Chapter 1

Introduction

This book provides an introduction to *continuous optimization*, the minimization of continuous, real-valued functions of real variables over convex domains. Continuous optimization covers a broad class of problems that admits many important subclasses. Here, we are primarily motivated on the sorts of problems that arise in machine learning, statistics, and data analysis more broadly.

It's useful to focus on a particular suite of examples to show how what might appear to be a narrow focus ends up touching on a variety of optimization concerns and including a wide set of mathematical formalisms.

The typical optimization problem in data analysis is to find a model that agrees with some collected data set but also adheres to some structural constraints that reflect our beliefs about what a good model should be. The data set in a typical analysis problem consists of m objects:

$$\mathcal{D} := \{(a_j, y_j),\ j = 1, 2, \dots, m\}, \tag{1.1}$$

where a_j is a vector (or matrix) of *features* and y_j is an *observation* or *label.* (Each pair (a_j, y_j) has the same size and shape for all $j = 1, 2, \dots, m$.) The analysis task then consists of discovering a function ϕ such that $\phi(a_j) \approx y_j$ for most $j = 1, 2, \dots, m$. The process of discovering the mapping ϕ is often called "learning" or "training."

The function ϕ is often defined in terms of a vector or matrix of parameters, which we denote by x or X. (Other notation also appears below.) With these parametrizations, the problem of identifying ϕ becomes a data-fitting problem: "Find the parameters x defining ϕ such that $\phi(a_j) \approx y_j$, $j = 1, 2, \dots, m$ in some optimal sense." Once we come up with a definition of the term "optimal," we have an optimization problem. Many such optimization formulations have objective functions of the "summation" type

$$\mathcal{L}_{\mathcal{D}}(x) := \frac{1}{m} \sum_{j=1}^{m} \ell(a_j, y_j; x)\,. \tag{1.2}$$

Here, the function ℓ here represents a *loss* paid for not properly aligning our prediction $\phi(a)$ with y. x is the vector of parameters that determines ϕ. The objective $\mathcal{L}_{\mathcal{D}}(x)$ measures the average loss accrued over the entire data set when the parameter vector is equal to x.

One use of ϕ is to make predictions about future data items. Given another previously unseen item of data $\hat{a}$ of the same type as a_j, $j = 1, 2, \dots, m$, we predict that the label $\hat{y}$ associated with $\hat{a}$ would be $\phi(\hat{a})$. The mapping may also expose other structure and properties in the data set. For example, it may reveal that only a small fraction of the features in a_j are needed to reliably predict the label y_j. (This is known as *feature selection.*) The function ϕ or its parameter x may also reveal important structure in the data. For example, X could reveal a low-dimensional subspace that contains most of the a_j, or X could reveal a matrix with particular structure (low-rank, sparse) such that observations of X prompted by the feature vectors a_j yield results close to y_j.

Examples of labels y_j include the following.

- A real number, leading to a *regression* problem.
- A label, say $y_j \in \{1, 2, \ldots, M\}$ indicating that a_j belongs to one of M classes. This is a *classification* problem. We have $M = 2$ for binary classification and $M > 2$ for multiclass classification.
- Null. Some problems only have feature vectors a_j and no labels. In this case, the data analysis task may consist of grouping the a_j into clusters (where the vectors within each cluster are deemed to be functionally similar), or identifying a low-dimensional subspace (or a collection of low-dimensional subspaces) that approximately contains the a_j. Such problems require the labels y_j to be learnt, alongside the function ϕ. For example, in a clustering problem, y_j could represent the cluster to which a_j is assigned.

Even after cleaning and preparation, the setup above may contain many complications that need to be dealt in formulating the problem in rigorous mathematical terms. The quantities (a_j, y_j) may contain noise, or may be otherwise corrupted. We would like the mapping ϕ to be robust to such errors. There may be *missing data*: parts of the vectors a_j may be missing, or we may not know all the labels y_j. The data may be arriving in *streaming* fashion rather than being available all at once. In this case, we would learn ϕ in an *online* fashion.

One particular consideration is that we wish to avoid *overfitting* the model to the data set $\mathcal{D}$ in (1.1). The particular data set $\mathcal{D}$ available to us can often be thought of as a finite sample drawn from some underlying larger (often infinite) collection of data, and we wish the function ϕ to perform well on the unobserved data points as well as the observed subset $\mathcal{D}$. In other words, we want ϕ to be not too sensitive to the particular sample $\mathcal{D}$ that is used to define empirical objective functions such as (1.2). The optimization formulation can be modified in various ways to achieve this goal, by the inclusion of constraints or penalty terms that limit some measure of "complexity" of the function (such techniques are typically called *regularization*). So, in sum, a generic "master problem" that balances data fit, model complexity, and model structure is

$$\begin{array}{ll} \text{minimize} & \frac{1}{m}\sum_{j=1}^{m} \ell(a_j, y_j; x) + \lambda\, \mathrm{pen}(x) \\ \text{subject to} x \in \Omega & \end{array} . \tag{1.3}$$

The problem seeks to minimize a cost function subject to constraints. The first term in the cost function is the average loss over the data set. The second term is a penalty term that aims to encourage models with low complexity. The scalar $\lambda > 0$ is called a *regularization parameter* and lets the practitioner tune between fitting the data and choosing a simple model. The set Ω in the constraints is the set of parameters that we would deem as valid solutions.

We now dive into a variety of special cases of problem (1.3), showing that a wide variety of applications can be formulated in this fashion and that there might not be a single simple algorithm appropriate for solving all instances.

1.1 Least Squares

Probably the oldest and best-known data analysis problem is linear least squares. Here, the data points (a_j, y_j) lie in $\mathbb{R}^n \times \mathbb{R}$, and we solve

$$\min_x \frac{1}{2m} \sum_{j=1}^{m} (a_j^T x - y_j)^2 = \frac{1}{2m} \|Ax - y\|_2^2, \tag{1.4}$$

where A the matrix whose rows are a_j^T, $j = 1, 2, \ldots, m$ and $y = (y_1, y_2, \ldots, y_m)^T$. In the terminology above, the function ϕ is defined by $\phi(a) := a^T x$. (We could also introduce a nonzero intercept by adding an extra parameter $\beta \in \mathbb{R}$ and defining $\phi(a) := a^T x + \beta$.) This formulation can be motivated statistically, as a maximum-likelihood estimate of x when the observations y_j are exact but for i.i.d. Gaussian noise. We can add a variety of penalty functions to this basic least squares problem to impose desirable structure on x and hence on ϕ. For example, Ridge regression adds a squared ℓ_2-norm penalty, resulting in

$$\min_x \frac{1}{2m} \|Ax - y\|_2^2 + \lambda \|x\|_2^2, \quad \text{for some parameter } \lambda > 0\,.$$

Ridge regression yields a solution x with less sensitivity to perturbations in the data (a_j, y_j). The LASSO formulation

$$\min_x \frac{1}{2m} \|Ax - y\|_2^2 + \lambda \|x\|_1 \tag{1.5}$$

tends to yield solutions x that are sparse, that is, containing relatively few nonzero components [33]. This formulation performs feature selection: The locations of the nonzero components in x reveal those components of a_j that are instrumental in determining the observation y_j. Besides its statistical appeal — predictors that depend on few features are potentially simpler and more comprehensible than those depending on many features — feature selection has practical appeal in making predictions about future data. Rather than gathering all components of a new data vector $\hat{a}$, we need to find only the "selected" features, since only these are needed to make a prediction.

The LASSO formulation (1.5) is an important prototype for many problems in data analysis, in that it involves a regularization term $\lambda\|x\|_1$ that is nonsmooth and convex, but with relatively simple structure that can potentially be exploited by algorithms.

1.2 Matrix Factorization Problems

There are a variety of data analysis problems that require estimating a low-rank matrix from some sparse collection of data. Such problems can be formulated as natural extension of least-squares to problems in which the data a_j are naturally represented as matrices rather than vectors.

Changing notation slightly, we suppose that each A_j is an $n \times p$ matrix, and we seek another $n \times p$ matrix X that solves

$$\min_X \frac{1}{2m} \sum_{j=1}^{m} (\langle A_j, X \rangle - y_j)^2, \tag{1.6}$$

where $\langle A, B \rangle := \text{trace}(A^T B)$. Here we can think of the A_j as "probing" the unknown matrix X. Commonly considered types of observations are random linear combinations (where the elements of A_j are selected i.i.d. from some distribution) or single-element observations (in which each A_j has 1 in a single location and zeros elsewhere). A regularized version of (1.6), leading to solutions X that are low-rank, is

$$\min_X \frac{1}{2m} \sum_{j=1}^m (\langle A_j, X \rangle - y_j)^2 + \lambda \|X\|_*, \tag{1.7}$$

where $\|X\|_*$ is the nuclear norm, which is the sum of singular values of X [28]. The nuclear norm plays a role analogous to the ℓ_1 norm in (1.5): where as the ℓ_1 norm favors sparse vectors, the nuclear norm favors low-rank matrices. Although the nuclear norm is a somewhat complex nonsmooth function, it is at least convex, so that the formulation (1.7) is also convex. This formulation can be shown to yield a statistically valid solution when the true X is low-rank and the observation matrices A_j satisfy a "restricted isometry" property, commonly satisfied by random matrices, but not by matrices with just one nonzero element. The formulation is also valid in a different context, in which the true X is incoherent (roughly speaking, it does not have a few elements that are much larger than the others), and the observations A_j are of single elements [10].

In another form of regularization, the matrix X is represented explicitly as a product of two "thin" matrices L and R, where $L \in \mathbb{R}^{n \times r}$ and $R \in \mathbb{R}^{p \times r}$, with $r \ll \min(n, p)$. We set $X = LR^T$ in (1.6) and solve

$$\min_{L,R} \frac{1}{2m} \sum_{j=1}^m (\langle A_j, LR^T \rangle - y_j)^2. \tag{1.8}$$

In this formulation, the rank r is "hard-wired" into the definition of X, so there is no need to include a regularizing term. This formulation is also typically much more compact than (1.7); the total number of elements in (L, R) is $(n + p)r$, which is much less than np. A disadvantage is that it is nonconvex. An active line of current research, pioneered in [9] and also drawing on statistical sources, shows that the nonconvexity is benign in many situations, and that under certain assumptions on the data (A_j, y_j), $j = 1, 2, \ldots, m$ and careful choice of algorithmic strategy, good solutions can be obtained from the formulation (1.8). A clue to this good behavior is that although this formulation is nonconvex, it is in some sense an approximation to a tractable problem: If we have a complete observation of X, then a rank-r approximation can be found by performing a singular value decomposition of X, and defining L and R in terms of the r leading left and right singular vectors.

Some applications in computer vision, chemometrics, and document clustering require us to find factors L and R like those in (1.8) in which all elements are nonnegative. If the full matrix $Y \in \mathbb{R}^{n \times p}$ is observed, this problem has the form

$$\min_{L,R} \|LR^T - Y\|_F^2, \quad \text{subject to } L \geq 0, \; R \geq 0$$

and is called *nonnegative matrix factorization.*

1.3 Support Vector Machines

Classification via support vector machines (SVM) is a classical optimization problem in machine learning with its origin in the 1960s. This problem takes as input data (a_j, y_j) with $a_j \in \mathbb{R}^n$ and

$y_j \in \{-1, 1\}$, and seeks a vector $x \in \mathbb{R}^n$ and a scalar $\beta \in \mathbb{R}$ such that

$$a_j^T x - \beta \geq 1 \quad \text{when } y_j = +1; \tag{1.9a}$$
$$a_j^T x - \beta \leq -1 \quad \text{when } y_j = -1. \tag{1.9b}$$

Any pair (x, β) that satisfies these conditions defines a *separating hyperplane* in $\mathbb{R}^n$, that separates the "positive" cases $\{a_j \,|\, y_j = +1\}$ from the "negative" cases $\{a_j \,|\, y_j = -1\}$. Among all separating hyperplanes, the one that minimizes $\|x\|^2$ is the one that maximizes the *margin* between the two classes, that is, the hyperplane whose distance to the nearest point a_j of either class is greatest.

We can formulate the problem of finding a separating hyperplane as an optimization problem by defining an objective with the summation form (1.2):

$$H(x, \beta) = \frac{1}{m} \sum_{j=1}^{m} \max(1 - y_j(a_j^T x - \beta), 0). \tag{1.10}$$

Note that the jth term in this summation is zero if the conditions (1.9) are satisfied, and positive otherwise. Even if no pair (x, β) exists for which $H(x, \beta) = 0$, a value (x, β) that minimizes (1.2) will be the one that comes as close as possible to satisfying (1.9), in some sense. A term $\lambda \|x\|_2^2$ (for some parameter $\lambda > 0$) is often added to (1.10), yielding the following regularized version:

$$H(x, \beta) = \frac{1}{m} \sum_{j=1}^{m} \max(1 - y_j(a_j^T x - \beta), 0) + \frac{1}{2}\lambda \|x\|_2^2. \tag{1.11}$$

Differing from our examples thus far, the SVM problem has a non-smooth loss function and a smooth regularizer.

If λ is sufficiently small, and if separating hyperplanes exist, the pair (x, β) that minimizes (1.11) is the maximum-margin separating hyperplane. The maximum-margin property is consistent with the goals of generalizability and robustness. For example, if the observed data (a_j, y_j) is drawn from an underlying "cloud" of positive and negative cases, the maximum-margin solution usually does a reasonable job of separating other empirical data samples drawn from the same clouds, whereas a hyperplane that passes close by several of the observed data points may not do as well (see Figure 1.1).

Often it is not possible to find a hyperplane that separates the positive and negative cases well enough to be useful as a classifier. One solution is to transform all of the raw data vectors a_j by a mapping ψ, then perform the support-vector-machine classification on the vectors $\psi(a_j)$, $j = 1, 2, \ldots, m$. The conditions (1.9) would thus be replaced by

$$\psi(a_j)^T x - \beta \geq 1 \quad \text{when } y_j = +1; \tag{1.12a}$$
$$\psi(a_j)^T x - \beta \leq -1 \quad \text{when } y_j = -1, \tag{1.12b}$$

leading to the following analog of (1.11):

$$H(x, \beta) = \frac{1}{m} \sum_{j=1}^{m} \max(1 - y_j(\psi(a_j)^T x - \beta), 0) + \frac{1}{2}\lambda \|x\|_2^2. \tag{1.13}$$

When transformed back to $\mathbb{R}^m$, the surface $\{a \,|\, \psi(a)^T x - \beta = 0\}$ is nonlinear and possibly disconnected, and is often a much more powerful classifier than the hyperplanes resulting from (1.11).

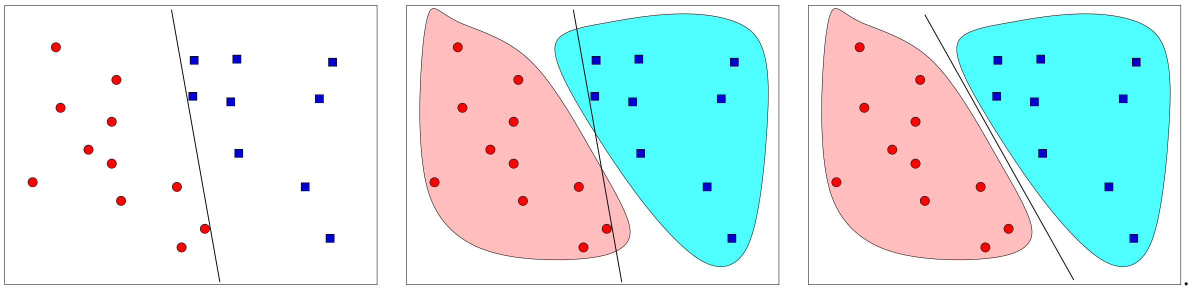

Figure 1.1: Linear support vector machine classification, with the one class represented by circles and the other by squares. One possible choice of separating hyperplane is shown at left. If the observed data is an empirical sample drawn from a cloud of underlying data points, this plane does not do well in separating the two clouds (middle). The maximum-margin separating hyperplane does better (right).

We note that the SVM is also naturally expressed as a minimization problem over a convex et. Indeed, by introducing artificial variables, the problem (1.13) (and (1.11)) can be formulated as a convex quadratic program, that is, a problem with a convex quadratic objective and linear constraints. By taking the dual of this problem, we obtain another convex quadratic program, in m variables:

$$\min_{\alpha \in \mathbb{R}^m} \frac{1}{2}\alpha^T Q \alpha - \mathbf{1}^T \alpha \quad \text{subject to } 0 \le \alpha \le \frac{1}{\lambda}\mathbf{1}, \;\; y^T\alpha = 0, \tag{1.14}$$

where

$$Q_{kl} = y_k y_l \psi(a_k)^T \psi(a_l), \quad y = (y_1, y_2, \dots, y_m)^T, \quad \mathbf{1} = (1, 1, \dots, 1)^T.$$

Interestingly, problem (1.14) can be formulated and solved without explicit knowledge or definition of the mapping ψ. We need only a technique to define the elements of Q. This can be done with the use of a *kernel function* $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, where $K(a_k, a_l)$ replaces $\psi(a_k)^T\psi(a_l)$ [5, 11]. This is the so-called "kernel trick." (The kernel function K can also be used to construct a classification function ϕ from the solution of (1.14).) A particularly popular choice of kernel is the Gaussian kernel:

$$K(a_k, a_l) := \exp(-\|a_k - a_l\|^2/(2\sigma)),$$

where σ is a positive parameter.

1.4 Logistic Regression

Logistic regression can be viewed as a variant of binary support-vector machine classification, in which rather than the classification function ϕ giving a unqualified prediction of the class in which a new data vector a lies, it returns an estimate of the *odds* of a belonging to one class or the other. We seek an "odds function" p parametrized by a vector $x \in \mathbb{R}^n$ as follows:

$$p(a; x) := (1 + \exp(a^T x))^{-1}, \tag{1.15}$$

and aim to choose the parameter x in so that

$$p(a_j; x) \approx 1 \quad \text{when } y_j = +1; \tag{1.16a}$$

$$p(a_j; x) \approx 0 \quad \text{when } y_j = -1. \tag{1.16b}$$

(Note the similarity to (1.9).) The optimal value of x can be found by maximizing a log-likelihood function:

$$L(x) := \frac{1}{m}\left[\sum_{j:y_j=-1}\log(1-p(a_j;x)) + \sum_{j:y_j=1}\log p(a_j;x)\right]. \tag{1.17}$$

We can perform feature selection using this model by introducing a regularizer $\lambda\|x\|_1$, as follows:

$$\max_x \frac{1}{m}\left[\sum_{j:y_j=-1}\log(1-p(a_j;x)) + \sum_{j:y_j=1}\log p(a_j;x)\right] - \lambda\|x\|_1, \tag{1.18}$$

where $\lambda > 0$ is a regularization parameter. As we see later, this term has the effect of producing a solution in which few components of x are nonzero, making it possible to evaluate $p(a;x)$ by knowing only those components of a that correspond to the nonzeros in x.

An important extension of this technique is to *multiclass* (or *multinomial*) logistic regression, in which the data vectors a_j belong to more than two classes. Such applications are common in modern data analysis. For example, in a speech recognition system, the M classes could each represent a *phoneme* of speech, one of the potentially thousands of distinct elementary sounds that can be uttered by humans in a few tens of milliseconds. A multinomial logistic regression problem requires a distinct odds function p_k for each class $k \in \{1,2,\dots,M\}$. These functions are parametrized by vectors $x_{[k]} \in \mathbb{R}^n$, $k = 1,2,\dots,M$, defined as follows:

$$p_k(a;X) := \frac{\exp(a^T x_{[k]})}{\sum_{l=1}^M \exp(a^T x_{[l]})}, \quad k = 1,2,\dots,M, \tag{1.19}$$

where we define $X := \{x_{[k]} \,|\, k = 1,2,\dots,M\}$. Note that for all a, we have $p_k(a) \in (0,1)$ for all $k = 1,2,\dots,M$ and that $\sum_{k=1}^M p_k(a) = 1$. The functions (1.19) are (somewhat colloquially) referred to as performing a "softmax" on the quantities $\{a^T x_{[l]} \,|\, l = 1,2,\dots,M\}$.

In the setting of multiclass logistic regression, the labels y_j are vectors in R^M, whose elements are defined as follows:

$$y_{jk} = \begin{cases} 1 & \text{when } a_j \text{ belongs to class } k, \\ 0 & \text{otherwise.} \end{cases} \tag{1.20}$$

Similarly to (1.16), we seek to define the vectors $x_{[k]}$ so that

$$p_k(a_j;X) \approx 1 \quad \text{when } y_{jk} = 1 \tag{1.21a}$$
$$p_k(a_j;X) \approx 0 \quad \text{when } y_{jk} = 0. \tag{1.21b}$$

The problem of finding values of $x_{[k]}$ that satisfy these conditions can again be formulated as one of maximizing a log-likelihood:

$$L(X) := \frac{1}{m}\sum_{j=1}^m \left[\sum_{\ell=1}^M y_{j\ell}(x_{[\ell]}^T a_j) - \log\left(\sum_{\ell=1}^M \exp(x_{[\ell]}^T a_j)\right)\right]. \tag{1.22}$$

"Group-sparse" regularization terms can be included in this formulation to select a set of features in the vectors a_j, common to each class, that distinguish effectively between the classes.

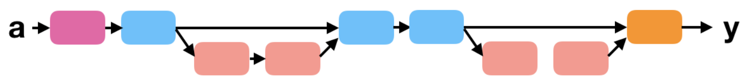

Figure 1.2: A deep neural network, showing connections between adjacent layers.

1.5 Deep Learning

Deep neural networks are often designed to perform the same function as multiclass logistic regression, that is, to classify a data vector a into one of M possible classes, where $M \geq 2$ is large in some key applications. The difference is that the mapping ϕ from data vector to prediction is now a nonlinear function, explicitly parameterized by a set of structured transformations.

The neural network shown in Figure 1.2 illustrates the basic ideas. In this figure, the data vector a_j enters at the bottom of the left of the network, each box represents a transformation that takes inputs and applies a nonlinear transformation of the data. We successively apply these nonlinear transformations according to the graph defined by this diagram. Each box has a set of its own parameters, and the collection of all of the parameters of all of the boxes comprises our optimization variable. The different colors here denote the fact that the types of transformations might differ, but we can compose them in whatever fashion suits our application.

A typical transformation, which converts the vector a_j^{l-1} at layer $l-1$ to vector a_j^l at layer l, is

$$a_j^l = \sigma(W^l a_j^{l-1} + g^l), \quad l = 1, 2, \dots, D,$$

where W^l is a matrix of dimension $|a_j^l| \times |a_j^{l-1}|$ and g^l is a vector of length $|a_j^l|$, σ is a *componentwise* nonlinear transformation, and D is the number of layer situated strictly between the bottom and top layers (referred to as "hidden layers"). The most common form of the function σ are the following, acting identically on each component $t \in \mathbb{R}$ of its input vector:

- Sigmoid: $t \to 1/(1 + e^{-t})$;
- Rectified Linear Unit: $t \to \max(t, 0)$;

Each node in the top layer corresponds to a particular class, and the output of each node corresponds to the odds of the input vector belonging to each class. As mentioned, the "softmax" operator is typically used to convert the transformed input vector in the second-top layer (layer D) to a set of odds at the top layer. Associated with each input vector a_j are labels y_{jk}, defined as in (1.20) to indicate which of the M classes that a_j belongs to.

The parameters in this neural network are the matrix-vector pairs (W^l, g^l), $l = 1, 2, \dots, D$ that transform the input vector a_j into its form a_j^D at the second-top layer, together with the parameters X of the softmax operation that takes place at the final (top) stage, where X is defined exactly as in the discussion of the previous section on multiclass logistic regression. We aim to choose all these parameters so that the network does a good job on classifying the training data correctly. Using the notation w for the layer-to-layer transformations, that is,

$$w := (W^1, g^1, W^2, g^2, \dots, W^D, g^D),$$

we can write the loss function for deep learning as follows:

$$L(w, X) := \frac{1}{m} \sum_{j=1}^{m} \left[\sum_{\ell=1}^{M} y_{j\ell} (x_{[\ell]}^T a_j^D(w)) - \log \left(\sum_{\ell=1}^{M} \exp(x_{[\ell]}^T a_j^D(w)) \right) \right]. \tag{1.23}$$

Here we write $a_j^D(w)$ to make explicit the dependence of a_j^D on the transformations w as well as on the input vector a_j. (We can view multiclass logistic regression as a special case of deep learning in which there are no hidden layers, so that $D = 0$, w is null, and $a_j^D = a_j$, $j = 1, 2, \ldots, m$.)

Neural networks in use for particular applications (in image recognition and speech recognition, for example, where they have been very successful) include many variants on the basic design above. These include restricted connectivity between layers (that is, enforcing structure on the matrices W^l, $l = 1, 2, \ldots, D$), layer arrangements that are more complex than the linear layout illustrated in Figure 1.2, with outputs coming from different levels, connections across non-adjacent layers, different componentwise transformations σ at different layers, and so on. Deep neural networks for practical applications are highly engineered objects.

The loss function (1.23) shares with many other applications the "summation" form (1.2), but it has several features that set it apart from the other applications discussed above. First, and possibly most important, it is *nonconvex* in the parameters w. There is reason to believe that the "landscape" of L is complex, with the global minimizer being exceedingly difficult to find. Second, the total number of parameters in (w, X) is usually very large. Effective training of deep learning classifiers typically requires a great deal of data and computation power. Huge clusters of powerful computers, often using multicore processors, GPUs, and even specially architected processing units, are devoted to this task.

1.6 Emphasis

The problems that we can formulate as (1.3) are varied: the cost functions might be convex or nonconvex, smooth or nonsmooth. But there are important features that they all share.

- They can be formulated as functions of *real variables*, which we typically arranged in a vector $\mathbb{R}^n$.

- The functions are continuous and often smooth. When nonsmoothness appears in the formulation, it does so in a structured way that can be exploited by the algorithm. Smoothness properties allow an algorithm to make good inferences about the behavior of the function on the basis of knowledge gained at points that have been visited previously.

- The objective is often made up in part of a summation of many terms, where each term depends on a single item of data.

- The objective is often a sum of two terms: a "loss term" (sometimes arising from a maximum likelihood expression for some statistical model) and a "regularization term" whose purpose is to impose structure and "generalizability" on the recovered model.

Our treatment emphasizes algorithms for solving the different kinds of problems discussed above, and the convergence properties of these algorithms. We pay attention to complexity guarantees, which are bounds on the amount of computational effort required to obtain solutions of a given accuracy. These bounds usually depend on fundamental properties of the objective function and the data that defines it, including the dimensions of the data set and the number of variables in the problem. This emphasis contrasts with much of the optimization literature, in which global convergence results do not usually involve complexity bound. (A notable exception is the analysis of interior-point methods; see [24, 36]).

At the same time, we try as much as possible to emphasize the practical concerns associated with solving these problems. There are a variety of trade-offs presented by any problem, and the optimizer has to evaluate which tools in her belt is most appropriate to use. On top of the problem formulation, it is imperative to account for the time budget for the task at hand, the sort of computer on where the problem will be solved, and the needed guarantees for the returned solution. Worst-case complexity guarantees are only a piece of the story here, and understanding the varied implementation heuristics are critical for building reliable solvers.

Notes and References

The examples in this chapter are drawn from the article [37] by one of the authors.