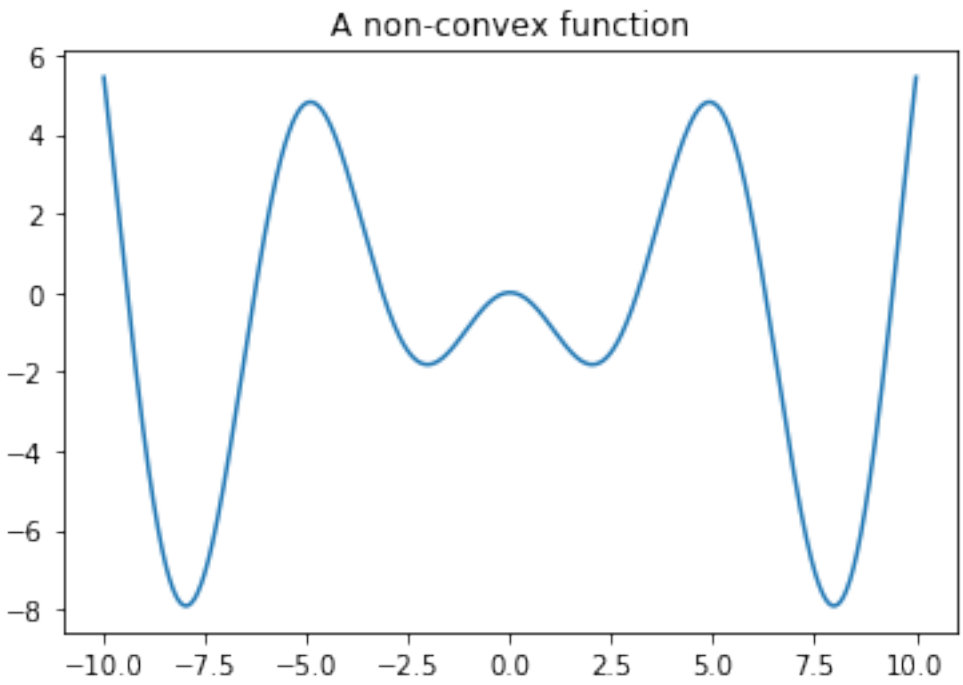
- There is a mathematical definition. You can read about it here.
- For our purposes, the key is for convex functions is that a local minimum is also a global minimum.
- (optional note) convex --> all local minima are global minima; but non-convex -/-> there exists local minima that are not global minima

```
In [2]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline
```

```
x = np.linspace(-10,10,1000)
plt.plot(x,x**2)
plt.title("A convex function")
plt.show()

plt.figure()
plt.plot(x, -x*np.sin(x))
plt.title("A non-convex function")
plt.show()
```





why do we care about convexity

- from a machine learning point of view, optimization is often the hard part and very often the time-consuming part.
- some methods don't involve minimization of a loss function, like kNN
- (although even there you need to find the nearest neighbours during prediction which is...)

How do we solve (continuous) optimization problems?

- From calculus we were told that to find the min/max of a function, take the derivative and set to 0
- So what's the big deal?
- The big deal is that "derivative=0" is an equation **we cannot solve most equations**!!!
- (I feel like high school gives the wrong impression about this...)

Here's one:

$$\sin(\cos(x)) + 999 \log x = \sqrt{x}$$

I just made that up and it might not make any sense, but the point is for it to look weird. Can we solve this? No. We cannot solve equations in closed form, in general. This is true for one variable ($x \in \mathbb{R}$) and more than one variable ($x \in \mathbb{R}^d$, $d > 1$). So we solve equations with *iterative methods*. (Remember Newton's method?) See **CPSC 302** for a lot more on this.

There's an entire field of study related to different such methods. Different methods work well for different classes of functions. And even "works well" needs to be defined carefully, but roughly it means "gets to a/the solution quickly".

So, yes, there is a mapping from optimization to solving equations, but that doesn't mean we're done because

1. solving equations is hard
2. even if we find points where the derivative (gradient) is zero, if the loss is non-convex we need to worry about local minima, saddle points (maxima are less of a problem)

What makes an optimization problem "hard"? This is a very difficult question. I can't necessarily do it justice but I will try.

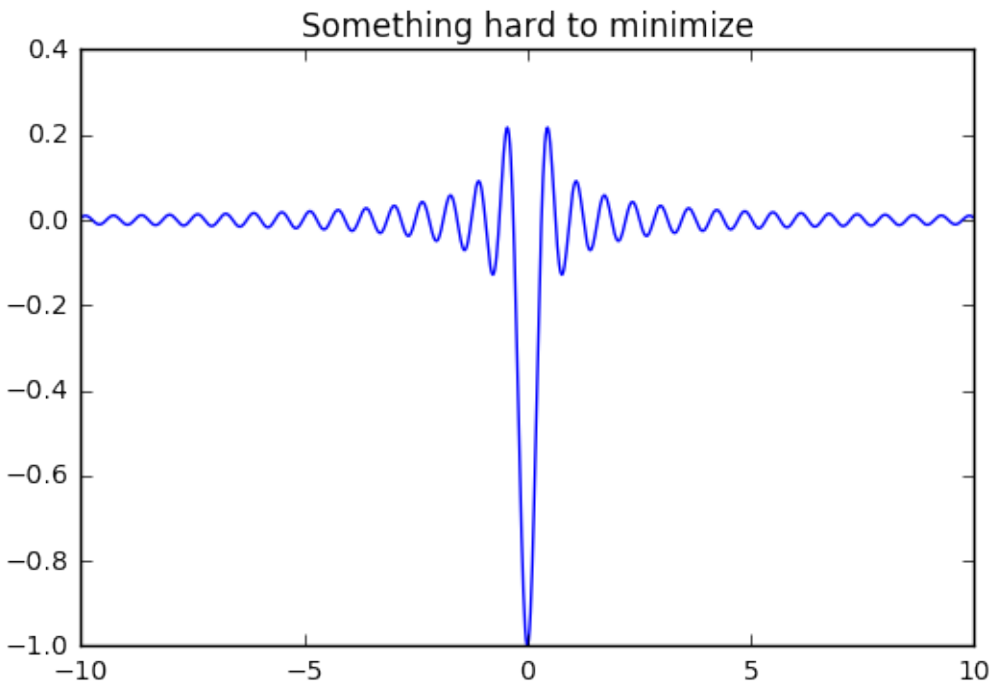
VERY roughly speaking, in ML we have, from least difficult to most difficult:

1. Problems that can be solved in closed form (OLS... and not much else)
2. Problems that are convex (e.g. logistic regression, SVM -- coming later in this course)
3. Problems that are not convex and have local minima (e.g. neural networks -- coming later in this course)

More reasons why optimization is hard...

- Discrete optimization tends to be even harder than continuous
- discrete optimization with N binary variables $x_i \in \{0,1\}$, $i = 1,\ldots,N$ often takes $\mathcal{O}(2^N)$ time in the worst case (or something bad like that). In other words, the worst case behaves like you have to try out all possibilities.
- If you are minimizing a non-convex continuous function, you may get "stuck" in a local minimum. **global optimization** refers to the (very ambitious) task of finding a global optimum. Sometimes we settle for just a **local** search.
- Sometimes the function you are evaluating is "expensive"; the most common expense is time. If $f(\mathbf{x})$ takes an hour to evaluate for a given $\mathbf{x}$ then your life is a lot harder. This comes up in hyperparameter optimization.
- Availability of "extra information" helps. Most notably the derivative but also 2nd derivatives, smoothness properties, etc.
- Understanding problem **structure** can help you solve the problem. For example if the objective and constraints are linear, we can solve the problem much faster than if we don't know anything.
- If you're really unlucky, you may only be able to perform *noisy* evaluations of your objective. This comes up in hyperparameter optimization.
- Higher dimensional problems tend to be harder. The "volume" to explore grows exponentially in d.
- Smoothness makes things easier. More local minima makes it harder. Consider the function below. This is not fun. Now imagine this in 100 dimensions with the spike hidden somewhere in 100-dimensional space. Yikes.

```
In [9]: plt.figure()
        plt.plot(x, -np.sin(10*x)/(10*x))
        plt.title("Something hard to minimize")
        plt.show()
```



In [ ]: