Comprehensive Data Analysis Collection

Norah Jones

2025-11-21

Table of contents

List of Figures

List of Tables

Preface

This is a Quarto book.

To learn more about Quarto books visit https://quarto.org/docs/books.

1 Introduction

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

Part I

Optimization Methods

2 Introduction to Optimization for Data Science

Fundamentals of Continuous Optimization

Chapter Overview

This chapter introduces the fundamentals of **continuous optimization algorithms** for data science applications. We explore how machine learning, statistics, and data analysis problems can be formulated as optimization challenges.
Key Topics Covered:

- Data analysis through optimization lens
- Classical optimization problems (Least Squares, LASSO)
- Matrix factorization and low-rank problems

- Machine learning formulations (SVM, Logistic Regression)
- Deep learning optimization challenges

2.1 Introduction

This book focuses on the **fundamentals of algorithms** for solving continuous optimization problems, which involve:

- **Minimizing functions** of multiple real-valued variables
- **Handling constraints** on variable values

- **Emphasizing convex problems** with data science applications
- **Connecting theory to practice** in machine learning, statistics, and data analysis

! Core Focus

Our choice of topics is **motivated by relevance to data science** — the formulations and algorithms discussed are directly useful for solving real-world problems in machine learning, statistics, and data analysis.

This chapter outlines several **key paradigms from data science** and demonstrates how they can be formulated as continuous optimization problems. Understanding the **smoothness properties and structure** of these formulations is crucial for selecting appropriate algorithms.

2.2 Data Analysis and Optimization

The typical optimization problem in data analysis involves finding a **model that balances** two competing objectives:

1. **Agreement with collected data**
2. **Adherence to structural constraints** reflecting our beliefs about good models

2.2.1 The Data Science Framework

In a typical analysis problem, our **dataset** consists of m objects:

$$D := \{(a_j, y_j), j = 1, 2, \dots, m\} \tag{2.1}$$

where:

Features (a_j)

- Vector or matrix of features
- Input variables
- Predictors

Labels/Observations (y_j)

- Target values
- Responses
- Outcomes

Data Preprocessing

We assume the data has been **cleaned** so that all pairs (a_j, y_j) have consistent size and shape.

2.2.2 The Learning Objective

The **data analysis task** consists of discovering a function ϕ such that:

$$\phi(a_j) \approx y_j \quad \text{for most } j = 1, 2, \ldots, m$$

Terminology

The process of discovering the mapping ϕ is often called **"learning"** or **"training"**.

2.2.3 Parametrization and Optimization Formulation

The function ϕ is often defined in terms of a **vector or matrix of parameters**, which we denote by x or X. With these parametrizations, the problem of identifying ϕ becomes a traditional **data-fitting problem**:

Optimization Goal

Find the parameters x defining ϕ such that $\phi(a_j) \approx y_j$ for $j = 1, 2, \ldots, m$ in some optimal sense.

Once we define "optimal" (and possibly add restrictions on allowable parameter values), we have an optimization problem.

2.2.4 Finite-Sum Formulation

Frequently, these optimization formulations have objective functions of the **finite-sum type**:

$$L_D(x) := \frac{1}{m} \sum_{j=1}^{m} \ell(a_j, y_j; x) \tag{2.2}$$

where:

- $\ell(a, y; x)$ represents the **"loss"** incurred for not properly aligning our prediction $\phi(a)$ with y
- $L_D(x)$ measures the **average loss** accrued over the entire data set when the parameter vector equals x

2.2.5 Prediction and Model Properties

Once an appropriate value of x (and thus ϕ) has been learned from the data, we can use it to make predictions about other items of data not in the set D (Equation 2.1).

Given an unseen item of data $\hat{a}$ of the same type as a_j, $j = 1, 2, \ldots, m$, we predict the label $\hat{y}$ associated with $\hat{a}$ to be $\phi(\hat{a})$.

The mapping ϕ may also expose other structures and properties in the data set:

- **Feature Selection**: Reveals that only a small fraction of the features in a_j are needed to reliably predict the label y_j
- **Subspace Discovery**: When the parameter x is a matrix, it could reveal a low-dimensional subspace that contains most of the vectors a_j
- **Structured Matrices**: 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

2.2.6 Types of Data Analysis Problems

The form of the labels y_j differs according to the nature of the data analysis problem:

2.2.6.1 Regression

If each y_j is a **real number**, we typically have a **regression problem**.

2.2.6.2 Classification

When each y_j is a **label** (an integer from the set $\{1, 2, \ldots, M\}$) indicating that a_j belongs to one of M classes:

- **Binary Classification**: $M = 2$
- **Multiclass Classification**: $M > 2$

Note

In data analysis problems arising in speech and image recognition, M can be very large, of the order of thousands or more.

2.2.6.3 Unsupervised Learning

The labels y_j may not even exist; the data set may contain only the feature vectors a_j, $j = 1, 2, \dots, m$. There are still interesting data analysis problems associated with these cases. For example, we may wish to group the a_j into **clusters** (where the vectors within each cluster are deemed to be functionally similar) or identify a **low-dimensional subspace** (or a collection of low-dimensional subspaces) that approximately contains the a_j.

In such problems, we are essentially learning the labels y_j alongside the function ϕ. For example, in a clustering problem, y_j could represent the cluster to which a_j is assigned.

2.2.7 Data Complications and Robustness

Even after cleaning and preparation, the preceding setup may contain many complications:

- **Noise and Corruption**: The quantities (a_j, y_j) may contain noise or be otherwise corrupted, requiring the mapping ϕ to be robust to such errors
- **Missing Data**: Parts of the vectors a_j may be missing, or we may not know all the labels y_j
- **Streaming Data**: The data may be arriving in streaming fashion rather than being available all at once, requiring **online learning** of ϕ

2.2.8 Regularization and Overfitting

One consideration that arises frequently is that we wish to avoid **overfitting** the model to the data set D in (Equation 2.1).

! Generalization Goal

The particular data set D available to us can often be thought of as a finite sample drawn from some underlying larger (perhaps infinite) collection of possible data points. We wish the function ϕ to perform well on the **unobserved data points** as well as the observed subset D.

In other words, we want ϕ to be not too sensitive to the particular sample D that is used to define empirical objective functions such as (Equation 2.2).

One way to avoid this issue is to modify the objective function by adding constraints or penalty terms, in a way that limits the "complexity" of the function ϕ. This process is typically called **regularization**.

An optimization formulation that balances fit to the training data D, model complexity, and model structure is:

$$\min_{x \in \Omega} L_D(x) + \lambda \text{pen}(x) \tag{2.3}$$

where:

- Ω is a set of allowable values for x
- $\text{pen}(\cdot)$ is a regularization function or **regularizer**
- $\lambda \geq 0$ is a **regularization parameter**

The regularizer usually takes lower values for parameters x that yield functions ϕ with lower complexity. For example, ϕ may depend on fewer of the features in the data vectors a_j or may be less oscillatory.

2.2.8.1 Tuning the Regularization Parameter

The parameter λ can be "tuned" to provide an appropriate balance:

- **Smaller values of λ**: Produce solutions that fit the training data D more accurately
- **Larger values of λ**: Lead to less complex models

Modern Perspective on Overfitting

Interestingly, the concept of overfitting has been reexamined in recent years, particularly in the context of deep learning, where models that perfectly fit the training data are sometimes observed to also do a good job of classifying previously unseen data. This phenomenon is a topic of intense current research in the machine learning community.

2.2.8.2 Constraint Sets

The constraint set Ω in (Equation 2.3) may be chosen to exclude values of x that are not relevant or useful in the context of the data analysis problem. For example:

- In some applications, we may not wish to consider values of x in which one or more components are negative
- We could set Ω to be the set of vectors whose components are all greater than or equal to zero

2.2.9 Framework Overview

We now examine some particular problems in data science that give rise to formulations that are special cases of our master problem (Equation 2.3). We will see that:

- A large variety of problems can be formulated using this general framework
- Within this framework, there is a wide range of structures that must be taken into account in choosing algorithms to solve these problems efficiently

2.3 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:

$$\min_x \frac{1}{2m} \sum_{j=1}^{m} (a_j^T x - y_j)^2 = \frac{1}{2m} \|Ax - y\|_2^2 \tag{2.4}$$

where:

- A is the matrix whose rows are a_j^T, $j = 1, 2, \dots, m$
- $y = (y_1, y_2, \dots, y_m)^T$

In the preceding terminology, the function ϕ is defined by $\phi(a) := a^T x$.

Adding an Intercept

We can introduce a nonzero intercept by adding an extra parameter $\beta \in \mathbb{R}$ and defining $\phi(a) := a^T x + \beta$.

2.3.1 Statistical Motivation

This formulation can be motivated statistically, as a **maximum-likelihood estimate** of x when the observations y_j are exact but for independent identically distributed (i.i.d.) Gaussian noise.

2.3.2 Ridge Regression

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:

$$\min_x \frac{1}{2m}\|Ax - y\|_2^2 + \lambda\|x\|_2^2 \tag{2.5}$$

for some parameter $\lambda > 0$. The solution x of this regularized formulation has less sensitivity to perturbations in the data (a_j, y_j).

2.3.3 LASSO Formulation

The **LASSO** (Least Absolute Shrinkage and Selection Operator) formulation:

$$\min_x \frac{1}{2m}\|Ax - y\|_2^2 + \lambda\|x\|_1 \tag{2.6}$$

tends to yield solutions x that are **sparse** – that is, containing relatively few nonzero components.

! Feature Selection

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.

2.3.3.1 Advantages of Feature Selection

1. **Statistical Appeal**: Predictors that depend on few features are potentially simpler and more comprehensible than those depending on many features
2. **Practical Benefits**: Rather than gathering all components of a new data vector $\hat{a}$, we need to find only the "selected" features because only these are needed to make a prediction

i LASSO as a Prototype

The LASSO formulation (Equation 2.6) 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 has relatively simple structure that can potentially be exploited by algorithms.

2.4 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.

2.4.1 Basic Matrix Sensing Problem

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:

$$\min_X \frac{1}{2m} \sum_{j=1}^{m} (\langle A_j, X \rangle - y_j)^2 \tag{2.7}$$

where $\langle A, B \rangle := \operatorname{trace}(A^T B)$.

Here we can think of the A_j as "probing" the unknown matrix X. Commonly considered types of observations are:

- **Random linear combinations**: Elements of A_j are selected i.i.d. from some distribution
- **Single-element observations**: Each A_j has 1 in a single location and zeros elsewhere

2.4.2 Nuclear Norm Regularization

A regularized version of (Equation 2.7), leading to solutions X that are low rank, is:

$$\min_X \frac{1}{2m} \sum_{j=1}^{m} (\langle A_j, X \rangle - y_j)^2 + \lambda \|X\|_* \tag{2.8}$$

where $\|X\|_*$ is the **nuclear norm**, which is the sum of singular values of X.

Nuclear Norm Properties

The nuclear norm plays a role analogous to the ℓ_1 norm in (Equation 2.6):

- 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 (Equation 2.8) is also convex.

This formulation can be shown to yield a statistically valid solution when:

- The true X is low rank
- 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)
- The observations A_j are of single elements

2.4.3 Factorized Representation

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 (Equation 2.7) and solve:

$$\min_{L,R} \frac{1}{2m} \sum_{j=1}^{m} (\langle A_j, LR^T \rangle - y_j)^2 \tag{2.9}$$

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 typically much more compact than (Equation 2.8); the total number of elements in (L,R) is $(n+p)r$, which is much less than np
- However, this function is **nonconvex** when considered as a function of (L,R) jointly

! Benign Nonconvexity

An active line of current research 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 (Equation 2.9).

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.

2.4.4 Nonnegative Matrix Factorization

Some applications in computer vision, chemometrics, and document clustering require us to find factors L and R like those in (Equation 2.9) in which all elements are nonnegative. If the full matrix $Y \in \mathbb{R}^{n \times p}$ is observed, this problem has the form:

$$\min_{L,R} \|LR^T - Y\|_F^2 \quad \text{subject to } L \geq 0,\ R \geq 0 \tag{2.10}$$

and is called **nonnegative matrix factorization**.

2.5 Support Vector Machines

Classical ML Problem

Classification via Support Vector Machines (SVM) is a classical optimization problem in machine learning, tracing its origins to the **1960s**.

Given the input data (a_j, y_j) with $a_j \in \mathbb{R}^n$ and $y_j \in \{-1, 1\}$, SVM seeks a vector $x \in \mathbb{R}^n$ and a scalar $\beta \in \mathbb{R}$ such that:

$$a_j^T x - \beta \geq 1 \quad \text{when } y_j = +1 \tag{2.11}$$

$$a_j^T x - \beta \leq -1 \quad \text{when } y_j = -1 \tag{2.12}$$

Any pair (x, β) that satisfies these conditions defines a **separating hyperplane** in $\mathbb{R}^n$, that separates the "positive" cases $\{a_j \mid y_j = +1\}$ from the "negative" cases $\{a_j \mid y_j = -1\}$.

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.

2.5.1 Hinge Loss Formulation

We can formulate the problem of finding a separating hyperplane as an optimization problem by defining an objective with the summation form (Equation 2.2):

$$H(x, \beta) = \frac{1}{m} \sum_{j=1}^{m} \max(1 - y_j(a_j^T x - \beta), 0) \tag{2.13}$$

Note that the j-th term in this summation is zero if the conditions (Equation 2.11)–(Equation 2.12) are satisfied, and it is positive otherwise. Even if no pair (x, β) exists for which $H(x, \beta) = 0$, a value (x, β) that minimizes (Equation 2.13) will be the one that comes as close as possible to satisfying the conditions in some sense.

A term $\frac{1}{2\lambda}\|x\|_2^2$ (for some parameter $\lambda > 0$) is often added to (Equation 2.13), yielding the following regularized version:

$$H(x, \beta) = \frac{1}{m}\sum_{j=1}^{m} \max(1 - y_j(a_j^T x - \beta), 0) + \frac{1}{2\lambda}\|x\|_2^2 \tag{2.14}$$

Loss vs Regularizer

In contrast to the examples presented so far, the SVM problem has a **nonsmooth loss function** and a **smooth regularizer**.

If λ is sufficiently small, and if separating hyperplanes exist, the pair (x, β) that minimizes (Equation 2.14) 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 to several of the observed data points may not do as well.

2.5.2 Kernel Methods

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 some nonlinear mapping ψ and then perform the support vector machine classification on the vectors $\psi(a_j)$, $j = 1, 2, \ldots, m$. The conditions (Equation 2.11)–(Equation 2.12) would thus be replaced by:

$$\psi(a_j)^T x - \beta \geq 1 \quad \text{when } y_j = +1 \tag{2.15}$$

$$\psi(a_j)^T x - \beta \leq -1 \quad \text{when } y_j = -1 \tag{2.16}$$

leading to the following analog of (Equation 2.14):

$$H(x, \beta) = \frac{1}{m}\sum_{j=1}^{m} \max(1 - y_j(\psi(a_j)^T x - \beta), 0) + \frac{1}{2\lambda}\|x\|_2^2 \tag{2.17}$$

When transformed back to $\mathbb{R}^m$, the surface $\{a \mid \psi(a)^T x - \beta = 0\}$ is nonlinear and possibly disconnected, and is often a much more powerful classifier than the hyperplanes resulting from (Equation 2.14).

2.5.3 Dual Formulation

We note that SVM can also be expressed naturally as a minimization problem over a convex set. By introducing artificial variables, the problem (Equation 2.17) (and (Equation 2.14)) 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:

$$\begin{aligned} \min_{\alpha \in \mathbb{R}^m} \quad & \frac{1}{2}\alpha^T Q \alpha - \mathbf{1}^T \alpha \\ \text{subject to} \quad & 0 \leq \alpha \leq \frac{1}{\lambda}\mathbf{1}, \quad y^T \alpha = 0 \end{aligned} \tag{2.18}$$

where:

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

! The Kernel Trick

Interestingly, problem (Equation 2.18) 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 $\psi(a_k)^T \psi(a_l)$. 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 (Equation 2.18). A particularly popular choice of kernel is the **Gaussian kernel**:

$$K(a_k, a_l) := \exp\left(-\frac{1}{2\sigma^2}\|a_k - a_l\|_2^2\right) \tag{2.19}$$

where σ is a positive parameter.

2.6 Logistic Regression

Logistic regression can be viewed as a softened form of binary support vector machine classification in which, rather than the classification function ϕ giving an 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 $x \in \mathbb{R}^n$:

$$p(a; x) := (1 + \exp(a^T x))^{-1} \tag{2.20}$$

and aim to choose the parameter x so that:

- $p(a_j; x) \approx 1$ when $y_j = +1$
- $p(a_j; x) \approx 0$ when $y_j = -1$

Note the similarity to (Equation 2.11)–(Equation 2.12).

2.6.1 Negative Log-Likelihood

The optimal value of x can be found by minimizing a **negative-log-likelihood function**:

$$L(x) := -\frac{1}{m}\left[\sum_{j: y_j = -1} \log(1 - p(a_j; x)) + \sum_{j: y_j = 1} \log p(a_j; x)\right] \tag{2.21}$$

Note that the definition (Equation 2.20) ensures that $p(a; x) \in (0, 1)$ for all a and x; thus, $\log(1 - p(a_j; x)) < 0$ and $\log p(a_j; x) < 0$ for all j and all x. When the conditions above are satisfied, these log terms will be only slightly negative, so values of x that satisfy them will be near optimal.

2.6.2 Feature Selection with LASSO

We can perform feature selection using the model (Equation 2.21) by introducing a regularizer $\lambda\|x\|_1$ (as in the LASSO technique for least squares):

$$\min_x -\frac{1}{m}\left[\sum_{j: y_j = -1} \log(1 - p(a_j; x)) + \sum_{j: y_j = 1} \log p(a_j; x)\right] + \lambda\|x\|_1 \tag{2.22}$$

where $\lambda > 0$ is a regularization parameter. 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.

2.6.3 Multiclass Logistic Regression

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.

Example: Speech Recognition

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:

$$p_k(a; X) := \frac{\exp(a^T x^{[k]})}{\sum_{l=1}^{M} \exp(a^T x^{[l]})}, \quad k = 1, 2, \dots, M \tag{2.23}$$

where we define $X := \{x^{[k]} \mid k = 1, 2, \dots, M\}$.

As in the binary case, we have $p_k(a) \in (0, 1)$ for all a and all $k = 1, 2, \dots, M$ and, in addition, that $\sum_{k=1}^{M} p_k(a) = 1$. The functions (Equation 2.23) perform a **"softmax"** on the quantities $\{a^T x^{[l]} \mid l = 1, 2, \dots, M\}$.

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

$$y_{jk} = \begin{cases} 1 & \text{when } a_j \text{ belongs to class } k, \\ 0 & \text{otherwise.} \end{cases} \tag{2.24}$$

Similarly to the binary case, we seek to define the vectors $x^{[k]}$ so that:

- $p_k(a_j; X) \approx 1$ when $y_{jk} = 1$
- $p_k(a_j; X) \approx 0$ when $y_{jk} = 0$

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

$$L(X) := -\frac{1}{m} \sum_{j=1}^{m} \left[\sum_{\ell=1}^{M} y_{j\ell} (x^{[\ell]T} a_j) - \log \left(\sum_{\ell=1}^{M} \exp(x^{[\ell]T} a_j) \right) \right] \tag{2.25}$$

"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.

2.7 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, often for large M. The major innovation is that the mapping ϕ from data vector to prediction is now a **nonlinear function**, explicitly parametrized by a set of structured transformations.

2.7.1 Neural Network Structure

The neural network shown in conceptual form illustrates the structure of a particular neural net. In this structure:

- The data vector a_j enters at the left of the network
- Each box (more often referred to as a "layer") represents a transformation that takes an input vector and applies a nonlinear transformation of the data to produce an output vector
- The output of each operator becomes the input for one or more subsequent layers
- Each layer has a set of its own parameters, and the collection of all of the parameters over all the layers comprises our optimization variable
- The different types of transformations might differ between layers, but we can compose them in whatever fashion suits our application

2.7.2 Layer Transformations

A typical transformation, which converts the vector a_j^{l-1} representing output from layer $l-1$ to the vector a_j^l representing output from layer l, is:

$$a_j^l = \sigma(W^l a_j^{l-1} + g^l) \tag{2.26}$$

where:

- W^l is a matrix of dimension $|a_j^l| \times |a_j^{l-1}|$
- g^l is a vector of length $|a_j^l|$
- σ is a componentwise nonlinear transformation, usually called an **activation function**

The most common forms of the activation function σ act independently on each component of their argument vector as follows:

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

Alternative transformations are needed when the input to box l comes from two or more preceding boxes.

2.7.3 Output Layer and Loss Function

The rightmost layer of the neural network (the **output layer**) typically has M outputs, one for each of the possible classes to which the input (a_j, say) could belong. These are compared to the labels y_{jk}, defined as in (Equation 2.24) to indicate which of the M classes that a_j belongs to. Often, a softmax is applied to the outputs in the rightmost layer, and a loss function similar to (Equation 2.25) is obtained.

2.7.4 Deep Learning Optimization Problem

Consider the special (but not uncommon) case in which the neural net structure is a linear graph of D levels, in which the output for layer $l-1$ becomes the input for layer l (for $l = 1, 2, \ldots, D$) with $a_j = a_j^0$, $j = 1, 2, \ldots, m$, and the transformation within each box has the form (Equation 2.26). A softmax is applied to the output of the rightmost layer to obtain a set of odds.

The parameters in this neural network are the matrix-vector pairs (W^l, g^l), $l = 1, 2, \ldots, D$ that transform the input vector $a_j = a_j^0$ into the output a_j^D of the final layer. We aim to choose all these parameters so that the network does a good job of classifying the training data correctly.

Using the notation w for the layer-to-layer transformations, that is:

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

we can write the loss function for deep learning as:

$$L(w) = -\frac{1}{m} \sum_{j=1}^{m} \left[\sum_{\ell=1}^{M} y_{j\ell} a_{j,\ell}^D(w) - \log\left(\sum_{\ell=1}^{M} \exp a_{j,\ell}^D(w) \right) \right] \tag{2.27}$$

where $a_{j,\ell}^D(w) \in \mathbb{R}$ is the output of the ℓ-th element in layer D corresponding to input vector a_j^0. (Here we write $a_{j,\ell}^D(w)$ to make explicit the dependence 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 with $D = 1$, so that $a^1_{j,\ell} = W^1_{\ell,\cdot} a^0_j$, where $W^1_{\ell,\cdot}$ denotes row ℓ of the matrix W^1.

2.7.5 Variants and Engineering

Neural networks in use for particular applications (for example, in image recognition and speech recognition, where they have been quite successful) include many variants on the basic design. These include:

- **Restricted connectivity** between the boxes (which corresponds to enforcing sparsity structure on the matrices W^l, $l = 1, 2, \dots, D$)
- **Sharing parameters**, which corresponds to forcing subsets of the elements of W^l to take the same value
- **Complex arrangements** of the boxes, with outputs coming from several layers, connections across nonadjacent layers, different componentwise transformations σ at different layers, and so on

Deep neural networks for practical applications are highly engineered objects.

2.7.6 Distinctive Features of Deep Learning

The loss function (Equation 2.27) shares with many other applications the finite-sum form (Equation 2.2), but it has several features that set it apart from the other applications discussed before:

1. **Nonconvexity**: Most importantly, it is nonconvex in the parameters w
2. **Large-scale**: The total number of parameters in w 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.

2.8 Emphasis

Many problems can be formulated as in the framework (Equation 2.3), and their properties may differ significantly. They might be convex or nonconvex, and smooth or nonsmooth. But there are important features that they all share:

Shared Features of Optimization Problems

- They can be formulated as **functions of real variables**, which we typically arrange in a vector of length n
- The functions are **continuous**. 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 nearby 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

2.8.1 Treatment Emphasis

Our treatment emphasizes **algorithms** for solving these various kinds of problems, with analysis of 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
- 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 bounds. (A notable exception is the analysis of interior-point methods.)

2.8.2 Practical Concerns

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 are 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 **type of computer** on which the problem will be solved
- The **guarantees needed** for the solution

Part II

Summary & References

3 Summary

In summary, this book has no content whatsoever.

References

Knuth, Donald E. 1984. "Literate Programming." *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.