# Chapter 7

## Optimization basics

### Contents (class version)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.0 Introduction (s,intro)</td>
<td>7.1</td>
</tr>
<tr>
<td>7.1 Convex Optimization (s,convex,def)</td>
<td>7.3</td>
</tr>
<tr>
<td>Properties of Convex Functions (s,convex,prop)</td>
<td>7.4</td>
</tr>
<tr>
<td>Gradient Descent (s,gd)</td>
<td>7.6</td>
</tr>
<tr>
<td>Stochastic Gradient Descent (s,sgd)</td>
<td>7.7</td>
</tr>
<tr>
<td>7.2 Empirical Risk Minimization (s,erm)</td>
<td>7.10</td>
</tr>
<tr>
<td>Loss Functions</td>
<td>7.10</td>
</tr>
<tr>
<td>Empirical Risk Minimization</td>
<td>7.12</td>
</tr>
<tr>
<td>Surrogate Losses</td>
<td>7.13</td>
</tr>
</tbody>
</table>

---

7.0 Introduction
Many of the previous topics have involved optimization formulations: LS, low-rank approximation, PCA, multidimensional scaling. In all these cases we derived analytical solutions, like the pseudo-inverse for minimum-norm LS problems and the truncated SVD for low-rank approximation.

But often we need iterative optimization algorithms, e.g.,

- if no closed-form minimizer exists, or
- if the analytical solution requires too much computation and/or memory, e.g., SVD for large problems.

To solve a problem like $\hat{x} = \arg\min_x f(x)$ via an iterative method, we start with some initial guess $x_0$, and then the algorithm produces a sequence $\{x_k\}$ where hopefully the sequence converges to $\hat{x}$, meaning $\|x_k - \hat{x}\| \to 0$ for some norm $\|\cdot\|$ (typically the 2-norm) as $k \to \infty$.

The homework has introduced one such optimization algorithm (gradient descent). In the remainder of the course, we will focus on gradient descent and its variants, as well as two other optimization approaches: Alternating Direction Method of Multipliers (ADMM) and Iteratively Reweighted Least Squares (IRLS).
Throughout the course, we’ve thrown around the term convex regarding functions and sets. In this section, we’ll formally define what that means. Consider first the following unconstrained optimization problem

\[ \min_{x \in \mathbb{R}^D} f(x) \]  

- \( f : \mathbb{R}^D \to \mathbb{R} \) is called the objective function or cost function
- the point \( x^* \in \mathbb{R}^D \) is called a local minimizer if there exists an \( r > 0 \) such that \( f(x^*) \leq f(x) \) for all \( x \) such that \( \|x - x^*\| < r \)
- the point \( x^* \in \mathbb{R}^D \) is called a global minimizer if \( f(x^*) \leq f(x) \) for all \( x \in \mathbb{R}^D \)

Define. A function \( f : \mathbb{R}^D \to \mathbb{R} \) is convex if

\[ f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2) \]  

for all \( t \in [0,1] \). Further, \( f \) is strictly convex if

\[ f(tx_1 + (1-t)x_2) < tf(x_1) + (1-t)f(x_2) \]  

for all \( t \in (0,1) \).
Properties of Convex Functions

The intuitive definition of convexity is that any line segment connecting two points \( f(x) \) and \( f(y) \) lies above the graph of \( f \). If \( f \) is convex, then the problem of minimizing \( f \) becomes easier to solve. Below are some useful properties.

- If \( f \) is convex, then every local minimizer is a global minimizer.
- If \( f \) is strictly convex, then \( f \) has at most one global minimizer.

A few more properties of interest require the use of gradients and the Hessian.

Define. Given a function \( f : \mathbb{R}^D \to \mathbb{R} \), the gradient and Hessian of \( f \) at \( x \in \mathbb{R}^D \) are defined as

We say that \( f \) is differentiable if \( \nabla f(x) \) exists for all \( x \in \mathbb{R}^D \) and twice differentiable if \( \nabla^2 f(x) \) exists for all \( x \in \mathbb{R}^D \). These definitions allow us to talk about a few more properties of convex functions.
• If $f$ is differentiable and $x^*$ is a local minimizer of $f$, then $\nabla f(x^*) = 0$.
• Suppose $f : \mathbb{R}^D \to \mathbb{R}$ is differentiable. Then $f$ is convex if and only if

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle$$

• If $f$ is convex and differentiable, then $x^*$ is a global minimizer if and only if $\nabla f(x^*) = 0$.
• Let $f$ be twice continuously differentiable. Then $f$ is convex if and only if $\nabla^2 f(x) \succeq 0$. Similarly, $f$ is strictly convex if $\nabla^2 f(x) \succ 0$.

Given the above properties, we know when we have found a global minimizer of a convex function. In the remainder of the lecture, we’ll discuss methods for finding a global minimizer based only on the gradient.
Gradient Descent

As you saw in the homework, gradient descent (GD) is a method for solving optimization problems that may be more computationally efficient than direct solutions. More often, we are not able to obtain a closed-form analytical solution, so we rely on numerical methods to solve problems. Recall that GD follows the iteration

$$x_{k+1} = x_k - \mu \nabla f(x_k).$$

(7.4)

The choice of $\mu$ is problem dependent. On the homework, we saw that for the linear least-squares problem, convergence is guaranteed by choosing $\mu$ smaller than the maximum singular value of the data matrix $A$.

Instead of using GD with a fixed step size $\alpha$, an alternative is to do a line search to find the best step size at each iteration. This variation is called steepest descent (or GD with a line search). Here is how steepest descent works:

$$d_k = -\nabla f(x_k)$$  

search direction

$$\alpha_k = \arg\min_{\alpha \in \mathbb{R}} f(x_k + \alpha d_k)$$  

step size

$$x_{k+1} = x_k + \alpha_k d_k$$  

update.

• By construction, this iteration is guaranteed to decrease the cost function monotonically: $f(x_{k+1}) \leq f(x_k)$ with strict decrease unless $x_k$ is already a minimizer.

• Computing $\alpha_k$ takes some extra work, especially for non-quadratic problems. As a result, adaptive and momentum-based methods are preferred. [3].
Stochastic Gradient Descent

Gradient descent is a first-order method because it only considers the first derivative of $f$. Other methods take advantage of the Hessian and are known as second-order methods. While these converge faster than first-order methods, they are computationally expensive to the point of being prohibitive for many problems. For very large problem sizes, even computing the full derivative is computationally prohibitive. For example, for the least-squares problem with $A \in \mathbb{R}^{D \times N}$, the complexity is $O(DN)$ per iteration.

To reduce computational complexity (and yield faster convergence rates), we can use an algorithm called stochastic gradient descent (SGD). SGD is applicable to cost functions that can be written as a sum of differentiable functions, i.e.,

$$f(x) = \sum_{i=1}^{N} f_i(x),$$

(7.5)

where $f_i(x)$ is differentiable. To reduce computational complexity, SGD “approximates” the gradient by taking the derivative with respect to only one of these $f_i$’s. This yields the following update

$$x_{k+1} = x_k - \mu \nabla f_i(x_k).$$

(7.6)
Example. Let’s return to the linear least-squares problem of minimizing $\frac{1}{2} \| Xw - y \|^2_2$ (note the differing variable names). As you saw in the homework, we often form $X$ by letting the rows of $X$ be the measurement locations $x_1, \ldots, x_N \in \mathbb{R}^D$ (features in machine learning) corresponding to the measurements in $y \in \mathbb{R}^N$ (labels). In this light, we can rewrite the least-squares residual as

$$Xw - y = \begin{bmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_N^T \end{bmatrix} w - \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} x_1^T w - y_1 \\ x_2^T w - y_2 \\ \vdots \\ x_N^T w - y_N \end{bmatrix}.$$ 

Now by the definition of the (vector) $\ell_2$-norm

$$\frac{1}{2} \| Xw - y \|^2_2 = \frac{1}{2} \sum_{i=1}^N (x_i^T w - y_i)^2 = \frac{1}{2} \sum_{i=1}^N f_i(w).$$

Differentiating with respect to $w$ yields

\[ (7.7) \]
Notes on SGD:
• You should cycle through $i$ in random order. If convergence is not achieved after one pass through the data, it is common to re-shuffle the data and make another pass.
• SGD is not a pure descent method in the sense that $\|x_k - \hat{x}_{LS}\|_2$ does not decrease monotonically.
• Practical implementations of SGD use an adaptive learning rate, similar to Nesterov-accelerated gradient descent.
• For a nice overview of GD, see this article.
7.2 Empirical Risk Minimization

Recall the supervised learning setup given in the first lecture. Assume we are given a set of feature vectors \( x_1, \ldots, x_N \in \mathcal{X} \) and their labels \( y_1, \ldots, y_N \in \mathcal{Y} \). We call \( \mathcal{X} \) the feature space (typically \( \mathcal{X} = \mathbb{R}^D \)) and \( \mathcal{Y} \) the label space. Typically, we have one of two scenarios:

\[
\begin{align*}
  y_i & \in \mathbb{R} \quad \text{(regression)} \quad (7.8) \\
  y_i & \in \{1, \ldots, K\} \quad \text{(classification)} \quad (7.9)
\end{align*}
\]

We call the \((x_i, y_i)\) pairs the training data, and our goal is to learn some function \( f : \mathcal{X} \to \mathcal{Y} \) that can be used to make predictions on new data.

Loss Functions

A general formulation for deriving machine learning algorithms is through minimizing the risk of a given optimization function. In the probabilistic setup for machine learning, we assume the feature-label pairs are random variables \((X, Y)\) that are jointly distributed according to some unknown distribution, i.e.,

\[
(X, Y) \sim P_{XY}.
\]
Define. A **loss function** (or simply loss) is a function $L(y, t)$, where $t \in \mathbb{R}$ and $y \in \mathcal{Y}$. Let $f : \mathcal{X} \to \mathcal{Y}$ be a function (e.g., a hypothesis classifier). The **L-risk** of $f$ is the expected (average) loss over all $(X, Y)$ pairs drawn according to $P_{XY}$, i.e.,

$$R_L(f) = \mathbb{E}[L(Y, f(X))] \tag{7.10}$$

**Example.** If $f$ is a regression function, two common losses are

1. **Squared-error loss**: $L(y, t) = (y - t)^2$ (7.11)
2. **Absolute deviation loss**: $L(y, t) = |y - t|$ (7.12)

**Example.** If $f$ is a binary classification function, the most natural loss is

$$L(y, t) = 1\{y \neq \text{sign}(t)\}$$ (0-1 loss) (7.13)

where $1\{S\}$ is the **indicator function**, which takes the value 1 on the set $S$ and 0 everywhere else.
Empirical Risk Minimization

The above definitions of risk and loss lead to a natural method for obtaining the learning rule $f$, referred to as **empirical risk minimization** (ERM). Given a set of training data $\{(x_i, y_i)\}_{i=1}^{N}$, ERM seeks to solve

$$
\min_{f \in \mathcal{F}} \sum_{i=1}^{N} L(y_i, f(x_i)) + \lambda R(f),
$$

(7.14)

where $\mathcal{F}$ denotes a set of possible functions to minimize over (e.g., linear functions), and $R(\cdot)$ is a **regularizer**, which is used to steer the solution toward functions of a certain structure (e.g., small 2-norm or sparse).

Define. The quantity

$$
\hat{R}_L(f) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(x_i)).
$$

(7.15)

is called the **empirical $L$-risk** or simply **empirical risk** of $f$.

ERM is useful in that it allows us to easily derive learning rules using convex optimization, especially first-order methods like SGD.
Fact. Let $f$ be an affine function parameterized by $\mathbf{w} \in \mathbb{R}^D$ and $b \in \mathbb{R}$. If $L(y, t)$ is a convex function of $t$ for every $y$, then

$$\hat{R}L(w, b) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \mathbf{w}^T \mathbf{x}_i + b)$$

(7.16)

is a convex function of $\bar{\mathbf{w}} = [\mathbf{w} \ b]^T$. Therefore, the ERM problem (7.14) can be solved via convex optimization.

We implicitly used this fact when solving the LS problem on the homework via gradient descent. Unfortunately, the same does not hold for the binary classification problem, since the 0-1 loss is neither convex nor differentiable.

---

**Surrogate Losses**

Since the 0-1 loss is incompatible with the ERM framework, one approach is to design loss functions to approximate this loss but that are convex. This idea also shows up in **sparse regression** a bit later in the course, where we will discuss **convex relaxations**. In the case of ERM, such loss functions are referred to as **surrogate losses**.
For binary classification, the most widely used surrogate losses are given below.

\[
L(y, t) = \log(1 + e^{-yt}) \quad \text{(logistic loss)} \tag{7.17}
\]

\[
L(y, t) = \max(0, 1 - yt) \quad \text{(hinge loss)} \tag{7.18}
\]

\[
L(y, t) = -\bar{y} \log(t) - (1 - \bar{y}) \log(1 - t) \quad \text{(cross entropy/log loss)} \tag{7.19}
\]

where \( \bar{y} = (1 + y)/2 \).

The logistic loss leads to the popular **logistic regression** algorithm, the hinge loss is used in both the **support vector machine (SVM)** and the **perceptron learning algorithm**, and the cross entropy is ubiquitous in deep learning. The ERM framework allows us to easily compare these algorithms, which you will do on the next few homework assignments.
Figure 7.1: Comparison of surrogate losses for binary classification.

Bibliography


