A PARTICULAR CLASS OF ITERATIVE ALGORITHMS TO COMPUTE

$$\min_{x \in \mathbb{R}^n} f(x)$$

↳ objective function

DESCENT METHODS

$$x_k = G(x_{k-1}) \rightarrow$$

$$\rightarrow \quad \underline{x}_k = \underline{x}_{k-1} + \alpha_{k-1} \underline{p}_{k-1}$$

x_{k-1} previous iterate $\in \mathbb{R}^n$

$\underline{p}_{k-1} \in \mathbb{R}^n \rightarrow$ search direction

$\alpha_{k-1} \in \mathbb{R}^+ \rightarrow$ step length

$p_{k-1} \rightarrow$ also called descent direction — why?

p_{k-1} is chosen so that:

$$f(x_k) < f(x_{k-1})$$

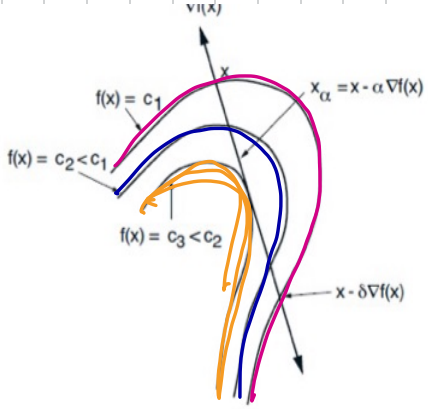
Definition of descent direction:

p is a descent direction in x for f if exists $\bar{\alpha} > 0$:

$$f(x + \alpha p) < f(x) \quad \forall \alpha \in (0, \bar{\alpha}]$$

$x = x_{k-1}$ $\quad x + \alpha p = x_k$

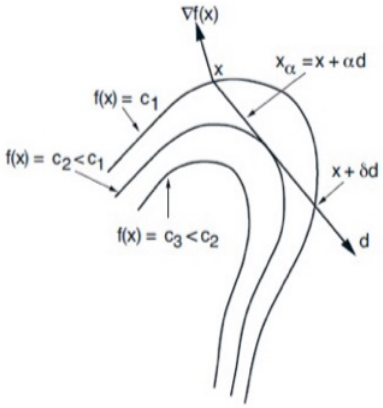
level curve → $\{x \in \mathbb{R}^n : f(x) = c\}$



If $\nabla f(x) \neq 0$, there is an interval $(0, \delta)$ of stepsizes such that

$$f\big(x - \alpha\nabla f(x)\big) < f(x)$$

for all $\alpha \in (0, \delta)$.



If d makes an angle with $\nabla f(x)$ that is greater than 90 degrees,

$$\nabla f(x)'d < 0,$$

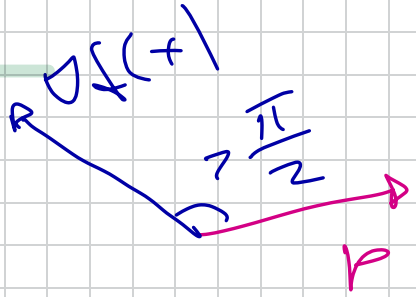
there is an interval $(0, \delta)$ of stepsizes such that $f(x + \alpha d) < f(x)$ for all $\alpha \in (0, \delta)$.

Property:

If f is continuous and differentiable in $\underline{x}$ and $\underline{p} \in \mathbb{R}^n$, $p \neq 0$, so that:

$$p^T \cdot \nabla f(x) < 0$$

then p is a descent direction for f in x.



$p = -\nabla f(x) \Rightarrow p$ is a descent direction for f in x.

Example

$$f(x) = x^2 + 3y^2 - 3xy$$

$f: \mathbb{R}^2 \to \mathbb{R}$

$\bar{x} = (1, 1)$ $\quad p = (1, 0)$

Is p a descent direction for f in $\bar{x}$?

$$\nabla f(x) = (2x - 3y, 6y - 3x)$$

$$\nabla f(\bar{x}) = (-1, 3)$$

$$p \cdot \nabla f^T(\bar{x}) = (1, 0) \cdot \begin{pmatrix} -1 \\ 3 \end{pmatrix} = -1 < 0$$

p is a descent direction for $\bar{x}$.

- If $p \cdot \nabla f^T(\bar{x}) > 0 \Rightarrow p$ is not a descent direction for f in $\bar{x}$

- If $p \cdot \nabla f^T(\bar{x}) = 0 \Rightarrow p$ can be a descent direction or not.

$$\underline{x}_{k+1} = \underline{x}_k + \alpha_k \underline{p}_k$$

REMARK

If p is a descent direction for f in x_k, the choice α_k:

$$f(x_k + \alpha_k p_k) = f(x_{k+1}) < f(x_k)$$

IS NOT SUFFICIENT TO GUARANTEE that $\lim_{k \to \infty} x_k = x^*$

HOW TO CHOOSE α_k (STEP LENGHT)

- Common choice in ML is α_k constant $\forall k$.

  $\alpha_k = \alpha$

  $\Rightarrow$ this does not guarantee the convergence.

- Inexact search

  a) α_k produces a "sufficient" decrease of f in x_k

  b) α_k produces a "sufficient" shift for x_{k-1} ($\| x_{k-1} - x_k \|$ sufficient)

BACKTRACKING PROCEDURE

- Set a "tentative value" $\bar{\alpha} = 1$
- Check if

$$f(x_k + \bar{\alpha} p_k) \leq f(x_k) + c_1 \bar{\alpha} \nabla f(x_k)^T p_k$$

x_{k+1}

$c_1 \in (0, 1)$

$\nabla f(x_k)^T p_k < 0$

If not true $\rightarrow \bar{\alpha} = \bar{\alpha} \cdot \rho$

$\rho = \frac{1}{2}$ i.e. $\rho < 1$

Backtracking algorithm

Set $\alpha = \bar{\alpha} = 1$

Choose $\rho < 1$, $c_1 \in (0,1)$

$j = 0$

while $f(x_k + \alpha p_k) > f(x_k) + c_1 \alpha \nabla f(x_k)^T p_k$

$\alpha = \alpha \cdot \rho$

$j = j + 1$

if $\alpha <$ Tolerance ($\sim 10^{-8}$) stop the algorithm

result: $\alpha_k = \alpha$

if $j = j^*$ then stop

Theorem CONVERGENCE

Consider a descent method

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

where α_k is chosen by a backtracking procedure.

Thesis:

$$\lim_{k \to \infty} x_k = x^*, \quad x^* \text{ local minimum.}$$

GRADIENT DESCENT ALGORITHM

Input: $f, \nabla f, x_0,$ tolerances

$k = 0$,

while stopping conditions are not satisfied

- α { const. α / backtracking
- $x = x_0 - \alpha_k \nabla f(x_0)$
- $x_0 = x$
- $k = k+1$

output: x

$p = -\nabla f(x)$

x^* local minimum $\Rightarrow \nabla f(x^*) = 0$

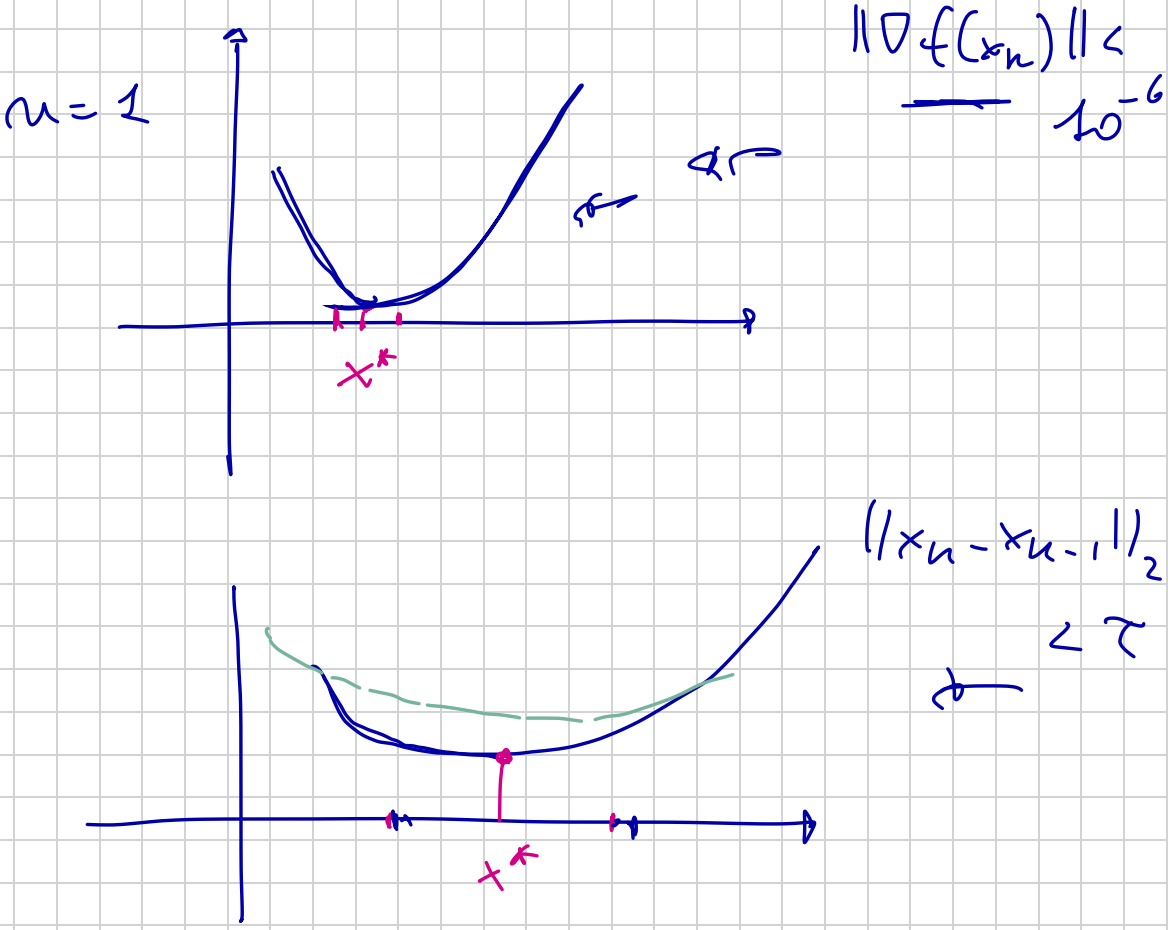
$\Downarrow$

1) $\|\nabla f(x_k)\| < \tau$ (absolute)

1b) $\dfrac{\|\nabla f(x_k)\|}{\|\nabla f(x_0)\|} < \tau_1$ (relative)

2) $\|x_k - x_{k-1}\| < \nu$ (absolute)

2b) $\dfrac{\|x_k - x_{k-1}\|}{\|x_{k-1}\|} < \nu_1$ (relative)



3) K = MAXIMUM NUMBER of ITERATIONS

IT DOES NOT GUARANTEE THAT THE SOLUTION IS CLOSE TO THE MINIMUM.

HOW TO CHOOSE x_0

x_0: influence the result →

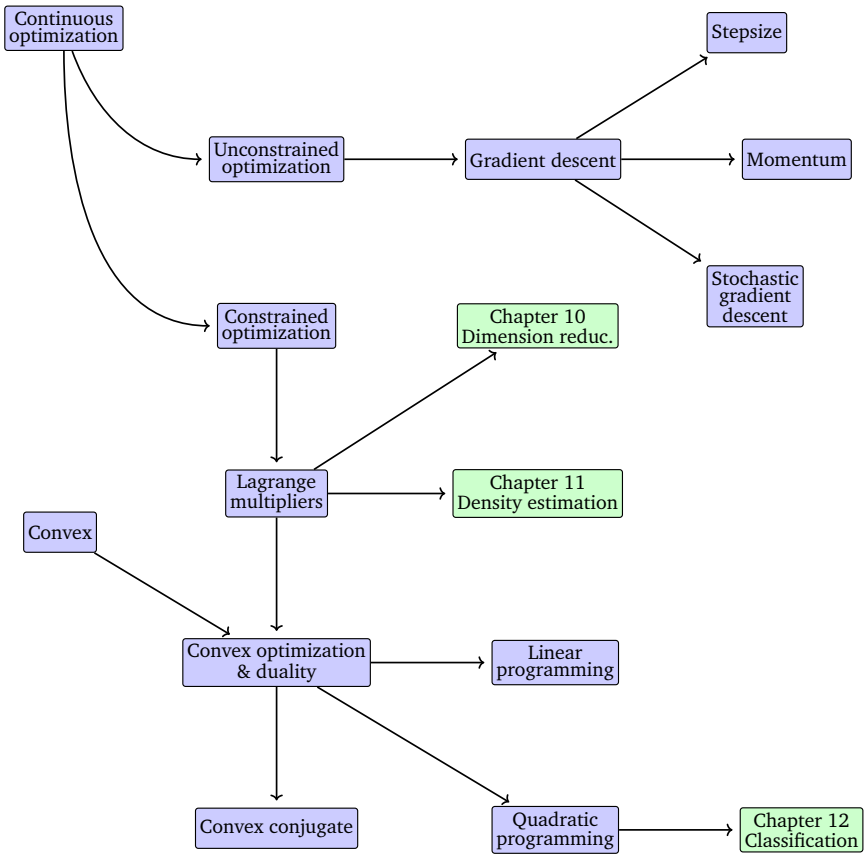


Figure 7.1 A mind map of the concepts related to optimization, as presented in this chapter. There are two main ideas: gradient descent and convex optimization.

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

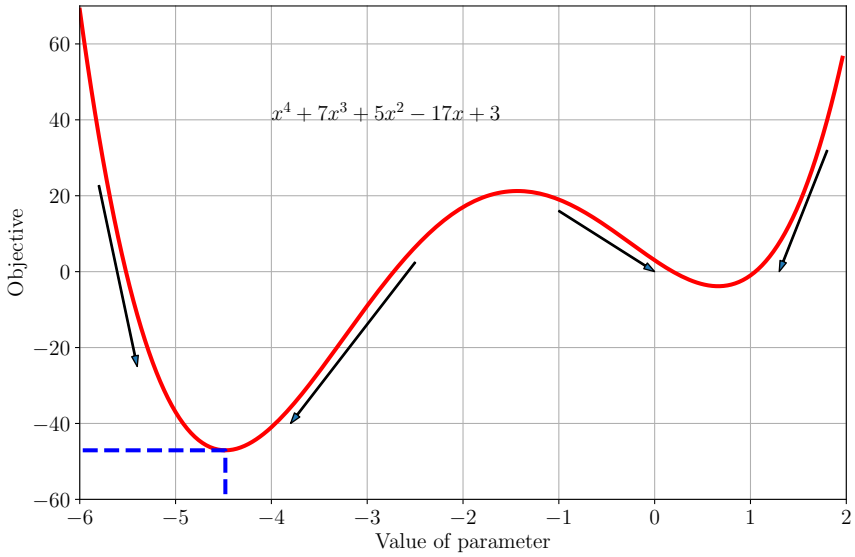


Figure 7.2 Example objective function. Negative gradients are indicated by arrows, and the global minimum is indicated by the dashed blue line.

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.

see notes

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

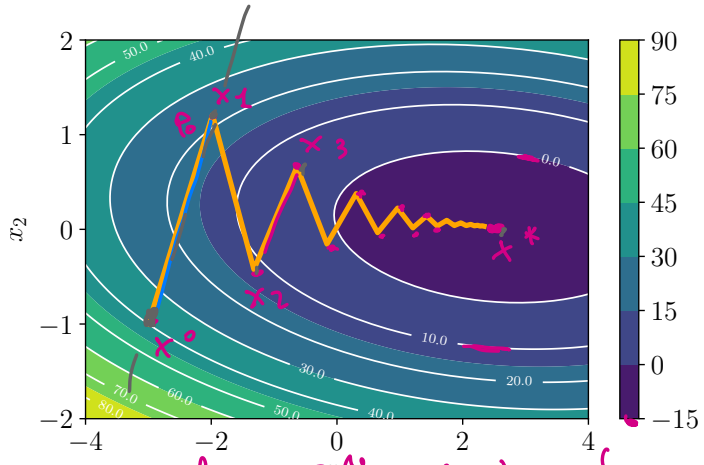


Figure 7.3 Gradient descent on a two-dimensional quadratic surface (shown as a heatmap). See Example 7.1 for a description.

(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

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.

The step-size is also called the learning rate.

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

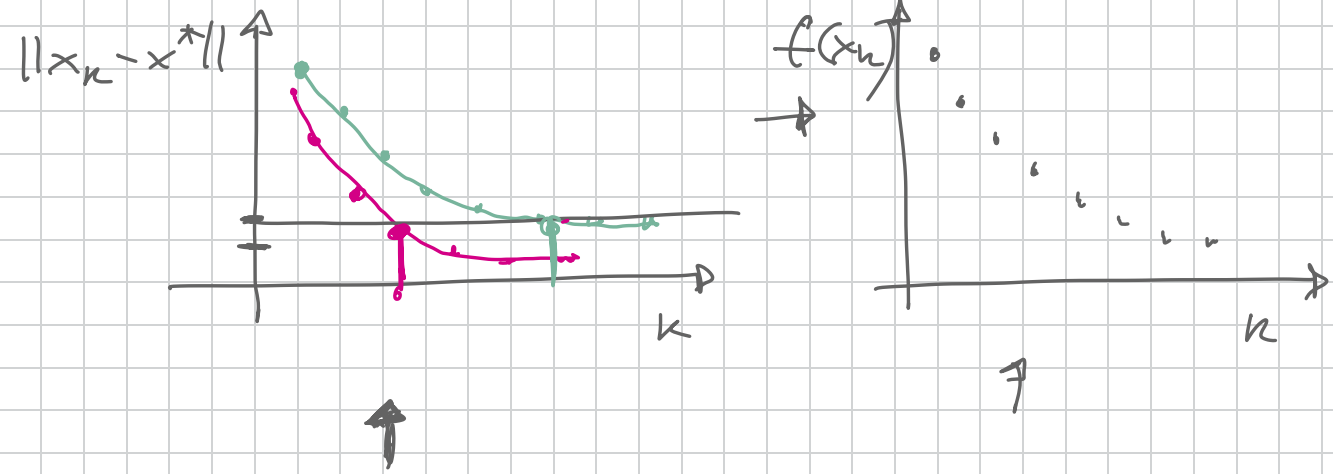
As illustrated in Figure 7.3, the convergence of gradient descent may be very slow if the curvature of the optimization surface is such that there are regions that are poorly scaled. The curvature is such that the gradient descent steps hops between the walls of the valley and approaches the optimum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

CONVERGENCE SPEED

→ Evaluate how fast the algorithm approximates the solution.



The method has Q convergence speed if:

$$\|x_k - x^*\| < C \|x_{k-1} - x^*\|^Q \quad , k > \bar{k}$$

$0 < Q$

1) IF $Q = 1 \Rightarrow$ LINEAR CONVERGENCE

$C < 1$

2) IF $1 < Q < 2 \Rightarrow$ SUPER LINEAR CONVERGENCE

3) IF $Q = 2 \Rightarrow$ QUADRATIC CONVERGENCE

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

$$\min_{\theta} L(\theta) = \min_{\theta} \sum_{n=1}^{N} L_n(\theta)$$

$$\nabla L(\theta) = \nabla \sum_{n=1}^{N} L_n(\theta) =$$

$$= \sum_{n=1}^{N} \nabla L_n(\theta)$$

$$\nabla L(\theta) \simeq \sum_{n \in S} \nabla L_n(\theta)$$

$$S = \{n_1, n_2 \dots n_K\} \subset \{1, 2 \dots N\}$$

$N = 100 \qquad K = 5$

S MINIBATCH of size K.

$$\nabla L_{n_2}(\theta) + \nabla L_{n_1}(\theta) \dots + \nabla L_{n_K}(\theta)$$

- AT each iteration, we choose a different minibatch of the same size.

$N = 10$ $\quad K = 1$

$K = 5$

→ EPOCH → number of iterations to select all the values in $\{1 \dots N\}$

STEP LENGTH α_K → LEARNING RATE

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),

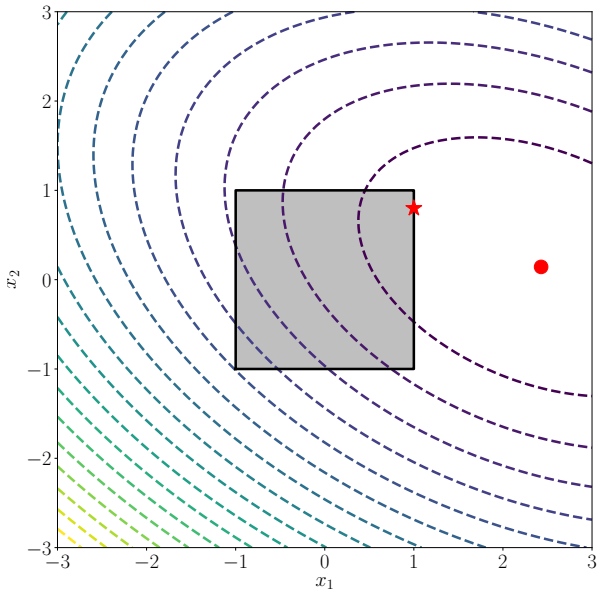


Figure 7.4 Illustration of constrained optimization. The unconstrained problem (indicated by the contour lines) has a minimum on the right side (indicated by the circle). The box constraints ($-1 \leqslant x \leqslant 1$ and $-1 \leqslant y \leqslant 1$) require that the optimal solution is within the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

Lagrangian dual problem

is known as the *primal problem*, corresponding to the primal variables x. The associated *Lagrangian dual problem* is given by

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

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.

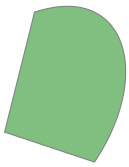
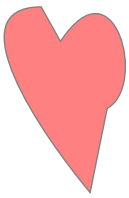


Figure 7.6 Example of a nonconvex set.



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

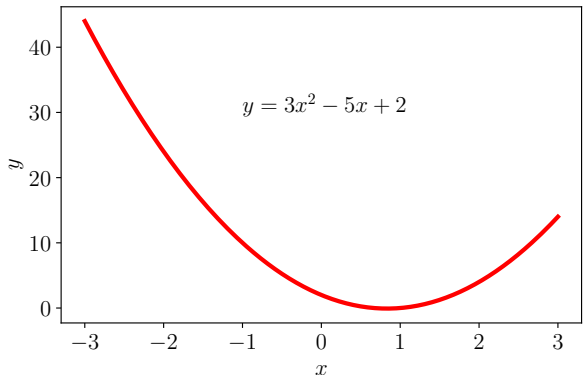


Figure 7.7 Example of a convex function.

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

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

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

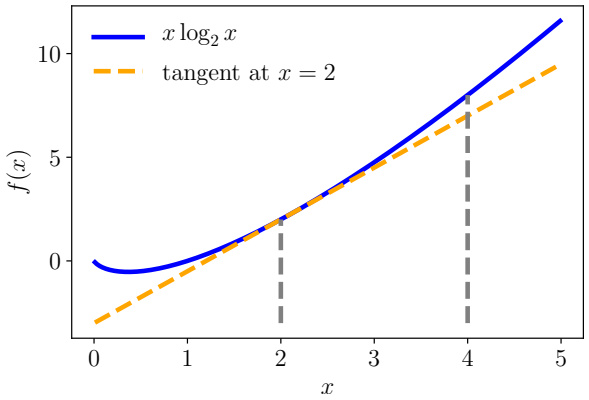


Figure 7.8 The negative entropy function (which is convex) and its tangent at $x = 2$.

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

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

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.

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

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