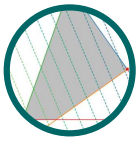
7

Continuous Optimization



Since machine learning algorithms are implemented on a computer, the mathematical formulations are expressed as numerical optimization methods. This chapter describes the basic numerical methods for training machine learning models. Training a machine learning model often boils down to finding a good set of parameters. The notion of "good" is determined by the objective function or the probabilistic model, which we will see examples of in the second part of this book. Given an objective function, finding the best value is done using optimization algorithms.

Since we consider data and models in $\mathbb{R}^D$, the optimization problems we face are *continuous* optimization problems, as opposed to *combinatorial* optimization problems for discrete variables.

This chapter covers two main branches of continuous optimization (Figure 7.1): unconstrained and constrained optimization. We will assume in this chapter that our objective function is differentiable (see Chapter 5), hence we have access to a gradient at each location in the space to help us find the optimum value. By convention, most objective functions in machine learning are intended to be minimized, that is, the best value is the minimum value. Intuitively finding the best value is like finding the valleys of the objective function, and the gradients point us uphill. The idea is to move downhill (opposite to the gradient) and hope to find the deepest point. For unconstrained optimization, this is the only concept we need, but there are several design choices, which we discuss in Section 7.1. For constrained optimization, we need to introduce other concepts to manage the constraints (Section 7.2). We will also introduce a special class of problems (convex optimization problems in Section 7.3) where we can make statements about reaching the global optimum.

global minimum

Consider the function in Figure 7.2. The function has a *global minimum* around $x = -4.5$, with a function value of approximately -47. Since the function is "smooth," the gradients can be used to help find the minimum by indicating whether we should take a step to the right or left. This assumes that we are in the correct bowl, as there exists another *local minimum* around $x = 0.7$. Recall that we can solve for all the stationary points of a function by calculating its derivative and setting it to zero. For

local minimum

Stationary points are the real roots of the derivative, that is, points that have zero gradient.

$$\ell(x) = x^4 + 7x^3 + 5x^2 - 17x + 3\,, \tag{7.1}$$

we obtain the corresponding gradient as

$$\frac{\mathrm{d}\ell(x)}{\mathrm{d}x} = 4x^3 + 21x^2 + 10x - 17\,. \tag{7.2}$$

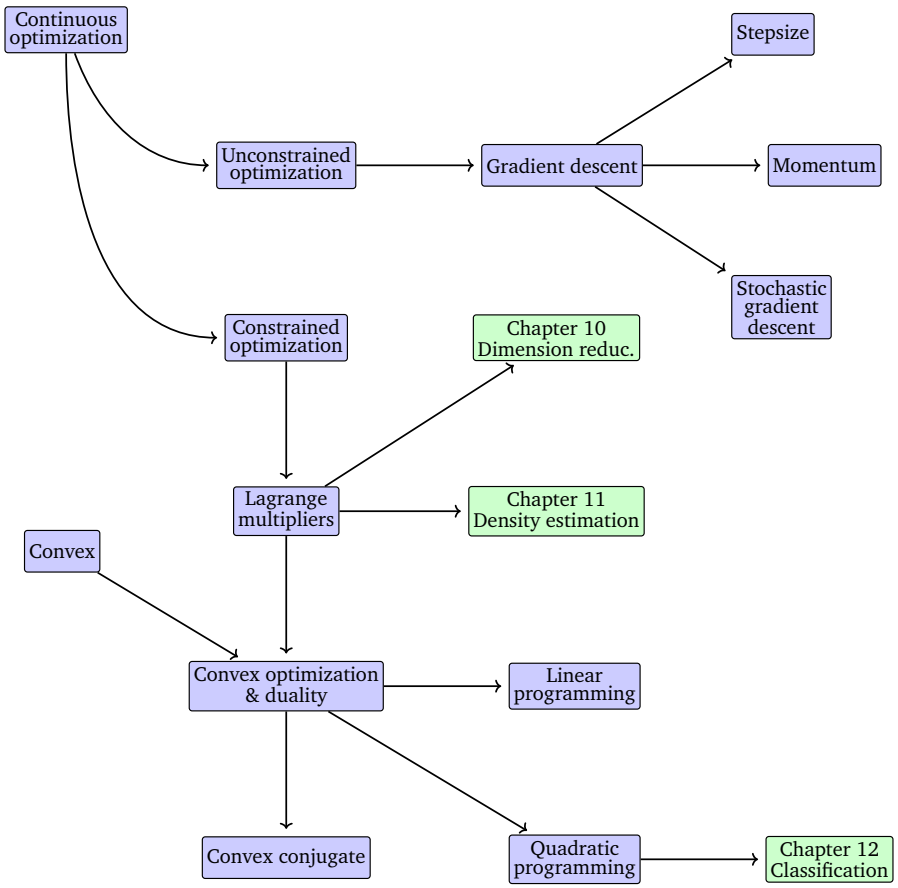


Figure 7.1 A mind map of the concepts related to optimization, as presented in this chapter. There are two main ideas: gradient descent and convex optimization.

Since this is a cubic equation, it has in general three solutions when set to zero. In the example, two of them are minimums and one is a maximum (around $x = -1.4$). To check whether a stationary point is a minimum or maximum, we need to take the derivative a second time and check whether the second derivative is positive or negative at the stationary point. In our case, the second derivative is

$$\frac{\mathrm{d}^2\ell(x)}{\mathrm{d}x^2} = 12x^2 + 42x + 10\,. \tag{7.3}$$

By substituting our visually estimated values of $x = -4.5, -1.4, 0.7$, we will observe that as expected the middle point is a maximum $\left(\frac{\mathrm{d}^2\ell(x)}{\mathrm{d}x^2} < 0\right)$ and the other two stationary points are minimums.

Note that we have avoided analytically solving for values of x in the previous discussion, although for low-order polynomials such as the preceding we could do so. In general, we are unable to find analytic solutions, and hence we need to start at some value, say $x_0 = -6$, and follow the negative gradient. The negative gradient indicates that we should go

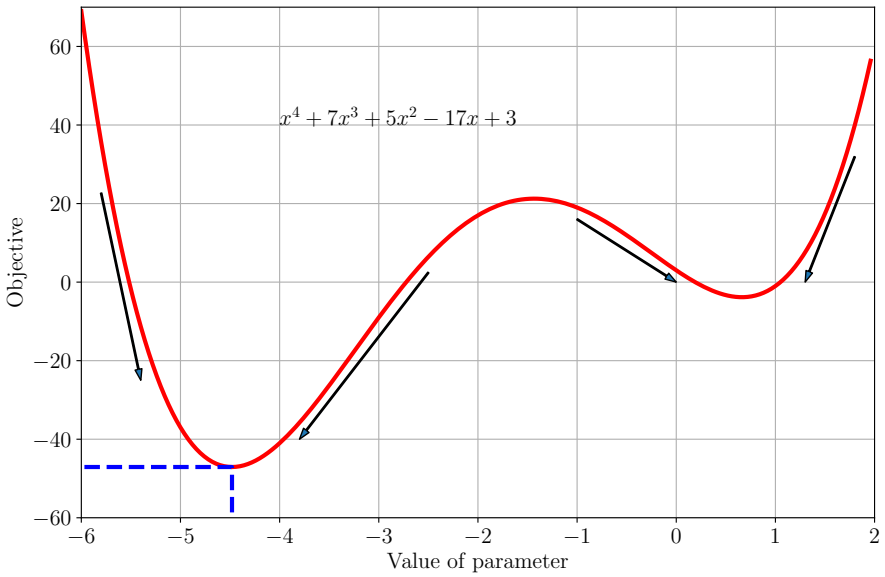


Figure 7.2 Example objective function. Negative gradients are indicated by arrows, and the global minimum is indicated by the dashed blue line.

right, but not how far (this is called the step-size). Furthermore, if we had started at the right side (e.g., $x_0 = 0$) the negative gradient would have led us to the wrong minimum. Figure 7.2 illustrates the fact that for $x > -1$, the negative gradient points toward the minimum on the right of the figure, which has a larger objective value.

According to the Abel–Ruffini theorem, there is in general no algebraic solution for polynomials of degree 5 or more (Abel, 1826).

In Section 7.3, we will learn about a class of functions, called convex functions, that do not exhibit this tricky dependency on the starting point of the optimization algorithm. For convex functions, all local minimums are global minimum. It turns out that many machine learning objective functions are designed such that they are convex, and we will see an example in Chapter 12.

For convex functions all local minima are global minimum.

The discussion in this chapter so far was about a one-dimensional function, where we are able to visualize the ideas of gradients, descent directions, and optimal values. In the rest of this chapter we develop the same ideas in high dimensions. Unfortunately, we can only visualize the concepts in one dimension, but some concepts do not generalize directly to higher dimensions, therefore some care needs to be taken when reading.

7.1 Optimization Using Gradient Descent

We now consider the problem of solving for the minimum of a real-valued function

$$\min_{\boldsymbol{x}} f(\boldsymbol{x})\,, \tag{7.4}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is an objective function that captures the machine learning problem at hand. We assume that our function f is differentiable, and we are unable to analytically find a solution in closed form.

Gradient descent is a first-order optimization algorithm. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient of the function at the current point. Recall from Section 5.1 that the gradient points in the direction of the steepest ascent. Another useful intuition is to consider the set of lines where the function is at a certain value ($f(\boldsymbol{x}) = c$ for some value $c \in \mathbb{R}$), which are known as the contour lines. The gradient points in a direction that is orthogonal to the contour lines of the function we wish to optimize.

We use the convention of row vectors for gradients.

Let us consider multivariate functions. Imagine a surface (described by the function $f(\boldsymbol{x})$) with a ball starting at a particular location $\boldsymbol{x}_0$. When the ball is released, it will move downhill in the direction of steepest descent. Gradient descent exploits the fact that $f(\boldsymbol{x}_0)$ decreases fastest if one moves from $\boldsymbol{x}_0$ in the direction of the negative gradient $-((\nabla f)(\boldsymbol{x}_0))^\top$ of f at $\boldsymbol{x}_0$. We assume in this book that the functions are differentiable, and refer the reader to more general settings in Section 7.4. Then, if

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 - \gamma((\nabla f)(\boldsymbol{x}_0))^\top \tag{7.5}$$

for a small *step-size* $\gamma \geqslant 0$, then $f(\boldsymbol{x}_1) \leqslant f(\boldsymbol{x}_0)$. Note that we use the transpose for the gradient since otherwise the dimensions will not work out.

This observation allows us to define a simple gradient descent algorithm: If we want to find a local optimum $f(\boldsymbol{x}_*)$ of a function $f : \mathbb{R}^n \to \mathbb{R},\ \boldsymbol{x} \mapsto f(\boldsymbol{x})$, we start with an initial guess $\boldsymbol{x}_0$ of the parameters we wish to optimize and then iterate according to

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \gamma_i((\nabla f)(\boldsymbol{x}_i))^\top . \tag{7.6}$$

For suitable step-size γ_i, the sequence $f(\boldsymbol{x}_0) \geqslant f(\boldsymbol{x}_1) \geqslant \ldots$ converges to a local minimum.

Example 7.1

Consider a quadratic function in two dimensions

$$f\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \frac{1}{2}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^\top \begin{bmatrix} 2 & 1 \\ 1 & 20 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 5 \\ 3 \end{bmatrix}^\top \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{7.7}$$

with gradient

$$\nabla f\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^\top \begin{bmatrix} 2 & 1 \\ 1 & 20 \end{bmatrix} - \begin{bmatrix} 5 \\ 3 \end{bmatrix}^\top . \tag{7.8}$$

Starting at the initial location $\boldsymbol{x}_0 = [-3, -1]^\top$, we iteratively apply (7.6) to obtain a sequence of estimates that converge to the minimum value

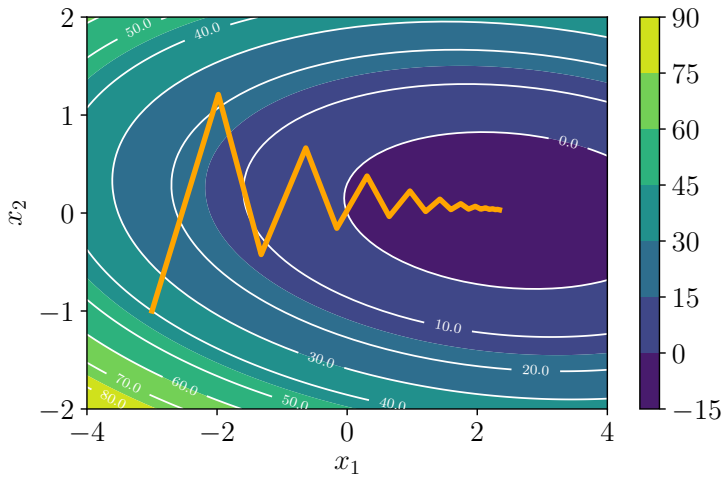


Figure 7.3 Gradient descent on a two-dimensional quadratic surface (shown as a heatmap). See Example 7.1 for a description.

(illustrated in Figure 7.3). We can see (both from the figure and by plugging $\boldsymbol{x}_0$ into (7.8) with $\gamma = 0.085$) that the negative gradient at $\boldsymbol{x}_0$ points north and east, leading to $\boldsymbol{x}_1 = [-1.98, 1.21]^\top$. Repeating that argument gives us $\boldsymbol{x}_2 = [-1.32, -0.42]^\top$, and so on.

Remark. Gradient descent can be relatively slow close to the minimum: Its asymptotic rate of convergence is inferior to many other methods. Using the ball rolling down the hill analogy, when the surface is a long, thin valley, the problem is poorly conditioned (Trefethen and Bau III, 1997). For poorly conditioned convex problems, gradient descent increasingly "zigzags" as the gradients point nearly orthogonally to the shortest direction to a minimum point; see Figure 7.3. ♢

7.1.1 Step-size

The step-size is also called the learning rate.

As mentioned earlier, choosing a good step-size is important in gradient descent. If the step-size is too small, gradient descent can be slow. If the step-size is chosen too large, gradient descent can overshoot, fail to converge, or even diverge. We will discuss the use of momentum in the next section. It is a method that smoothes out erratic behavior of gradient updates and dampens oscillations.

Adaptive gradient methods rescale the step-size at each iteration, depending on local properties of the function. There are two simple heuristics (Toussaint, 2012):

- When the function value increases after a gradient step, the step-size was too large. Undo the step and decrease the step-size.
- When the function value decreases the step could have been larger. Try to increase the step-size.

Although the "undo" step seems to be a waste of resources, using this heuristic guarantees monotonic convergence.

Example 7.2 (Solving a Linear Equation System)
When we solve linear equations of the form $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, in practice we solve $\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = \boldsymbol{0}$ approximately by finding $\boldsymbol{x}_*$ that minimizes the squared error

$$\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2 = (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})^\top (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \tag{7.9}$$

if we use the Euclidean norm. The gradient of (7.9) with respect to $\boldsymbol{x}$ is

$$\nabla_{\boldsymbol{x}} = 2(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})^\top \boldsymbol{A}\,. \tag{7.10}$$

We can use this gradient directly in a gradient descent algorithm. However, for this particular special case, it turns out that there is an analytic solution, which can be found by setting the gradient to zero. We will see more on solving squared error problems in Chapter 9.

Remark. When applied to the solution of linear systems of equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, gradient descent may converge slowly. The speed of convergence of gradient descent is dependent on the *condition number* $\kappa = \frac{\sigma(\boldsymbol{A})_{\max}}{\sigma(\boldsymbol{A})_{\min}}$, which is the ratio of the maximum to the minimum singular value (Section 4.5) of $\boldsymbol{A}$. The condition number essentially measures the ratio of the most curved direction versus the least curved direction, which corresponds to our imagery that poorly conditioned problems are long, thin valleys: They are very curved in one direction, but very flat in the other. Instead of directly solving $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, one could instead solve $\boldsymbol{P}^{-1}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) = \boldsymbol{0}$, where $\boldsymbol{P}$ is called the *preconditioner*. The goal is to design $\boldsymbol{P}^{-1}$ such that $\boldsymbol{P}^{-1}\boldsymbol{A}$ has a better condition number, but at the same time $\boldsymbol{P}^{-1}$ is easy to compute. For further information on gradient descent, preconditioning, and convergence we refer to Boyd and Vandenberghe (2004, chapter 9). ♢

condition number

preconditioner

7.1.2 Gradient Descent With Momentum

As illustrated in Figure 7.3, the convergence of gradient descent may be very slow if the curvature of the optimization surface is such that there are regions that are poorly scaled. The curvature is such that the gradient descent steps hops between the walls of the valley and approaches the optimum in small steps. The proposed tweak to improve convergence is to give gradient descent some memory.

Goh (2017) wrote an intuitive blog post on gradient descent with momentum.

Gradient descent with momentum (Rumelhart et al., 1986) is a method that introduces an additional term to remember what happened in the previous iteration. This memory dampens oscillations and smoothes out the gradient updates. Continuing the ball analogy, the momentum term emulates the phenomenon of a heavy ball that is reluctant to change directions. The idea is to have a gradient update with memory to implement

a moving average. The momentum-based method remembers the update $\Delta\boldsymbol{x}_i$ at each iteration i and determines the next update as a linear combination of the current and previous gradients

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \gamma_i((\nabla f)(\boldsymbol{x}_i))^\top + \alpha\Delta\boldsymbol{x}_i \tag{7.11}$$

$$\Delta\boldsymbol{x}_i = \boldsymbol{x}_i - \boldsymbol{x}_{i-1} = \alpha\Delta\boldsymbol{x}_{i-1} - \gamma_{i-1}((\nabla f)(\boldsymbol{x}_{i-1}))^\top , \tag{7.12}$$

where $\alpha \in [0, 1]$. Sometimes we will only know the gradient approximately. In such cases, the momentum term is useful since it averages out different noisy estimates of the gradient. One particularly useful way to obtain an approximate gradient is by using a stochastic approximation, which we discuss next.

7.1.3 Stochastic Gradient Descent

Computing the gradient can be very time consuming. However, often it is possible to find a "cheap" approximation of the gradient. Approximating the gradient is still useful as long as it points in roughly the same direction as the true gradient.

stochastic gradient descent

Stochastic gradient descent (often shortened as SGD) is a stochastic approximation of the gradient descent method for minimizing an objective function that is written as a sum of differentiable functions. The word stochastic here refers to the fact that we acknowledge that we do not know the gradient precisely, but instead only know a noisy approximation to it. By constraining the probability distribution of the approximate gradients, we can still theoretically guarantee that SGD will converge.

In machine learning, given $n = 1, \ldots, N$ data points, we often consider objective functions that are the sum of the losses L_n incurred by each example n. In mathematical notation, we have the form

$$L(\boldsymbol{\theta}) = \sum_{n=1}^{N} L_n(\boldsymbol{\theta}) , \tag{7.13}$$

where $\boldsymbol{\theta}$ is the vector of parameters of interest, i.e., we want to find $\boldsymbol{\theta}$ that minimizes L. An example from regression (Chapter 9) is the negative log-likelihood, which is expressed as a sum over log-likelihoods of individual examples so that

$$L(\boldsymbol{\theta}) = -\sum_{n=1}^{N} \log p(y_n|\boldsymbol{x}_n, \boldsymbol{\theta}) , \tag{7.14}$$

where $\boldsymbol{x}_n \in \mathbb{R}^D$ are the training inputs, y_n are the training targets, and $\boldsymbol{\theta}$ are the parameters of the regression model.

Standard gradient descent, as introduced previously, is a "batch" optimization method, i.e., optimization is performed using the full training set

by updating the vector of parameters according to

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \gamma_i (\nabla L(\boldsymbol{\theta}_i))^\top = \boldsymbol{\theta}_i - \gamma_i \sum_{n=1}^{N} (\nabla L_n(\boldsymbol{\theta}_i))^\top \tag{7.15}$$

for a suitable step-size parameter γ_i. Evaluating the sum gradient may require expensive evaluations of the gradients from all individual functions L_n. When the training set is enormous and/or no simple formulas exist, evaluating the sums of gradients becomes very expensive.

Consider the term $\sum_{n=1}^{N}(\nabla L_n(\boldsymbol{\theta}_i))$ in (7.15), we can reduce the amount of computation by taking a sum over a smaller set of L_n. In contrast to batch gradient descent, which uses all L_n for $n = 1, \ldots, N$, we randomly choose a subset of L_n for mini-batch gradient descent. In the extreme case, we randomly select only a single L_n to estimate the gradient. The key insight about why taking a subset of data is sensible is to realize that for gradient descent to converge, we only require that the gradient is an unbiased estimate of the true gradient. In fact the term $\sum_{n=1}^{N}(\nabla L_n(\boldsymbol{\theta}_i))$ in (7.15) is an empirical estimate of the expected value (Section 6.4.1) of the gradient. Therefore, any other unbiased empirical estimate of the expected value, for example using any subsample of the data, would suffice for convergence of gradient descent.

Remark. When the learning rate decreases at an appropriate rate, and subject to relatively mild assumptions, stochastic gradient descent converges almost surely to local minimum (Bottou, 1998). ♢

Why should one consider using an approximate gradient? A major reason is practical implementation constraints, such as the size of central processing unit (CPU)/graphics processing unit (GPU) memory or limits on computational time. We can think of the size of the subset used to estimate the gradient in the same way that we thought of the size of a sample when estimating empirical means (Section 6.4.1). Large mini-batch sizes will provide accurate estimates of the gradient, reducing the variance in the parameter update. Furthermore, large mini-batches take advantage of highly optimized matrix operations in vectorized implementations of the cost and gradient. The reduction in variance leads to more stable convergence, but each gradient calculation will be more expensive.

In contrast, small mini-batches are quick to estimate. If we keep the mini-batch size small, the noise in our gradient estimate will allow us to get out of some bad local optima, which we may otherwise get stuck in. In machine learning, optimization methods are used for training by minimizing an objective function on the training data, but the overall goal is to improve generalization performance (Chapter 8). Since the goal in machine learning does not necessarily need a precise estimate of the minimum of the objective function, approximate gradients using mini-batch approaches have been widely used. Stochastic gradient descent is very effective in large-scale machine learning problems (Bottou et al., 2018),

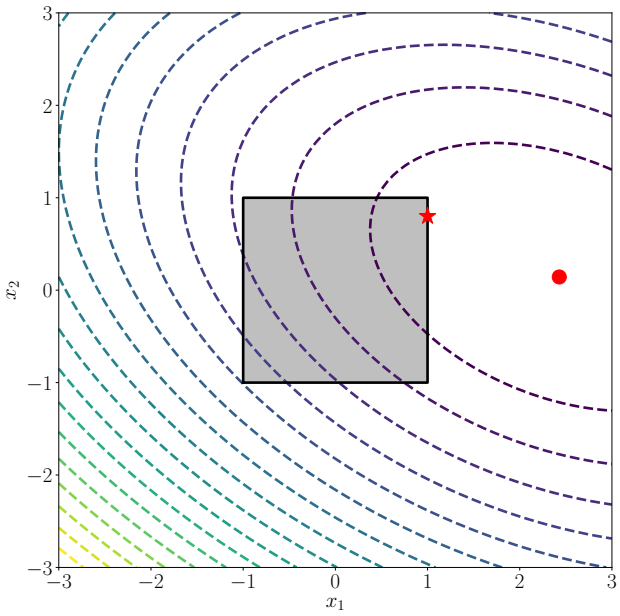


Figure 7.4 Illustration of constrained optimization. The unconstrained problem (indicated by the contour lines) has a minimum on the right side (indicated by the circle). The box constraints ($-1 \leqslant x \leqslant 1$ and $-1 \leqslant y \leqslant 1$) require that the optimal solution is within the box, resulting in an optimal value indicated by the star.

such as training deep neural networks on millions of images (Dean et al., 2012), topic models (Hoffman et al., 2013), reinforcement learning (Mnih et al., 2015), or training of large-scale Gaussian process models (Hensman et al., 2013; Gal et al., 2014).

7.2 Constrained Optimization and Lagrange Multipliers

In the previous section, we considered the problem of solving for the minimum of a function

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \,, \tag{7.16}$$

where $f : \mathbb{R}^D \to \mathbb{R}$.

In this section, we have additional constraints. That is, for real-valued functions $g_i : \mathbb{R}^D \to \mathbb{R}$ for $i = 1, \ldots, m$, we consider the constrained optimization problem (see Figure 7.4 for an illustration)

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & g_i(\boldsymbol{x}) \leqslant 0 \quad \text{for all} \quad i = 1, \ldots, m \,. \end{aligned} \tag{7.17}$$

It is worth pointing out that the functions f and g_i could be non-convex in general, and we will consider the convex case in the next section.

One obvious, but not very practical, way of converting the constrained problem (7.17) into an unconstrained one is to use an indicator function

$$J(\boldsymbol{x}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \mathbf{1}(g_i(\boldsymbol{x})) \,, \tag{7.18}$$

where $\mathbf{1}(z)$ is an infinite step function

$$\mathbf{1}(z) = \begin{cases} 0 & \text{if } z \leqslant 0 \\ \infty & \text{otherwise} \end{cases} . \tag{7.19}$$

This gives infinite penalty if the constraint is not satisfied, and hence would provide the same solution. However, this infinite step function is equally difficult to optimize. We can overcome this difficulty by introducing *Lagrange multipliers*. The idea of Lagrange multipliers is to replace the step function with a linear function.

Lagrange multiplier

Lagrangian

We associate to problem (7.17) the *Lagrangian* by introducing the Lagrange multipliers $\lambda_i \geqslant 0$ corresponding to each inequality constraint respectively (Boyd and Vandenberghe, 2004, chapter 4) so that

$$\mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i g_i(\boldsymbol{x}) \tag{7.20a}$$

$$= f(\boldsymbol{x}) + \boldsymbol{\lambda}^\top \boldsymbol{g}(\boldsymbol{x}) , \tag{7.20b}$$

where in the last line we have concatenated all constraints $g_i(\boldsymbol{x})$ into a vector $\boldsymbol{g}(\boldsymbol{x})$, and all the Lagrange multipliers into a vector $\boldsymbol{\lambda} \in \mathbb{R}^m$.

We now introduce the idea of Lagrangian duality. In general, duality in optimization is the idea of converting an optimization problem in one set of variables $\boldsymbol{x}$ (called the primal variables), into another optimization problem in a different set of variables $\boldsymbol{\lambda}$ (called the dual variables). We introduce two different approaches to duality: In this section, we discuss Lagrangian duality; in Section 7.3.3, we discuss Legendre-Fenchel duality.

Definition 7.1. The problem in (7.17)

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & g_i(\boldsymbol{x}) \leqslant 0 \quad \text{for all} \quad i = 1, \dots, m \end{aligned} \tag{7.21}$$

primal problem

is known as the *primal problem*, corresponding to the primal variables x.

Lagrangian dual problem

The associated *Lagrangian dual problem* is given by

$$\begin{aligned} \max_{\boldsymbol{\lambda} \in \mathbb{R}^m} \quad & \mathfrak{D}(\boldsymbol{\lambda}) \\ \text{subject to} \quad & \boldsymbol{\lambda} \geqslant \mathbf{0} , \end{aligned} \tag{7.22}$$

where $\boldsymbol{\lambda}$ are the dual variables and $\mathfrak{D}(\boldsymbol{\lambda}) = \min_{\boldsymbol{x} \in \mathbb{R}^d} \mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda})$.

Remark. In the discussion of Definition 7.1, we use two concepts that are also of independent interest (Boyd and Vandenberghe, 2004).

minimax inequality

First is the *minimax inequality*, which says that for any function with two arguments $\varphi(\boldsymbol{x}, \boldsymbol{y})$, the maximin is less than the minimax, i.e.,

$$\max_{\boldsymbol{y}} \min_{\boldsymbol{x}} \varphi(\boldsymbol{x}, \boldsymbol{y}) \leqslant \min_{\boldsymbol{x}} \max_{\boldsymbol{y}} \varphi(\boldsymbol{x}, \boldsymbol{y}) . \tag{7.23}$$

This inequality can be proved by considering the inequality

$$\text{For all } \boldsymbol{x}, \boldsymbol{y} \qquad \min_{\boldsymbol{x}} \varphi(\boldsymbol{x}, \boldsymbol{y}) \leqslant \max_{\boldsymbol{y}} \varphi(\boldsymbol{x}, \boldsymbol{y}) \,. \tag{7.24}$$

Note that taking the maximum over $\boldsymbol{y}$ of the left-hand side of (7.24) maintains the inequality since the inequality is true for all $\boldsymbol{y}$. Similarly, we can take the minimum over $\boldsymbol{x}$ of the right-hand side of (7.24) to obtain (7.23).

The second concept is *weak duality*, which uses (7.23) to show that primal values are always greater than or equal to dual values. This is described in more detail in (7.27). ♢

weak duality

Recall that the difference between $J(\boldsymbol{x})$ in (7.18) and the Lagrangian in (7.20b) is that we have relaxed the indicator function to a linear function. Therefore, when $\lambda \geqslant 0$, the Lagrangian $\mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda})$ is a lower bound of $J(\boldsymbol{x})$. Hence, the maximum of $\mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda})$ with respect to $\boldsymbol{\lambda}$ is

$$J(\boldsymbol{x}) = \max_{\boldsymbol{\lambda} \geqslant \boldsymbol{0}} \mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda}) \,. \tag{7.25}$$

Recall that the original problem was minimizing $J(\boldsymbol{x})$,

$$\min_{\boldsymbol{x} \in \mathbb{R}^d} \max_{\boldsymbol{\lambda} \geqslant \boldsymbol{0}} \mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda}) \,. \tag{7.26}$$

By the minimax inequality (7.23), it follows that swapping the order of the minimum and maximum results in a smaller value, i.e.,

$$\min_{\boldsymbol{x} \in \mathbb{R}^d} \max_{\boldsymbol{\lambda} \geqslant \boldsymbol{0}} \mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda}) \geqslant \max_{\boldsymbol{\lambda} \geqslant \boldsymbol{0}} \min_{\boldsymbol{x} \in \mathbb{R}^d} \mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda}) \,. \tag{7.27}$$

This is also known as *weak duality*. Note that the inner part of the right-hand side is the dual objective function $\mathfrak{D}(\boldsymbol{\lambda})$ and the definition follows.

weak duality

In contrast to the original optimization problem, which has constraints, $\min_{\boldsymbol{x} \in \mathbb{R}^d} \mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda})$ is an unconstrained optimization problem for a given value of $\boldsymbol{\lambda}$. If solving $\min_{\boldsymbol{x} \in \mathbb{R}^d} \mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda})$ is easy, then the overall problem is easy to solve. We can see this by observing from (7.20b) that $\mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda})$ is affine with respect to $\boldsymbol{\lambda}$. Therefore $\min_{\boldsymbol{x} \in \mathbb{R}^d} \mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda})$ is a pointwise minimum of affine functions of $\boldsymbol{\lambda}$, and hence $\mathfrak{D}(\boldsymbol{\lambda})$ is concave even though $f(\cdot)$ and $g_i(\cdot)$ may be nonconvex. The outer problem, maximization over $\boldsymbol{\lambda}$, is the maximum of a concave function and can be efficiently computed.

Assuming $f(\cdot)$ and $g_i(\cdot)$ are differentiable, we find the Lagrange dual problem by differentiating the Lagrangian with respect to $\boldsymbol{x}$, setting the differential to zero, and solving for the optimal value. We will discuss two concrete examples in Sections 7.3.1 and 7.3.2, where $f(\cdot)$ and $g_i(\cdot)$ are convex.

Remark (Equality Constraints). Consider (7.17) with additional equality constraints

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & g_i(\boldsymbol{x}) \leqslant 0 \quad \text{for all} \quad i = 1, \ldots, m \\ & h_j(\boldsymbol{x}) = 0 \quad \text{for all} \quad j = 1, \ldots, n \,. \end{aligned} \tag{7.28}$$

We can model equality constraints by replacing them with two inequality constraints. That is for each equality constraint $h_j(\boldsymbol{x}) = 0$ we equivalently replace it by two constraints $h_j(\boldsymbol{x}) \leqslant 0$ and $h_j(\boldsymbol{x}) \geqslant 0$. It turns out that the resulting Lagrange multipliers are then unconstrained.

Therefore, we constrain the Lagrange multipliers corresponding to the inequality constraints in (7.28) to be non-negative, and leave the Lagrange multipliers corresponding to the equality constraints unconstrained. ♢

7.3 Convex Optimization

We focus our attention of a particularly useful class of optimization problems, where we can guarantee global optimality. When $f(\cdot)$ is a convex function, and when the constraints involving $g(\cdot)$ and $h(\cdot)$ are convex sets, this is called a *convex optimization problem*. In this setting, we have *strong duality*: The optimal solution of the dual problem is the same as the optimal solution of the primal problem. The distinction between convex functions and convex sets are often not strictly presented in machine learning literature, but one can often infer the implied meaning from context.

convex optimization problem

strong duality

convex set

Definition 7.2. A set $\mathcal{C}$ is a *convex set* if for any $x, y \in \mathcal{C}$ and for any scalar θ with $0 \leqslant \theta \leqslant 1$, we have

$$\theta x + (1-\theta)y \in \mathcal{C}\,. \tag{7.29}$$

Figure 7.5 Example of a convex set.

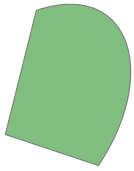
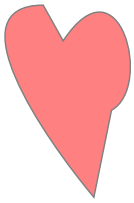


Figure 7.6 Example of a nonconvex set.



Convex sets are sets such that a straight line connecting any two elements of the set lie inside the set. Figures 7.5 and 7.6 illustrate convex and nonconvex sets, respectively.

Convex functions are functions such that a straight line between any two points of the function lie above the function. Figure 7.2 shows a nonconvex function, and Figure 7.3 shows a convex function. Another convex function is shown in Figure 7.7.

Definition 7.3. Let function $f : \mathbb{R}^D \to \mathbb{R}$ be a function whose domain is a convex set. The function f is a *convex function* if for all $\boldsymbol{x}, \boldsymbol{y}$ in the domain of f, and for any scalar θ with $0 \leqslant \theta \leqslant 1$, we have

$$f(\theta\boldsymbol{x} + (1-\theta)\boldsymbol{y}) \leqslant \theta f(\boldsymbol{x}) + (1-\theta)f(\boldsymbol{y})\,. \tag{7.30}$$

Remark. A *concave function* is the negative of a convex function. ♢

convex function

concave function

The constraints involving $g(\cdot)$ and $h(\cdot)$ in (7.28) truncate functions at a scalar value, resulting in sets. Another relation between convex functions and convex sets is to consider the set obtained by "filling in" a convex function. A convex function is a bowl-like object, and we imagine pouring water into it to fill it up. This resulting filled-in set, called the *epigraph* of the convex function, is a convex set.

epigraph

If a function $f : \mathbb{R}^n \to \mathbb{R}$ is differentiable, we can specify convexity in

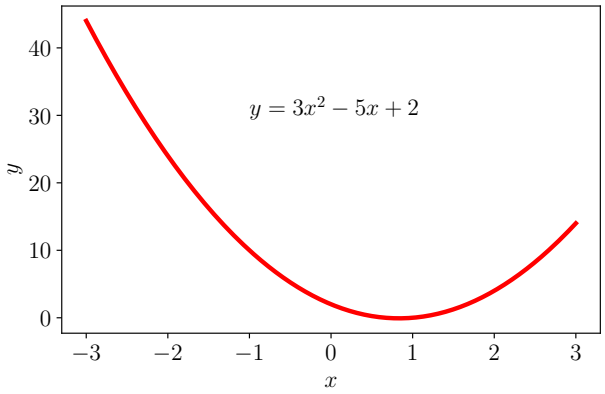


Figure 7.7 Example of a convex function.

terms of its gradient $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ (Section 5.2). A function $f(\boldsymbol{x})$ is convex if and only if for any two points $\boldsymbol{x}, \boldsymbol{y}$ it holds that

$$f(\boldsymbol{y}) \geqslant f(\boldsymbol{x}) + \nabla_{\boldsymbol{x}} f(\boldsymbol{x})^\top (\boldsymbol{y} - \boldsymbol{x}) \,. \tag{7.31}$$

If we further know that a function $f(\boldsymbol{x})$ is twice differentiable, that is, the Hessian (5.147) exists for all values in the domain of $\boldsymbol{x}$, then the function $f(\boldsymbol{x})$ is convex if and only if $\nabla^2_{\boldsymbol{x}} f(\boldsymbol{x})$ is positive semidefinite (Boyd and Vandenberghe, 2004).

Example 7.3

The negative entropy $f(x) = x \log_2 x$ is convex for $x > 0$. A visualization of the function is shown in Figure 7.8, and we can see that the function is convex. To illustrate the previous definitions of convexity, let us check the calculations for two points $x = 2$ and $x = 4$. Note that to prove convexity of $f(x)$ we would need to check for all points $x \in \mathbb{R}$.

Recall Definition 7.3. Consider a point midway between the two points (that is $\theta = 0.5$); then the left-hand side is $f(0.5 \cdot 2 + 0.5 \cdot 4) = 3 \log_2 3 \approx 4.75$. The right-hand side is $0.5(2 \log_2 2) + 0.5(4 \log_2 4) = 1 + 4 = 5$. And therefore the definition is satisfied.

Since $f(x)$ is differentiable, we can alternatively use (7.31). Calculating the derivative of $f(x)$, we obtain

$$\nabla_x (x \log_2 x) = 1 \cdot \log_2 x + x \cdot \frac{1}{x \log_e 2} = \log_2 x + \frac{1}{\log_e 2} \,. \tag{7.32}$$

Using the same two test points $x = 2$ and $x = 4$, the left-hand side of (7.31) is given by $f(4) = 8$. The right-hand side is

$$f(\boldsymbol{x}) + \nabla_{\boldsymbol{x}}^\top (\boldsymbol{y} - \boldsymbol{x}) = f(2) + \nabla f(2) \cdot (4 - 2) \tag{7.33a}$$

$$= 2 + (1 + \frac{1}{\log_e 2}) \cdot 2 \approx 6.9 \,. \tag{7.33b}$$

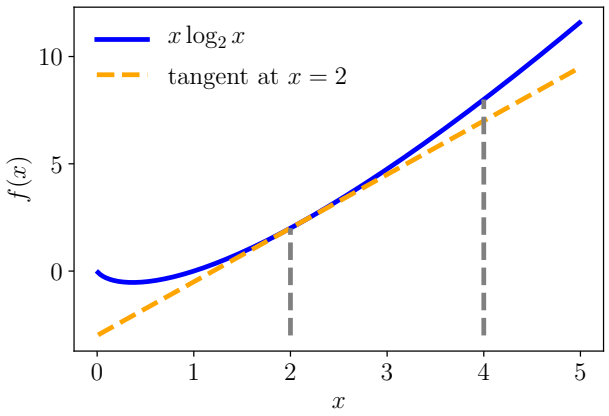


Figure 7.8 The negative entropy function (which is convex) and its tangent at $x = 2$.

We can check that a function or set is convex from first principles by recalling the definitions. In practice, we often rely on operations that preserve convexity to check that a particular function or set is convex. Although the details are vastly different, this is again the idea of closure that we introduced in Chapter 2 for vector spaces.

Example 7.4

A nonnegative weighted sum of convex functions is convex. Observe that if f is a convex function, and $\alpha \geqslant 0$ is a nonnegative scalar, then the function αf is convex. We can see this by multiplying α to both sides of the equation in Definition 7.3, and recalling that multiplying a nonnegative number does not change the inequality.

If f_1 and f_2 are convex functions, then we have by the definition

$$f_1(\theta \boldsymbol{x} + (1-\theta)\boldsymbol{y}) \leqslant \theta f_1(\boldsymbol{x}) + (1-\theta) f_1(\boldsymbol{y}) \tag{7.34}$$

$$f_2(\theta \boldsymbol{x} + (1-\theta)\boldsymbol{y}) \leqslant \theta f_2(\boldsymbol{x}) + (1-\theta) f_2(\boldsymbol{y})\,. \tag{7.35}$$

Summing up both sides gives us

$$\begin{aligned} &f_1(\theta \boldsymbol{x} + (1-\theta)\boldsymbol{y}) + f_2(\theta \boldsymbol{x} + (1-\theta)\boldsymbol{y}) \\ &\leqslant \theta f_1(\boldsymbol{x}) + (1-\theta) f_1(\boldsymbol{y}) + \theta f_2(\boldsymbol{x}) + (1-\theta) f_2(\boldsymbol{y})\,, \end{aligned} \tag{7.36}$$

where the right-hand side can be rearranged to

$$\theta(f_1(\boldsymbol{x}) + f_2(\boldsymbol{x})) + (1-\theta)(f_1(\boldsymbol{y}) + f_2(\boldsymbol{y}))\,, \tag{7.37}$$

completing the proof that the sum of convex functions is convex.

Combining the preceding two facts, we see that $\alpha f_1(\boldsymbol{x}) + \beta f_2(\boldsymbol{x})$ is convex for $\alpha, \beta \geqslant 0$. This closure property can be extended using a similar argument for nonnegative weighted sums of more than two convex functions.

Remark. The inequality in (7.30) is sometimes called *Jensen's inequality*. In fact, a whole class of inequalities for taking nonnegative weighted sums of convex functions are all called Jensen's inequality. ♢

Jensen's inequality

In summary, a constrained optimization problem is called a *convex optimization problem* if

convex optimization problem

$$\begin{aligned} &\min_{\boldsymbol{x}} f(\boldsymbol{x}) \\ &\text{subject to } g_i(\boldsymbol{x}) \leqslant 0 \quad \text{for all} \quad i = 1, \dots, m \\ &\qquad\quad h_j(\boldsymbol{x}) = 0 \quad \text{for all} \quad j = 1, \dots, n\,, \end{aligned} \tag{7.38}$$

where all functions $f(\boldsymbol{x})$ and $g_i(\boldsymbol{x})$ are convex functions, and all $h_j(\boldsymbol{x}) = 0$ are convex sets. In the following, we will describe two classes of convex optimization problems that are widely used and well understood.

7.3.1 Linear Programming

Consider the special case when all the preceding functions are linear, i.e.,

$$\begin{aligned} \min_{\boldsymbol{x} \in \mathbb{R}^d} \quad & \boldsymbol{c}^\top \boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} \leqslant \boldsymbol{b}\,, \end{aligned} \tag{7.39}$$

where $\boldsymbol{A} \in \mathbb{R}^{m \times d}$ and $\boldsymbol{b} \in \mathbb{R}^m$. This is known as a *linear program*. It has d variables and m linear constraints. The Lagrangian is given by

linear program

Linear programs are one of the most widely used approaches in industry.

$$\mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{\lambda}^\top (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})\,, \tag{7.40}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^m$ is the vector of non-negative Lagrange multipliers. Rearranging the terms corresponding to $\boldsymbol{x}$ yields

$$\mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = (\boldsymbol{c} + \boldsymbol{A}^\top \boldsymbol{\lambda})^\top \boldsymbol{x} - \boldsymbol{\lambda}^\top \boldsymbol{b}\,. \tag{7.41}$$

Taking the derivative of $\mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda})$ with respect to $\boldsymbol{x}$ and setting it to zero gives us

$$\boldsymbol{c} + \boldsymbol{A}^\top \boldsymbol{\lambda} = \boldsymbol{0}\,. \tag{7.42}$$

Therefore, the dual Lagrangian is $\mathfrak{D}(\boldsymbol{\lambda}) = -\boldsymbol{\lambda}^\top \boldsymbol{b}$. Recall we would like to maximize $\mathfrak{D}(\boldsymbol{\lambda})$. In addition to the constraint due to the derivative of $\mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda})$ being zero, we also have the fact that $\boldsymbol{\lambda} \geqslant \boldsymbol{0}$, resulting in the following dual optimization problem

It is convention to minimize the primal and maximize the dual.

$$\begin{aligned} \max_{\boldsymbol{\lambda} \in \mathbb{R}^m} \quad & -\boldsymbol{b}^\top \boldsymbol{\lambda} \\ \text{subject to} \quad & \boldsymbol{c} + \boldsymbol{A}^\top \boldsymbol{\lambda} = \boldsymbol{0} \\ & \boldsymbol{\lambda} \geqslant \boldsymbol{0}\,. \end{aligned} \tag{7.43}$$

This is also a linear program, but with m variables. We have the choice of solving the primal (7.39) or the dual (7.43) program depending on

whether m or d is larger. Recall that d is the number of variables and m is the number of constraints in the primal linear program.

Example 7.5 (Linear Program)
Consider the linear program

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathbb{R}^2} \quad & -\begin{bmatrix}5\\3\end{bmatrix}^\top \begin{bmatrix}x_1\\x_2\end{bmatrix} \\ \text{subject to} \quad & \begin{bmatrix}2 & 2\\2 & -4\\-2 & 1\\0 & -1\\0 & 1\end{bmatrix} \begin{bmatrix}x_1\\x_2\end{bmatrix} \leqslant \begin{bmatrix}33\\8\\5\\-1\\8\end{bmatrix} \end{aligned} \tag{7.44}$$

with two variables. This program is also shown in Figure 7.9. The objective function is linear, resulting in linear contour lines. The constraint set in standard form is translated into the legend. The optimal value must lie in the shaded (feasible) region, and is indicated by the star.

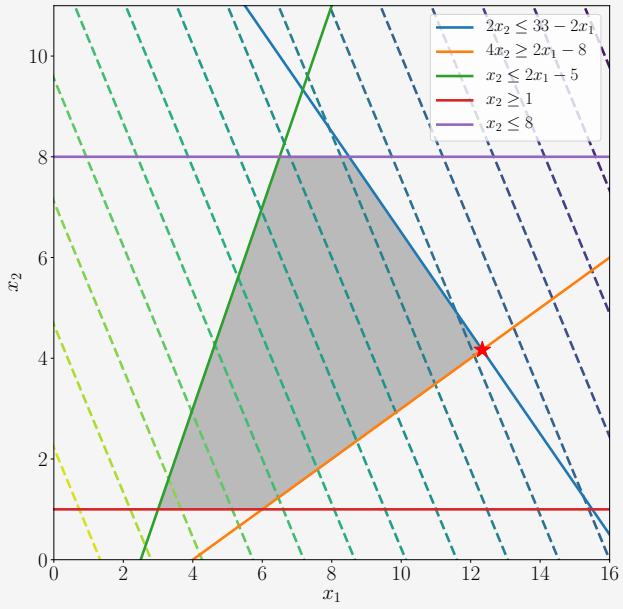


Figure 7.9 Illustration of a linear program. The unconstrained problem (indicated by the contour lines) has a minimum on the right side. The optimal value given the constraints are shown by the star.

7.3.2 Quadratic Programming

Consider the case of a convex quadratic objective function, where the constraints are affine, i.e.,

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathbb{R}^d} \quad & \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}^\top \boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} \leqslant \boldsymbol{b}\,, \end{aligned} \tag{7.45}$$

where $\boldsymbol{A} \in \mathbb{R}^{m\times d}$, $\boldsymbol{b} \in \mathbb{R}^m$, and $\boldsymbol{c} \in \mathbb{R}^d$. The square symmetric matrix $\boldsymbol{Q} \in \mathbb{R}^{d\times d}$ is positive definite, and therefore the objective function is convex. This is known as a *quadratic program*. Observe that it has d variables and m linear constraints.

Example 7.6 (Quadratic Program)
Consider the quadratic program

$$\min_{\boldsymbol{x}\in\mathbb{R}^2} \quad \frac{1}{2}\begin{bmatrix}x_1\\x_2\end{bmatrix}^\top \begin{bmatrix}2 & 1\\1 & 4\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} + \begin{bmatrix}5\\3\end{bmatrix}^\top \begin{bmatrix}x_1\\x_2\end{bmatrix} \tag{7.46}$$

$$\text{subject to} \quad \begin{bmatrix}1 & 0\\-1 & 0\\0 & 1\\0 & -1\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} \leqslant \begin{bmatrix}1\\1\\1\\1\end{bmatrix} \tag{7.47}$$

of two variables. The program is also illustrated in Figure 7.4. The objective function is quadratic with a positive semidefinite matrix $\boldsymbol{Q}$, resulting in elliptical contour lines. The optimal value must lie in the shaded (feasible) region, and is indicated by the star.

The Lagrangian is given by

$$\mathfrak{L}(\boldsymbol{x},\boldsymbol{\lambda}) = \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{\lambda}^\top(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \tag{7.48a}$$

$$= \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} + (\boldsymbol{c} + \boldsymbol{A}^\top\boldsymbol{\lambda})^\top \boldsymbol{x} - \boldsymbol{\lambda}^\top \boldsymbol{b}\,, \tag{7.48b}$$

where again we have rearranged the terms. Taking the derivative of $\mathfrak{L}(\boldsymbol{x},\boldsymbol{\lambda})$ with respect to $\boldsymbol{x}$ and setting it to zero gives

$$\boldsymbol{Q}\boldsymbol{x} + (\boldsymbol{c} + \boldsymbol{A}^\top\boldsymbol{\lambda}) = \boldsymbol{0}\,. \tag{7.49}$$

Assuming that $\boldsymbol{Q}$ is invertible, we get

$$\boldsymbol{x} = -\boldsymbol{Q}^{-1}(\boldsymbol{c} + \boldsymbol{A}^\top\boldsymbol{\lambda})\,. \tag{7.50}$$

Substituting (7.50) into the primal Lagrangian $\mathfrak{L}(\boldsymbol{x},\boldsymbol{\lambda})$, we get the dual Lagrangian

$$\mathfrak{D}(\boldsymbol{\lambda}) = -\frac{1}{2}(\boldsymbol{c} + \boldsymbol{A}^\top\boldsymbol{\lambda})^\top \boldsymbol{Q}^{-1}(\boldsymbol{c} + \boldsymbol{A}^\top\boldsymbol{\lambda}) - \boldsymbol{\lambda}^\top \boldsymbol{b}\,. \tag{7.51}$$

Therefore, the dual optimization problem is given by

$$\begin{aligned} \max_{\boldsymbol{\lambda} \in \mathbb{R}^m} \quad & -\frac{1}{2}(\boldsymbol{c} + \boldsymbol{A}^\top \boldsymbol{\lambda})^\top \boldsymbol{Q}^{-1}(\boldsymbol{c} + \boldsymbol{A}^\top \boldsymbol{\lambda}) - \boldsymbol{\lambda}^\top \boldsymbol{b} \\ \text{subject to} \quad & \boldsymbol{\lambda} \geqslant \boldsymbol{0}\,. \end{aligned} \tag{7.52}$$

We will see an application of quadratic programming in machine learning in Chapter 12.

7.3.3 Legendre-Fenchel Transform and Convex Conjugate

Let us revisit the idea of duality from Section 7.2, without considering constraints. One useful fact about a convex set is that it can be equivalently described by its supporting hyperplanes. A hyperplane is called a *supporting hyperplane* of a convex set if it intersects the convex set, and the convex set is contained on just one side of it. Recall that we can fill up a convex function to obtain the epigraph, which is a convex set. Therefore, we can also describe convex functions in terms of their supporting hyperplanes. Furthermore, observe that the supporting hyperplane just touches the convex function, and is in fact the tangent to the function at that point. And recall that the tangent of a function $f(\boldsymbol{x})$ at a given point $\boldsymbol{x}_0$ is the evaluation of the gradient of that function at that point $\frac{\mathrm{d}f(\boldsymbol{x})}{\mathrm{d}\boldsymbol{x}}\Big|_{\boldsymbol{x}=\boldsymbol{x}_0}$. In summary, because convex sets can be equivalently described by its supporting hyperplanes, convex functions can be equivalently described by a function of their gradient. The *Legendre transform* formalizes this concept.

supporting hyperplane

Legendre transform

Physics students are often introduced to the Legendre transform as relating the Lagrangian and the Hamiltonian in classical mechanics.

We begin with the most general definition, which unfortunately has a counter-intuitive form, and look at special cases to relate the definition to the intuition described in the preceding paragraph. The *Legendre-Fenchel transform* is a transformation (in the sense of a Fourier transform) from a convex differentiable function $f(\boldsymbol{x})$ to a function that depends on the tangents $s(\boldsymbol{x}) = \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$. It is worth stressing that this is a transformation of the function $f(\cdot)$ and not the variable $\boldsymbol{x}$ or the function evaluated at $\boldsymbol{x}$. The Legendre-Fenchel transform is also known as the *convex conjugate* (for reasons we will see soon) and is closely related to duality (Hiriart-Urruty and Lemaréchal, 2001, chapter 5).

Legendre-Fenchel transform

convex conjugate

convex conjugate

Definition 7.4. The *convex conjugate* of a function $f : \mathbb{R}^D \to \mathbb{R}$ is a function f^* defined by

$$f^*(\boldsymbol{s}) = \sup_{\boldsymbol{x} \in \mathbb{R}^D} \left(\langle \boldsymbol{s}, \boldsymbol{x} \rangle - f(\boldsymbol{x}) \right) . \tag{7.53}$$

Note that the preceding convex conjugate definition does not need the function f to be convex nor differentiable. In Definition 7.4, we have used a general inner product (Section 3.2) but in the rest of this section we

will consider the standard dot product between finite-dimensional vectors ($\langle \boldsymbol{s}, \boldsymbol{x} \rangle = \boldsymbol{s}^\top \boldsymbol{x}$) to avoid too many technical details.

This derivation is easiest to understand by drawing the reasoning as it progresses.

To understand Definition 7.4 in a geometric fashion, consider a nice simple one-dimensional convex and differentiable function, for example $f(x) = x^2$. Note that since we are looking at a one-dimensional problem, hyperplanes reduce to a line. Consider a line $y = sx + c$. Recall that we are able to describe convex functions by their supporting hyperplanes, so let us try to describe this function $f(x)$ by its supporting lines. Fix the gradient of the line $s \in \mathbb{R}$ and for each point $(x_0, f(x_0))$ on the graph of f, find the minimum value of c such that the line still intersects $(x_0, f(x_0))$. Note that the minimum value of c is the place where a line with slope s "just touches" the function $f(x) = x^2$. The line passing through $(x_0, f(x_0))$ with gradient s is given by

$$y - f(x_0) = s(x - x_0)\,. \tag{7.54}$$

The y-intercept of this line is $-sx_0 + f(x_0)$. The minimum of c for which $y = sx + c$ intersects with the graph of f is therefore

$$\inf_{x_0} -sx_0 + f(x_0)\,. \tag{7.55}$$

The preceding convex conjugate is by convention defined to be the negative of this. The reasoning in this paragraph did not rely on the fact that we chose a one-dimensional convex and differentiable function, and holds for $f : \mathbb{R}^D \to \mathbb{R}$, which are nonconvex and non-differentiable.

The classical Legendre transform is defined on convex differentiable functions in $\mathbb{R}^D$.

Remark. Convex differentiable functions such as the example $f(x) = x^2$ is a nice special case, where there is no need for the supremum, and there is a one-to-one correspondence between a function and its Legendre transform. Let us derive this from first principles. For a convex differentiable function, we know that at x_0 the tangent touches $f(x_0)$ so that

$$f(x_0) = sx_0 + c\,. \tag{7.56}$$

Recall that we want to describe the convex function $f(x)$ in terms of its gradient $\nabla_x f(x)$, and that $s = \nabla_x f(x_0)$. We rearrange to get an expression for $-c$ to obtain

$$-c = sx_0 - f(x_0)\,. \tag{7.57}$$

Note that $-c$ changes with x_0 and therefore with s, which is why we can think of it as a function of s, which we call

$$f^*(s) := sx_0 - f(x_0)\,. \tag{7.58}$$

Comparing (7.58) with Definition 7.4, we see that (7.58) is a special case (without the supremum). ♢

The conjugate function has nice properties; for example, for convex functions, applying the Legendre transform again gets us back to the original function. In the same way that the slope of $f(x)$ is s, the slope of $f^*(s)$

is x. The following two examples show common uses of convex conjugates in machine learning.

Example 7.7 (Convex Conjugates)
To illustrate the application of convex conjugates, consider the quadratic function

$$f(\boldsymbol{y}) = \frac{\lambda}{2}\boldsymbol{y}^\top \boldsymbol{K}^{-1}\boldsymbol{y} \tag{7.59}$$

based on a positive definite matrix $\boldsymbol{K} \in \mathbb{R}^{n\times n}$. We denote the primal variable to be $\boldsymbol{y} \in \mathbb{R}^n$ and the dual variable to be $\boldsymbol{\alpha} \in \mathbb{R}^n$.

Applying Definition 7.4, we obtain the function

$$f^*(\boldsymbol{\alpha}) = \sup_{\boldsymbol{y}\in\mathbb{R}^n} \langle \boldsymbol{y}, \boldsymbol{\alpha}\rangle - \frac{\lambda}{2}\boldsymbol{y}^\top \boldsymbol{K}^{-1}\boldsymbol{y}\,. \tag{7.60}$$

Since the function is differentiable, we can find the maximum by taking the derivative and with respect to $\boldsymbol{y}$ setting it to zero.

$$\frac{\partial\left[\langle \boldsymbol{y}, \boldsymbol{\alpha}\rangle - \frac{\lambda}{2}\boldsymbol{y}^\top \boldsymbol{K}^{-1}\boldsymbol{y}\right]}{\partial \boldsymbol{y}} = (\boldsymbol{\alpha} - \lambda \boldsymbol{K}^{-1}\boldsymbol{y})^\top \tag{7.61}$$

and hence when the gradient is zero we have $\boldsymbol{y} = \frac{1}{\lambda}\boldsymbol{K}\boldsymbol{\alpha}$. Substituting into (7.60) yields

$$f^*(\boldsymbol{\alpha}) = \frac{1}{\lambda}\boldsymbol{\alpha}^\top \boldsymbol{K}\boldsymbol{\alpha} - \frac{\lambda}{2}\left(\frac{1}{\lambda}\boldsymbol{K}\boldsymbol{\alpha}\right)^\top \boldsymbol{K}^{-1}\left(\frac{1}{\lambda}\boldsymbol{K}\boldsymbol{\alpha}\right) = \frac{1}{2\lambda}\boldsymbol{\alpha}^\top \boldsymbol{K}\boldsymbol{\alpha}\,. \tag{7.62}$$

Example 7.8
In machine learning, we often use sums of functions; for example, the objective function of the training set includes a sum of the losses for each example in the training set. In the following, we derive the convex conjugate of a sum of losses $\ell(t)$, where $\ell : \mathbb{R} \to \mathbb{R}$. This also illustrates the application of the convex conjugate to the vector case. Let $\mathcal{L}(\boldsymbol{t}) = \sum_{i=1}^n \ell_i(t_i)$. Then,

$$\mathcal{L}^*(\boldsymbol{z}) = \sup_{\boldsymbol{t}\in\mathbb{R}^n} \langle \boldsymbol{z}, \boldsymbol{t}\rangle - \sum_{i=1}^n \ell_i(t_i) \tag{7.63a}$$

$$= \sup_{\boldsymbol{t}\in\mathbb{R}^n} \sum_{i=1}^n z_i t_i - \ell_i(t_i) \qquad \text{definition of dot product} \tag{7.63b}$$

$$= \sum_{i=1}^n \sup_{t\in\mathbb{R}^n} z_i t_i - \ell_i(t_i) \tag{7.63c}$$

$$= \sum_{i=1}^{n} \ell_i^*(z_i)\,. \qquad \text{definition of conjugate} \tag{7.63d}$$

Recall that in Section 7.2 we derived a dual optimization problem using Lagrange multipliers. Furthermore, for convex optimization problems we have strong duality, that is the solutions of the primal and dual problem match. The Legendre-Fenchel transform described here also can be used to derive a dual optimization problem. Furthermore, when the function is convex and differentiable, the supremum is unique. To further investigate the relation between these two approaches, let us consider a linear equality constrained convex optimization problem.

Example 7.9
Let $f(\boldsymbol{y})$ and $g(\boldsymbol{x})$ be convex functions, and $\boldsymbol{A}$ a real matrix of appropriate dimensions such that $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}$. Then

$$\min_{\boldsymbol{x}} f(\boldsymbol{A}\boldsymbol{x}) + g(\boldsymbol{x}) = \min_{\boldsymbol{A}\boldsymbol{x}=\boldsymbol{y}} f(\boldsymbol{y}) + g(\boldsymbol{x}). \tag{7.64}$$

By introducing the Lagrange multiplier $\boldsymbol{u}$ for the constraints $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}$,

$$\min_{\boldsymbol{A}\boldsymbol{x}=\boldsymbol{y}} f(\boldsymbol{y}) + g(\boldsymbol{x}) = \min_{\boldsymbol{x},\boldsymbol{y}} \max_{\boldsymbol{u}} f(\boldsymbol{y}) + g(\boldsymbol{x}) + (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y})^\top \boldsymbol{u} \tag{7.65a}$$

$$= \max_{\boldsymbol{u}} \min_{\boldsymbol{x},\boldsymbol{y}} f(\boldsymbol{y}) + g(\boldsymbol{x}) + (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y})^\top \boldsymbol{u}\,, \tag{7.65b}$$

where the last step of swapping max and min is due to the fact that $f(\boldsymbol{y})$ and $g(\boldsymbol{x})$ are convex functions. By splitting up the dot product term and collecting $\boldsymbol{x}$ and $\boldsymbol{y}$,

$$\max_{\boldsymbol{u}} \min_{\boldsymbol{x},\boldsymbol{y}} f(\boldsymbol{y}) + g(\boldsymbol{x}) + (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y})^\top \boldsymbol{u} \tag{7.66a}$$

$$= \max_{\boldsymbol{u}} \left[\min_{\boldsymbol{y}} -\boldsymbol{y}^\top \boldsymbol{u} + f(\boldsymbol{y})\right] + \left[\min_{\boldsymbol{x}} (\boldsymbol{A}\boldsymbol{x})^\top \boldsymbol{u} + g(\boldsymbol{x})\right] \tag{7.66b}$$

$$= \max_{\boldsymbol{u}} \left[\min_{\boldsymbol{y}} -\boldsymbol{y}^\top \boldsymbol{u} + f(\boldsymbol{y})\right] + \left[\min_{\boldsymbol{x}} \boldsymbol{x}^\top \boldsymbol{A}^\top \boldsymbol{u} + g(\boldsymbol{x})\right] \tag{7.66c}$$

Recall the convex conjugate (Definition 7.4) and the fact that dot products are symmetric,

For general inner products, $\boldsymbol{A}^\top$ is replaced by the adjoint $\boldsymbol{A}^*$.

$$\max_{\boldsymbol{u}} \left[\min_{\boldsymbol{y}} -\boldsymbol{y}^\top \boldsymbol{u} + f(\boldsymbol{y})\right] + \left[\min_{\boldsymbol{x}} \boldsymbol{x}^\top \boldsymbol{A}^\top \boldsymbol{u} + g(\boldsymbol{x})\right] \tag{7.67a}$$

$$= \max_{\boldsymbol{u}} -f^*(\boldsymbol{u}) - g^*(-\boldsymbol{A}^\top \boldsymbol{u})\,. \tag{7.67b}$$

Therefore, we have shown that

$$\min_{\boldsymbol{x}} f(\boldsymbol{A}\boldsymbol{x}) + g(\boldsymbol{x}) = \max_{\boldsymbol{u}} -f^*(\boldsymbol{u}) - g^*(-\boldsymbol{A}^\top \boldsymbol{u})\,. \tag{7.68}$$

The Legendre-Fenchel conjugate turns out to be quite useful for machine learning problems that can be expressed as convex optimization problems. In particular, for convex loss functions that apply independently to each example, the conjugate loss is a convenient way to derive a dual problem.

7.4 Further Reading

Continuous optimization is an active area of research, and we do not try to provide a comprehensive account of recent advances.

From a gradient descent perspective, there are two major weaknesses which each have their own set of literature. The first challenge is the fact that gradient descent is a first-order algorithm, and does not use information about the curvature of the surface. When there are long valleys, the gradient points perpendicularly to the direction of interest. The idea of momentum can be generalized to a general class of acceleration methods (Nesterov, 2018). Conjugate gradient methods avoid the issues faced by gradient descent by taking previous directions into account (Shewchuk, 1994). Second-order methods such as Newton methods use the Hessian to provide information about the curvature. Many of the choices for choosing step-sizes and ideas like momentum arise by considering the curvature of the objective function (Goh, 2017; Bottou et al., 2018). Quasi-Newton methods such as L-BFGS try to use cheaper computational methods to approximate the Hessian (Nocedal and Wright, 2006). Recently there has been interest in other metrics for computing descent directions, resulting in approaches such as mirror descent (Beck and Teboulle, 2003) and natural gradient (Toussaint, 2012).

The second challenge is to handle non-differentiable functions. Gradient methods are not well defined when there are kinks in the function. In these cases, *subgradient methods* can be used (Shor, 1985). For further information and algorithms for optimizing non-differentiable functions, we refer to the book by Bertsekas (1999). There is a vast amount of literature on different approaches for numerically solving continuous optimization problems, including algorithms for constrained optimization problems. Good starting points to appreciate this literature are the books by Luenberger (1969) and Bonnans et al. (2006). A recent survey of continuous optimization is provided by Bubeck (2015).

Hugo Gonçalves' blog is also a good resource for an easier introduction to Legendre–Fenchel transforms: `https://tinyurl.com/ydaal7hj`

Modern applications of machine learning often mean that the size of datasets prohibit the use of batch gradient descent, and hence stochastic gradient descent is the current workhorse of large-scale machine learning methods. Recent surveys of the literature include Hazan (2015) and Bottou et al. (2018).

For duality and convex optimization, the book by Boyd and Vandenberghe (2004) includes lectures and slides online. A more mathematical treatment is provided by Bertsekas (2009), and recent book by one of

the key researchers in the area of optimization is Nesterov (2018). Convex optimization is based upon convex analysis, and the reader interested in more foundational results about convex functions is referred to Rockafellar (1970), Hiriart-Urruty and Lemaréchal (2001), and Borwein and Lewis (2006). Legendre–Fenchel transforms are also covered in the aforementioned books on convex analysis, but a more beginner-friendly presentation is available at Zia et al. (2009). The role of Legendre–Fenchel transforms in the analysis of convex optimization algorithms is surveyed in Polyak (2016).

Exercises

7.1 Consider the univariate function

$$f(x) = x^3 + 6x^2 - 3x - 5.$$

Find its stationary points and indicate whether they are maximum, minimum, or saddle points.

7.2 Consider the update equation for stochastic gradient descent (Equation (7.15)). Write down the update when we use a mini-batch size of one.

7.3 Consider whether the following statements are true or false:

a. The intersection of any two convex sets is convex.
b. The union of any two convex sets is convex.
c. The difference of a convex set A from another convex set B is convex.

7.4 Consider whether the following statements are true or false:

a. The sum of any two convex functions is convex.
b. The difference of any two convex functions is convex.
c. The product of any two convex functions is convex.
d. The maximum of any two convex functions is convex.

7.5 Express the following optimization problem as a standard linear program in matrix notation

$$\max_{\boldsymbol{x}\in\mathbb{R}^2,\ \xi\in\mathbb{R}} \boldsymbol{p}^\top \boldsymbol{x} + \xi$$

subject to the constraints that $\xi \geqslant 0$, $x_0 \leqslant 0$ and $x_1 \leqslant 3$.

7.6 Consider the linear program illustrated in Figure 7.9,

$$\min_{\boldsymbol{x}\in\mathbb{R}^2} -\begin{bmatrix}5\\3\end{bmatrix}^\top \begin{bmatrix}x_1\\x_2\end{bmatrix}$$
$$\text{subject to } \begin{bmatrix}2 & 2\\2 & -4\\-2 & 1\\0 & -1\\0 & 1\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} \leqslant \begin{bmatrix}33\\8\\5\\-1\\8\end{bmatrix}$$

Derive the dual linear program using Lagrange duality.

7.7 Consider the quadratic program illustrated in Figure 7.4,

$$\min_{\boldsymbol{x}\in\mathbb{R}^2} \frac{1}{2}\begin{bmatrix}x_1\\x_2\end{bmatrix}^\top \begin{bmatrix}2 & 1\\1 & 4\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} + \begin{bmatrix}5\\3\end{bmatrix}^\top \begin{bmatrix}x_1\\x_2\end{bmatrix}$$
$$\text{subject to } \begin{bmatrix}1 & 0\\-1 & 0\\0 & 1\\0 & -1\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} \leqslant \begin{bmatrix}1\\1\\1\\1\end{bmatrix}$$

Derive the dual quadratic program using Lagrange duality.

7.8 Consider the following convex optimization problem

$$\begin{aligned}\min_{\boldsymbol{w}\in\mathbb{R}^D} \quad & \frac{1}{2}\boldsymbol{w}^\top\boldsymbol{w}\\ \text{subject to} \quad & \boldsymbol{w}^\top\boldsymbol{x} \geqslant 1\,.\end{aligned}$$

Derive the Lagrangian dual by introducing the Lagrange multiplier λ.

7.9 Consider the negative entropy of $\boldsymbol{x} \in \mathbb{R}^D$,

$$f(\boldsymbol{x}) = \sum_{d=1}^{D} x_d \log x_d\,.$$

Derive the convex conjugate function $f^*(s)$, by assuming the standard dot product.

Hint: Take the gradient of an appropriate function and set the gradient to zero.

7.10 Consider the function

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^\top\boldsymbol{x} + c\,,$$

where $\boldsymbol{A}$ is strictly positive definite, which means that it is invertible. Derive the convex conjugate of $f(\boldsymbol{x})$.

Hint: Take the gradient of an appropriate function and set the gradient to zero.

7.11 The hinge loss (which is the loss used by the support vector machine) is given by

$$L(\alpha) = \max\{0, 1-\alpha\}\,,$$

If we are interested in applying gradient methods such as L-BFGS, and do not want to resort to subgradient methods, we need to smooth the kink in the hinge loss. Compute the convex conjugate of the hinge loss $L^*(\beta)$ where β is the dual variable. Add a ℓ_2 proximal term, and compute the conjugate of the resulting function

$$L^*(\beta) + \frac{\gamma}{2}\beta^2\,,$$

where γ is a given hyperparameter.