Chapter 2

Learning Linear and Independent Structures

> "*The art of doing mathematics consists in finding that special case which contains all the germs of generality.*"
>
> – David Hilbert

Real data has low-dimensional structure. To see why this is true, let us consider the unassuming case of static on a TV when the satellite isn't working. At each frame (approximately every $\frac{1}{30}$ seconds), the RGB static on a screen of size $H \times W$ is, roughly, sampled independently from a uniform distribution on $[0,1]^{3\times H\times W}$. In theory, the static *could* resolve to a natural image on any given frame, but even if you spend a thousand years looking at the TV screen, it will not. This discrepancy is explained by the fact that the set of $H \times W$ natural images takes up a vanishingly small fraction of the hypercube $[0,1]^{3\times H\times W}$. In particular, it is extremely low-dimensional, compared to the ambient space dimension. Similar phenomena occur for all other types of natural data, such as text, audio, and video. Thus, when we design systems and methods to process natural data and learn its structure or distribution, this is a central property of natural data which we need to take into account.

Therefore, our central task is to learn a distribution that has low intrinsic dimension in a high-dimensional space. In the remainder of this chapter, we discuss several *classical* methods to perform this task for several somewhat *idealistic* models for the distribution, namely models that are geometrically linear or statistically independent. While these models and methods are important and useful in their own right, we discuss them here as they motivate, inspire, and serve as a predecessor or analogue to more modern methods for more general distributions that involve deep (representation) learning.

Our main approach (and general problem formulation) can be summarized as:

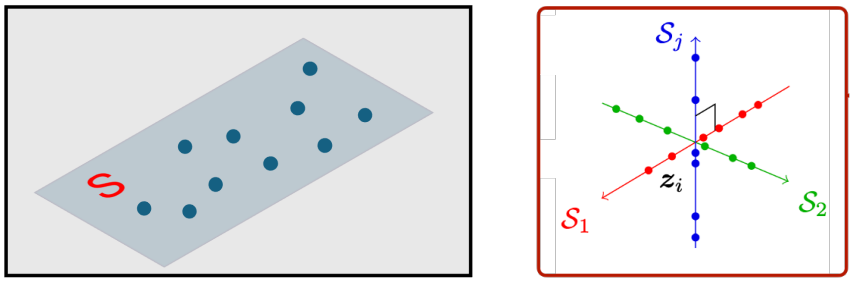


Figure 2.1: Data on a single low-dimensional subspace (left), say $\mathcal{S} = \text{col}(\boldsymbol{U})$, or a mixture of low-dimensional subspaces (right), say $\mathcal{S}_j = \text{col}(\boldsymbol{U}_j)$.

Problem: *Given one or several (noisy or incomplete) observations of a ground truth sample from the data distribution, obtain an estimate of this sample.*

This approach underpins several classical methods for data processing, which we discuss in this chapter.

- Section 2.1 — Principal Components Analysis (PCA): Given noisy samples from a distribution supported on *one low-dimensional subspace*, obtain an estimate of the true sample that lies on this subspace.
- Section 2.2 — Complete Dictionary Learning and Independent Components Analysis (ICA): Given noisy samples from a distribution supported on *a union (not the span) of a few low-dimensional subspaces*, obtain an estimate of the true samples.
- Section 2.3 — Sparse Coding and Dictionary Learning: Given noisy samples from *a distribution supported on combinations of a few incoherent vectors*, such as the coordinate axes, obtain an estimate of the true sample, which also has this property.

In this chapter, as described above, we make simplifying modeling assumptions that essentially assume (the support of) the data distribution has geometrically (nearly, piece-wise) linear structures and statistically independent components, as illustrated in Figure 2.1. In Chapter 1, we have referred to such models of the data as "analytical models". These modeling assumptions allow us to derive efficient algorithms with provable efficiency guarantees, both in terms of data and computational complexity, for processing data at scale. However, they are imperfect models for often-complex real-world data distributions, and so their underlying assumptions only approximately hold. This means that the guarantees afforded by detailed analysis of these algorithms also only approximately hold in the case of real data.

Nonetheless, the techniques discussed in this chapter are useful in their own right, and beyond that, serve as the "special case with the germ of generality",

so to speak, in that they present a guiding motivation and intuition for the more general paradigms within (deep) learning of more general distributions that we later address. As we will soon reveal in later chapters, in the deep learning era, modern "data-driven" or "non-parametric" approaches to learn (low-dimensional) data distributions essentially adopt the same concepts and ideas to learn.

A Note on Notation. We encourage the reader to consult the Notation chapter, particularly the "Modeling and Optimization Notation" section, as they begin to go through this chapter. We use the same notational conventions for modeling data and optimizing the parameters of these models throughout the book, and the simple, explicit models we study in this chapter help make the 'semantic' content of these conventions clear.

2.1 A Low-Dimensional Subspace

2.1.1 Principal Components Analysis (PCA)

Perhaps the simplest modeling assumption possible for low-dimensional structure is the so-called *low-rank* assumption. Letting D be the dimension of our data space, we assume that our data belong to a low-dimensional subspace of dimension $d \ll D$, possibly plus some small disturbances.[1] This ends up being a nearly valid assumption for some surprisingly complex data, such as images of handwritten digits and face data [RD03] as shown in Figure 2.2, yet as we will see, it will lend itself extremely well to comprehensive analysis.

Problem Formulation. To write this in mathematical notation, we represent a subspace $\mathcal{S} \subseteq \mathbb{R}^D$ of dimension d by an orthonormal matrix $\boldsymbol{U} \in \mathsf{O}(D, d) \subseteq \mathbb{R}^{D \times d}$ such that the columns of $\boldsymbol{U}$ span $\mathcal{S}$. Then, we say that our data $\{\boldsymbol{x}_i\}_{i=1}^N \subseteq \mathbb{R}^D$ have (approximate) low-rank structure if there exists an orthonormal matrix $\boldsymbol{U} \in \mathsf{O}(D, d)$, vectors $\{\boldsymbol{z}_i\}_{i=1}^N \subseteq \mathbb{R}^d$, and *small* vectors $\{\boldsymbol{\varepsilon}_i\}_{i=1}^N \subseteq \mathbb{R}^D$ such that

$$\boldsymbol{x}_i = \boldsymbol{U}\boldsymbol{z}_i + \boldsymbol{\varepsilon}_i, \quad \forall i \in [N]. \tag{2.1.1}$$

Here $\boldsymbol{\varepsilon}_i$ are disturbances that prevent the data from being perfectly low rank; their presence in our model allows us to quantify the degree to which our analysis remains relevant in the presence of deviations from our model. The true supporting subspace is $\mathcal{S} \doteq \operatorname{col}(\boldsymbol{U})$, the span of the columns of $\boldsymbol{U}$. To process all we can from these data, we need to recover $\mathcal{S}$; to do this it is sufficient to recover $\boldsymbol{U}$, also called the *principal components*. Fortunately, this is a computationally tractable task named **Principal Component Analysis**, and we discuss now how to solve it.

[1] Plus, potentially, a non-trivial offset from $\mathbf{0}$; in practice we account for this offset by subtracting the empirical mean over data samples from each sample, and in the following Chapter we pay no more heed to it.

Figure 2.2: Images of aligned human faces and handwritten digits. Despite the seemingly large variety in their appearances, each set of these data spans (approximately) a very low-dimensional (nearly) linear subspace. This can be made technically precise in the notation of (2.1.1): for a relatively small subspace dimension d (say, ≈ 20, compared to the dimension $D = 784$ of MNIST digits), the error vectors $\boldsymbol{\varepsilon}_i$ are small for every $i \in [N]$ (where $N = 6 \cdot 10^4$ for MNIST is the dataset size). We can *learn* the optimal such subspace from data.

Given data $\{\boldsymbol{x}_i\}_{i=1}^N$ distributed as in (2.1.1), we aim to recover the model $\boldsymbol{U}$. A natural approach is to find the subspace $\boldsymbol{U}^\star$ and latent vectors $\{\boldsymbol{z}_i^\star\}_{i=1}^N$ which yield the best approximation $\boldsymbol{x}_i \approx \boldsymbol{U}^\star \boldsymbol{z}_i^\star$. Namely, we aim to solve the problem

$$\min_{\tilde{\boldsymbol{U}}, \{\tilde{\boldsymbol{z}}_i\}_{i=1}^N} \frac{1}{N} \sum_{i=1}^N \|\boldsymbol{x}_i - \tilde{\boldsymbol{U}} \tilde{\boldsymbol{z}}_i\|_2^2, \tag{2.1.2}$$

where $\tilde{\boldsymbol{U}}$ is constrained to be an orthonormal matrix, as above. We will omit this constraint in similar statements below for the sake of concision.

Subspace Encoding-Decoding via Denoising. To simplify this problem, for a fixed $\tilde{\boldsymbol{U}}$, we have (proof as exercise):

$$\min_{\{\tilde{\boldsymbol{z}}_i\}_{i=1}^N} \frac{1}{N} \sum_{i=1}^N \|\boldsymbol{x}_i - \tilde{\boldsymbol{U}} \tilde{\boldsymbol{z}}_i\|_2^2 = \frac{1}{N} \sum_{i=1}^N \min_{\tilde{\boldsymbol{z}}_i} \|\boldsymbol{x}_i - \tilde{\boldsymbol{U}} \tilde{\boldsymbol{z}}_i\|_2^2 \tag{2.1.3}$$

$$= \frac{1}{N} \sum_{i=1}^N \|\boldsymbol{x}_i - \tilde{\boldsymbol{U}} \tilde{\boldsymbol{U}}^\top \boldsymbol{x}_i\|_2^2. \tag{2.1.4}$$

That is, the optimal solution $(\boldsymbol{U}^\star, \{\boldsymbol{z}_i^\star\}_{i=1}^N)$ to the above optimization problem has $\boldsymbol{z}_i^\star = (\boldsymbol{U}^\star)^\top \boldsymbol{x}_i$.

Now, we can write the original optimization problem in $\tilde{\boldsymbol{U}}$ and $\{\tilde{\boldsymbol{z}}_i\}_{i=1}^N$ as an optimization problem just over $\tilde{\boldsymbol{U}}$, i.e., to obtain the basis $\boldsymbol{U}^\star$ and compact

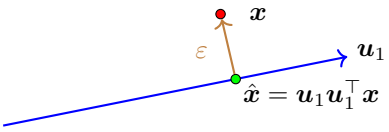


Figure 2.3: **Geometry of PCA.** A data point $\boldsymbol{x}$ (red) is projected onto the one-dimensional learned subspace spanned by the unit basis vector $\boldsymbol{u}_1$ (blue arrow). The projection $\boldsymbol{U}\boldsymbol{U}^\top\boldsymbol{x} = \boldsymbol{u}_1\boldsymbol{u}_1^\top\boldsymbol{x}$ (green) is the denoised version of $\boldsymbol{x}$ using the low-dimensional structure given by $\boldsymbol{u}_1$, and $\boldsymbol{\varepsilon}$ (brown arrow) represents the projection residual or noise.

codes $\{\boldsymbol{z}_i^\star\}_{i=1}^N$ it suffices to solve either of the two following equivalent problems:

$$\min_{\tilde{\boldsymbol{U}},\{\tilde{\boldsymbol{z}}_i\}_{i=1}^N} \frac{1}{N}\sum_{i=1}^N \|\boldsymbol{x}_i - \tilde{\boldsymbol{U}}\tilde{\boldsymbol{z}}_i\|_2^2 = \min_{\tilde{\boldsymbol{U}}} \frac{1}{N}\sum_{i=1}^N \|\boldsymbol{x}_i - \tilde{\boldsymbol{U}}\tilde{\boldsymbol{U}}^\top\boldsymbol{x}_i\|_2^2. \tag{2.1.5}$$

Note that the problem on the right-hand side of (2.1.5) is a *denoising* problem: given noisy observations $\boldsymbol{x}_i$ of low-rank data, we aim to find the *noise-free* copy of $\boldsymbol{x}_i$, which under the model (2.1.1) is $\boldsymbol{U}\boldsymbol{z}_i$. That is, the denoised input $\hat{\boldsymbol{x}}_i = \boldsymbol{U}\boldsymbol{U}^\top\boldsymbol{x}_i$. Notice that this is the point on the subspace that is closest to $\boldsymbol{x}_i$, as visualized in Figure 2.3. Here by solving the equivalent problem of finding the best subspace, parameterized by the learned basis $\boldsymbol{U}^\star$, we learn an approximation to the *denoiser*, i.e., the projection matrix $\boldsymbol{U}^\star(\boldsymbol{U}^\star)^\top \approx \boldsymbol{U}\boldsymbol{U}^\top$ that projects the noisy data point $\boldsymbol{x}_i$ onto the subspace $\mathcal{S}$.

Putting the above process together, we essentially obtain a simple encoding-decoding scheme that maps a data point $\boldsymbol{x}$ in $\mathbb{R}^D$ to a lower-dimensional (latent) space $\mathbb{R}^d$ and then back to $\mathbb{R}^D$:

$$\boldsymbol{x} \xrightarrow{\mathcal{E}=(\boldsymbol{U}^\star)^\top} \boldsymbol{z} \xrightarrow{\mathcal{D}=\boldsymbol{U}^\star} \hat{\boldsymbol{x}}. \tag{2.1.6}$$

Here, $\boldsymbol{z} \in \mathbb{R}^d$ can be viewed as the low-dimensional compact code (or a latent representation) of a data point $\boldsymbol{x} \in \mathbb{R}^D$ and the learned subspace basis $\boldsymbol{U}^\star$ as the associated codebook whose columns are the (learned) optimal code words. The process achieves the function of denoising $\boldsymbol{x}$ by projecting it onto the subspace spanned by $\boldsymbol{U}^\star$.

Computing the Subspace Basis. For now, we continue our calculation. Let $\boldsymbol{X} = \big[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\big] \in \mathbb{R}^{D\times N}$ be the matrix whose columns are the observations $\boldsymbol{x}_i$. We have (proof as exercise)

$$\arg\min_{\tilde{\boldsymbol{U}}} \frac{1}{N}\sum_{i=1}^N \|\boldsymbol{x}_i - \tilde{\boldsymbol{U}}\tilde{\boldsymbol{U}}^\top\boldsymbol{x}_i\|_2^2 = \arg\max_{\tilde{\boldsymbol{U}}} \frac{1}{N}\sum_{i=1}^N \|\tilde{\boldsymbol{U}}^\top\boldsymbol{x}_i\|_2^2 \tag{2.1.7}$$

$$= \arg\max_{\tilde{\boldsymbol{U}}} \operatorname{tr}\left\{\tilde{\boldsymbol{U}}^\top\left(\frac{\boldsymbol{X}\boldsymbol{X}^\top}{N}\right)\tilde{\boldsymbol{U}}\right\}. \tag{2.1.8}$$

Thus, to compute the principal components, we find the orthogonal matrix $\tilde{\boldsymbol{U}}$ which maximizes the term $\operatorname{tr}(\tilde{\boldsymbol{U}}^\top(\boldsymbol{X}\boldsymbol{X}^\top/N)\tilde{\boldsymbol{U}})$. We can prove via induction that this matrix $\boldsymbol{U}^\star$ has columns which are the *top d unit eigenvectors* of $\boldsymbol{X}\boldsymbol{X}^\top/N$. We leave the whole proof to the reader in Exercise 2.1, but we handle the base case of the induction here. Suppose that $d = 1$. Then we only have a single unit vector $\boldsymbol{u}$ to recover, so the above problem reduces to

$$\max_{\tilde{\boldsymbol{u}}:\ \|\tilde{\boldsymbol{u}}\|_2=1} \tilde{\boldsymbol{u}}^\top(\boldsymbol{X}\boldsymbol{X}^\top/N)\tilde{\boldsymbol{u}}. \tag{2.1.9}$$

This is the so-called *Rayleigh quotient* of $\boldsymbol{X}\boldsymbol{X}^\top/N$. By invoking the spectral theorem we diagonalize $\boldsymbol{X}\boldsymbol{X}^\top/N = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top$, where $\boldsymbol{Q}$ is orthogonal and $\boldsymbol{\Lambda}$ is diagonal with non-negative entries. Hence

$$\tilde{\boldsymbol{u}}^\top(\boldsymbol{X}\boldsymbol{X}^\top/N)\tilde{\boldsymbol{u}} = \tilde{\boldsymbol{u}}^\top\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top\tilde{\boldsymbol{u}} = (\boldsymbol{Q}^\top\tilde{\boldsymbol{u}})^\top\boldsymbol{\Lambda}(\boldsymbol{Q}^\top\tilde{\boldsymbol{u}}). \tag{2.1.10}$$

Since $\boldsymbol{Q}$ is an invertible orthogonal transformation, $\boldsymbol{Q}^\top\tilde{\boldsymbol{u}}$ is a unit vector, and optimizing over $\tilde{\boldsymbol{u}}$ is equivalent to optimizing over $\tilde{\boldsymbol{w}} \doteq \boldsymbol{Q}^\top\tilde{\boldsymbol{u}}$. Hence, we can write

$$\tilde{\boldsymbol{u}}^\top(\boldsymbol{X}\boldsymbol{X}^\top/N)\tilde{\boldsymbol{u}} = \tilde{\boldsymbol{w}}^\top\boldsymbol{\Lambda}\tilde{\boldsymbol{w}}, \tag{2.1.11}$$

whose optimal solutions $\boldsymbol{w}^\star$ among unit vectors are one-hot vectors whose only nonzero (hence unit) entry is in one of the indices corresponding to the largest eigenvalue of $\boldsymbol{X}\boldsymbol{X}^\top/N$. This means that $\boldsymbol{u}^\star = \boldsymbol{Q}\boldsymbol{w}^\star$, the optimal solution to the original problem, corresponds to a unit eigenvector of $\boldsymbol{X}\boldsymbol{X}^\top/N$ (i.e., column of $\boldsymbol{Q}$) which corresponds to the largest eigenvalue. Suitably generalizing this to the case $d > 1$, and summarizing all the previous discussion, we have the following informal Theorem.

Theorem 2.1. *Suppose that our dataset $\{\boldsymbol{x}_i\}_{i=1}^N \subseteq \mathbb{R}^D$ can be written as*

$$\boldsymbol{x}_i = \boldsymbol{U}\boldsymbol{z}_i + \boldsymbol{\varepsilon}_i, \qquad \forall i \in [N], \tag{2.1.12}$$

where $\boldsymbol{U} \in \mathsf{O}(D, d)$ captures the low-rank structure, $\{\boldsymbol{z}_i\}_{i=1}^N \subseteq \mathbb{R}^d$ are the compact codes of the data, and $\{\boldsymbol{\varepsilon}_i\}_{i=1}^N \subseteq \mathbb{R}^D$ are small vectors indicating the deviation of our data from the low-rank model. Then the principal components $\boldsymbol{U}^\star \in \mathsf{O}(D, d)$ of our dataset are given by the top d eigenvectors of $\boldsymbol{X}\boldsymbol{X}^\top/N$, where $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N] \in \mathbb{R}^{D\times N}$, and approximately correspond to the optimal linear denoiser: $\boldsymbol{U}^\star(\boldsymbol{U}^\star)^\top \approx \boldsymbol{U}\boldsymbol{U}^\top$.

We do not give explicit rates of approximation here as they can become rather technical. In the special case that $\boldsymbol{\varepsilon}_i = \boldsymbol{0}$ for all i, the learned $\boldsymbol{U}^\star$ spans the support of the samples $\{\boldsymbol{x}_i\}_{i=1}^N$. If in addition the $\boldsymbol{z}_i$ are sufficiently diverse (say, spanning all of $\mathbb{R}^d$) then we would have perfect recovery: $\boldsymbol{U}^\star(\boldsymbol{U}^\star)^\top = \boldsymbol{U}\boldsymbol{U}^\top$.

Remark 2.1. In some data analysis tasks, the data matrix $\boldsymbol{X}$ is formatted such that each data point is a *row* rather than a *column* as is presented here. In this case the principal components are the top d eigenvectors of $\boldsymbol{X}^\top\boldsymbol{X}/N$.

Remark 2.2 (Basis Selection via Denoising Eigenvalues). In many cases, either our data will not truly be distributed according to a subspace-plus-noise model, or we will not know the true underlying dimension d of the subspace. In this case, we have to choose d; this problem is called *model selection*. In the restricted case of PCA, one way to perform model selection is to compute $\boldsymbol{X}\boldsymbol{X}^\top/N$ and look for instances where adjacent eigenvalues sharply decrease; this is one indicator that the index of the larger eigenvalue is the "true dimension d", and the rest of the eigenvalues of $\boldsymbol{X}\boldsymbol{X}^\top/N$ are contributed by the noise or disturbances $\boldsymbol{\varepsilon}_i$. Model selection is a difficult problem and, nowadays in the era of deep learning where it is called "hyperparameter optimization", is usually done via brute force or Bayesian optimization, often using the loss on a holdout ("validation") dataset as an objective.

Remark 2.3 (Denoising Samples). The expression on the right-hand side of (2.1.5), that is,

$$\min_{\tilde{\boldsymbol{U}}} \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{x}_i - \tilde{\boldsymbol{U}}\tilde{\boldsymbol{U}}^\top \boldsymbol{x}_i\|_2^2, \tag{2.1.13}$$

is what's known as a *denoising problem*, thus named because it is an optimization problem whose solution *removes the noise from the samples so it fits on the subspace*. Denoising—learning a map which removes noise from noisy samples so that it fits on the data structure (such as in (2.1.1), but maybe more complicated)—is a common method for learning distributions that will be discussed in the sequel and throughout the manuscript. Note that we have already discussed this notion, but it bears repeating due to its central importance in later chapters.

Remark 2.4 (Neural Network Interpretation). If we do a PCA, we approximately recover the support of the distribution encoded by the parameter $\boldsymbol{U}^\star$. The learned denoising map then takes the form $\boldsymbol{U}^\star(\boldsymbol{U}^\star)^\top$. On top of being a denoiser, this can be viewed as a *simple two-layer weight-tied linear neural network*: the first layer multiplies by $(\boldsymbol{U}^\star)^\top$, and the second layer multiplies by $\boldsymbol{U}^\star$, namely

$$\text{denoise}(\boldsymbol{x}) = \underbrace{\boldsymbol{U}^\star \circ \underbrace{\text{id} \circ \underbrace{(\boldsymbol{U}^\star)^\top \boldsymbol{x}}_{\text{first "layer"}}}_{\text{post-activation of first "layer"}}}_{\text{output of "NN"}} \tag{2.1.14}$$

Contrasting this to a standard two-layer neural network, we see a structural similarity:

$$\text{NN}(\boldsymbol{x}) = \underbrace{\boldsymbol{W}^\star \circ \underbrace{\text{ReLU} \circ \underbrace{(\boldsymbol{U}^\star)^\top \boldsymbol{x}}_{\text{first layer}}}_{\text{post-activation of first layer}}}_{\text{output of NN}} \tag{2.1.15}$$

In particular, PCA can be interpreted as *learning a simple two-layer denoising*

autoencoder,[2] one of the simplest examples of a non-trivial neural network. In this framework, the *learned representations* would just be $(\boldsymbol{U}^\star)^\top \boldsymbol{x} \approx \boldsymbol{z}$. In this way, PCA serves as a model problem for (deep) representation learning, which we will draw upon further in the monograph. Notice that in this analogy, the representations reflect, or are projections of, the input data towards a learned low-dimensional structure. This property will be particularly relevant in the future.

2.1.2 Pursuing Low-Rank Structure via Power Iteration

There is a computationally efficient way to estimate the top eigenvectors of $\boldsymbol{X}\boldsymbol{X}^\top/N$ or any symmetric positive semidefinite matrix $\boldsymbol{M}$, called *power iteration*. This method is the building block of several algorithmic approaches to high-dimensional data analysis that we discuss later in the chapter, so we discuss it here.

Let $\boldsymbol{M}$ be a symmetric positive semidefinite matrix. There exists an orthonormal basis for $\mathbb{R}^D$ consisting of eigenvectors $(\boldsymbol{w}_i)_{i=1}^D$ of $\boldsymbol{M}$, with corresponding eigenvalues $\lambda_1 \geq \cdots \geq \lambda_D \geq 0$. By definition, any eigenvector $\boldsymbol{w}_i$ satisfies $\lambda_i \boldsymbol{w}_i = \boldsymbol{M}\boldsymbol{w}_i$. Therefore, for any $\lambda_i > 0$, $\boldsymbol{w}_i$ is a "fixed point" to the following equation:

$$\boldsymbol{w} = \frac{\boldsymbol{M}\boldsymbol{w}}{\|\boldsymbol{M}\boldsymbol{w}\|_2}. \tag{2.1.16}$$

Theorem 2.2 (Power Iteration)**.** *Assume that $\lambda_1 > \lambda_i$ for all $i > 1$. If we compute the fixed point of (2.1.16) using the following iteration:*

$$\boldsymbol{v}_0 \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{1}), \qquad \boldsymbol{v}_{t+1} \leftarrow \frac{\boldsymbol{M}\boldsymbol{v}_t}{\|\boldsymbol{M}\boldsymbol{v}_t\|_2}, \tag{2.1.17}$$

then, in the limit, $\boldsymbol{v}_t$ will converge to a top unit-norm eigenvector of $\boldsymbol{M}$.

Proof. First, note that for all t, we have

$$\boldsymbol{v}_t = \frac{\boldsymbol{M}\boldsymbol{v}_{t-1}}{\|\boldsymbol{M}\boldsymbol{v}_{t-1}\|_2} = \frac{\boldsymbol{M}^2\boldsymbol{v}_{t-2}}{\|\boldsymbol{M}\boldsymbol{v}_{t-1}\|_2\|\boldsymbol{M}\boldsymbol{v}_{t-2}\|_2} = \cdots = \frac{\boldsymbol{M}^t\boldsymbol{v}_0}{\prod_{s=1}^t \|\boldsymbol{M}\boldsymbol{v}_s\|_2}. \tag{2.1.18}$$

Thus, $\boldsymbol{v}_t$ has the same direction as $\boldsymbol{M}^t\boldsymbol{v}_0$ and is unit norm, so we can write

$$\boldsymbol{v}_t = \frac{\boldsymbol{M}^t\boldsymbol{v}_0}{\|\boldsymbol{M}^t\boldsymbol{v}_0\|_2}. \tag{2.1.19}$$

Because all the eigenvectors $\boldsymbol{w}_i$ of $\boldsymbol{M}$ form an orthonormal basis for $\mathbb{R}^D$, we can write

$$\boldsymbol{v}_0 = \sum_{i=1}^D \alpha_i \boldsymbol{w}_i, \tag{2.1.20}$$

[2] In fact, as we have mentioned in the previous chapter, PCA was one of the first problems that neural networks were used to solve [BH89; Oja82].

where because $\boldsymbol{v}_0$ is Gaussian the α_i are all nonzero with probability 1. Thus, we can use our earlier expression for $\boldsymbol{v}_t$ to write

$$\boldsymbol{v}_t = \frac{\boldsymbol{M}^t \boldsymbol{v}_0}{\|\boldsymbol{M}^t \boldsymbol{v}_0\|_2} = \frac{\sum_{i=1}^D \lambda_i^t \alpha_i \boldsymbol{w}_i}{\|\sum_{i=1}^D \lambda_i^t \alpha_i \boldsymbol{w}_i\|_2} = \frac{\sum_{i=1}^D \lambda_i^t \alpha_i \boldsymbol{w}_i}{\sqrt{\sum_{i=1}^D \lambda_i^{2t} \alpha_i^2}}. \tag{2.1.21}$$

Now, let us consider the case where $\lambda_1 > \lambda_2 \geq \cdots \geq \lambda_D \geq 0$. (The case with repeated top eigenvalues goes similarly.) Then we can write

$$\boldsymbol{v}_t = \frac{\alpha_1 \boldsymbol{w}_1 + \sum_{i=2}^D (\lambda_i/\lambda_1)^t \alpha_i \boldsymbol{w}_i}{\sqrt{\alpha_1^2 + \sum_{i=2}^D (\lambda_i/\lambda_1)^{2t} \alpha_i^2}}. \tag{2.1.22}$$

Because $\lambda_1 > \lambda_i$ for all $i > 1$, the terms inside the summation go to 0 exponentially fast, and the remainder is the limit

$$\lim_{t\to\infty} \boldsymbol{v}_t = \frac{\alpha_1}{|\alpha_1|} \boldsymbol{w}_1 = \operatorname{sign}(\alpha_1) \boldsymbol{w}_1, \tag{2.1.23}$$

which is a top unit eigenvector of $\boldsymbol{M}$. The top eigenvalue λ_1 of $\boldsymbol{M}$ can be estimated by $\boldsymbol{v}_t^\top \boldsymbol{M} \boldsymbol{v}_t$, which converges similarly rapidly to λ_1. □

Remark 2.5 (Power Iteration as a Denoising Process). Notice that computing the principal component via power iteration can be interpreted as a "denoising" process: starting from random noise, it converges iteratively onto a one-dimensional subspace. Indeed, as we will show later in Chapter 3, if we assume *the support of the data distribution is a single subspace*, the associated denoising operator (or the score function) is precisely the same as the above power iteration. In Chapter 3, we will show how to generalize such a denoising process to distributions of more general low-dimensional structures. This is essentially a universal and most efficient approach to learn any low-dimensional distributions.

To find the second top eigenvector, we apply the power iteration algorithm to $\boldsymbol{M} - \lambda_1 \boldsymbol{v}_1 \boldsymbol{v}_1^\top$, which has eigenvectors $(\boldsymbol{w}_i)_{i=2}^D$ and corresponding eigenvalues $(\lambda_i)_{i=2}^D$. By repeating this procedure d times in sequence, we can very efficiently estimate the top d eigenvectors of $\boldsymbol{M}$ for any symmetric positive semidefinite matrix $\boldsymbol{M}$. Thus we can also apply it to $\boldsymbol{X}\boldsymbol{X}^\top/N$ to recover the top d principal components, which is what we were after in the first place. Notice that this approach recovers one principal component at a time; we will contrast this to other algorithmic approaches, such as gradient descent on all eigenvectors, in future sections.

2.1.3 Local and Incremental Learning via Online PCA

Notice that in the above construction, the linear transform $\boldsymbol{U}$ (i.e., the matrix of principal components) used for encoding and decoding is computed "offline" from all the input data beforehand. One question is whether this transform can be learned "online" as the input data comes in order. This question was answered by the work of Oja in 1982 [Oja82].

Example 2.1 (Normalized Hebbian learning scheme for PCA). Consider a sequence of i.i.d. random vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i, \ldots \in \mathbb{R}^n$ with covariance $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times n}$. Let $\boldsymbol{u}_0 \in \mathbb{R}^n$ and define the response of an input vector $\boldsymbol{x}_i$ against a weight vector $\boldsymbol{u}_i$ to be their inner product:

$$\eta_i = \boldsymbol{u}_i^T \boldsymbol{x}_i \tag{2.1.24}$$

and we update the weight vector according to the following scheme:

$$\boldsymbol{u}_{i+1} = \frac{\boldsymbol{u}_i + \gamma \eta_i \boldsymbol{x}_i}{\|\boldsymbol{u}_i + \gamma \eta_i \boldsymbol{x}_i\|_2} \tag{2.1.25}$$

for some small gain $\gamma > 0$. This update scheme can be viewed as a normalized Hebbian scheme, in which the weights of connections between neurons become stronger if (products of) both the input $\boldsymbol{x}$ and output η are strong. One may view the vector of weights $\boldsymbol{u}$ as "learned" based on a form of feedback from the output η. Then, under reasonable assumptions, Oja [Oja82] has shown that $\boldsymbol{u}_i$ converges to the eigenvector associated with the largest eigenvalue of $\boldsymbol{\Sigma}$. ■

To interpret the normalized Hebbian scheme (2.1.25), let us think of the data sample PCA objective (2.1.13) as an approximation to the *population* version of the objective, defined via an expectation over the distribution of samples $\boldsymbol{x}$:

$$\min_{\boldsymbol{U}^\top \boldsymbol{U} = \boldsymbol{I}} \mathbb{E}_{\boldsymbol{x}} \left[\left\| \boldsymbol{x} - \boldsymbol{U}\boldsymbol{U}^\top \boldsymbol{x} \right\|_2^2 \right] \tag{2.1.26}$$

We will dig into this objective further in the following section. Then the normalized Hebbian scheme can be interpreted as a first-order approximation to a *stochastic* projected gradient descent scheme on Equation (2.1.26) (with batch size 1, and with the number of columns of $\boldsymbol{U}$ equal to 1) as long as $\|\boldsymbol{u}\|_2 = 1$, which is maintained by the projection operation in (2.1.25). Such an *online algorithm* is a useful primitive for more complex algorithms that process sequentially-structured data with causal structure, as we will study more in Section 5.3.3 and Section 8.11. Interestingly, and more generally, the simple online PCA algorithm is suggestive of the fact that *simpler algorithms than (end-to-end) back propagation can succeed in learning encoder-decoder pairs*, which we will briefly revisit in Chapter 9 as an important goal for the future study of intelligence.

2.1.4 The Statistical View: Probabilistic PCA

Notice that the above formulation makes no statistical assumptions on the data-generating process. However, it is common to include statistical elements within a given data model, as it may add further enlightening interpretations about the result of the analysis. As such, we ask the natural question: *what is the statistical analogue to low-dimensional structure?* Our answer is that a low-dimensional *distribution* is one whose support is concentrated around a low-dimensional geometric structure.

To illustrate this point, we discuss *probabilistic principal component analysis (PPCA)* [Row97; TB99]. This formulation can be viewed as a statistical variant of regular PCA. Mathematically, we now consider our data as samples from a random variable $\boldsymbol{x}$ taking values in $\mathbb{R}^D$ (also sometimes called a *random vector*). We say that $\boldsymbol{x}$ has (approximate) low-rank statistical structure if and only if there exists an orthonormal matrix $\boldsymbol{U} \in \mathsf{O}(D, d)$, a random variable $\boldsymbol{z}$ taking values in $\mathbb{R}^d$, and a *small* random variable $\boldsymbol{\varepsilon}$ taking values in $\mathbb{R}^D$ such that $\boldsymbol{z}$ and $\boldsymbol{\varepsilon}$ are independent, and

$$\boldsymbol{x} = \boldsymbol{U}\boldsymbol{z} + \boldsymbol{\varepsilon}. \tag{2.1.27}$$

Our goal is again to recover $\boldsymbol{U}$. Towards this end, we set up the analogous problem as in Section 2.1.1, i.e., optimizing over subspace supports $\tilde{\boldsymbol{U}}$ and random variables $\boldsymbol{z}$ to solve the following problem:

$$\min_{\tilde{\boldsymbol{U}}, \tilde{\boldsymbol{z}}} \mathbb{E}\, \|\boldsymbol{x} - \tilde{\boldsymbol{U}}\tilde{\boldsymbol{z}}\|_2^2. \tag{2.1.28}$$

Since we are finding the best such random variable $\boldsymbol{z}$ we can find its realization $\boldsymbol{z}(\boldsymbol{x})$ separately for each value of $\boldsymbol{x}$. Performing the same calculations as in Section 2.1.1, we obtain

$$\min_{\tilde{\boldsymbol{U}}, \tilde{\boldsymbol{z}}} \mathbb{E}\, \|\boldsymbol{x} - \tilde{\boldsymbol{U}}\tilde{\boldsymbol{z}}\|_2^2 = \min_{\tilde{\boldsymbol{U}}} \mathbb{E} \min_{\tilde{\boldsymbol{z}}(\boldsymbol{x})} \|\boldsymbol{x} - \tilde{\boldsymbol{U}}\tilde{\boldsymbol{z}}(\boldsymbol{x})\|_2^2 = \min_{\tilde{\boldsymbol{U}}} \mathbb{E}\, \|\boldsymbol{x} - \tilde{\boldsymbol{U}}\tilde{\boldsymbol{U}}^\top \boldsymbol{x}\|_2^2, \tag{2.1.29}$$

again re-emphasizing the fact that the estimated subspace with principal components $\boldsymbol{U}^\star$ corresponds to a denoiser $\boldsymbol{U}^\star(\boldsymbol{U}^\star)^\top$ which projects onto that subspace. As before, we obtain

$$\arg\min_{\tilde{\boldsymbol{U}}} \mathbb{E}\, \|\boldsymbol{x} - \tilde{\boldsymbol{U}}\tilde{\boldsymbol{U}}^\top \boldsymbol{x}\|_2^2 = \arg\max_{\tilde{\boldsymbol{U}}} \mathbb{E}\, \|\tilde{\boldsymbol{U}}^\top \boldsymbol{x}\|_2^2 \tag{2.1.30}$$

$$= \arg\max_{\tilde{\boldsymbol{U}}} \operatorname{tr}(\tilde{\boldsymbol{U}}^\top\, \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^\top]\tilde{\boldsymbol{U}}), \tag{2.1.31}$$

and the solution to the latter problem is just the top d principal components of the second moment matrix $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^\top]$. Actually, the above problems are visually very similar to the equations for computing the principal components in the previous subsection, except with $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^\top]$ replacing $\boldsymbol{X}\boldsymbol{X}^\top/N$. In fact, the latter quantity is an estimate for the former. Both formulations effectively do the same thing, and have the same practical solution—compute the left singular vectors of the data matrix $\boldsymbol{X}$, or equivalently the top eigenvectors of the estimated covariance matrix $\boldsymbol{X}\boldsymbol{X}^\top/N$. The statistical formulation, however, has an additional interpretation. Suppose that $\mathbb{E}[\boldsymbol{z}] = \boldsymbol{0}$ and $\mathbb{E}[\boldsymbol{\varepsilon}] = \boldsymbol{0}$. We have

$$\mathbb{E}[\boldsymbol{x}] = \boldsymbol{U}\, \mathbb{E}[\boldsymbol{z}] + \mathbb{E}[\boldsymbol{\varepsilon}] = \boldsymbol{0}, \tag{2.1.32}$$

so that $\operatorname{Cov}[\boldsymbol{x}] = \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^\top]$. Now working out $\operatorname{Cov}[\boldsymbol{x}]$ we have

$$\operatorname{Cov}[\boldsymbol{x}] = \boldsymbol{U} \operatorname{Cov}[\boldsymbol{z}]\boldsymbol{U}^\top + \operatorname{Cov}[\boldsymbol{\varepsilon}] = \boldsymbol{U}\, \mathbb{E}[\boldsymbol{z}\boldsymbol{z}^\top]\boldsymbol{U}^\top + \operatorname{Cov}[\boldsymbol{\varepsilon}]. \tag{2.1.33}$$

In particular, if $\operatorname{Cov}[\boldsymbol{\varepsilon}]$ is small, it holds that $\operatorname{Cov}[\boldsymbol{x}] = \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^\top]$ is approximately a low-rank matrix, in particular rank d. Thus the top d eigenvectors of $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^\top]$

essentially summarize the whole covariance matrix. But they are also the principal components, so we can interpret principal component analysis as performing a low-rank decomposition of $\mathrm{Cov}[\boldsymbol{x}]$.

Remark 2.6. By using the probabilistic viewpoint of PCA, we achieve a clearer and more quantitative understanding of how it relates to denoising. First, consider the denoising problem in (2.1.29), namely

$$\min_{\tilde{\boldsymbol{U}}} \mathbb{E}\,\|\boldsymbol{x} - \tilde{\boldsymbol{U}}\tilde{\boldsymbol{U}}^\top \boldsymbol{x}\|_2^2. \tag{2.1.34}$$

It is not too hard to prove that if $\tilde{\boldsymbol{U}}$ has d columns and if $\boldsymbol{\varepsilon}$ is an isotropic Gaussian random variable, i.e., with distribution $\boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$,[3] then for *any* optimal solution $\boldsymbol{U}^\star$ to this problem, we have

$$\boldsymbol{U}^\star (\boldsymbol{U}^\star)^\top = \boldsymbol{U}\boldsymbol{U}^\top \tag{2.1.35}$$

and so the true supporting subspace, say $\mathcal{S} \doteq \mathrm{col}(\boldsymbol{U})$, is recovered as the span of columns of $\boldsymbol{U}^\star$, since

$$\mathcal{S} = \mathrm{col}(\boldsymbol{U}) = \mathrm{col}(\boldsymbol{U}\boldsymbol{U}^\top) = \mathrm{col}(\boldsymbol{U}^\star(\boldsymbol{U}^\star)^\top) = \mathrm{col}(\boldsymbol{U}^\star). \tag{2.1.36}$$

In particular, the learned *denoising map* $\boldsymbol{U}^\star(\boldsymbol{U}^\star)^\top$ is an orthogonal projection onto $\mathcal{S}$, pushing noisy points onto the ground truth supporting subspace. We can establish a similar technical result in the case where we only have finite samples, as in Theorem 2.1, but this takes more effort and technicality. Summarizing this discussion, we have the following informal Theorem.

Theorem 2.3. *Suppose that the random variable $\boldsymbol{x}$ can be written as*

$$\boldsymbol{x} = \boldsymbol{U}\boldsymbol{z} + \boldsymbol{\varepsilon} \tag{2.1.37}$$

where $\boldsymbol{U} \in \mathsf{O}(D, d)$ captures the low-rank structure, $\boldsymbol{z}$ is a random vector taking values in $\mathbb{R}^d$, and $\boldsymbol{\varepsilon}$ is a random vector taking values in $\mathbb{R}^D$ such that $\boldsymbol{z}$ and $\boldsymbol{\varepsilon}$ are independent, and $\boldsymbol{\varepsilon}$ is small. Then the principal components $\boldsymbol{U}^\star \in \mathsf{O}(D, d)$ of our dataset are given by the top d eigenvectors of $\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^\top]$, and approximately correspond to the optimal linear denoiser: $\boldsymbol{U}^\star(\boldsymbol{U}^\star)^\top \approx \boldsymbol{U}\boldsymbol{U}^\top$.

2.1.5 Masked Low-Rank Matrix Completion

In the previous subsections, we discussed the problem of *learning a low-rank geometric or statistical distribution*, where the data were sampled from a subspace with additive noise. But this is not the only type of disturbance from a low-dimensional distribution that is worthwhile to study. In this subsection, we introduce one more class of non-additive errors which become increasingly important in deep learning. Let us consider the case where we have some data $\{\boldsymbol{x}_i\}_{i=1}^N$ generated according to (2.1.1). Now we arrange them into a matrix

[3] Other distributions work so long as they support all of $\mathbb{R}^D$, but the Gaussian is the easiest to work with here.

$\boldsymbol{X} = \begin{bmatrix}\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\end{bmatrix} \in \mathbb{R}^{D\times N}$. Unlike before, we do not observe $\boldsymbol{X}$ directly; we instead imagine that our observation was corrupted en route and we obtained

$$\boldsymbol{Y} = \boldsymbol{M} \odot \boldsymbol{X}, \tag{2.1.38}$$

where $\boldsymbol{M} \in \{0,1\}^{D\times N}$ is a *mask* that is known by us, and $\odot$ is element-wise multiplication. In this case, our goal is to recover $\boldsymbol{X}$ (from which point we can use PCA to recover $\boldsymbol{U}$, etc), given only the corrupted observation $\boldsymbol{Y}$, the mask $\boldsymbol{M}$, and the knowledge that $\boldsymbol{X}$ is (approximately) low-rank. This is called *low-rank matrix completion*.

This and similar generalizations of this low-rank structure recovery problem are referred to as "Robust PCA" problems. There have been many excellent resources discussing algorithms and approaches to solve these problems [WM22]. Hence we will not go into the solution method here. Instead, we will discuss under what conditions this problem is *plausible* to solve. On one hand, in the most absurd case, suppose that each entry of the matrix $\boldsymbol{X}$ were chosen independently from all the others. Then there would be no hope of recovering $\boldsymbol{X}$ exactly even if only one entry were missing and we had $DN - 1$ entries. On the other hand, suppose that we knew that $\boldsymbol{X}$ has rank 1 exactly, which is an extremely strong condition on the low-dimensional structure of the data, and we were dealing with the mask

$$\boldsymbol{M} = \begin{bmatrix}\mathbf{1}_{(D-1)\times 1} & \mathbf{0}_{(D-1)\times(N-1)} \\ 1 & \mathbf{1}_{1\times(N-1)}\end{bmatrix}. \tag{2.1.39}$$

Then we know that the data are distributed on a line, and we know a vector on that line—it is just the first column of the matrix $\boldsymbol{Y} = \boldsymbol{M} \odot \boldsymbol{X}$. From the last coordinate of each column, also revealed to us by the mask, we can solve for each column since for each final coordinate there is only one vector on the line with this coordinate. Thus we can reconstruct $\boldsymbol{X}$ with perfect accuracy, and we only need a linear number of observations $D + N - 1$.

In the real world, actual problems are somewhere in between the two limiting cases discussed above. Yet the differences between these two extremes, as well as the earlier discussion of PCA, reveal a general kernel of truth:

> *The lower-dimensional and more structured the data distribution is, the easier it is to process, and the fewer observations are needed—provided that the algorithm effectively utilizes this low-dimensional structure.*

As is perhaps predictable, we will encounter this motif repeatedly in the remainder of the manuscript, starting in the very next section.[4]

[4] In later chapters, we will see that one can complete real-world data such as images with only a fraction of unmasked observations, e.g., the masked autoencoding of images in Section 8.5. This is also because the distribution of natural images has rather low-dimensional (albeit nonlinear) structures in the space of all images.

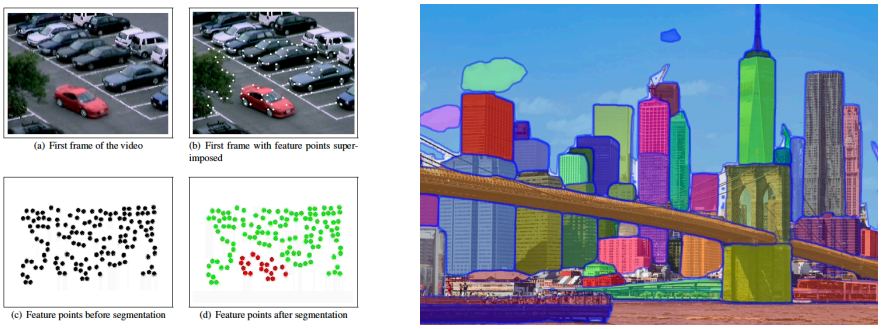


Figure 2.4: **Left:** features tracked on independently moving objects in a scene. **Right:** image patches associated with different regions of an image.

2.2 A Mixture of Complete Low-Dimensional Subspaces

As we have seen, low-rank signal models are rich enough to provide a full picture of the interplay between low-dimensionality in data and efficient and scalable computational algorithms for representation and recovery under errors. These models imply a *linear* and symmetric representation learning pipeline (2.1.6):

$$\boldsymbol{z} = \mathcal{E}(\boldsymbol{x}) = \boldsymbol{U}^\top \boldsymbol{x}, \quad \hat{\boldsymbol{x}} = \mathcal{D}(\boldsymbol{z}) = \boldsymbol{U}\boldsymbol{z},$$

which can be provably learned from finite samples of $\boldsymbol{x}$ with principal component analysis (solved efficiently, say, with the power method) whenever the distribution of $\boldsymbol{x}$ truly is linear. This is a restrictive assumption—for as Harold Hotelling, the distinguished 20th century statistician,[5] objected following George Dantzig's presentation of his theory of linear programming for the first time [Dan02],

"...we all know the world is nonlinear."

Even accounting for its elegance and simplicity, the low-rank assumption is too restrictive to be broadly applicable to modeling real-world data. A key limitation is the assumption of a *single* linear subspace that is responsible for generating the structured observations. In many practical applications, structure generated by a *mixture* of distinct low-dimensional subspaces provides a more realistic model. For example, consider a video sequence that captures the motion of several distinct objects, each subject to its own independent displacement (Figure 2.4 left). Under suitable assumptions on the individual motions, each object becomes responsible for an independent low-dimensional subspace in the concatenated sequence of video frames [VM04]. As another example, consider modeling natural images via learning a model for the distribution of

[5] By coincidence, also famous for his contributions to the development and naming of Principal Component Analysis [Hot33].

patches, spatially-contiguous collections of pixels, within an image (Figure 2.4 right). Unlike in the simplistic setting of Figure 2.2, where images of faces with matched poses can be well-approximated by a single low-dimensional subspace, the patch at a specific location in a natural image can correspond to objects with very different properties—for example, distinct color or shape due to occlusion boundaries. Therefore, modeling the distribution of patches with a single subspace is futile, but a *mixture* of subspaces, one for each region, performs surprisingly well in practice, say for segmenting or compressing purposes [MRY+11].[6] We will see a concrete example in a later chapter (Example 4.10).

In this section, we will begin by discussing the conceptual and algorithmic foundations for structured representation learning when the data distribution can be modeled by *a mixture of low-dimensional subspaces*, as illustrated in Figure 2.1. In this setting, the decoder mapping will be almost as simple as the case of a single subspace: we simply represent via

$$\hat{\boldsymbol{x}} = \mathcal{D}(\boldsymbol{z}) = \left(\sum_{k=1}^{K} \pi_k(\boldsymbol{z}) \boldsymbol{U}_k \right) \boldsymbol{z}, \tag{2.2.1}$$

where $\pi_k : \mathbb{R}^d \to \{0, 1\}$ are a set of *sparse* weighting random variables, such that only a single subspace $\mathcal{S}_k = \mathrm{col}(\boldsymbol{U}_k)$ is selected in the sum. However, the task of encoding such data $\boldsymbol{x}$ to suitable representations $\boldsymbol{z}$, and learning such an encoder-decoder pair from data, will prove to be more involved.

We will see how ideas from the rich literature on *sparse representation* and *independent component analysis* lead to a natural reformulation of the above decoder architecture through the lens of sparsity, the corresponding encoder architecture (obtained through a power-method-like algorithm analogous to that of principal component analysis), and strong guarantees of correctness and efficiency for learning such encoder-decoder pairs from data. In this sense, the case of mixed linear low-dimensional structure already leads to many of the key conceptual components of structured representation learning that we will develop in far greater generality in this book.

2.2.1 Mixtures of Subspaces and Sparsely-Used Dictionaries

Let $\boldsymbol{U}_1, \ldots, \boldsymbol{U}_K$, each of size $D \times d$, denote a collection of orthonormal bases for K subspaces of dimension d in $\mathbb{R}^D$. To say that $\boldsymbol{x}$ follows a mixture-of-subspaces distribution parameterized by $\boldsymbol{U}_1, \ldots, \boldsymbol{U}_K$ means, geometrically speaking, that

$$\boldsymbol{x} = \boldsymbol{U}_k \boldsymbol{z} \quad \text{for some } \; k \in [K], \;\; \boldsymbol{z} \in \mathbb{R}^d. \tag{2.2.2}$$

The statistical analogue of this geometric model, as we saw for the case of PCA and linear structure, is that $\boldsymbol{x}$ follows a *mixture of Gaussians* distribution: that

[6] We will return to this observation in Chapter 5, where we will show it can be significantly generalized to learn representations for large-scale modern datasets.

is,

$$\boldsymbol{x} \sim \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{0}, \boldsymbol{U}_k \boldsymbol{U}_k^\top), \quad \text{for some } \pi_k \geq 0, \ \sum_{k=1}^{K} \pi_k = 1. \tag{2.2.3}$$

In other words, for each $k \in [K]$, $\boldsymbol{x}$ is Gaussian on the low-dimensional subspace $\mathrm{col}(\boldsymbol{U}_k)$ with probability π_k.

Remark 2.7 (A Mixture of Gaussians v.s. A Superposition of Gaussians). One should be aware that the above model (2.2.3) is a *mixture* of Gaussian distributions, not to be confused with a mixture of Gaussian variables by superposition, say

$$\boldsymbol{x} = \sum_{i=1}^{n} w_i \boldsymbol{x}_i, \quad \boldsymbol{x}_i \sim \mathcal{N}(\mathbf{0}, \boldsymbol{U}_i \boldsymbol{U}_i^\top), \tag{2.2.4}$$

where $\boldsymbol{x}_i$ are independent random Gaussian vectors and w_i are a set of fixed weights. As we know from the properties of Gaussian vectors, such a superposition $\boldsymbol{x}$ will remain a Gaussian distribution.

For now, we focus on the geometric perspective offered by (2.2.2). There is an algebraically convenient alternative to this conditional representation. Consider a *lifted* representation vector $\boldsymbol{z} = [\boldsymbol{z}_1^\top, \ldots, \boldsymbol{z}_K^\top]^\top \in \mathbb{R}^{dK}$, such that $\boldsymbol{z}$ is *d-sparse* with support on one of the K consecutive non-overlapping blocks of d coordinates out of dK. Then (2.2.2) can be written equivalently as

$$\boldsymbol{x} = \underbrace{\begin{bmatrix} | & & | \\ \boldsymbol{U}_1 & \cdots & \boldsymbol{U}_K \\ | & & | \end{bmatrix}}_{\boldsymbol{U}} \underbrace{\begin{bmatrix} \boldsymbol{z}_1 \\ \vdots \\ \boldsymbol{z}_K \end{bmatrix}}_{\boldsymbol{z}}, \quad \left\| \begin{bmatrix} \|\boldsymbol{z}_1\|_2 \\ \vdots \\ \|\boldsymbol{z}_K\|_2 \end{bmatrix} \right\|_0 = 1. \tag{2.2.5}$$

Here, the ℓ^0 "norm" $\|\cdot\|_0$ measures sparsity by counting the number of nonzero entries:

$$\|\boldsymbol{z}\|_0 = |\{i \mid z_i \neq 0\}|, \tag{2.2.6}$$

and the matrix $\boldsymbol{U} \in \mathbb{R}^{D \times Kd}$ is called a *dictionary* with all the $\{\boldsymbol{U}_i\}_{i=1}^K$ as code words. In general, if the number of subspaces in the mixture K is large enough, there is no bound on the number of columns contained in the dictionary $\boldsymbol{U}$. In the case where $Kd < D$, $\boldsymbol{U}$ is called *undercomplete*; when $Kd = D$, it is called *complete*; and when $Kd > D$, it is called *overcomplete*.

Now, (2.2.5) suggests a convenient relaxation for tractability of analysis: rather than modeling $\boldsymbol{x}$ as coming from a mixture of K *specific* subspaces $\boldsymbol{U}_1, \ldots, \boldsymbol{U}_K$, we may instead start with a dictionary $\boldsymbol{U} \in \mathbb{R}^{D \times m}$, where m may be smaller or larger than D, and simply seek to represent $\boldsymbol{x} = \boldsymbol{U}\boldsymbol{z}$ with the sparsity $\|\boldsymbol{z}\|_0$ sufficiently small. This leads to the *sparse dictionary model* for $\boldsymbol{x}$:

$$\boldsymbol{x} = \boldsymbol{U}\boldsymbol{z} + \boldsymbol{\varepsilon}, \quad \|\boldsymbol{z}\|_0 \ll d, \tag{2.2.7}$$

where $\boldsymbol{\varepsilon}$ represents an unknown noise vector. Geometrically, this implies that $\boldsymbol{x}$ lies close to the span of a subset of $\|\boldsymbol{z}\|_0$ columns of $\boldsymbol{U}$, making this an instantiation of the mixture-of-subspaces model (2.2.2) with a very large value of K, and specific correlations between the subspaces $\boldsymbol{U}_k$.

Orthogonal dictionary for sparse coding. Now we can formulate the structured representation learning problem for mixtures of low-dimensional subspaces that we will study in this section. We assume that we have samples $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]$ from an unknown sparse dictionary model (2.2.7), possibly with added noises $\boldsymbol{\varepsilon}_i$. Let us begin from the assumption that the dictionary $\boldsymbol{U}$ in the sparse dictionary model (2.2.7) is complete and orthogonal,[7] and that each coefficient vector $\boldsymbol{z}$ is d-sparse, with $d \ll D$. In this setting, representation learning amounts to correctly learning the orthogonal dictionary $\boldsymbol{U}$ via optimization: we can then take $\mathcal{E}(\boldsymbol{x}) = \boldsymbol{U}^\top \boldsymbol{x}$ as the encoder and $\mathcal{D}(\boldsymbol{z}) = \boldsymbol{U}\boldsymbol{z}$ as the decoder, and $\mathcal{D} = \mathcal{E}^{-1}$. In diagram form:

$$\boldsymbol{x} \xrightarrow{\mathcal{E}=\boldsymbol{U}^\top} \boldsymbol{z} \xrightarrow{\mathcal{D}=\boldsymbol{U}} \hat{\boldsymbol{x}}. \tag{2.2.8}$$

We see that the autoencoding pair $(\mathcal{E}, \mathcal{D})$ for complete dictionary learning is symmetric, as in the case of a single linear subspace, making the computational task of encoding and decoding no more difficult than in the linear case. On the other hand, the task of learning the dictionary $\boldsymbol{U}$ is strictly more difficult than learning a single linear subspace by PCA. To see why we cannot simply use PCA to learn the orthogonal dictionary $\boldsymbol{U}$ correctly, note that the loss function that gave rise to PCA, namely (2.1.5), is completely invariant to rotations of the rows of the matrix $\boldsymbol{U}$: that is, if $\boldsymbol{Q}$ is any $d \times d$ orthogonal matrix, then $\boldsymbol{U}$ and $\boldsymbol{U}\boldsymbol{Q}$ are both feasible and have an identical loss for (2.1.5). The sparse dictionary model is decidedly not invariant to such transformations: if we replaced $\boldsymbol{U}$ by $\boldsymbol{U}\boldsymbol{Q}$ and made a corresponding rotation $\boldsymbol{Q}^\top \boldsymbol{z}$ of the representation coefficients $\boldsymbol{z}$, we would in general destroy the sparsity structure of $\boldsymbol{z}$, violating the modeling assumption. Thus, we need new algorithms for learning orthogonal dictionaries.

2.2.2 Complete Dictionary Learning

In this section, we will derive algorithms for solving the orthogonal dictionary learning problem. To be more precise, we assume that the observed vector $\boldsymbol{x} \in \mathbb{R}^D$ follows a statistical model

$$\boldsymbol{x} = \boldsymbol{U}\boldsymbol{z} + \boldsymbol{\varepsilon}, \tag{2.2.9}$$

where $\boldsymbol{U} \in \mathbb{R}^{D \times D}$ is an unknown orthogonal dictionary, $\boldsymbol{z}$ is a random vector with statistically independent components z_i, each with zero mean, and $\boldsymbol{\varepsilon} \in \mathbb{R}^D$ is an independent random vector of small (Gaussian) noises. The goal is to recover $\boldsymbol{U}$ (and hence $\boldsymbol{z}$) from samples of $\boldsymbol{x}$.

Here we assume that each independent component z_i is distributed as

$$z_i \sim \mathrm{Ber}(\theta) \cdot \mathcal{N}(0, 1/\theta).$$

That is, it is the product of a Bernoulli random variable with probability θ of being 1 and $1 - \theta$ of being 0, and an independent Gaussian random variable with variance $1/\theta$. This distribution is formally known as the *Bernoulli-Gaussian* distribution. The normalization is chosen so that $\mathrm{Var}(z_i) = 1$ and

[7] It can be shown that for the complete case, we do not lose any generality by making the orthogonal assumption (Exercise 2.4).

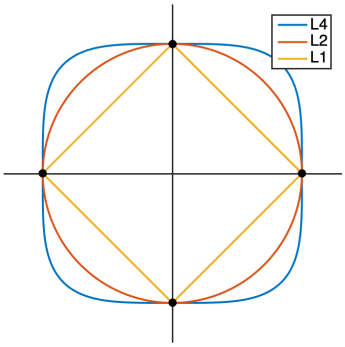


Figure 2.5: Maximizing ℓ^4 norm or minimizing ℓ^1 norm promotes sparsity (for vectors on the sphere).

hence $\mathbb{E}[\|\boldsymbol{z}\|_2^2] = d$. This modeling assumption implies that the vector of independent components $\boldsymbol{z}$ is typically very sparse: we calculate $\mathbb{E}\left[\|\boldsymbol{z}\|_0\right] = d\theta$, which is small when θ is inversely proportional to d.

Remark 2.8 (The Orthogonal Assumption). At first sight, the assumption that the dictionary $\boldsymbol{U}$ is orthogonal might seem to be somewhat restrictive. But there is actually no loss of generality. One may consider a complete dictionary to be any square invertible matrix $\boldsymbol{U}$. With samples generated from this dictionary: $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{Z} \in \mathbb{R}^{D\times N}$, it is easy to show that with some preconditioning of the data matrix $\boldsymbol{X}$:

$$\bar{\boldsymbol{X}} = \left(\frac{1}{N\theta}\boldsymbol{X}\boldsymbol{X}^\top\right)^{-\frac{1}{2}}\boldsymbol{X}, \tag{2.2.10}$$

then there exists an orthogonal matrix $\boldsymbol{U}_o \in \mathsf{O}(D)$ such that

$$\bar{\boldsymbol{X}} = \boldsymbol{U}_o\boldsymbol{Z}. \tag{2.2.11}$$

See Exercise 2.4 or [SQW17b] for more details.

Dictionary learning via the MSP algorithm. Now suppose that we are given a set of observations:

$$\boldsymbol{x}_i = \boldsymbol{U}\boldsymbol{z}_i + \boldsymbol{\varepsilon}_i,\ \forall i \in [N]. \tag{2.2.12}$$

Let $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]$ and $\boldsymbol{Z} = [\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N]$. The goal is to recover $\boldsymbol{U}$ from the data $\boldsymbol{X}$. Therefore, given any orthogonal matrix $\boldsymbol{A} \in \mathsf{O}(D)$,

$$\boldsymbol{A}\boldsymbol{x}_i = \boldsymbol{A}\boldsymbol{U}\boldsymbol{z}_i + \boldsymbol{A}\boldsymbol{\varepsilon}_i \tag{2.2.13}$$

would be nearly sparse if $\boldsymbol{A} = \boldsymbol{U}^\top$ (as by assumption the noise $\boldsymbol{\varepsilon}_i$ is of small magnitude).

Also, given $\boldsymbol{U}$ is orthogonal and the fact $\boldsymbol{\varepsilon}$ is small, the vector $\boldsymbol{x}$ has a predictable expected norm, i.e., $\mathbb{E}[\|\boldsymbol{x}\|_2^2] \approx \mathbb{E}[\|\boldsymbol{z}\|_2^2] = d$. It is a known fact that for vectors on a sphere, maximizing the ℓ^4 norm is equivalent to minimizing the ℓ^0 norm (for promoting sparsity),

$$\arg\max_{\boldsymbol{z}\in\mathbb{S}^n} \|\boldsymbol{z}\|_4 \quad \Leftrightarrow \quad \arg\min_{\boldsymbol{z}\in\mathbb{S}^n} \|\boldsymbol{z}\|_0. \tag{2.2.14}$$

This is illustrated in Figure 2.5.

An orthogonal matrix $\boldsymbol{A}$ preserves the Euclidean (ℓ^2) norm: $\|\boldsymbol{A}\boldsymbol{x}\|_2^2 = \|\boldsymbol{x}\|_2^2$. Therefore, to find the correct orthogonal dictionary $\boldsymbol{U}$ from $\boldsymbol{X}$, we may try to solve the following optimization program:

$$\max_{\tilde{\boldsymbol{A}}\in\mathsf{O}(D)} \frac{1}{4}\left\|\tilde{\boldsymbol{A}}\boldsymbol{X}\right\|_4^4 = \frac{1}{4}\sum_{i=1}^{N}\left\|\tilde{\boldsymbol{A}}\boldsymbol{x}_i\right\|_4^4 \tag{2.2.15}$$

This is known as the ℓ^4 maximization problem [ZMZ+20]. After we find the solution $\boldsymbol{A}^\star$, we can take the transpose $\boldsymbol{U}^\star = (\boldsymbol{A}^\star)^\top$.

Remark 2.9. It is also known that for vectors on a sphere, minimizing the ℓ^1 norm is equivalent to minimizing the ℓ^0 norm (for promoting sparsity),

$$\arg\min_{\boldsymbol{z}\in\mathbb{S}^n} \|\boldsymbol{z}\|_1 \quad \Leftrightarrow \quad \arg\min_{\boldsymbol{z}\in\mathbb{S}^n} \|\boldsymbol{z}\|_0,$$

which is also illustrated in Figure 2.5. This fact can also be exploited to learn the dictionary $\boldsymbol{A}$ effectively and efficiently. This was actually explored earlier than the ℓ^4 norm used here. Interested readers may refer to the work of [QZL+20b].

Note that the above problem is equivalent to the following constrained optimization problem:

$$\min -\frac{1}{4}\left\|\tilde{\boldsymbol{A}}\boldsymbol{X}\right\|_4^4 \quad \text{subject to} \quad \tilde{\boldsymbol{A}}^\top\tilde{\boldsymbol{A}} = \boldsymbol{I}. \tag{2.2.16}$$

As shown in [WM22], using the Lagrange multiplier method, one can derive that the optimal solution to the problem should satisfy the following "fixed point" condition:

$$\boldsymbol{A}^\star = \mathcal{P}_{\mathrm{O}(D)}[(\boldsymbol{A}^\star\boldsymbol{X})^{\odot 3}\boldsymbol{X}^\top], \tag{2.2.17}$$

where $\mathcal{P}_{\mathrm{O}(D)}[\,\cdot\,]$ is a projection onto the space of orthogonal matrices $\mathrm{O}(D)$.[8]

To compute the fixed point for the above equation, similar to how we computed eigenvectors for PCA (2.1.16), we may take the following power iteration:

$$\tilde{\boldsymbol{A}}_{t+1} = \mathcal{P}_{\mathrm{O}(D)}[(\tilde{\boldsymbol{A}}_t\boldsymbol{X})^{\odot 3}\boldsymbol{X}^\top]. \tag{2.2.18}$$

This is known as the *matching, stretching, and projection* (MSP) algorithm proposed by [ZMZ+20]. It was shown that under broad conditions such a greedy algorithm indeed converges to the correct solution at a superlinear rate.

[8] For any matrix $\boldsymbol{M}$ with SVD $\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$, $\mathcal{P}_{\mathrm{O}(D)}[\boldsymbol{M}] = \boldsymbol{U}\boldsymbol{V}^\top$. We leave this as an exercise for the reader.

Remark 2.10 (Global Optimality of ℓ^4 Maximization). The constrained ℓ^4 maximization problem is a nonconvex program. In general one should *not* expect that any greedy (say gradient-descent type) algorithms would converge to the globally optimal solution. Surprisingly, one can show that, unlike general nonconvex programs, the landscape of ℓ^4 maximization over a sphere

$$\min -\frac{1}{4}\left\|\tilde{\boldsymbol{q}}^\top \boldsymbol{X}\right\|_4^4 \quad \text{subject to} \quad \tilde{\boldsymbol{q}}^\top \tilde{\boldsymbol{q}} = 1. \tag{2.2.19}$$

is very benign: All local minima are close to the global optima and all critical points are saddle points with a direction of negative curvature. Hence, any descent method with the ability of escaping strict saddle points provably finds global optimal solutions. For more precise statements, interested readers may refer to [QZL+20a].

Remark 2.11 (Stable Deep Linear Network). The above iterative process of computing the dictionary has a natural incremental "deep learning" interpretation. Let us define $\delta \boldsymbol{A}_{t+1} = \tilde{\boldsymbol{A}}_{t+1}\tilde{\boldsymbol{A}}_t^\top$ and $\boldsymbol{Z}_t = \tilde{\boldsymbol{A}}_t \boldsymbol{X}$, then it is easy to show that

$$\delta \boldsymbol{A}_{t+1} = \mathcal{P}_{\mathrm{O}(D)}[(\boldsymbol{Z}_t)^{\circ 3}\boldsymbol{Z}_t^\top].$$

If $\tilde{\boldsymbol{A}}_t$ converges to the correct dictionary $\boldsymbol{U}$, then the above iterative encoding process is essentially equivalent to a "deep linear network":

$$\boldsymbol{Z} \longleftarrow \boldsymbol{Z}_{t+1} = \underbrace{\delta \boldsymbol{A}_{t+1}\delta \boldsymbol{A}_t \ldots \delta \boldsymbol{A}_1}_{\text{forward constructed layers}} \boldsymbol{X}.$$

Note that the computation of the increment transforms $\delta \boldsymbol{A}_{t+1}$ at each layer depends only on the feature output from the previous layer $\boldsymbol{Z}_t$. The network is naturally stable as each layer is a norm-preserving orthogonal transform. Despite its resemblance to a linear deep network, backpropagation is unnecessary to learn each layer. All layers are learned in one forward pass!

2.2.3 Connection to ICA and Kurtosis

With the Bernoulli-Gaussian model, the variables z_i are independent and non-Gaussian. Then, there is a clear correspondence between the dictionary learning and the classic independent component analysis (ICA), to the extent that algorithms to solve one problem can be used to solve the other.[9]

Towards deriving an algorithm based on ICA, we focus on an objective function known as *kurtosis*, which is used in ICA as a direct consequence of the non-Gaussianity of the components. The *kurtosis*, or fourth-order cumulant, of a zero-mean random variable X is defined as

$$\operatorname{kurt}(X) = \mathbb{E}\, X^4 - 3(\mathbb{E}\, X^2)^2. \tag{2.2.20}$$

[9] We explore this issue in more depth in Exercise 2.3, where a connection between non-Gaussianity of the independent components and the purely geometric notion of symmetry is made. This issue is related to our observation above that PCA does not work for recovering sparsely-used orthogonal dictionaries: in the statistical setting, it can be related to rotational invariance of the Gaussian distribution (Exercise 2.2).

If we have only finite samples from the random variable X arranged into a vector $\boldsymbol{x} = [x_1, \dots, x_N]$, we define kurtosis through their empirical average, which yields

$$\operatorname{kurt}(\boldsymbol{x}) = \frac{1}{N}\|\boldsymbol{x}\|_4^4 - \frac{3}{N^2}\|\boldsymbol{x}\|_2^4. \tag{2.2.21}$$

Finally, for random vectors, we define their kurtosis as the sum of each component's scalar kurtosis. Kurtosis is a natural loss function for ICA because for Gaussian X, kurtosis is zero; the reader can verify further that the Bernoulli-Gaussian distribution has positive kurtosis. Thus a natural procedure for seeking non-Gaussian independent components is to search for a set of mutually-orthogonal directions $\tilde{\boldsymbol{U}} \in \mathbb{R}^{d \times k}$, such that $\tilde{\boldsymbol{U}}^\top \boldsymbol{X}$ has maximal kurtosis, where $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{Z} \in \mathbb{R}^{D \times N}$ is the Bernoulli-Gaussian ICA data matrix (such that $\tilde{\boldsymbol{U}}$ approximates $\boldsymbol{U}$ up to permutations of the columns). Formally, we seek to solve the problem

$$\max_{\tilde{\boldsymbol{U}}^\top \tilde{\boldsymbol{U}} = \boldsymbol{I}} \operatorname{kurt}(\tilde{\boldsymbol{U}}^\top \boldsymbol{X}). \tag{2.2.22}$$

At one extreme, we may set $k = D$ and seek to recover the entire dictionary $\boldsymbol{U}$ in a single shot. It can be shown that this problem can be solved with the MSP algorithm we have seen previously. At the other extreme, we may set $k = 1$ and seek to recover a single direction (column of $\boldsymbol{U}$) at a time, performing *deflation*, i.e., replacing the data matrix $\boldsymbol{X}$ by $(\boldsymbol{I} - \boldsymbol{u}\boldsymbol{u}^\top)\boldsymbol{X}$, after each step before finding another direction. There is a natural tradeoff between the scalability of the $k = 1$ incremental approach and the efficiency and robustness of the $k = D$ approach.

Incremental ICA: correctness and FastICA algorithm. The FastICA algorithm, advanced by Hyvärinen and Oja [HO97], is a fast fixed-point algorithm for solving the $k = 1$ kurtosis maximization scheme for ICA. The problem at hand is

$$\max_{\|\tilde{\boldsymbol{u}}\|_2^2 = 1} \operatorname{kurt}(\boldsymbol{X}^\top \tilde{\boldsymbol{u}}). \tag{2.2.23}$$

First, we will perform some very basic analysis of this objective to verify its correctness. Notice by the change of variables $\tilde{\boldsymbol{w}} = \boldsymbol{U}^\top \tilde{\boldsymbol{u}}$ that this problem is equivalent to

$$\max_{\|\tilde{\boldsymbol{w}}\|_2^2 = 1} \operatorname{kurt}(\boldsymbol{Z}^\top \tilde{\boldsymbol{w}}).$$

This objective is simple enough that we can make strong statements about its correctness as a formulation for recovering the dictionary $\boldsymbol{U}$. For example, in the population setting where $N \to \infty$, we may use additivity properties of the kurtosis (Exercise 2.5) and our assumed normalization on the independent components to write the previous problem equivalently as

$$\max_{\|\tilde{\boldsymbol{w}}\|_2^2 = 1} \sum_{i=1}^{d} \operatorname{kurt}(z_i)\tilde{w}_i^4. \tag{2.2.24}$$

It can be shown that under the Bernoulli-Gaussian assumption, the optimization landscape of this problem is "benign" (Exercise 2.7)—meaning that all local maxima of the objective function correspond to the recovery of one of the independent components.

In practice, we cannot directly compute the change of variables sending $\tilde{\boldsymbol{u}}$ to $\tilde{\boldsymbol{w}}$ because it requires the ground-truth dictionary $\boldsymbol{U}$, so we develop algorithms directly for the $\tilde{\boldsymbol{u}}$ parameterization and the data matrix $\boldsymbol{X}$. Then one efficient and scalable way to compute one of these maxima is via first-order optimization algorithms, which iteratively follow the gradient of the objective function and project onto the constraint set $\{\tilde{\boldsymbol{u}} \mid \|\tilde{\boldsymbol{u}}\|_2^2 = 1\}$. Since we have assumed that each z_i satisfies $\mathrm{Var}(z_i) = 1$, we have for any $\tilde{\boldsymbol{u}}$, for large enough N

$$\mathrm{kurt}(\boldsymbol{X}^\top \tilde{\boldsymbol{u}}) \approx \tfrac{1}{N}\|\boldsymbol{X}^\top \tilde{\boldsymbol{u}}\|_4^4 - 3\|\tilde{\boldsymbol{u}}\|_2^4. \tag{2.2.25}$$

We can then derive a corresponding approximation to the gradient:

$$\nabla_{\tilde{\boldsymbol{u}}} \mathrm{kurt}(\boldsymbol{X}^\top \tilde{\boldsymbol{u}}) \approx \tfrac{4}{N}\boldsymbol{X}(\boldsymbol{X}^\top \tilde{\boldsymbol{u}})^{\odot 3} - 12\|\tilde{\boldsymbol{u}}\|_2^2 \tilde{\boldsymbol{u}}.$$

The FastICA algorithm uses a fixed-point method to compute directions of maximum kurtosis. It starts from the first-order optimality conditions for the kurtosis maximization problem, given the preceding gradient approximation and the constraint set, which read

$$\boldsymbol{X}(\boldsymbol{X}^\top \tilde{\boldsymbol{u}})^{\odot 3} = \underbrace{\left\langle \tilde{\boldsymbol{u}}, \boldsymbol{X}(\boldsymbol{X}^\top \tilde{\boldsymbol{u}})^{\odot 3} \right\rangle}_{\lambda} \tilde{\boldsymbol{u}}, \tag{2.2.26}$$

where the specific value of λ is determined using the unit norm constraint on $\tilde{\boldsymbol{u}}$. Exercise 2.6 describes the mathematical background necessary to derive these optimality conditions from first principles. Equation (2.2.26) is satisfied by *any* critical point of the kurtosis maximization problem; we want to derive an equation satisfied by only the maximizers. After noticing that $\lambda = \|\boldsymbol{X}^\top \tilde{\boldsymbol{u}}\|_4^4$, we equivalently re-express (2.2.26) as the modified equation

$$\frac{1}{N}\boldsymbol{X}(\boldsymbol{X}^\top \tilde{\boldsymbol{u}})^{\odot 3} - 3\tilde{\boldsymbol{u}} = \left(\frac{\lambda}{N} - 3\right)\tilde{\boldsymbol{u}}, \tag{2.2.27}$$

and realize that any maximizer of (2.2.23) must satisfy $\lambda/N - 3 > 0$, assuming that N is sufficiently large. Hence, we may *normalize* both sides of (2.2.27), giving the following fixed-point equation satisfied by every maximizer of (2.2.23):

$$\frac{\frac{1}{N}\boldsymbol{X}(\boldsymbol{X}^\top \tilde{\boldsymbol{u}})^{\odot 3} - 3\tilde{\boldsymbol{u}}}{\left\|\frac{1}{N}\boldsymbol{X}(\boldsymbol{X}^\top \tilde{\boldsymbol{u}})^{\odot 3} - 3\tilde{\boldsymbol{u}}\right\|_2} = \tilde{\boldsymbol{u}}. \tag{2.2.28}$$

Iterating the mapping defined by the lefthand side of this fixed point expression then yields the FastICA algorithm of Hyvärinen and Oja [HO97]:

$$\tilde{\boldsymbol{v}}^+ = \tfrac{1}{N}\boldsymbol{X}(\boldsymbol{X}^\top \tilde{\boldsymbol{u}})^{\odot 3} - 3\tilde{\boldsymbol{u}}, \tag{2.2.29}$$

$$\tilde{\boldsymbol{u}}^+ = \tilde{\boldsymbol{v}}^+ / \left\|\tilde{\boldsymbol{v}}^+\right\|_2. \tag{2.2.30}$$

It turns out that the FastICA algorithm converges extremely rapidly (actually at a *cubic* rate) to columns of the dictionary $\boldsymbol{U}$; interested readers may consult [HO97] for details. Comparing the FastICA algorithm (2.2.29) to the power method studied in Section 2.1.1 for the PCA problem and the MSP algorithm (2.2.18), we notice a striking similarity. Indeed, FastICA is essentially a modified power method, involving the gradient of the empirical kurtosis rather than the simpler linear gradient of the PCA objective.

2.3 A Mixture of Overcomplete Low-Dimensional Subspaces

As we have seen, complete dictionary learning enjoys an elegant computational theory in which we maintain a symmetric autoencoding structure $\mathcal{E}(\boldsymbol{x}) = \boldsymbol{U}^\top \boldsymbol{x}$, $\mathcal{D}(\boldsymbol{z}) = \boldsymbol{U}\boldsymbol{z}$, with a scalable power-method-like algorithm (the MSP algorithm) for learning an orthogonal dictionary/codebook $\boldsymbol{U}$ from data. Nevertheless, for learning representations of general high-dimensional data distributions, one must expand the size of the codebook beyond the orthogonality requirement—meaning that we must have $\boldsymbol{A} \in \mathbb{R}^{D\times m}$, with $m \gg D$, corresponding to the case of an *overcomplete* dictionary/codebook,[10] and the signal model

$$\boldsymbol{x} = \boldsymbol{A}\boldsymbol{z} + \boldsymbol{\varepsilon}, \quad \|\boldsymbol{z}\|_0 = d \ll m. \tag{2.3.1}$$

There are both geometric and physical/modeling motivations for passing to the overcomplete case. Geometrically, recall that in our original reduction from the mixture of subspace data model to the sparse dictionary model, a mixture of K subspaces in $\mathbb{R}^D$, each of dimension d, led to a dictionary of shape $\boldsymbol{A} \in \mathbb{R}^{D\times Kd}$. In other words, overcomplete dictionaries correspond to *richer* mixtures of subspaces, with more distinct modes of variability for modeling the high-dimensional data distribution. On the modeling side, we may run a computational experiment on real-world data that reveals the additional modeling power conferred by an overcomplete representation.

Example 2.2. Given sampled images of hand-written digits, Figure 2.6(a) shows the result of fitting an orthogonal dictionary to the dataset. In contrast, Figure 2.6(b) shows the result of running an optimization algorithm for learning overcomplete dictionaries (which we will present in detail later in the Chapter) on these samples. Notice that the representations become far sparser and the codebooks far more interpretable—they consist of fundamental primitives for the strokes composing the digits, i.e. oriented edges. ■

In fact, overcomplete dictionary learning was originally proposed as a biologically plausible algorithm for image representation based on empirical evidence of how early stages of the visual cortex represent visual stimuli [OF97; OF96].

In the remainder of this section, we will overview the conceptual and computational foundations of overcomplete dictionary learning. Supposing that the

[10] We change the notation here from $\boldsymbol{U}$ to $\boldsymbol{A}$ in order to emphasize the non-orthogonality and non-square-shape of the overcomplete dictionary $\boldsymbol{A}$.

model (2.3.1) is satisfied with sparse codes $\boldsymbol{z}$, overcomplete dictionary $\boldsymbol{A}$, and sparsity level d, and given samples $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]$ of $\boldsymbol{x}$, we want to learn an encoder $\mathcal{E} : \mathbb{R}^D \to \mathbb{R}^m$ mapping each $\boldsymbol{x}$ to its sparse code $\boldsymbol{z}$, and a decoder $\mathcal{D}(\boldsymbol{z}) = \boldsymbol{A}\boldsymbol{z}$ reconstructing each $\boldsymbol{x}$ from its sparse code. In diagram form:

$$\boldsymbol{x} \xrightarrow{\quad\mathcal{E}\quad} \boldsymbol{z} \xrightarrow{\quad\mathcal{D}=\boldsymbol{A}\quad} \hat{\boldsymbol{x}}. \tag{2.3.2}$$

We will start from the construction of the encoder $\mathcal{E}$. We will work incrementally: first, *given the true dictionary* $\boldsymbol{A}$, we will show how the problem of *sparse coding* gives an elegant, scalable, and provably-correct algorithm for recovering the sparse code $\boldsymbol{z}$ of $\boldsymbol{x}$. Although this problem is NP-hard in the worst case, it can be solved efficiently and scalably for dictionaries $\boldsymbol{A}$ which are *incoherent*, i.e. having columns that are not too correlated. The encoder architecture encompassed by this solution will no longer be symmetric: we will see it has the form of a primitive deep network, which depends on the dictionary $\boldsymbol{A}$.

Then we will proceed to the task of learning the decoder $\mathcal{D}$, or equivalently the overcomplete dictionary $\boldsymbol{A}$. We will derive an algorithm for overcomplete dictionary learning that allows us to simultaneously learn the codebook $\boldsymbol{A}$ and the sparse codes $\boldsymbol{z}$, using ideas from sparse coding. Finally, we will discuss a more modern perspective on learnable sparse coding that leads us to a fully asymmetric encoder-decoder structure, as an alternative to (2.3.2). Here, the decoder will correspond to an incremental solution to the sparse dictionary learning problem, and yield a pair of deep network-like encoder decoders for sparse dictionary learning. This structure will foreshadow many developments to come in the remainder of the monograph, as we progress from analytical models to modern neural networks.

2.3.1 Sparse Coding with an Overcomplete Dictionary

In this section, we will consider the data model (2.3.1), which accommodates sparse linear combinations of many motifs, or *atoms*. Given data $\{\boldsymbol{x}_i\}_{i=1}^N \subseteq \mathbb{R}^D$ satisfying this model, i.e. expressible as

$$\boldsymbol{x}_i = \boldsymbol{A}\boldsymbol{z}_i + \boldsymbol{\varepsilon}_i, \qquad \forall i \in [N] \tag{2.3.3}$$

for some dictionary $\boldsymbol{A} \in \mathbb{R}^{D\times m}$ with m atoms, sparse codes $\boldsymbol{z}_i$ such that $\|\boldsymbol{z}_i\|_0 \leq d$, and small errors $\boldsymbol{\varepsilon}_i$, the sparse coding problem is to recover the codes $\boldsymbol{z}_i$ as accurately as possible from the data $\boldsymbol{x}_i$, given the dictionary $\boldsymbol{A}$. Efficient algorithms to solve this problem succeed when the dictionary $\boldsymbol{A}$ is *incoherent* in the sense that the inner products $\boldsymbol{a}_i^\top \boldsymbol{a}_j$ are uniformly small, hence the atoms are nearly orthogonal.[11]

Note that we can collect the $\boldsymbol{x}_i$ into $\boldsymbol{X} = \big[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\big] \in \mathbb{R}^{D\times N}$, collect the $\boldsymbol{z}_i$ into $\boldsymbol{Z} = \big[\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\big] \in \mathbb{R}^{d\times N}$, and collect the $\boldsymbol{\varepsilon}_i$ into $\boldsymbol{E} = \big[\boldsymbol{\varepsilon}_1, \ldots, \boldsymbol{\varepsilon}_N\big] \in \mathbb{R}^{D\times N}$, to rewrite (2.3.3) as

$$\boldsymbol{X} = \boldsymbol{A}\boldsymbol{Z} + \boldsymbol{E}. \tag{2.3.4}$$

[11] As it turns out, in a high-dimensional space, it is rather easy to pack a number of nearly orthogonal vectors that is much larger than the ambient dimension [WM22].

A natural approach to solving the sparse coding problem is to seek the sparsest signals that are consistent with our observations, and this naturally leads to the following optimization problem:

$$\min_{\tilde{\boldsymbol{Z}} \in \mathbb{R}^{d \times N}} \left\{ \|\boldsymbol{X} - \boldsymbol{A}\tilde{\boldsymbol{Z}}\|_F^2 + \lambda \|\tilde{\boldsymbol{Z}}\|_1 \right\}, \tag{2.3.5}$$

where the ℓ^1 norm $\|\tilde{\boldsymbol{Z}}\|_1$, taken elementwise, is known to promote sparsity of the solution [WM22]. This optimization problem is known as the LASSO problem.

However, unlike PCA or the complete dictionary learning problem, there is no clear power iteration-type algorithm to compute the optimal $\boldsymbol{Z}^\star$. A natural alternative is to consider solving the above optimization problem with gradient descent type algorithms. Let $\mathcal{L}(\tilde{\boldsymbol{Z}}) = \|\boldsymbol{X} - \boldsymbol{A}\tilde{\boldsymbol{Z}}\|_2^2 + \lambda\|\tilde{\boldsymbol{Z}}\|_1$. Conceptually, we could try to compute $\boldsymbol{Z}^\star$ with the following iterations:

$$\tilde{\boldsymbol{Z}}_{t+1} \leftarrow \tilde{\boldsymbol{Z}}_t + \eta \nabla \mathcal{L}(\tilde{\boldsymbol{Z}}_t). \tag{2.3.6}$$

However, because the term associated with the ℓ^1 norm $\|\tilde{\boldsymbol{Z}}\|_1$ is non-smooth, we cannot just run gradient descent. For this type of function, we need to replace the gradient $\nabla\mathcal{L}(\tilde{\boldsymbol{Z}})$ with something that generalizes the notion of gradient, known as the subgradient $\partial\mathcal{L}(\tilde{\boldsymbol{Z}})$:

$$\tilde{\boldsymbol{Z}}_{t+1} \leftarrow \tilde{\boldsymbol{Z}}_t + \eta \partial \mathcal{L}(\tilde{\boldsymbol{Z}}_t). \tag{2.3.7}$$

However, it is known that the convergence of subgradient descent is usually very slow. Hence, for this type of optimization problems, it is conventional to adopt a so-called *proximal gradient descent*-type algorithm. We give a technical overview of this method in Section A.1.3.

Applying proximal gradient to the LASSO objective function (2.3.5) leads to the classic *iterative shrinkage-thresholding algorithm* (ISTA), which implements the iteration

$$\tilde{\boldsymbol{Z}}_1 \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}), \tag{2.3.8}$$

$$\tilde{\boldsymbol{Z}}_{t+1} = S_{\eta\lambda}\Big(\tilde{\boldsymbol{Z}}_t - 2\eta \boldsymbol{A}^\top(\boldsymbol{A}\tilde{\boldsymbol{Z}}_t - \boldsymbol{X})\Big), \quad \forall t \geq 1, \tag{2.3.9}$$

with step size $\eta \geq 0$, and the soft-thresholding operator S_α defined on scalars by

$$S_\alpha(x) \doteq \begin{cases} x - \alpha, & x \geq \alpha, \\ 0, & -\alpha < x < \alpha, \\ x + \alpha, & x \leq -\alpha \end{cases} \tag{2.3.10}$$

$$= \operatorname{sign}(x) \max\{|x| - \alpha, 0\}, \tag{2.3.11}$$

and applied element-wise to the input matrix. As a proximal gradient descent algorithm applied to a convex problem, convergence to a global optimum is assured, and a precise convergence rate can be derived straightforwardly [WM22].

We now take a moment to remark on the functional form of the update operator in (2.3.9). It takes the form

$$\tilde{\boldsymbol{Z}}_{t+1} = \texttt{nonlinearity}(\tilde{\boldsymbol{Z}}_t + \texttt{linear}^\top(\texttt{linear}(\tilde{\boldsymbol{Z}}_t) + \texttt{bias})). \tag{2.3.12}$$

This functional form is very similar to that of a residual network layer, namely,

$$\tilde{\boldsymbol{Z}}_{t+1} = \tilde{\boldsymbol{Z}}_t + \texttt{linear}_1^\top(\texttt{nonlinearity}(\texttt{linear}_2(\tilde{\boldsymbol{Z}}_t) + \texttt{bias}_1) + \texttt{bias}_2, \tag{2.3.13}$$

only decoupling the linear maps (i.e., matrix multiplications), adding a bias, and moving the nonlinearity. The nonlinearity in (2.3.9) is notably similar to the commonly-used ReLU activation function $x \mapsto \max\{x, 0\}$ in deep learning—as visualized in Figure 2.7, the soft-thresholding operator is like a "signed" ReLU activation, which is also shifted by a bias. The ISTA, then, can be viewed as a forward pass of a primitive (recurrent one-layer) neural network. We argue in Chapter 5 that such operations are essential to deep representation learning.

2.3.2 Overcomplete Dictionary Learning

Recall that we have the data model

$$\boldsymbol{X} = \boldsymbol{A}\boldsymbol{Z} + \boldsymbol{E}, \tag{2.3.14}$$

where $\boldsymbol{Z}$ is sparse, and our goal previously was to estimate $\boldsymbol{Z}$ given knowledge of the data $\boldsymbol{X}$ and the dictionary atoms $\boldsymbol{A}$. Now we turn to the more practical and more difficult case where we do not know either $\boldsymbol{A}$ or $\boldsymbol{Z}$ and seek to learn them from a large dataset.

A direct generalization of Equation (2.3.5) suggests solving the problem

$$\min_{\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{Z}}} \left\{ \|\boldsymbol{X} - \tilde{\boldsymbol{A}}\tilde{\boldsymbol{Z}}\|_F^2 + \lambda \|\tilde{\boldsymbol{Z}}\|_1 \right\}. \tag{2.3.15}$$

However, the bilinear term in Equation (2.3.15) introduces a scale ambiguity that hinders convergence: given any point $(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{Z}})$ and any constant $c > 0$, the substitution $(c^{-1}\tilde{\boldsymbol{A}}, c\tilde{\boldsymbol{Z}})$ gives loss value

$$\|\boldsymbol{X} - \tilde{\boldsymbol{A}}\tilde{\boldsymbol{Z}}\|_F^2 + c\lambda \|\tilde{\boldsymbol{Z}}\|_1. \tag{2.3.16}$$

This loss is evidently minimized over c by taking $c \to 0$, which corresponds to making the rescaled dictionary $c^{-1}\tilde{\boldsymbol{A}}$ go 'to infinity'. In particular, the iterates of any optimization algorithm solving (2.3.15) will not converge.

This issue with (2.3.15) is dealt with by adding a constraint on the scale of the columns of the dictionary $\tilde{\boldsymbol{A}}$. For example, it is common to assume that each column $\boldsymbol{A}_j$ of the dictionary $\boldsymbol{A}$ in (2.3.14) has bounded ℓ^2 norm—say, without loss of generality, by 1. We then enforce this as a constraint, giving the objective

$$\min_{\tilde{\boldsymbol{Z}}, \tilde{\boldsymbol{A}} : \|\tilde{\boldsymbol{A}}_j\|_2 \le 1} \left\{ \|\boldsymbol{X} - \tilde{\boldsymbol{A}}\tilde{\boldsymbol{Z}}\|_F^2 + \lambda \|\tilde{\boldsymbol{Z}}\|_1 \right\}. \tag{2.3.17}$$

Constrained optimization problems such as (2.3.17) can be solved by a host of sophisticated algorithms [NW06]. However, a simple and scalable one is actually furnished by the same proximal gradient descent algorithm that we used to solve the sparse coding problem in the previous section. We can encode each constraint as additional regularization term, via the characteristic function for the constraint set—details are given in Example A.1. Applying proximal gradient descent to the resulting regularized problem is equivalent to *projected gradient descent*, in which, at each iteration, the iterates after taking a gradient descent step are projected onto the constraint set.

Remark 2.12 (ℓ^4 maximization versus ℓ^1 minimization). Note that the above problem formulation follows naturally from the LASSO formulation (2.3.5) for sparse coding. We promote the sparsity of the solution via the ℓ^1 norm. Nevertheless, if we are only interested in recovering the over-complete dictionary $\boldsymbol{A}$, the ℓ^4 maximization scheme introduced in Section 2.2.2 also generalizes to the over-complete case, without any significant modification. Interested readers may refer to the work of [QZL+20a].

The above problem (2.3.17), which we call *overcomplete dictionary learning*, is nonconvex as here both $\boldsymbol{A}$ and $\boldsymbol{Z}$ are unknown. It cannot be solved easily by the standard convex optimization toolkit. Nevertheless, because it is interesting, simple to state, and practically important, there have been many important works dedicated to different algorithms and theoretical analysis for this problem. Here, for the interest of this manuscript, we present an idiomatic approach to solve this problem which is closer to the spirit of deep learning.

From our experience with the LASSO problem above, it is easy to see that, for the two unknowns $\boldsymbol{A}$ and $\boldsymbol{Z}$, if we fix one and optimize the other, each subproblem is in fact convex and easy to solve. This naturally suggests that we could attempt to solve the above program (2.3.17) by minimizing against $\boldsymbol{Z}$ or $\boldsymbol{A}$ alternatively, say using gradient descent. Coupled with a natural choice of initialization, this leads to the following iterative scheme:

$$\tilde{\boldsymbol{Z}}^{\ell+1} = S_{\eta\lambda}\Big(\tilde{\boldsymbol{Z}}^{\ell} - 2\eta \boldsymbol{A}_+^\top(\boldsymbol{A}_+\tilde{\boldsymbol{Z}}^{\ell} - \boldsymbol{X})\Big), \quad \tilde{\boldsymbol{Z}}^1 = \boldsymbol{0}, \quad \forall \ell \in [L] \tag{2.3.18}$$

$$\boldsymbol{Z}^+ = \tilde{\boldsymbol{Z}}^L, \tag{2.3.19}$$

$$\tilde{\boldsymbol{A}}_{t+1} = \mathrm{proj}_{\|(\,\cdot\,)_j\|_2 \le 1,\, \forall j}\left(\tilde{\boldsymbol{A}}_t - 2\nu(\tilde{\boldsymbol{A}}_t\boldsymbol{Z}^+ - \boldsymbol{X})(\boldsymbol{Z}^+)^\top\right),$$
$$\text{where} \quad (\tilde{\boldsymbol{A}}_1)_j \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\boldsymbol{0}, \tfrac{1}{D}\boldsymbol{I}), \ \ \forall j \in [m], \qquad \forall t \in [T], \tag{2.3.20}$$

$$\boldsymbol{A}_+ = \tilde{\boldsymbol{A}}_T, \tag{2.3.21}$$

where the projection operation in the update for $\boldsymbol{A}$ ensures each column has at most unit ℓ^2 norm, via $\boldsymbol{A}_j \mapsto \boldsymbol{A}_j / \max\{\|\boldsymbol{A}_j\|_2, 1\}$, and where $\boldsymbol{A}_+$ is initialized with each column i.i.d. $\mathcal{N}(\boldsymbol{0}, \frac{1}{D}\boldsymbol{I})$. The above consists of one 'block' of alternating minimization, and we repeatedly perform such blocks, each with independent initializations, until convergence. Above, we have used two separate indices $\{t\}$ and $\{\ell\}$ to indicate the iterations. As we will see later, this allows us to interpret the two updates separately in the context of deep learning.

Despite the dictionary learning problem being a nonconvex problem, it has been shown that alternating minimization type algorithms indeed converge to the correct solution, at least locally. See, for example, [AAJ+16]. As a practical demonstration, the above algorithm (with $L = T = 1$) was used to generate the results for overcomplete dictionary learning in Figure 2.6.

2.3.3 Learned Deep Sparse Coding

The main insight from the alternating minimization algorithm for overcomplete dictionary learning in the previous section (Equations (2.3.18) and (2.3.20)) is to notice that *when we fix* $\boldsymbol{A}$*, the ISTA update for* $\boldsymbol{Z}^\ell$ (2.3.18) *looks like the forward pass of a deep neural network with weights given by* $\boldsymbol{A}$ *(and* $\boldsymbol{A}^\top$*).* But in general, we do not know the true $\boldsymbol{A}$, and the current estimate $\boldsymbol{A}_+$ could be erroneous. Hence it needs to be further updated using (2.3.20) based on the residual of using the current estimate of the sparse codes $\boldsymbol{Z}^+$ to reconstruct $\boldsymbol{X}$. The alternating minimization algorithm iterates these two procedures until convergence. But we can instead extrapolate, and design other learning procedures by combining these insights with techniques from deep learning. This leads to more interpretable network architectures, which will be a recurring theme throughout this manuscript.

Learned ISTA. The above deep-network interpretation of the alternating minimization is more conceptual than practical, as the process could be rather inefficient and take many layers or iterations to converge. But this is mainly because we try to infer both $\boldsymbol{Z}$ and $\boldsymbol{A}$ from $\boldsymbol{X}$. The problem can be significantly simplified and the above iterations can be made much more efficient in the *supervised* setting, where we have a dataset of input and output pairs $(\boldsymbol{X}, \boldsymbol{Z})$ distributed according to (2.3.14) and we only seek to learn $\boldsymbol{A}^\ell$ for the layerwise learnable sparse coding iterations (2.3.29):[12]

$$\boldsymbol{Z}^{\ell+1} = S_{\eta\lambda}\big(\boldsymbol{Z}^\ell - 2\eta(\boldsymbol{A}^\ell)^\top(\boldsymbol{A}^\ell\boldsymbol{Z}^\ell - \boldsymbol{X})\big), \quad \forall \ell \in [L]. \tag{2.3.22}$$

If we denote the operator for each iteration as $\boldsymbol{Z}^{\ell+1} = f(\boldsymbol{A}^\ell, \boldsymbol{Z}^\ell)$, the above iteration can be illustrated in terms of a diagram:

$$\boldsymbol{X}, \boldsymbol{Z}^1 \xrightarrow{f(\boldsymbol{A}^1, \cdot)} \boldsymbol{Z}^2 \xrightarrow{f(\boldsymbol{A}^2, \cdot)} \boldsymbol{Z}^3 \xrightarrow{f(\boldsymbol{A}^3, \cdot)} \cdots \boldsymbol{Z}^L \xrightarrow{f(\boldsymbol{A}^L, \cdot)} \boldsymbol{Z}^{L+1} \approx \boldsymbol{Z}.$$

Thus, given the sequential architecture, to learn the operator $\boldsymbol{A}^\ell$ at each layer, it is completely natural to learn it, say via back propagation (BP),[13] by minimizing the error between the final code $\boldsymbol{Z}^L$ and the ground truth $\boldsymbol{Z}$:

$$\min_{\{\boldsymbol{A}^\ell\}} \big\|\boldsymbol{Z}^L(\boldsymbol{A}^1, \ldots, \boldsymbol{A}^L) - \boldsymbol{Z}\big\|_2^2. \tag{2.3.23}$$

[12] We drop the "tilde" accents here for concision, as the indexing makes the optimization-dependence clear.

[13] See Section A.2 for a brief description of BP.

This is the basis of the Learned ISTA (LISTA) algorithm [GL10], which can be viewed as the learning algorithm for a deep neural network, which tries to emulate the sparse encoding process from $\boldsymbol{X}$ to $\boldsymbol{Z}$. In particular, it can be viewed as a *simple representation learning algorithm*. In fact, this same methodology can be used as a basis to understand the representations computed in more powerful network architectures, such as transformers. We develop these implications in detail in Chapter 5.

Sparse Autoencoders. The original motivation for overcomplete dictionary learning was to provide a simple generative model for high-dimensional data. We have seen with LISTA that, in addition, iterative algorithms for learning sparsely-used overcomplete dictionaries provide an interpretation for ReLU-like deep networks, which we will generalize in later chapters to more complex data distributions than (2.3.14). But it is also worth noting that even in the modern era of large models, the data generating model (2.3.14) provides a useful practical basis for *interpreting features in pretrained large-scale deep networks*, such as transformers, following the hypothesis that the (non-interpretable, *a priori*) features in these networks consist of sparse "superpositions" of underlying features, which are themselves interpretable [EHO+22b]. These *unsupervised* learning paradigms are generally more data friendly than LISTA, as well, which requires large amounts of labeled $(\boldsymbol{X}, \boldsymbol{Z})$ pairs for supervised training.

We can use our development of the LISTA algorithm above to understand common practices in this field of research. In the most straightforward instantiation (see [GTT+25; HCS+24]), a large number of features from a pretrained deep network h are collected from different inputs $\boldsymbol{x}_i$, which themselves are chosen based on a desired interpretation task.[14] For simplicity, we will use h to denote the pre-selected feature map in question, with D-dimensional features; given N sample inputs, let $\boldsymbol{H} \in \mathbb{R}^{D \times N}$ denote the full matrix of features of h. Then a so-called sparse autoencoder $f : \mathbb{R}^D \to \mathbb{R}^d$, with decoder $g : \mathbb{R}^d \to \mathbb{R}^D$, is trained via the LASSO loss (2.3.5):

$$\min_{f,g} \|\boldsymbol{H} - g(f(\boldsymbol{H}))\|_F^2 + \lambda \|f(\boldsymbol{H})\|_1, \tag{2.3.24}$$

where the sparse autoencoder f takes the form of a one-layer neural network, i.e. $f(\boldsymbol{h}_i) = \sigma(\boldsymbol{W}_{\text{enc}}(\boldsymbol{h}_i - \boldsymbol{b}) + \boldsymbol{b}_{\text{enc}})$, where $\sigma(x) = \max\{x, 0\}$ is the ReLU activation function, and the decoder g is linear, so that $g(\boldsymbol{z}_i) = \boldsymbol{W}_{\text{dec}}\boldsymbol{z} + \boldsymbol{b}$.[15]

[14] For example, the inputs $\boldsymbol{x}_i$ could correspond to texts containing samples of computer code in different programming languages, with our task being to try to identify interpretable features in a transformer feature map h corresponding to different salient aspects of the input, such as the specific programming language (distinct across input "classes") or the need to insert a matching parenthesis at the current position (common across input "classes"). We discuss the use of deep networks, and in particular transformers, for text representation learning in greater detail in Chapter 8.

[15] Since f is a sparsifying operator, we denote it analogously to the previously used f, say in LISTA where f denotes an optimization algorithm to sparsely encode the input. Throughout the book, f will be used to denote such sparse and otherwise structured transformations, including those of *deep network encoders* (Chapters 4 and 5).

The parameterization and training procedure (2.3.24) may initially seem to be an arbitrary application of deep learning to the sparse coding problem, but it is actually highly aligned with the algorithms we have studied above for layerwise sparse coding with a learned dictionary. In particular, recall the LISTA architecture $\boldsymbol{Z}^L = f(\boldsymbol{A}^L, f(\boldsymbol{A}^{L-1}, \cdots, f(\boldsymbol{A}^1, \boldsymbol{X}) \cdots))$. In the special case $L = 2$, we have

$$\boldsymbol{Z}^2 = f(\boldsymbol{A}^1, \boldsymbol{X}) = S_{\eta\lambda}\big(\boldsymbol{Z}^1 - 2\eta(\boldsymbol{A}^1)^\top(\boldsymbol{A}^1\boldsymbol{Z}^1 - \boldsymbol{X})\big). \tag{2.3.25}$$

Let us assume that the sparse codes $\boldsymbol{Z}$ in question are nonnegative, i.e., that $\boldsymbol{Z} \geq \boldsymbol{0}$.[16] Then (see Example A.3), we can consider an equivalent LISTA architecture obtained from the sparse coding objective with an additional nonnegativity constraint on $\boldsymbol{Z}$ as

$$\boldsymbol{Z}^2 = f(\boldsymbol{A}^1, \boldsymbol{X}) = \max\left\{\boldsymbol{Z}^1 - 2\eta(\boldsymbol{A}^1)^\top(\boldsymbol{A}^1\boldsymbol{Z}^1 - \boldsymbol{X}) - \lambda\eta\boldsymbol{1}, 0\right\}, \tag{2.3.26}$$

and after some algebra, express this as

$$\boldsymbol{Z}^2 = f(\boldsymbol{A}^1, \boldsymbol{X}) = \max\left\{2\eta(\boldsymbol{A}^1)^\top + \left(\boldsymbol{Z}^1 - 2\eta(\boldsymbol{A}^1)^\top\boldsymbol{A}^1\boldsymbol{Z}^1 - \lambda\eta\boldsymbol{1}\right), 0\right\}. \tag{2.3.27}$$

Given the ability to change the sparse code initialization $\boldsymbol{Z}^1$ as a learnable parameter (which, in the current framework, must have all columns equal to the same learnable vector), this has the form of a ReLU neural network with learnable bias—identical to the sparse autoencoder f! Moreover, to *decode* the learned sparse codes $\boldsymbol{Z}^2$, it is natural to apply the learned dictionary $\boldsymbol{Z}^2 \mapsto \boldsymbol{A}^1\boldsymbol{Z}^2$. Then the only difference between this and the SAE decoder g is the additional bias $\boldsymbol{b}$, which can technically be absorbed into $\boldsymbol{H}$ and f in the training objective (2.3.24).

Thus, the SAE parameterization and training procedure coincides with LISTA training with $L = 1$, and a modified training objective—using the LASSO objective (2.3.5), which remains *unsupervised*, instead of the supervised reconstruction loss (2.3.23) used in vanilla LISTA. In particular, we can understand the SAE architecture in terms of our interpretation of the LISTA architecture in terms of layerwise sparse coding in (2.3.29). This connection is suggestive of a host of new design strategies for improving practical interpretability methodology, many of which remain tantalizingly unexplored. We begin to lay out some connections to broader autoencoding methodology in Chapter 6.

Layerwise learned sparse coding? In the supervised setting, LISTA provides a deep neural network analogue of the sparse coding iteration, with layerwise-learned dictionaries, inspired by alternating minimization; even in the unsupervised setting, the same methodology can be applied to learning, as with sparse autoencoders. But the connection between low-dimensional-structure-seeking optimization algorithms and deep network architectures goes much deeper than

[16] In the data generating model (2.3.14), an arbitrary dictionary-and-sparse-code pair $(\boldsymbol{A}, \boldsymbol{Z})$ can be replaced by one in which $\boldsymbol{Z} \geq \boldsymbol{0}$ simply by doubling the number of columns in $\boldsymbol{A}$, so from a modeling perspective, this is a very mild assumption.

this, and suggests an array of scalable and natural neural learning architectures which may even be usable without backpropagation.

As a simple illustration, we return the alternating minimization iterations (2.3.18) and (2.3.20). This scheme randomly re-initializes the dictionary $\boldsymbol{A}_1$ on every such update. An improvement uses instead *warm starting*, where the residual is generated using the previous estimate $\boldsymbol{A}_+$ for the dictionary. If we then view each ISTA update (2.3.18) as a layer and allow the associated dictionary, now coupled with the sparse code updates as $\boldsymbol{A}_\ell$, to update in time, this leads to a "layerwise-learnable" sparse coding scheme:

$$\boldsymbol{Z}^1 = \boldsymbol{0}, \quad (\boldsymbol{A}_1)_j \overset{\text{i.i.d.}}{\sim} \mathcal{N}(\boldsymbol{0}, \tfrac{1}{D}\boldsymbol{I}), \ \ \forall j \in [m], \tag{2.3.28}$$

$$\boldsymbol{Z}^{\ell+1} = S_{\eta\lambda}\big(\boldsymbol{Z}^\ell - 2\eta(\boldsymbol{A}_\ell)^\top(\boldsymbol{A}_\ell\boldsymbol{Z}^\ell - \boldsymbol{X})\big), \tag{2.3.29}$$

$$\boldsymbol{A}_{\ell+1} = \boldsymbol{A}_\ell - 2\nu(\boldsymbol{A}_\ell\boldsymbol{Z}^{\ell+1} - \boldsymbol{X})(\boldsymbol{Z}^{\ell+1})^\top. \tag{2.3.30}$$

Note that this iteration corresponds to a relabeling of (2.3.18) and (2.3.20) for $T = L = 1$, over infinitely many blocks. Each of the 'inner' steps updating $\boldsymbol{Z}$ can be considered as a one-layer forward pass, while each of the 'outer' steps updating $\boldsymbol{A}$ can be considered as a one-layer backward pass, of a primitive deep neural network. In particular, this algorithm is the simplest case in which a clear divide between forward optimization and backward learning manifests. This distinction between forward updates to the ephemeral "features" $\boldsymbol{Z}$ of data $\boldsymbol{X}$ versus backward updates to the persistent parameters $\boldsymbol{A}$ is still observed in current neural networks and autoencoders—we will have much more to say about it in Chapter 5 and in Chapter 6.

Notice that the above layer-wise scheme also suggests a plausible alternative to the current end-to-end optimization strategy that primarily relies on back propagation (BP) detailed in Section A.2.3. Freeing training large networks from BP would be one of the biggest challenges and opportunities in the future, as we will discuss more at the end of the book in Chapter 9.

2.4 Summary and Notes

Key Messages. The idealistic models we have presented in this chapter—PCA, ICA, and dictionary learning—were developed over the course of the twentieth century. Many books have been written solely about each method, so we only attempted here to give a broad overview of the key concepts and results. As we will see in coming chapters, although these are arguably rather idealistic simple models for low-dimensional data distributions, the basic ideas and even resulting algorithms to identify such distributions can be naturally generalized or extended to arbitrary low-dimensional distributions – the thesis of this book. In addition, as we will see in coming chapters, these basic (linear) models, to a large extent, can also serve as local prototypes to effectively approximately model general distributions with nonlinear low-dimensional structures.

More History and References. Jolliffe [Jol86] attributes principal component analysis to Pearson [Pea01], and independently Hotelling [Hot33]. In mathematics, the main result on the related problem of low-rank approximation in unitarily invariant norms is attributed to Eckart and Young [EY36], and to Mirsky for full generality [Mir60]. PCA continues to play an important role in research as perhaps the simplest model problem for unsupervised representation learning: as early as the 1980s, works such as Oja [Oja82] and Baldi and Hornik [BH89] used the problem to understand learning in primitive neural networks, and more recently, it has served as a tool for understanding more complex representation learning frameworks, such as diffusion models [WZZ+24].

Independent component analysis was proposed by B. Ans et al. [BJC85] and pioneered by Aapo Hyvärinen in the 1990s and early 2000s in a series of influential works: see Hyvärinen and Oja [HO00b] for a summary. As a simple model for structure that arises in practical data, it initially saw significant use in applications such as blind source separation, where each independent component z_i represents an independent source (such as sound associated to a distinct instrument in a musical recording) that is superimposed to produce the observation $\boldsymbol{x} = \boldsymbol{U}\boldsymbol{z}$.

The problem of dictionary learning can, in the complete or orthogonal case, be seen as one of the foundational problems of twentieth-century signal processing, particularly in linear systems theory, where the Fourier basis plays the key role; from the 1980s onward, the field of computational harmonic analysis crystallized around the study of alternate such dictionaries for classes of signals in which optimal approximation could only be realized in a basis other than Fourier (e.g., wavelets) [DVD+98]. However, the importance of the case of redundant bases, or overcomplete dictionaries, was only highlighted following the pioneering work of Olshausen and Field [OF97; OF96]. Early subsequent work established the conceptual and algorithmic foundations for learning sparsely-used overcomplete dictionaries, often aimed at representing natural images [AEB06; DM03; Don01; EA06; GJB15; MBP14; MK07]. Later, a significant amount of theoretical interest in the problem, as an important and nontrivial model problem for unsupervised representation learning, led to its study by the signal processing, theoretical machine learning, and theoretical computer science communities, in particular focused on conditions under which the problem could be provably and efficiently solved. A non-exhaustive list of notable works in this line include those of Spielman et al. [SWW12] and Sun et al. [SQW17a] on the complete case; Arora et al. [AGM+15]; Barak et al. [BKS15]; and Qu et al. [QZL+20a]. Many deep theoretical questions about this simple-to-state problem remain open, perhaps in part due to a tension with the problem's worst-case NP-hardness (e.g., see Tillmann [Til15]).

One point that we wish to highlight from the study of these classical analytical models for low-dimensional structure is the common role played by various *generalized power methods*—algorithms that very rapidly converge, at least locally, to various types of low-dimensional structures. The terminology for this class of algorithms follows the work of M. Journée et al. [MYP+10]. At a high level, modeled on the classical power iteration for computation of the top

Table 2.1: Summary of (generalized) power methods presented in this chapter.

Problem	Algorithm	Iteration	Type of Structure Enforced
PCA	Power Method	$\boldsymbol{u}_t = \frac{\boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{u}_t}{\|\boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{u}_t\|_2}$	1-dim. subspace (unit vector)
ICA	FastICA	$\boldsymbol{u}_{t+1} = \frac{\frac{1}{N}\boldsymbol{X}(\boldsymbol{X}^\top \boldsymbol{u}_t)^{\circ 3} - 3\boldsymbol{u}_t}{\left\|\frac{1}{N}\boldsymbol{X}(\boldsymbol{X}^\top \boldsymbol{u}_t)^{\circ 3} - 3\boldsymbol{u}_t\right\|_2}$	1-dim. subspace (unit vector)
Complete DL	MSP Algorithm	$\boldsymbol{U}_{t+1} = \mathcal{P}_{\mathrm{O}(D)}[(\boldsymbol{U}_t\boldsymbol{X})^{\circ 3}\boldsymbol{X}^\top]$	D-dim. subspace (ortho. matrix)

eigenvector of a semidefinite matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, that is

$$\boldsymbol{u}_{t+1} = \frac{\boldsymbol{A}\boldsymbol{u}_t}{\|\boldsymbol{A}\boldsymbol{u}_t\|_2}, \tag{2.4.1}$$

this class of algorithms consists of a "powering" operation involving a matrix $\boldsymbol{A}$ associated to the data, along with a "projection" operation that enforces a desired type of structure. Table 2.1 presents a summary of the algorithms we have studied in this chapter that follow this structure. The reader may appreciate the applicability of this methodology to different types of low-dimensional structure, and different losses (i.e., both the quadratic loss from PCA, and the kurtosis-type losses from ICA), as well as the lack of such an algorithm for overcomplete dictionary learning, despite the breadth of the literature on these algorithms. We see the development of power methods for further families of low-dimensional structures, particularly those relevant to applications where deep learning is prevalent, as one of the more important (and open) research questions suggested by this chapter.

The connection we make in Section 2.2.1 between the geometric mixture-of-subspaces distributional assumption and the more analytically-convenient sparse dictionary assumption has been mentioned in prior work, especially by those focused on generalized principal component analysis and applications such as subspace clustering, e.g., work of Vidal et al. [VMS16]. The mixture of subspaces assumption will continue to play a significant role throughout this manuscript, both as an analytical test case for different algorithmic paradigms, and as a foundation for deriving different deep network architectures, as with LISTA in Section 2.3.3, but which can scale to more complex data distributions.

2.5 Exercises and Extensions

Exercise 2.1. Prove that, for any symmetric matrix $\boldsymbol{A}$, the solution to the problem $\max_{\boldsymbol{U}\in\mathsf{O}(D,d)} \operatorname{tr}\left(\boldsymbol{U}^\top \boldsymbol{A}\boldsymbol{U}\right)$ is the matrix $\boldsymbol{U}^\star$ whose columns are the top d unit eigenvectors of $\boldsymbol{A}$.

Exercise 2.2. Let $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2\boldsymbol{I})$ be a Gaussian random variable with independent components, each with variance σ^2. Prove that for any orthogonal matrix $\boldsymbol{Q}$ (i.e., $\boldsymbol{Q}^\top\boldsymbol{Q} = \boldsymbol{I}$), the random variable $\boldsymbol{Q}\boldsymbol{z}$ is distributed identically to $\boldsymbol{z}$. *(Hint: recall the formula for the Gaussian probability density function, and the formula for the density of a linear function of a random variable.)*

Exercise 2.3. The notion of statistical identifiability discussed above can be related to *symmetries* of the model class, allowing estimation to be understood in a purely deterministic fashion without any statistical assumptions.

Consider the model $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{Z}$ for matrices $\boldsymbol{X}, \boldsymbol{U}, \boldsymbol{Z}$ of compatible sizes.

1. Show that if $\boldsymbol{A}$ is any square invertible matrix of compatible size, then the pair $(\boldsymbol{U}\boldsymbol{A}, \boldsymbol{A}^{-1}\boldsymbol{Z})$ also equals $\boldsymbol{X}$ under the model. We call this a $\mathsf{GL}(d)$ *symmetry.*

2. Suppose each column of $\boldsymbol{Z}$ is an independent and identically distributed observation from a common statistical model $\boldsymbol{z}$, which moreover has zero mean and independent components z_i with positive variance. Show that for any square invertible matrix $\boldsymbol{A}$, if $\boldsymbol{A}\boldsymbol{z}$ has uncorrelated components, then $\boldsymbol{A}$ can be written as $\boldsymbol{D}_1\boldsymbol{Q}\boldsymbol{D}_2$, where $\boldsymbol{Q}$ is an orthogonal matrix and $\boldsymbol{D}_1$, $\boldsymbol{D}_2$ are diagonal matrices. *This links the "independence" assumption in ICA to a "symmetry breaking" effect, which only allows scale and rotational symmetries.*

Exercise 2.4. Consider the model $\boldsymbol{x} = \boldsymbol{U}\boldsymbol{z}$, where $\boldsymbol{U} \in \mathbb{R}^{D\times d}$ with $D \geq d$ is fixed and has rank d, and $\boldsymbol{z}$ is a zero-mean random variable. Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ denote i.i.d. observations from this model.

1. Show that the matrix $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]$ has rank no larger than d, and therefore there is an orthonormal matrix $\boldsymbol{V} \in \mathbb{R}^{D\times d}$ so that $\boldsymbol{X} = \boldsymbol{V}\boldsymbol{Y}$, where $\boldsymbol{Y} \in \mathbb{R}^{d\times N}$. (*Hint: use PCA.*)

2. Show that the *whitened matrix* $(\boldsymbol{Y}\boldsymbol{Y}^\top)^{-1/2}\boldsymbol{Y}$ exists in expectation whenever $\mathrm{Cov}(\boldsymbol{z})$ is nonsingular, and that it has identity empirical covariance.[17]

3. Show, by using the singular value decomposition of $\boldsymbol{U}$, that the matrix $\boldsymbol{V}$ can be chosen so that the whitened matrix satisfies $(\boldsymbol{Y}\boldsymbol{Y}^\top)^{-1/2}\boldsymbol{Y} = \boldsymbol{W}[\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N]$, where $\boldsymbol{W}$ is an orthonormal matrix.

Exercise 2.5. Let X and Y be zero-mean independent random variables.

1. Show that $\mathrm{kurt}(X + Y) = \mathrm{kurt}(X) + \mathrm{kurt}(Y)$.

2. For any $\alpha \in \mathbb{R}$, show that $\mathrm{kurt}(\alpha X) = \alpha^4 \mathrm{kurt}(X)$.

Exercise 2.6. Let $f : \mathbb{R}^d \to \mathbb{R}$ be a given twice-continuously-differentiable objective function. Consider the spherically-constrained optimization problem

$$\max_{\|\boldsymbol{u}\|_2^2 = 1} f(\boldsymbol{u}). \tag{2.5.1}$$

In this exercise, we will derive the expressions we gave in the FastICA derivation for maximizing kurtosis over the sphere via a gradient ascent algorithm. These

[17] In particular, it can be proved mathematically that this is enough to guarantee that the whitened matrix exists with high probability whenever $\boldsymbol{z}$ satisfies a suitable concentration inequality and N is sufficiently large.

expressions are special cases of a rich theory of calculus and optimization on manifolds, of which the sphere is a particular example. A deep technical study of this field is out-of-scope for our purposes, so we only mention two key references for the interested reader: the pioneering textbook by Absil, Mahony, and Sepulchre [AMS09], and a more recent introductory treatise by Boumal [Bou23].

1. For any constraint set $\mathcal{M}$ that is a differentiable submanifold of $\mathbb{R}^d$, the *tangent space* at a point $\boldsymbol{u} \in \mathcal{M}$ is, informally, the best local linear approximation to the manifold $\mathcal{M}$ at the point $\boldsymbol{u}$. In the important special case where $\mathcal{M}$ is defined locally at $\boldsymbol{u}$ as a level set of a function $F : \mathbb{R}^d \to \mathbb{R}$, that is
$$U \cap \mathcal{M} = F^{-1}(\{0\})$$
for some open set $U \subset \mathcal{M}$ with $\boldsymbol{u} \in U$, the tangent space to $\mathcal{M}$ at $\boldsymbol{u}$ can be calculated via differentiation:
$$T_{\boldsymbol{u}}\mathcal{M} = \operatorname{Ker}(DF_{\boldsymbol{u}}).$$
It is easily seen that the sphere has the defining equation $F(\boldsymbol{u}) = \|\boldsymbol{u}\|_2^2 - 1$. Show, using these facts, that the tangent space to the sphere at $\boldsymbol{u}$ is given by
$$T_{\boldsymbol{u}}\mathbb{S}^{d-1} = \{\boldsymbol{v} \in \mathbb{R}^d \mid \langle \boldsymbol{v}, \boldsymbol{u} \rangle = 0\},$$
and that the orthogonal projection onto this subspace is $\boldsymbol{P}_{\boldsymbol{u}}^{\perp} = \boldsymbol{I} - \boldsymbol{u}\boldsymbol{u}^{\top}$.

2. The vector field
$$\operatorname{grad} f(\boldsymbol{u}) = \boldsymbol{P}_{\boldsymbol{u}}^{\perp} \nabla f \tag{2.5.2}$$
is known as the *Riemannian gradient* of the function f restricted to the sphere. The *first order optimality conditions* for the optimization problem (2.5.1) can be expressed in terms of the Riemann gradient:
$$\operatorname{grad} f(\boldsymbol{u}) = \boldsymbol{0}.$$
Geometrically, this says that the Euclidean gradient of f at $\boldsymbol{u}$ must be orthogonal to the tangent space to the sphere at $\boldsymbol{u}$. Now suppose $\boldsymbol{v} \in \mathbb{R}^d$ is nonzero. Show that
$$\operatorname{proj}_{\mathbb{S}^{d-1}}(\boldsymbol{v}) \doteq \min_{\|\boldsymbol{u}\|_2^2 = 1} \|\boldsymbol{u} - \boldsymbol{v}\|_2 = \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|_2},$$
using the first-order optimality conditions.

3. In optimization over $\mathbb{R}^d$, one checks the second-order optimality conditions (to determine whether a critical point is a maximizer, a minimizer, or a saddle point) using the *Hessian matrix* $\nabla^2 f(\boldsymbol{u})$. Show, by differentiating the Riemann gradient $\operatorname{grad} f(\boldsymbol{u})$ for the sphere with respect to $\boldsymbol{u}$ as in the first part of this exercise, that the corresponding object for determining second-order optimality conditions for sphere-constrained optimization is the *Riemannian Hessian*, defined as
$$\operatorname{Hess} f(\boldsymbol{u}) = \boldsymbol{P}_{\boldsymbol{u}}^{\perp} \left(\nabla^2 f(\boldsymbol{u}) - \langle \nabla f(\boldsymbol{u}), \boldsymbol{u} \rangle \boldsymbol{I} \right) \boldsymbol{P}_{\boldsymbol{u}}^{\perp}. \tag{2.5.3}$$

Exercise 2.7. In this exercise, we sketch an argument referred to in the literature as a *landscape analysis* for the spherically-constrained population kurtosis maximization problem (2.2.24). We will show that when there is at least one independent component with positive kurtosis, its global maximizers indeed lead to the recovery of one column of the dictionary $\boldsymbol{U}$. For simplicity, we will assume that $\operatorname{kurt}(z_i) \neq 0$ for each $i = 1, \dots, d$.

1. Using the results of Part 1 of Exercise 2.6, show that the first-order optimality condition for (2.2.24) is

$$\left(\sum_{i=1}^{d} \operatorname{kurt}(z_i) w_i^4 \right) \boldsymbol{w} = \operatorname{kurt}(\boldsymbol{z}) \odot \boldsymbol{w}^{\odot 3}, \tag{2.5.4}$$

   where the kurtosis is calculated elementwise, $\odot$ denotes elementwise multiplication of vectors and $\boldsymbol{w}^{\odot 3}$ denotes the elementwise cube of its argument.

2. Show that the vectors $\boldsymbol{w}$ with unit norm that also satisfy (2.5.4) all take the following form. Let $S^+ = \{i \in [d] \mid \operatorname{kurt}(z_i) > 0\}$, and $S^- = \{i \in [d] \mid \operatorname{kurt}(z_i) < 0\}$. Let S be a subset either of S^+ or S^-. Then

$$\boldsymbol{w}_S = \sum_{i \in S} \pm \sqrt{\frac{1}{\operatorname{kurt}(z_i) \sum_{j \in S} \frac{1}{\operatorname{kurt}(z_j)}}} \boldsymbol{e}_i \tag{2.5.5}$$

   satisfies (2.5.4), where $\boldsymbol{e}_i$ is the vector with a 1 in the i-th position and 0s elsewhere, and the $\pm$ sign denotes the choice of either a positive or negative sign.

3. Assume that there is at least one i such that $\operatorname{kurt}(z_i) > 0$. Using the results of Part 2 of Exercise 2.6, show that the only local maxima of the objective of (2.2.24) are the signed one-sparse vectors $\pm \boldsymbol{e}_i$ with $i \in S^+$. Conclude that the global maximizers of (2.2.24) are the signed one-sparse vectors corresponding to components with maximum kurtosis. (*Hint: count the number of positive and negative eigenvalues of the Riemannian Hessian* (2.5.3) *at each critical point.*)

4. Now assume that $\operatorname{kurt}(z_j) < 0$ for every $j = 1, \dots, d$. This corresponds to an "over-deflated" instantiation of the kurtosis maximization problem. Using again the results of Part 2 of Exercise 2.6, show that the only local maxima of the objective of (2.2.24) are the signed dense vectors $\sum_{i=1}^{d} \pm \boldsymbol{e}_i$. This shows that the optimization formulation (2.2.24) cannot be applied naively.

Exercise 2.8. This exercise follows the structure and formalism introduced in Exercise 2.6, but applies it instead to the orthogonal group $\mathsf{O}(d) = \{\boldsymbol{U} \in \mathbb{R}^{d \times d} \mid \boldsymbol{U}^\top \boldsymbol{U} = \boldsymbol{I}\}$. Consult the description of Exercise 2.6 for the necessary conceptual background; the formalism applies identically to the case where the ambient space is the set of $d \times d$ matrices as long as one recalls that the relevant inner product on matrices is $\langle \boldsymbol{X}, \boldsymbol{Y} \rangle = \operatorname{tr}(\boldsymbol{X}^\top \boldsymbol{Y})$. An excellent general reference for

facts about optimization on the orthogonal group is Edelman, Arias, and Smith [EAS98].

Let $f : \mathbb{R}^{d\times d} \to \mathbb{R}$ be a given twice-continuously-differentiable objective function. Consider the orthogonally-constrained optimization problem

$$\max_{\boldsymbol{Q}^\top \boldsymbol{Q} = \boldsymbol{I}} f(\boldsymbol{Q}). \tag{2.5.6}$$

1. It is easily seen that the orthogonal group has the defining equation $F(\boldsymbol{Q}) = \boldsymbol{Q}^\top \boldsymbol{Q} = \boldsymbol{I}$. Show, using this fact, that the tangent space to the orthogonal group at $\boldsymbol{Q}$ is given by

   $$T_{\boldsymbol{Q}}\mathsf{O}(d) = \{\boldsymbol{Q}\boldsymbol{\Omega} \in \mathbb{R}^{d\times d} \mid \boldsymbol{\Omega}^\top = -\boldsymbol{\Omega}\},$$

   and that the orthogonal projection onto this subspace is

   $$\mathcal{P}_{T_{\boldsymbol{Q}}\mathsf{O}(d)}(\boldsymbol{\Delta}) = \boldsymbol{Q}\,\mathrm{skew}(\boldsymbol{Q}^\top \boldsymbol{\Delta}),$$

   where $\mathrm{skew}(\boldsymbol{\Delta}) = \frac{1}{2}(\boldsymbol{\Delta} - \boldsymbol{\Delta}^\top)$ is the orthogonal projection onto the set of skew-symmetric matrices. The vector field

   $$\mathrm{grad}\, f(\boldsymbol{Q}) = \mathcal{P}_{T_{\boldsymbol{Q}}\mathsf{O}(d)}\left(\nabla f(\boldsymbol{Q})\right) \tag{2.5.7}$$

   is known as the *Riemannian gradient* of the function f restricted to the orthogonal group. The *first order optimality conditions* for the optimization problem (2.5.6) can be expressed in terms of the Riemann gradient:

   $$\mathrm{grad}\, f(\boldsymbol{Q}) = \boldsymbol{0}.$$

2. Show, by differentiating the Riemann gradient $\mathrm{grad}\, f(\boldsymbol{Q})$ for the orthogonal group with respect to $\boldsymbol{Q}$ as in the first part of this exercise, that the *Riemannian Hessian* is given by

   $$\mathrm{Hess}\, f(\boldsymbol{Q}) = \mathcal{P}_{T_{\boldsymbol{Q}}\mathsf{O}(d)}\left(\nabla^2 f(\boldsymbol{Q}) - \mathrm{sym}(\boldsymbol{Q}^\top \nabla f(\boldsymbol{Q})) \otimes \boldsymbol{I}\right) \mathcal{P}_{T_{\boldsymbol{Q}}\mathsf{O}(d)}, \tag{2.5.8}$$

   where $\mathrm{sym}(\boldsymbol{\Delta}) = \frac{1}{2}(\boldsymbol{\Delta} + \boldsymbol{\Delta}^\top)$ denotes the orthogonal projection onto the set of symmetric matrices, and $\otimes$ denotes the Kronecker product of matrices. Take care to interpret the operators appearing in the previous expression as *linear transformations on $d \times d$ matrices*, **not** as $d \times d$ matrices themselves. The *second-order optimality conditions* for the optimization problem (2.5.6) can be expressed in terms of the Riemann Hessian:

   $$\mathrm{Hess}\, f(\boldsymbol{Q}) \preceq \boldsymbol{0}.$$

   For a minimization problem, the sign is reversed.

   (*Hint: The key is to manipulate one's calculations to obtain the form* (2.5.8)*, which is as compact as possible. To this end, make use of the following*

isomorphism of the Kronecker product: if $\boldsymbol{A}$*,* $\boldsymbol{X}$*, and* $\boldsymbol{B}$ *are matrices of compatible sizes, then one has*

$$(\boldsymbol{B}^\top \otimes \boldsymbol{A})\,\mathrm{vec}(\boldsymbol{X}) = \mathrm{vec}(\boldsymbol{A}\boldsymbol{X}\boldsymbol{B}),$$

where $\mathrm{vec}(\boldsymbol{X})$ *denotes the "left-to-right" stacking of the columns of the matrix argument into a vector. We use this isomorphism in* (2.5.8) *in order to define the Kronecker product of two matrices as an operator on matrices in a canonical way.*)

3. Now suppose $\boldsymbol{X} \in \mathbb{R}^{d\times d}$ is full-rank. In this and the next part of the exercise, we consider the projection onto the orthogonal group of $\boldsymbol{X}$:

$$\mathrm{proj}_{\mathsf{O}(d)}(\boldsymbol{X}) \doteq \min_{\boldsymbol{Q}\in\mathsf{O}(d)} \|\boldsymbol{Q}-\boldsymbol{X}\|_F^2. \tag{2.5.9}$$

We will prove that the solution to this problem is given by

$$\mathrm{proj}_{\mathsf{O}(d)}(\boldsymbol{X}) = \boldsymbol{U}\boldsymbol{V}^\top,$$

where $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$ is a singular value decomposition of $\boldsymbol{X}$.

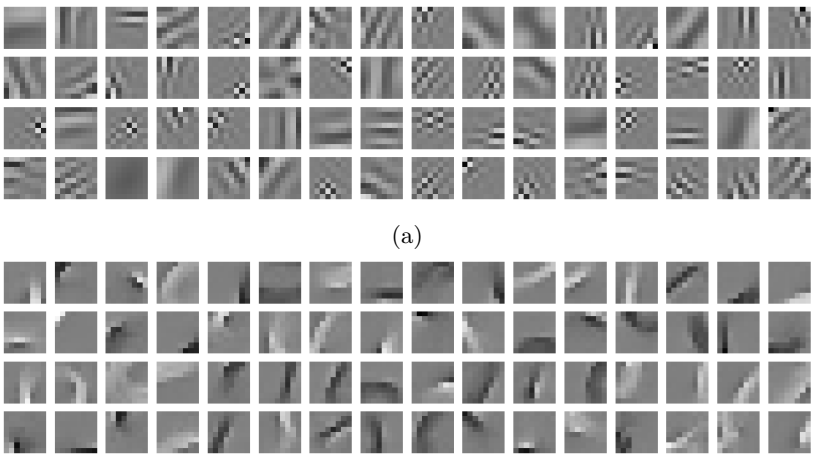
1. Using the first and second-order optimality conditions, show that every local minimizer $\boldsymbol{Q}$ of (2.5.9) satisfies

$$\begin{aligned}(\boldsymbol{Q}^\top\boldsymbol{X})^\top &= \boldsymbol{Q}^\top\boldsymbol{X},\\ \boldsymbol{Q}^\top\boldsymbol{X} &\succeq \boldsymbol{0}.\end{aligned}$$

(*Hint: use linearity of the Kronecker product in either of its two arguments when the other is fixed.*)

2. Using these conditions, argue that at every local minimizer $\boldsymbol{Q}$ of (2.5.9), one has $\boldsymbol{Q}^\top\boldsymbol{X} = (\boldsymbol{X}^\top\boldsymbol{X})^{1/2}$. (*Hint: Use the fact from linear algebra that if* $\boldsymbol{S}\succeq\boldsymbol{0}$ *is a symmetric positive semidefinite matrix, then* $(\boldsymbol{S}^\top\boldsymbol{S})^{1/2} = \boldsymbol{S}$.)
3. Using the singular value decomposition $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$, conclude that

$$\boldsymbol{U}\boldsymbol{V}^\top = \mathrm{proj}_{\mathsf{O}(d)}(\boldsymbol{X}).$$



(a)

(b)

Figure 2.6: Comparison of learned dictionary atoms for complete (orthogonal) and overcomplete dictionaries, trained to reconstruct 8 by 8 patches taken from MNIST digits. Both dictionaries are trained for 6000 epochs on 10^4 random patches with nontrivial content, and sparse codes are computed with the LASSO objective and $\lambda = 0.1$ (see (2.3.5)). Colormaps have black for negative values, and white for positive values. **Top:** An orthogonal dictionary learned with the MSP algorithm (2.2.18) is constrained to have no more than 64 atoms; the learned atoms roughly correspond to a "spike and slab" dictionary, and achieve relatively poor reconstruction sparsity levels on held-out test data (codes are approximately 17-sparse on average, with respect to a threshold of 10^{-1}). **Bottom:** In contrast, an overcomplete dictionary (here, with 8^3 atoms; we visualize a random subset of 64) learns semantically-meaningful dictionary atoms corresponding to signed oriented edges, which can be pieced together to create digit patches and achieve superior reconstruction and sparsity levels. Codes are approximately 20-sparse on average, while being 8 times larger than those of the orthogonal dictionary. To compute the dictionary, we use an optimizer based on proximal alternating linearized minimization on a suitably-regularized version of (2.3.17).

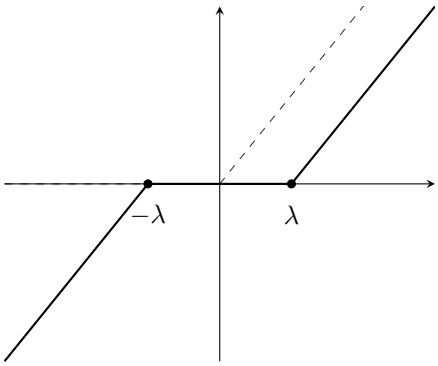


Figure 2.7: The graph of the soft thresholding function S_λ (solid) compared to the ReLU activation $x \mapsto \max\{x, 0\}$ (dashed). Between $-\lambda$ and $+\lambda$, the value of S_λ is clipped to 0, while large inputs' (absolute) value is simply shrunk by λ. Unlike ReLU, S_λ is symmetric and acts on both positive and negative inputs.