

**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.

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.

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.

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

For convex functions all local minima are global minimum.

## 7.1 Optimization Using Gradient Descent

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

$$\min_{\mathbf{x}} f(\mathbf{x}), \quad (7.4)$$

where  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  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 ( $f(\mathbf{x}) = c$  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.

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

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

for a small *step-size*  $\gamma \geq 0$ , then  $f(\mathbf{x}_1) \leq f(\mathbf{x}_0)$ . 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  $f(\mathbf{x}_*)$  of a function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$ ,  $\mathbf{x} \mapsto f(\mathbf{x})$ , we start with an initial guess  $\mathbf{x}_0$  of the parameters we wish to optimize and then iterate according to

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

For suitable step-size  $\gamma_i$ , the sequence  $f(\mathbf{x}_0) \geq f(\mathbf{x}_1) \geq \dots$  converges to a local minimum.

### Example 7.1

Consider a quadratic function in two dimensions

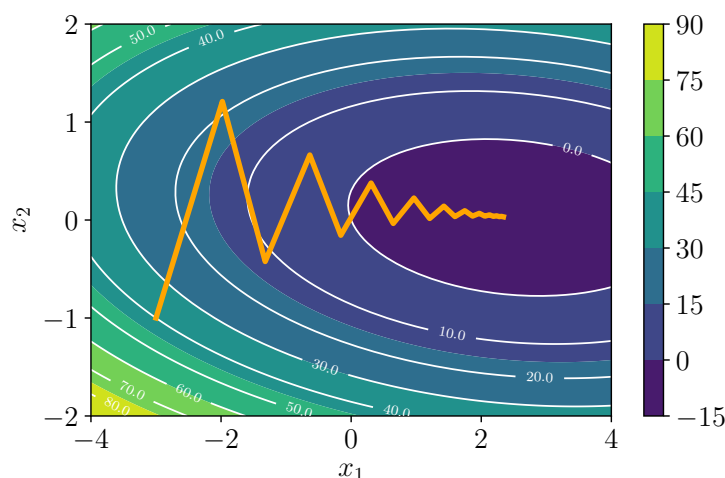
$$f\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \frac{1}{2} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^\top \begin{bmatrix} 2 & 1 \\ 1 & 20 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 5 \\ 3 \end{bmatrix}^\top \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad (7.7)$$

with gradient

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

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

We use the convention of row vectors for gradients.



**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  $\mathbf{x}_0$  into (7.8) with  $\gamma = 0.085$ ) that the negative gradient at  $\mathbf{x}_0$  points north and east, leading to  $\mathbf{x}_1 = [-1.98, 1.21]^\top$ . Repeating that argument gives us  $\mathbf{x}_2 = [-1.32, -0.42]^\top$ , 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.  $\diamond$

### 7.1.1 Step-size

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.

The step-size is also called the learning rate.

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  $\mathbf{Ax} = \mathbf{b}$ , in practice we solve  $\mathbf{Ax} - \mathbf{b} = \mathbf{0}$  approximately by finding  $\mathbf{x}_*$  that minimizes the squared error

$$\|\mathbf{Ax} - \mathbf{b}\|^2 = (\mathbf{Ax} - \mathbf{b})^\top (\mathbf{Ax} - \mathbf{b}) \quad (7.9)$$

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

$$\nabla_{\mathbf{x}} = 2(\mathbf{Ax} - \mathbf{b})^\top \mathbf{A}. \quad (7.10)$$

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  $\mathbf{Ax} = \mathbf{b}$ , gradient descent may converge slowly. The speed of convergence of gradient descent is dependent on the *condition number*  $\kappa = \frac{\sigma(\mathbf{A})_{\max}}{\sigma(\mathbf{A})_{\min}}$ , which is the ratio of the maximum to the minimum singular value (Section 4.5) of  $\mathbf{A}$ . 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  $\mathbf{Ax} = \mathbf{b}$ , one could instead solve  $\mathbf{P}^{-1}(\mathbf{Ax} - \mathbf{b}) = \mathbf{0}$ , where  $\mathbf{P}$  is called the *preconditioner*. The goal is to design  $\mathbf{P}^{-1}$  such that  $\mathbf{P}^{-1}\mathbf{A}$  has a better condition number, but at the same time  $\mathbf{P}^{-1}$  is easy to compute. For further information on gradient descent, preconditioning, and convergence we refer to Boyd and Vandenberghe (2004, chapter 9).  $\diamond$

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.

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

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

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

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

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

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  $n = 1, \dots, N$  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

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

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

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

where  $\mathbf{x}_n \in \mathbb{R}^D$  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

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

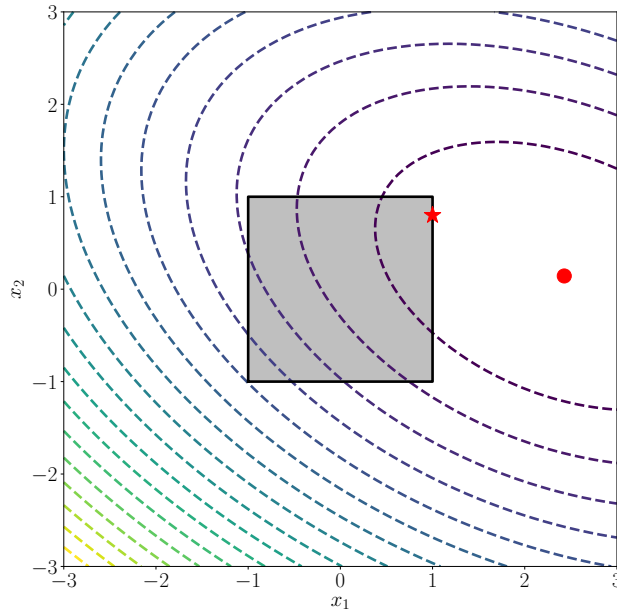
for a suitable step-size parameter  $\gamma_i$ . Evaluating the sum gradient may require expensive evaluations of the gradients from all individual functions  $L_n$ . When the training set is enormous and/or no simple formulas exist, evaluating the sums of gradients becomes very expensive.

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  $n = 1, \dots, N$ , 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).  $\diamond$

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 ( $-1 \leq x \leq 1$  and  $-1 \leq y \leq 1$ ) 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

$$\min_{\mathbf{x}} f(\mathbf{x}), \quad (7.16)$$

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

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

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

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

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

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

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

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  $\lambda_i \geq 0$  corresponding to each inequality constraint respectively (Boyd and Vandenberghe, 2004, chapter 4) so that

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

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

where in the last line we have concatenated all constraints  $g_i(\mathbf{x})$  into a vector  $\mathbf{g}(\mathbf{x})$ , 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  $\mathbf{x}$  (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)

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

primal problem  
Lagrangian dual  
problem

is known as the *primal problem*, corresponding to the primal variables  $\mathbf{x}$ . The associated *Lagrangian dual problem* is given by

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

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

*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  $\varphi(\mathbf{x}, \mathbf{y})$ , the maximin is less than the minimax, i.e.,

$$\max_{\mathbf{y}} \min_{\mathbf{x}} \varphi(\mathbf{x}, \mathbf{y}) \leq \min_{\mathbf{x}} \max_{\mathbf{y}} \varphi(\mathbf{x}, \mathbf{y}). \quad (7.23)$$



This inequality can be proved by considering the inequality

$$\text{For all } \mathbf{x}, \mathbf{y} \quad \min_{\mathbf{x}} \varphi(\mathbf{x}, \mathbf{y}) \leq \max_{\mathbf{y}} \varphi(\mathbf{x}, \mathbf{y}). \quad (7.24)$$

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

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

$$J(\mathbf{x}) = \max_{\lambda \geq 0} \mathcal{L}(\mathbf{x}, \lambda). \quad (7.25)$$

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

$$\min_{\mathbf{x} \in \mathbb{R}^d} \max_{\lambda \geq 0} \mathcal{L}(\mathbf{x}, \lambda). \quad (7.26)$$

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

$$\min_{\mathbf{x} \in \mathbb{R}^d} \max_{\lambda \geq 0} \mathcal{L}(\mathbf{x}, \lambda) \geq \max_{\lambda \geq 0} \min_{\mathbf{x} \in \mathbb{R}^d} \mathcal{L}(\mathbf{x}, \lambda). \quad (7.27)$$

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

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

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

We can model equality constraints by replacing them with two inequality constraints. That is for each equality constraint  $h_j(\mathbf{x}) = 0$  we equivalently replace it by two constraints  $h_j(\mathbf{x}) \leq 0$  and  $h_j(\mathbf{x}) \geq 0$ . 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.  $\diamond$

### 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.

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

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

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 non-convex function, and Figure 7.3 shows a convex function. Another convex function is shown in Figure 7.7.

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

$$f(\theta \mathbf{x} + (1 - \theta)\mathbf{y}) \leq \theta f(\mathbf{x}) + (1 - \theta)f(\mathbf{y}). \quad (7.30)$$

*Remark.* A *concave function* is the negative of a convex function.  $\diamond$

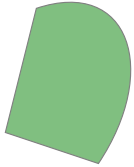
The constraints involving  $g(\cdot)$  and  $h(\cdot)$  in (7.28) truncate functions at a scalar value, resulting in sets. Another relation between convex functions and convex sets is to consider the set obtained by “filling in” a convex function. A convex function is a bowl-like object, and we imagine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

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

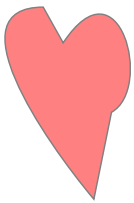
convex optimization  
problem  
strong duality

convex set

**Figure 7.5** Example of a convex set.



**Figure 7.6** Example of a nonconvex set.



convex function  
concave function

epigraph